SUBSET SELECTION ROUTING: MODELLING AND HEURISTICS

A thesis presented in partial fulfilment
of the requirements for
the degree of Doctor of Philosophy in
Operations Research at
Massey University

Ian R. Beale

2002

Abstract

This theoretically practical thesis relates to the field of subset selection routing problems, in which there are a set of customers available to be serviced and decisions of which customers to service and how to carry out the service are required. We develop models for problems of this kind, particularly accounting for customer service in decision making. We consider possibilities that relate to not servicing customers, or servicing them poorly with respect to their desired service, and we model some possible implications of these decisions. We consider different constraints that may appear within these problems and exploit these within an overall model for a problem which we term the Maximum Collection Problem.

We develop effective, generic solution methods for these problems and tailor specific routines to certain types of problem. We devise new methods for generating problems with specific characteristics and we use these to test the effectiveness of our methods. We extensively test our methods, identify shortcomings of existing methods and develop new methods for overcoming the identified weaknesses of the methods.

We introduce a new version of subset selection routing problems, involving decision making in dynamic situations. We create models involving next day and same day service and develop fast, practical methods for obtaining effective solutions to these problems and test their effectiveness and robustness on a number of varied test problems.

Acknowledgements

As with all theses, there has to be a lot of support in order for the completion of the thesis to be more than just a pipedream. I would like to thank everyone who has contributed in whatever way, be it encouragement or whether their disparaging remarks have hardened my resolve to complete this puppy.

All up then, I would like to particularly thank:

- my parents, Syd and Cela, for their emotional and financial support.
- my chief supervisor John Giffin for his philosphical and whimsical support. Plus I would like to thank my other supervisors, Chin Diew Lai and Mark Bebbington for their signatorial support.
- Mark Johnston for his technical and practical support.
- Ron Diamond @ `www.rondiamond.com` for his acoustic support.
- Stephen and Debra White, and family, for their nutritional and accommodational support.
- the Villa, Braves, Rams, Colts and White Sox for giving me a reason to come to Massey each day, which had a positive externality of me getting some work done.

Table of Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

Many practical problems are modelled within the field of Operations Research (OR), in pursuit of enabling effective decision making for difficult problems. One area of OR that has attracted a lot of attention is vehicle routing, in which the objective is to devise effective solution routes that service a set of customers. One assumption that is commonly made with vehicle routing problems is that all the available customers must be serviced, and this assumption requires the service company to have the capacity to be able to handle all the service requests.

In this thesis we are interested in the case where the service company may be operating under a situation of heavy demand, where the demand may not be able to be handled, given the current set of vehicles that are available for servicing. If no fleet expansion is possible during the planning period, this case requires judicious decisions to be made, as to which customers to service, with different criteria required for different situations. The problem then becomes one of selecting which customers to service and determining how to best carry out the service.

We develop theoretical models for problems of this kind, particularly accounting for customer service in decision making. We use this approach in order to address some of the issues relating to the practicality of the developed models. Previous studies in this area have ignored the consideration of customer service within their models, and we see this as an important area to address. We consider possibilities in whcih poor service or non-service is allowed, and we model some possible implications of these decisions. We consider different constraints that may appear within these problems and exploit these within our overall models.

Our other major area of concern is with the development of practical solution methods that are designed to obtain good solutions to reasonable sized problems within an acceptable amount of computational time. With this in mind, we look to develop fast methods for solving problem instances, and we develop generic solution methods that are able to be used with a wide range of problems. We also attempt to characterize some of the problem types considered, in order to be able to tailor specific methods to specific types of problem instances.

This thesis therefore involves the development of a model for a new, general routing problem, which we call the Maximum Collection Problem (MCP). In the MCP there are a set of customers, each with a reward for servicing and/or a penalty for unsatisfactory service, and there is an objective relating to the maximization of the net rewards. This problem involves the routing of service vehicles to meet customer requirements, and we introduce the fundamentals of vehicle routing and scheduling problems in Chapter 2. Other elements of the MCP are discussed in Chapters 3, 4 and 5, with these including subset selection, time considerations and different forms of customers. An overall model for the MCP, together with its potential implications and appropriate heuristic solution methods, is proposed in Chapter 6; this model enables the MCP to be considered in the context of existing vehicle routing and scheduling problems. In Chapter 7 we investigate methods for generating problem instances, and we give computational results for various versions of the MCP in Chapter 8.

We consider one particular extension of the model in Chapter 9 — the dynamic situation, where customer requests arrive for service and these may need to be processed on-line. This is an important extension, which relates to practical situations where requests for service arrive as routing decisions are being made, and this requires many more (and different) considerations than in the static case. We again consider the case where the customer requests exceed the capacity to handle these requests, and so selection decisions need to be made while requests are arriving. In Chapter 10, we give conclusions about the MCP and direct attention to areas requiring further consideration.

We see the major contribution of this thesis being in the modelling of previously ignored phenomena, the unification of a number of areas of vehicle routing problems within our single model, and the development of appropriate techniques for dealing with particular identified problem types.

CHAPTER 2

Fundamentals of VRSPs

Vehicle Routing and Scheduling Problems (VRSPs) form an important practical component of the field of Operations Research. They involve devising minimum cost routes for vehicles servicing a set of customers. Here the routing relates to the order in which the customers are serviced, and is concerned with the spatial aspects of the problem. The scheduling relates to the time at which the customers are serviced, and is concerned with the temporal aspects of the problem.

The general VRSP model considered is:

$$\begin{aligned} &\min \textstyle\sum_{i \in V} C(i) \\ &s.t. \quad \text{connectivity constraints} \\ &\qquad \text{vehicle constraints} \\ &\qquad \text{customer constraints} \\ &\qquad \text{inclusion constraints} \end{aligned}$$

V is the set of available vehicles; the solution route of each vehicle $i \in V$, has an associated cost $C(i)$, and the objective is to minimize the total cost of servicing the customers. The connectivity constraints ensure that legitimate routes are obtained, through ensuring that a vehicle route consists of a connected sequence of sections of travel, where the destination of one section is the origin of the succeeding section. Vehicle constraints relate to the feasible operation of the service vehicles, with aspects such as load capacity and limits on the allowed route duration being

relevant. The customer constraints involve the type or level of service that the customers require, which, for example, may involve restrictions on the time at which the customer is available to be serviced. Inclusion constraints are used to ensure that the required customers are serviced.

The most basic type of VRSP, the Travelling Salesman Problem (TSP), involves a number of locations being assigned to a single vehicle, with a route devised so as to minimize the total distance travelled. The TSP has been well-studied as it is a sub-problem of many other VRSPs.

The Vehicle Routing Problem (VRP) involves a fleet of vehicles visiting a set of locations, where each location requires the delivery of a load of a given size and the objective is to minimize the total distance travelled in order to service the customers. Solving this problem involves assigning customers to vehicles so that the total load of the customers assigned to a vehicle does not exceed the vehicle's capacity. Each vehicle route in a solution for the VRP consists of the solution of a TSP on the set of customers assigned to the vehicle. Many different VRSPs have been studied, with a number of variants obtained through altering the constraints on the customers or the vehicles. These variant problems require additional componentry to deal with them efficiently, but they still can be dealt with in the same manner with varying degrees of effectiveness. The common theme is that VRSPs involve assigning the customers to the vehicles and creating a minimum cost, feasible route through each set of customers.

Other variants of VRSPs that introduce aspects which create fundamentally different problems, have also been studied. We call problems of this type General VRSPs (GVRSPs). One aspect that elicits such behaviour, is the relaxation of the requirement that all customers be serviced, and we investigate this in more detail in Chapter 3. Another aspect is to allow customers to arrive dynamically during the course of the time horizon available for servicing, and we consider the effect of this dynamism in Chapter 9. An example of a dynamic VRSP is the Dynamic Traveling Salesman Problem (DTSP), which was defined by Psaraftis [145]. He defines the DTSP to be a problem where all the customer locations are known, but the time at which each customer becomes available for servicing is generated according to a Poisson process. This problem enables, and in practice requires, different objectives from the static TSP, as we seek to optimize some function of the service given, such as maximizing the throughput of customers or minimizing

the total delay in servicing.

The addition of stochasticity to VRSPs creates fundamentally different problems. The Stochastic VRP (SVRP, see [169]) involves customers having a demand that is not known with certainty until the vehicle arrives at the customer's location. There are two ways in which this problem has commonly been modelled. The first involves minimizing the distance, as in the VRP, but with the additional constraint that the probability of exceeding the capacity of the vehicles is less than some allowable threshold. The other approach involves minimizing the sum of the routing costs and the expected penalty costs for when the capacity of a vehicle is exceeded. The uncertainty in the customer requirements leads to a different decision making process from the VRSP.

Another variant involving uncertainty is the Probabilistic TSP (PTSP) of Jaillet [88]. This problem involves finding the best TSP solution through a set of customers, where there is a probability, each time period, that each customer will require servicing and we service the customers in the order of the TSP tour, skipping those who do not require service. The objective is to minimize the expected distance travelled for each time period. The incorporation of multiple time periods, as is considered within the PTSP, may create different problems where the required decisions may involve determining in which period to service the customers. For problems with multiple depots, an additional phase of allocating the sets of customers (and the service vehicles to which these sets are assigned) to the depots, may be required.

We concentrate our focus now on the cases that fit within the class of VRSPs (rather than GVRSPs). We are interested in the techniques that have been used for these problems, and their applicability to the types of problems we are considering.

2.1 Exact Solution Methods

The ultimate aim associated with the VRSP is to find the optimal solution to a given problem. To this purpose, many methods have been developed which can optimally solve the desired problem. The difficulty however, is that most versions of the VRSP are $\mathcal{NP}$-hard (see [118]), which means that no exact algorithm will be able to solve each instance of the problem within a polynomial number of steps.

Most of the recent techniques for solving versions of the VRSP exactly, have

involved the use of column generation techniques. This technique involves restricting the size of the linear program considered, by only generating the columns as they are required within the search procedure. Lübbecke [115, page 25] described the importance of this procedure, by stating:

> "Today, more than ever, *column generation* is a prominent—and sometimes the solely applicable—method to cope with linear programs having a colossal number of variables."

The column generation method he developed was then applied to a practical problem involving the scheduling of locomotives, which was formulated as a form of the VRSP which was able to be solved with up to 50 customers.

Larsen [111] developed another column generation method for a version of the VRSP, in which there are time windows on when the customers can be serviced. This method was tested on a standard set of test problems, with a limit of 30 minutes on the computational time allowed. The algorithm was able to solve 13 of the 27 problems with 25 customers and only four of the 27 test problems containing 50 customers, within the allotted time limit.

Ralphs, Kopmanz, Pulleyblank and Trotter [147], in their very recent working paper, use column generation to solve versions of the VRSP in which there are constraints on the total load that may be carried within each vehicle. The algorithm was tested on a number of known test problems, with problems containing up to 64 customers being able to be solved in a reasonable amount of time (a maximum time of 23 minutes was required for these problems, although most were able to be solved in under 2 minutes). They also managed to solve a problem containing 100 customers, although the computational time required for this instance was 527 hours, which means that the method is unlikely to frequently be of practical value for this problem.

One recent article, which describes the current state of the art in exact algorithms, is by Van Breedam [179, page 348], who states that:

> "The VRP with or without side-constraints is an $\mathcal{NP}-$hard problem. This property makes it difficult to solve the VRP to optimality. Exact algorithms have been used to solve problems of up to 50 stops in a reasonable amount of computing time."

Due to the limited size of problems which could be handled by exact algorithms,

Van Breedam proceeded to develop inexact, practical solution methods for constrained versions of the VRSP.

Advances in technology in recent times have enabled the application of exact techniques to become more viable, and future improvements will no doubt enable larger problems to be solved within reasonable amounts of time. The current state of the art in algorithms, does not allow for large sized problems to be quickly solved, and with the ultimate goal of this study incorporating the fast updates required for dynamically developing problems we will, in the rest of this thesis, mainly focus on inexact or heuristic methods.

2.2 Basic Heuristic Techniques for VRSPs

We now introduce some important techniques that have been used for VRSPs. These are to be used in some form in our solution methods, and have also formed the basis of heuristic methods employed in previous research. We demonstrate our techniques on unconstrained problems, involving points to be serviced, with later chapters demonstrating how they may be adapted and extended to solving other problems.

A basic construction method is INSERTION, where an unscheduled customer is inserted into an existing vehicle route. The basic insertion rationale is shown in Figure 2.1, where the new customer j is inserted after customer i of the current route. This involves the inclusion of the arcs $i \rightarrow j$ and $j \rightarrow i+1$ and the deletion of the arc $i \rightarrow i+1$. We give a pseudo-code representation of this approach in Algorithm 2.1, in which there are n customers available to be serviced. The determination of the cost of the insertion, δ_{ij}, the set of available positions for insertion, $\{\nu_1, .., \nu_{numpos}\}$, and the minimum cost of insertion that we will accept, *min_allowed*, defines the implementation of the Algorithm. For example, if we are only allowing the customer to be inserted at the end of the current path, then we set $numpos = 1$, as only one insertion position is allowed.

One method for applying INSERTION in problems where there are rewards that are gained from servicing the customers, is to use Bang-for-Buck. In this method the insertion that is carried out is the one where the ratio of $\frac{\text{extra reward received}}{\text{extra distance required}}$, is maximized, i.e., the cost of the insertion is the negative of this ratio. If we require the insertion to increase the reward of the solution, then we simply set *min_allowed*

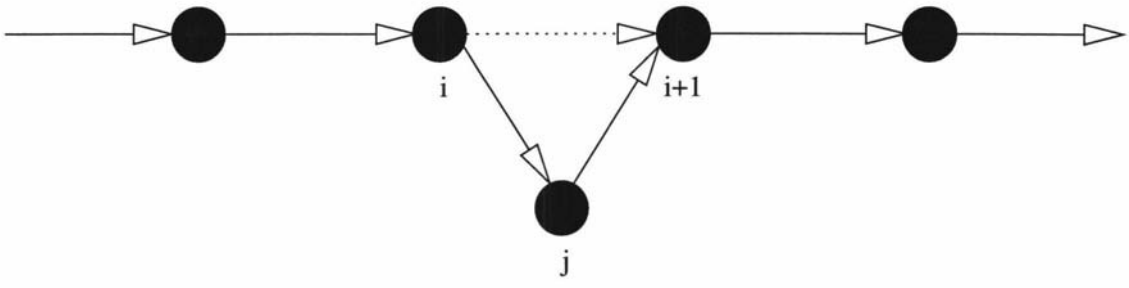

Figure 2.1: Inserting a Customer

to zero.

For improving the quality of the routes obtained, a number of local search techniques have been applied. One method that is commonly employed for reducing the distance travelled, is the k-exchange of Lin [113]. This routine involves k arcs being removed from the current solution and k others being inserted so as to ensure the route is still connected. An illustration, showing the route before and after the 2-exchange, is shown in Figure 2.2. Here arcs $i \rightarrow i+1$ and $j \rightarrow j+1$ have been replaced by arcs $i \rightarrow j$ and $i+1 \rightarrow j+1$, with the sub-path $i+1 \longrightarrow j$ being reversed. The process of performing k-exchanges until no k-exchange may improve the solution further, is known as k-opt.

Another improvement method involves the moving of a sub-path of a given number of customers to another position within the route. Or [129] developed a method, in which either 1, 2 or 3 customers are moved, and we subsequently use the terminology that a k-Or-exchange is where k customers are moved to another position of the route. A k-Or-exchange is illustrated in Figure 2.3, where the sub-path from $i \longrightarrow i+k$–1 is moved before customer j. This process involves the replacement of three arcs from the solution route and replacing these with three others, and so this is a particular version of a 3-exchange. The overall improvement routine described by Or, is known as Or-opt.

As well as the intra-route methods described above, a number of improvement methods have been developed which enable customers to be transferred between routes. The Inter-route Move method, involves a customer being moved from its current vehicle to a position in another vehicle. This is shown in Figure 2.4, where customer i from route 1 is moved to before customer j of route 2. Another basic improvement technique, is the Inter-route Exchange, which is shown in Figure 2.5. Here customer i from route 1 and customer j from route 2 are removed and customer i is placed before customer m of route 2 and customer j is placed before customer

Algorithm 2.1 function INSERTION

```
// Set all customers as initially available
numavail ← n
for (i = 1 to n) do
    avail_i ← i
end
while (numavail > 0) do
    min_cost ← min_allowed
    Determine numpos and the set {ν_1, .., ν_numpos} of insertion positions
    for (i = 1 to numavail) do
        for (j = 1 to numpos) do
            δ_ij ← the cost of inserting avail_i in position ν_j
            if (δ_ij < min_cost) then
                best_i ← i, best_j ← j, min_cost ← δ_ij
            end
        end
        if (no insertion of avail_i was feasible) then
            numavail ← numavail − 1
            for (j = i to numavail) do
                avail_j ← avail_{j+1}
            end
        end
    end
    if (min_cost < min_allowed) then
        Insert customer avail_{best_i} in position ν_{best_j}
        numavail ← numavail − 1
        for (j = best_i to numavail) do
            avail_j ← avail_{j+1}
        end
    else
        numavail ← 0
    end
end

end
```

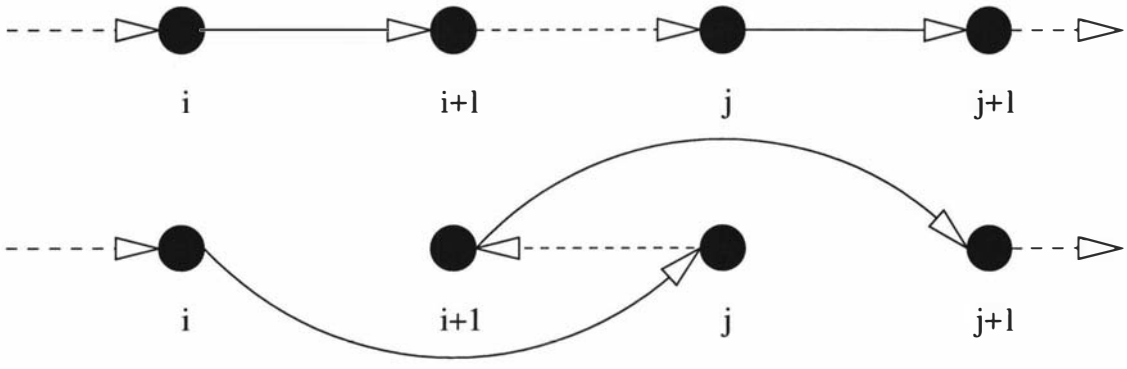

Figure 2.2: An Example of a 2-Exchange

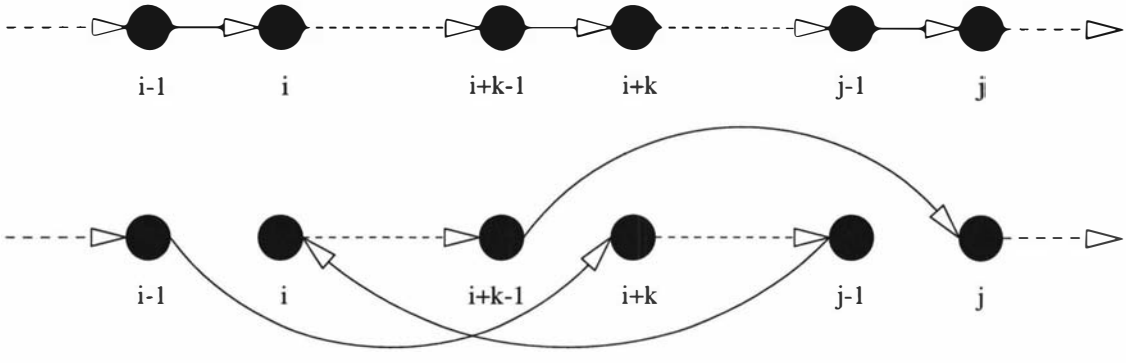

Figure 2.3: An Example of a k-Or-Exchange

k of route 1.

A recently developed inter-route improvement technique is the Cross exchange routine of Taillard, Badeau, Gendreau, Guertin and Potvin [171], which is a generalized exchange technique involving the exchange of sub-paths of varying lengths. This procedure is illustrated in Figure 2.6, where the sub-paths $i{+}1 \longrightarrow k$ from route 1 and $j{+}1 \longrightarrow m$ from route 2 are exchanged. The procedure successively increments i, j, k and m, while feasible exchanges may be found. In this form of exchange, the position of the sub-path in the new route is restricted to occupy the position of the other sub-path.

One method for implementing the local search is lexicographic search. This has been used by Savelsbergh [154, 155, 156, 157] to efficiently perform the local search for constrained problems, and is applicable when there are a number of loops in the search method. The parameters of the loops of the search procedure are incremented in a given order, and the difference between the potential solutions obtained by successive iterations are stored. These differences are exploited in order to reduce the computational requirement involved with the search procedure,

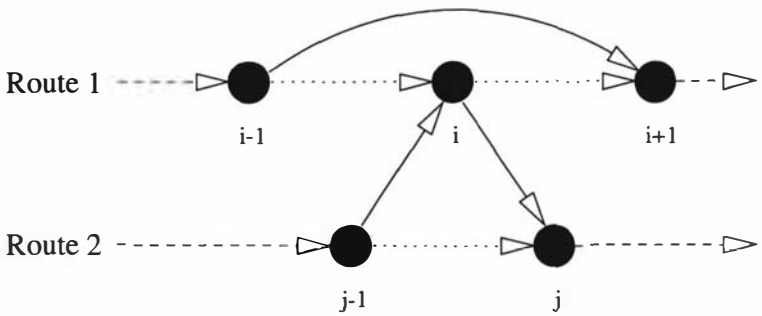

Figure 2.4: An Example of an Inter-Route Move

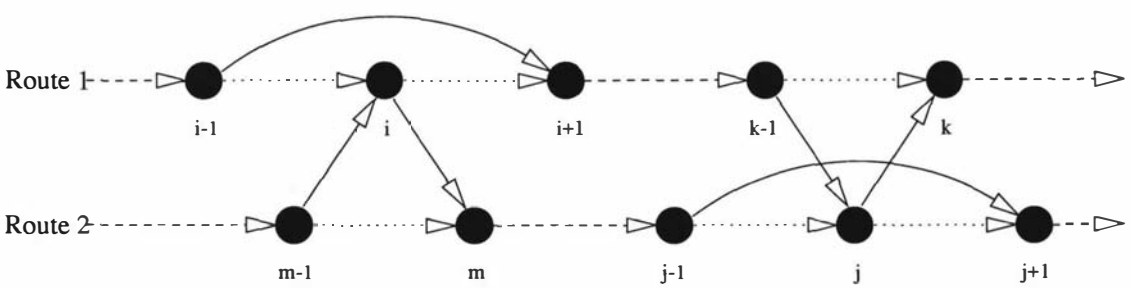

Figure 2.5: An Example of an Inter-Route Exchange

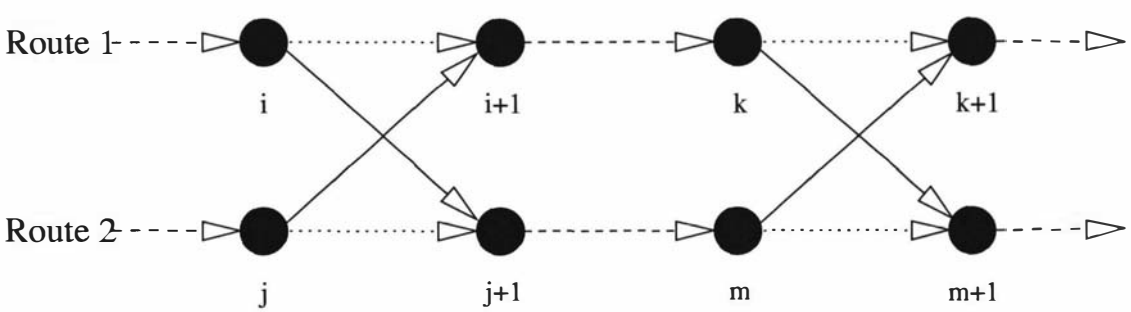

Figure 2.6: An Example of a Cross-Exchange

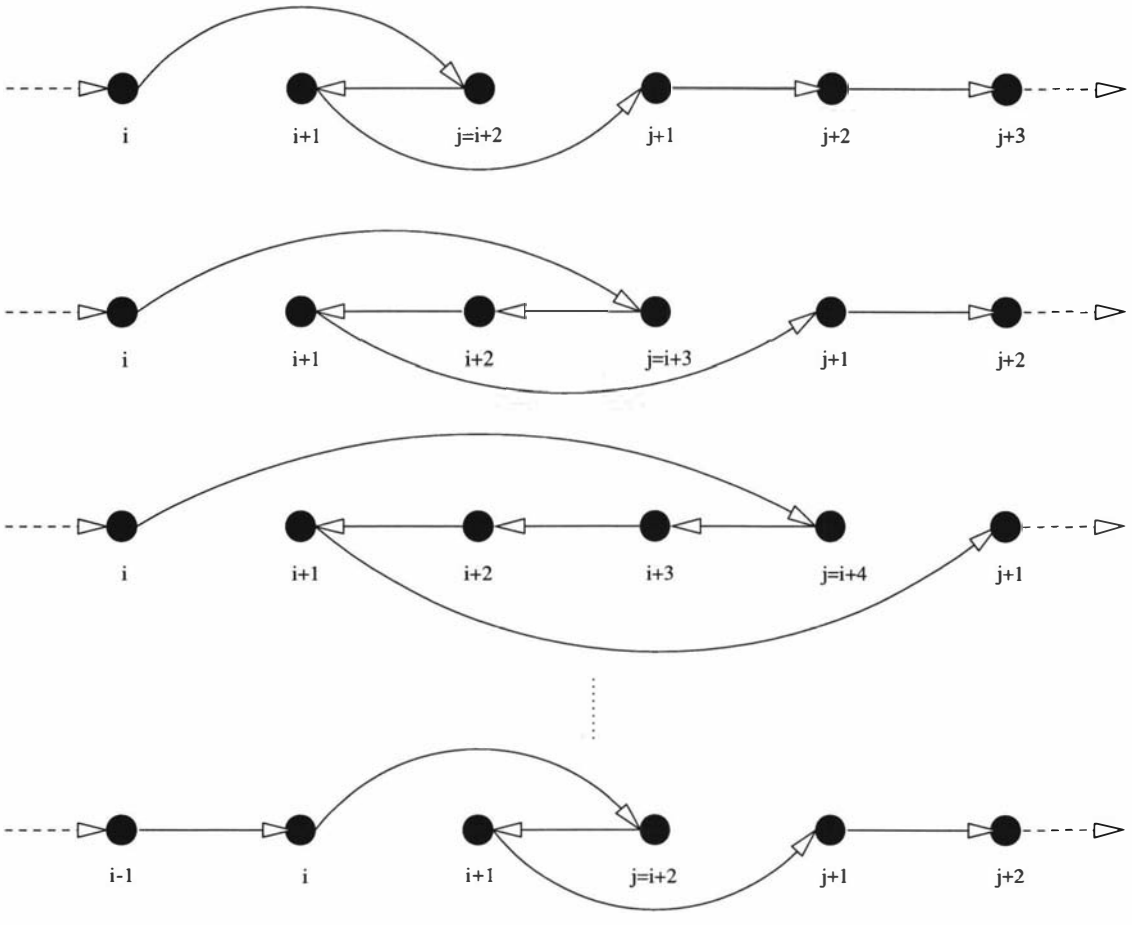

Figure 2.7: The Lexicographic Search for the 2-exchange

particularly in the presence of constraints on the routes. We demonstrate the lexicographic search for the 2-exchange in Figure 2.7. We fix i and set j to be $i+2$, which is the first value for which a 2-exchange exists, and calculate the 2-exchange as shown in the top diagram. We then successively increment j, and consider the 2-exchanges, as shown in the second and third diagram. The last diagram shows where we increment i, after coming to the end of the j loop. Thus we start again with our new i and with j initially equal to $i+2$.

For Or-exchanges, lexicographic search may also be carried out. The sub-path that is moved is fixed, and we successively increment the position in which to place this sub-path. Once we place the sub-path in the last position, we increment the first point of the moved sub-path, and continue by moving this new sub-path.

The lexicographic search may also be employed with the Cross exchange in order to reduce the computational requirements. We illustrate part of this search in Algorithm 2.2, where the Cross exchange is to be performed on two given vehicles,

Algorithm 2.2 function LEXICOCROSS

```
go_on_k ← 1
for ((k = i + 1 to n(V2)) and (go_on_k ≥ 1)) do
    Check feasibility of customer k
    if (k is infeasible) then
        go_on_k ← 0
        go_on_m ← 0
    end
    else
        go_on_m ← 1
    end
    for ((m = j + 1 to n(V2)) and (go_on_m ≥ 1)) do
        Check feasibility of customer m
        if (m is feasible) then
            Include arcs (k, m + 1) and (m, k + 1)
            Check feasibility of customers k + 1 ... n(V1)
            Check feasibility of customers m + 1 ... n(V2)
            if (the solution is feasible) then
                Evaluate the effect of the exchange
            end
        end
        else
            go_on_m ← 0
        end
    end
end

end
```

$V1$ and $V2$, containing respectively $n(V1)$ and $n(V2)$ customers. We fix the values of i and j and proceed with k=i+1 and m=j+1. If the m'th point of vehicle $V2$ can not feasibly be included in the required position in the solution, we stop the current loop, increment k and continue with $m = j$+1. Similarly, if the k'th point of vehicle $V1$ would become infeasible, we stop the loop, and increment j. The new loop would thus begin with $m = j$+1 and $k = m$+1.

Another improvement routine that has been applied to VRSPs, is the Generalized Insertion procedure (GENI) of Gendreau, Hertz and Laporte [60], which enables improvements to be made to the tour while inserting new customers. Instead of requiring the customer be inserted between two adjacent points on the

current tour, as was the case with the Inter-route Move routine, there are two additional insertion moves that are used. The first involves reversing two sub-paths within the current tour, while the second involves two sub-path reversals and the moving of another sub-path. An associated improvement routine that is used with GENI is Unstringing Stringing (US). With US there are two ways of removing a point from the current tour, with the first involving two sub-paths being reversed and the second involving two sub-paths being reversed and another sub-path being moved. The point is then replaced in the tour using GENI. The combined method of route construction, using GENI, and improvement, using US, is known as GENIUS.

For GENI and US, the current route is considered to be a path and so alterations are considered in either orientation of the tour. Since there are so many feasible insertions, the position of insertion is restricted via a parameter p, whereby a customer can only be inserted adjacent to one of its p closest neighbours within the tour, and the new arcs included can be between a point and one of its p closest neighbours. This restricts the computational requirement of GENI to $O(np^4 + n^2)$ in order to create a TSP solution.

2.3 Local Search and Metaheuristics

We now consider how the routines described above may be incorporated within solution routines for VRSPs. The simplest form of searching is to explore a certain neighbourhood of a solution, and move to a neighbouring solution with an improved objective function value, if one exists. If we carry out the move that improves the objective function the most, this method is called steepest descent for a minimization problem or steepest ascent for maximization. This search procedure terminates when there are no feasible improving solutions in the neighbourhood of the current solution.

The main difficulty of this solution procedure is shown in Figure 2.8, where we show a 2-D solution landscape. Here, from an initial solution with objective function value A, steepest ascent is performed until the solution *local opt* with objective function value B is obtained. There are no neighbouring solutions that can improve this solution, so the search terminates. The best solution over the whole landscape is at *global opt*, which has objective function value C.

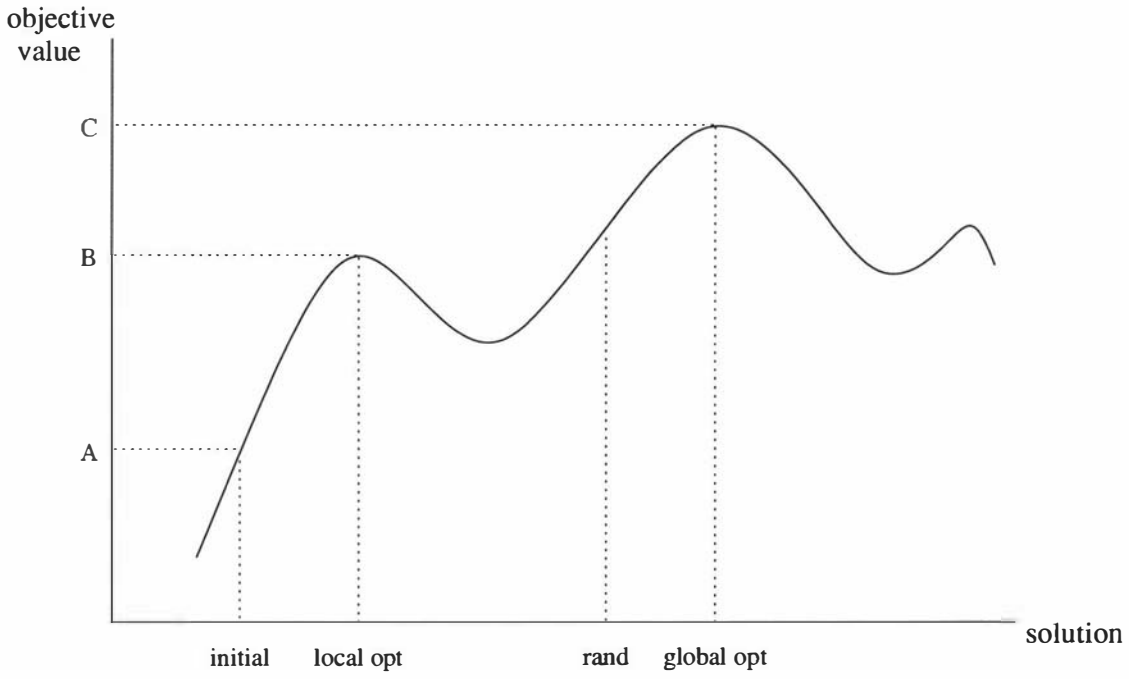

Figure 2.8: An Example Solution Landscape for Local Search

Through repeated random generation of solutions, we may increase the chances of obtaining a solution that may be improved to obtain the global optimum. Thus in Figure 2.8, if we start at the solution *rand* and use steepest ascent, we would get to the solution *global opt*.

There are a number of more specific techniques that have been developed in order to prevent search procedures from getting stuck in a local optimum. These come into the category of metaheuristics, where a metaheuristic is literally a solution method that uses heuristic methods as a part of the overall solution process. The advantage of metaheuristics over steepest ascent is that a metaheuristic may allow us to carry out a move that deteriorates the current solution, in order that we may obtain a better solution at some later stage. Thus it is possible to move away from the local optimum through carrying out non-improving moves.

A metaheuristic technique that has been found to be effective in a number of cases is the Tabu Search method of Glover [68, 69]. Tabu Search is a very general method, based upon the use of flexible memory structures, that is used for guiding local search. The key features of Tabu Search are:

- the use of flexible, attribute-based memory systems, with some attribute of the search process being stored for later use;

- a method of controlling the local search, to restrict or free the search;
- the inclusion of memory functions of different time spans, which allow intensification into areas that have previously been found to be good, or diversification into new areas;

A typical Tabu Search involves starting with an initial solution, which are either constructed or obtained from previous solutions found. We generate the neighbourhood of the current solution according to the type of move that is being used, carry out the best admissible move and then check to see if the stopping criteria have been met. If not, we update the memory according to the move just executed and continue; otherwise we either terminate the search or intensify or diversify the current search and continue from the new solution. Some feature (known as an attribute) of the solution, is stored in the memory and made tabu for a number of moves. Checking the admissibility of a particular move, involves checking with the search memory to see whether the attribute to be changed is tabu, and if so, the move may only be carried out if the aspiration criterion is met. This tabu status typically restricts the search so a move that has been recently executed can not be undone immediately.

The Tabu Search paradigm has a number of options that affect the way it is implemented. The user must select:

- the method of initializing the search
- the method of generating the neighbourhood of the current solution
- the attribute that is stored in the tabu list
- the length of the tabu list, which is the number of iterations for which the attribute of the executed move is tabu
- the definition of the aspiration criterion
- the method of intensifying and diversifying the search
- the stopping criteria.

An example of how Tabu Search may be applied to the VRP is given by Gendreau, Hertz and Laporte [61]. The objective is to minimize the total distance travelled to service a set of customers, and there are constraints on the total load

that may be carried in each vehicle and on the route duration of each vehicle. This implementation of Tabu Search was found to be effective, and it was able to consistently outperform the best previously published solutions for a set of benchmark problems. For the Tabu Search approach in the article:

- to create the initial solution, all customers are placed in a TSP solution using GENIUS. This route is broken up so as to create feasible vehicle routes, and, thus, this method involves a route-first, cluster-second approach.

- the neighbourhood of the current solution is all solutions that may be obtained through removing a customer from its route and placing it in another route, using GENI

- the route the customer was moved from is stored in the tabu list

- the moved customer can not be put back in the route it came from, in the next θ iterations, where θ is a random integer in a given range

- the tabu status of a move is overridden if the move results in a new overall best solution

- for intensifying the search, only the customers that have been moved the most frequently may be moved. For diversifying, the change for a move is penalized by a factor proportional to the frequency that this customer has been moved.

- the heuristic stops if the objective function has not been improved for a given number of iterations

The method of Gendreau et al. [61] also incorporates the technique of strategic oscillation, in which solutions are allowed to oscillate between being feasible and infeasible during periods of the search. The objective function was penalized for the level of constraint violation, with the rates of the penalties being 'self-correcting'. Thus, if the solution has been infeasible throughout the recent history of the search, the penalties are increased in order to bring the solution back to feasibility. If the solution has been consistently feasible, the penalties are decreased in order to encourage the creation of infeasible solutions, which effectively diversify the search.

Simulated Annealing (SA), is a metaheuristic method in which any improving move is accepted, but a non-improving move will be accepted according to a probability function of the associated deterioration and the stage of the search. The probability of accepting the move is lower for moves that deteriorate the current solution more, and it is reduced as the search proceeds. Thus SA has the ability to move away from local optima early on in the search, but as the search continues, it is more likely to move towards the best solution in the current search area.

Genetic Algorithms (GAs) are another form of metaheuristic, where previous good solutions are combined in order to generate new solutions. This is based upon the process of development of genetic reproduction. The procedure begins through generating a set of solutions, known as chromosomes, which constitute the initial population. Each chromosome is given a fitness value that relates to the effectiveness of the individual chromosomes. In order to obtain new solutions, sets of chromosomes are selected, with probability proportional to their fitness values. These chromosomes are then altered through using appropriate improvement routines that may simulate mutation (alteration of a single solution route) or crossover (exchanging portions of solutions from different routes). The new solutions, or offspring, are added to the population of available solutions, and so generations of solutions are obtained.

Many other metaheuristic methods have been used to obtain solutions to versions of VRSPs, most notably to the TSP. Ant Colony Systems (ACSs) [37] model the behaviour of ants, in which the shortest paths contain the greatest amount of pheromones, and these are the ones that are most likely to be followed. Therefore ACSs involve solution routes being selected from the current set of solutions according to the attractiveness of the solutions. Other features of ACSs include the evaporation of the pheromones, so more recent events have greater importance, and the inclusion of different methods for applying the pheromones during the search procedure. Other metaheuristic methods that have been applied to the TSP include Threshold Acceptance [40] (which is a deterministic version of SA), Memetic Algorithms [126] (which are hybrid GAs that incorporate local search) and Neural Networks [137]. The Greedy Randomized Adaptive Search Procedure (GRASP) [48] is another metaheuristic which has been applied to various versions of VRSPs.

Applicability to VRSPs

Metaheuristics have been applied to a number of different problems, within a number of different fields of Operations Research. They have been used to obtain high quality solutions to problems where the computational time requirements prevent the application of exact solution methods. The metaheuristics typically are able to continue searching and identifying improved solutions, for as long a period of computational time as is allowed for obtaining a solution, although stopping criteria are often used to restrict the time that is actually used. It appears that particular metaheuristics perform well on some problems and perform poorly on others. Hence it is important not to prejudge a metaheuristic before applying it to a new problem or variation of an existing problem, but rather to try it out. Indeed, the effectiveness of the solution methods often appears to be influenced more by the routines and the method of implementing these within the method, than the underlying metaheuristic method employed.

We will use Tabu Search as one of the solution methods used within this thesis, with the specific search routines including the improvement routines described in Section 2.2. We consider Tabu Search to be a widely applicable technique, which has been found to be effective for a number of different applications, e.g., constrained versions of the VRSP [28, 65, 139], machine scheduling [105, 178] and location analysis [63, 163]. Tabu Search is a very general method which enables many different techniques to be used within the actual implementation, with the selection of the appropriate routines and parameter settings for a particular application, being critical in determining the effectiveness of the particular method.

Therefore we see Tabu Search as the skeleton upon which a solution method may be created. The important steps are in the identification of the appropriate tools to use as part of the search process, and selecting the exact details of the implementation. The development of new techniques within the metaheuristic methods are important features, and we will subsequently introduce some new techniques for use within a Tabu Search method for the subset selection versions of the VRSP that we investigate in later chapters.

CHAPTER 3

Subset Selection Routing

Many of the problems in the vehicle routing literature relate to a service company needing to service a given set of customers. We now consider problems where it may not be feasible or desirable to service all the customers, and the server may select a subset of customers to service. Decisions as to which customers to service are made according to the objective of the problem, where rewards and/or penalties are frequently used to encourage service or discourage non-service. We describe subset selection routing problems that have previously been defined, and we initially separate these problems according to the restrictions that force the subset selection elements to be relevant, i.e., restrictions on the time available for servicing, the minimum reward that must be collected and unrestricted problems where the form of the objective function restricts the number of customers that are serviced.

3.1 Previous Problems

3.1.1 Time Constrained Subset Routing

Two of the earlier references to time constrained subset selection routing where the objective is to maximize the reward received, were by Gensch [66] and Golden, Levy and Dahl [72]. Gensch [66] introduced the Travelling Salesman Subtour Problem (TSSP), which is the first routing problem in the literature where only a subset of customers need be visited within a solution route. The objective of the TSSP

is to maximize the net rewards received within a single solution route that is of a restricted length, where the net reward is the total reward derived from visiting the selected customers minus the costs of the route. Gensch used Lagrangean relaxation to obtain an upper bound that is used within a branch-and-bound algorithm to create solutions with up to 30 customers. Golden et al. [72] defined the Time-Constrained Traveling Salesman Problem (TCTSP), where the objective is to maximize the profit received in a single tour with a given time limit. In this problem the rewards are associated with the edges, thus the reward is received for travelling along the edge. Golden, Assad and Dahl [71] described a vehicle routing problem in which they use the TCTSP as a subproblem. Customers are given a priority rating according to the urgency of their needs, and the TCTSP subproblem selects customers for service so as to maximize the sum of priority ratings.

The initial problem we focus on has the following characteristics:

- there are a fixed number of customers
- each customer has a fixed reward that is received upon the customer being serviced
- there is a fixed time that the service vehicle is available
- the objective is to maximize the total reward received within the time limit.

There is one less dimension to consider in this problem than there is for Gensch's problem, as we only need to consider the distance of a tour (since the time travelled is assumed to be a function of the distance), rather than both the distance and the costs. Here the decisions to be made are the selection of which customers to service, and the sequencing of these customers within a time-feasible route. The constraint here is the time available for servicing and the objective is to maximize the reward collected. The rewards give a natural form of prioritizing which customers to include. We describe previous references to this problem from the literature, and give conclusions at the end of this section.

This problem can be formulated as follows:

$$\max \quad \sum_{i=1}^{n} r_i(1 - x_{ii}) \qquad \text{(i)}$$

$$\text{s.t.} \quad \sum_{j=0}^{n} x_{ij} = 1, \quad i = 0, \ldots, n \qquad \text{(ii)}$$

$$\sum_{i=0}^{n} x_{ij} = 1, \quad j = 0, \ldots, n \qquad \text{(iii)}$$

$$\sum_{i=0}^{n} \sum_{j=0}^{n} c_{ij} x_{ij} \leq C \qquad \text{(iv)}$$

$$2 \sum_{\nu_k \in S} (1 - x_{kk}) \leq |S| \left(\sum_{\substack{\nu_i \in S \\ \nu_j \notin S}} x_{ij} + \sum_{\substack{\nu_i \notin S \\ \nu_j \in S}} x_{ij} \right), \quad |S| \geq 3 \qquad \text{(v)}$$

$$x_{ij} \in \{0, 1\}, \quad i = 0, ..., n, \quad j = 0, \ldots, n, \qquad \text{(vi)}$$

Thus, there are n customers, each with an associated reward, r_i, that is gained if the customer is serviced within the solution route. In this formulation, we set $x_{ij} = 0$ if j directly follows i in the solution route and $x_{ii} = 1$ if the customer is not serviced. The constraints (ii) and (iii) ensure that each customer is entered and exited one time in the solution, constraint (iv) gives the restriction on the maximum distance of the solution route (c_{ij} is the distance between customers i and j) and constraints (v) ensure that there is only one subtour in the solution that contains more than one customer.

This problem was first introduced by Tsiligirides [176], and he termed this the Generalized Travelling Salesman Problem (GTSP). This problem relates directly to a form of the sport of orienteering, where there are a number of checkpoints available, each with an associated score that is received for visiting this checkpoint. The winner of the event is the competitor who receives the highest score in a route that is completed within a given time limit, where competitors who exceed the time limit are disqualified. Tsiligirides created the first test problems for this problem and these have subsequently become the standard problems for testing the effectiveness of solution methods. His solution methods constructed feasible solutions through using either a stochastic heuristic that appends one of the best customers to the end of the current tour, or a sectoring heuristic which breaks the solution area into radial sections, with a sweep of the section being carried

out, and customers added to the solution if doing so will not lead to a violation of the time limit. A major component of the implementation of both of these construction methods is the use of repetition; 3000 solutions were generated by the stochastic heuristic, while the sectoring heuristic generated 48 solutions through varying the angles and the radii used to describe the sectors. The solutions obtained were then improved using the distance reduction techniques of 2-exchange and interchanging the position of two points within the solution. Techniques of inserting unincluded customers and inserting a customer while deleting another, were applied to the solutions to obtain the final results. The stochastic algorithm obtained higher reward values than those obtained by the sectoring heuristic, although after applying the improvement routines, similar rewards were obtained.

The above problem is now commonly known as the Orienteering Problem (OP), the name that was introduced by Golden, Levy and Vohra [73]. They demonstrated that this problem is $\mathcal{NP}$-hard, as it has the TSP as a special case. They created a new solution heuristic where the attractiveness of a customer relates to the reward of the customer, the distance the customer is away from the start and end points of the route, and the distance from a Centre of Gravity, which is a reward weighted centre of the current solution. The solution obtained was improved using 2-opt distance reduction and insertion, before a new solution was constructed using the centre of gravity of the current solution.

Golden, Liu and Wang [74] developed a new heuristic for the OP. Their heuristic incorporates the centre of gravity approach mentioned above, and also includes two new features for determining the attractiveness of a customer. They used a learning measure, which gives higher scores for customers that have been previously placed in tours with high overall reward. They also used a "subgravity" approach, where the score for a customer i is given by the function:

$$S(i) = \sum_j r_j e^{-\mu d_{ij}}$$

where r_j is the reward of customer j, μ is a weighting factor and d_{ij} is the distance between customers i and j. This subgravity measure therefore takes into account not only the reward for the customer, but also the reward of the customers near it, with the effect on the subgravity score being a decreasing function of the distance. The solution method also included randomization, through randomly selecting one of the best candidate customers to be included in the solution.

Keller [94] developed a model for what he called the Multi-objective Vending Problem (MVP), where the objective is to simultaneously minimize the total distance travelled and maximize the total reward received. To achieve these aims he attempted to create a non-inferior solution set, which is the set of solutions where the reward received cannot be increased without increasing the time required and the time used cannot be decreased without reducing the reward. The simplest version of this problem is the OP, where for a fixed time limit the objective is to maximize the reward. Keller and Goodchild [96] described their solution method to this problem, which involves constructing an initial solution either randomly or using bang-for-buck, and successively using a series of point and path moves and exchanges to improve upon the solution. One interesting improvement routine they developed, attempts to remove isolated clusters of customers from the solution. This was developed after they had identified a shortcoming of the solutions obtained, where clusters of customers could not be removed from the solution route by any of the other improvement routines. They demonstrate the effectiveness of their solution method on a single data set for the OP. Keller [95] compared the effectiveness of the heuristic, with the effectiveness of the heuristics of Tsiligirides [176] and Golden et al. [73]. He stated that the solution quality of this heuristic compares favourably with that obtained by previously published methods, as it is the highest scoring method when applied to the most test problems considered. This conclusion doesn't agree with the results in the paper, as Keller and Goodchild's heuristic is only superior in one of the three problem sets considered. Since this problem set was the one with the greatest number of instances, they obtained their desired conclusion.

Kataoka and Morito [92] independently formulated the OP, which they call the Single Constraint Maximum Collection Problem. They used branch and bound on the relaxed linear programming problem to find the optimal solution to problems with up to 10 customers.

Laporte and Martello [108] gave a new name for the OP, the Selective Travelling Salesman Problem (STSP), and they developed an exact algorithm. They developed an integer programming formulation for the OP, and created lower and upper bounds for the value of the solution. These were used within a branch-and-bound framework to find the optimal solution. A number of test problems with uniformly distributed distances were generated, with problems with 10 customers able to be

consistently solved, and with larger problems, with up to 90 customers, able to be solved with short time limits.

Sokkappa [164] gave the OP yet another name, the Cost-Constrained Traveling Salesman Problem (CCTSP), and dedicated a PhD thesis to this problem. She developed a new exact branch-and-bound algorithm using improved bounds via the knapsack relaxation of the problem. She determined that the important characteristics of a heuristic for the CCTSP are the methods of selecting which node to insert and where to insert it, the ability to remove customers from the solution, the use of intra-route improvement and the repetition of solutions. She adapted features of the heuristic of Golden et al. [74] to create a new improved heuristic, and tested the effect on the solution quality of a number of features of the heuristic. A number of new test problems for the CCTSP were created through random generation, and varying the reward values, the form of the distances and the relative locations of the customers.

Ramesh and Brown [148] developed a new heuristic for the OP, for a problem they called the Generalized Orienteering Problem (GOP), where there may be a distinct origin and depot for the solution route, and where customers may be revisited, although without receiving any reward. The customer revisitation case is only relevant with non-Euclidean distances that don't satisfy the triangle inequality. The heuristic uses four phases, where the three phases of insertion, distance reduction and deletion are repeated until the stopping criterion is met, with a final phase consisting of greedy insertion. The authors claimed that this heuristic quickly generates near optimal solutions.

Ramesh, Yoon and Karwan [149] introduced yet another name, this time for the problem where the origin and depot are the same, and they called this problem the Orienteering Tour Problem (OTP). They solved this problem optimally, using Lagrangean relaxation within a branch-and-bound framework, and were able to solve random problems with up to 150 customers.

Leifer and Rosenwein [112] developed upper bounds for the solution of the OP. They improved upon the linear programming relaxation of the problem, by adding further constraints and re-solving. This is an iterative procedure, to improve the bounds, where these improved bounds may be used to improve the speed of an optimal solution algorithm.

Wang, Sun, Golden and Jia [185] developed a neural network for solving the OP.

The neural network is used to create candidate solutions, and local search routines of 2-opt, insertion and deletion required to ensure feasibility or to improve upon the solution obtained. This heuristic therefore has the ability to generate infeasible solutions which need to be altered to bring about feasibility. This method was tested on Tsiligirides' data, and was found to be as effective as the best previous heuristic for most of the problem instances.

Arkin, Mitchell and Narasimhan [3] described a version of the OP on networks. Their formulation requires the rewards to be integer, as their objective is to maximize the number of customers visited, and therefore the rewards are represented as repeated customers. They obtained approximation bounds for variants of the OP, where there may or may not be certain customers that need to be included in the solution.

Millar [122] described an application of the OP to fish scouting, which he called the Fish Scouting Problem. Here the rewards are the expected gain from sampling a given area of a fishing ground, and there is a time associated with sampling each area. He gave some basic heuristics and a procedure for determining the upper bound of the number of areas included in the optimal solution. Millar and Kiragu [123] basically replicated the above paper, although this time for an Operations Research audience, and demonstrated how the bounding procedure may be used on some randomly generated problems. Both papers mentioned that the bounding procedure may be used with multiple vehicles, although they didn't do so in their problem instances.

Gendreau, Laporte and Semet [64] developed an optimal branch-and-cut algorithm for a version of the OP where there is a set of customers that must be in the solution. They created two heuristics that are used to obtain lower bounds on the value of the optimal solution. The first heuristic involves placing the compulsory customers via the generalized insertion heuristic GENIUS [60]. Pairs of customers and single customers are added to the solution, which is improved through removing pairs of customers, applying GENIUS, and then using the insertion phase again. The second heuristic creates a tour through all the customers using GENIUS, then removing customers until the distance of the path is less than 80% of the time limit. Insertion and swapping are then used to improve the solution. The swapping involves removing one customer from the route and including one non-solution customer, such that the reward of the route would be increased. GENIUS is then

applied to the solution set, in an attempt to obtain a feasible path.

The heuristics are heavily reliant upon the use of GENIUS to construct solutions. Each time GENIUS is called, the route is started again from scratch, in order to obtain the shortest length tour through the selected customers. This is a computationally expensive approach that emphasizes the importance of obtaining the minimum distance tour. The branch-and-cut algorithm uses a number of valid inequalities and the new heuristics to restrict the search. They solved Euclidean problems with relatively short time limit, with up to 300 customers, and non-Euclidean problems with longer time limits and again up to 300 customers. In fact, they claimed that it is the heuristic, i.e., the memory requirements for GENIUS, that restricts the size of the problem that they were able to solve.

Fischetti, González and Toth [51] developed an optimal branch-and-cut algorithm for the OP. They use several valid inequalities to speed up the solution process and they introduce a conditional cut to remove sub-optimal solutions. The algorithm is used to solve a number of different classes of problems, with non-Euclidean problems with up to 500 customers able to be solved. They tested their algorithm on the same Euclidean problems as Gendreau et al. [64], and claimed that their method is able to solve a number of instances substantially larger than those solved by Gendreau *et al.*

Kataoka, Yamada and Morito [93] developed a new relaxation for the OP, and developed new methods for finding a lower bound for this relaxation. They test the effectiveness of their bound on some random test problems and found that their bound was generally more effective than using a relaxation based upon the assignment problem.

Butt and Cavalier [21] extended the OP to cases where there is more than one service route available, and created the Multiple Tour Maximum Collection Problem (MTMCP). Here the objective is to maximize the total reward collected over all the solution routes. They created a new heuristic that builds up tours through finding pairs of customers that should appear in the same route, and attempting to include the pairs of customers in the same route. Repeated solutions are obtained through varying the parameter that is used to assess the viability of including the customers within a pair. The heuristic was tested on small, random problem instances and obtained optimal solutions for most of the cases, and solutions were obtained within a reasonable amount of computational time for problem instances

containing up to 100 customers.

Butt and Ryan [22, 23] attempted to optimally solve the MTMCP using column generation. Their model allows for there to be differing time limits for the different service routes. The method selects candidate subsets of customers and assesses their viability through creating a TSP tour through the customers. One important technique that is used within the solution method is to use a tree structure to store the subsets of customers that can and can not allow a time feasible tour to be obtained. Scanning the trees, will prevent the unnecessary application of the TSP solution method. The column generation method obtained will obtain optimal solutions to the MTMCP when they find the optimal TSP solution. This phase can be computationally restrictive, so they suggested that a TSP heuristic may substantially decrease the computational time without degrading the solution quality too much. Optimal solutions were obtained for problems containing up to 100 available customers, but the problem instances were such that the maximum number of customers in each solution route was approximately five. The restriction in the number of customers per route enabled the TSP algorithm to be implemented quickly and enabled the TSP heuristic to obtain optimal solutions in most cases.

Chao [25] modelled a number of problems as examples of what he called the Multi-Level Vehicle Routing Problem (MLVRP). He developed a general solution metaheuristic based on Threshold Acceptance, and adapted it to be applicable to the different versions of these problems. Two such problems considered were the OP, and the multiple vehicle version of the OP which he termed the Team Orienteering Problem (TOP). The solution method for these problems consisted of creating a number of initial solutions through greedy insertion, and taking the best of these to be the initial solution. All eligible points are included in vehicle routes, with the best k routes, where k is the number of vehicles in the solution, considered to be the solution. The initial solution is improved using insertion and exchange techniques, with moves carried out if they improve the solution or if they don't deteriorate the solution by too much. Distance reduction is applied to the routes, before new initial solutions are created through removing a number of points from the current solution. The OP and TOP were further analyzed by Chao, Golden and Wasil [26, 27]. The Threshold Acceptance heuristic appears to outperform the previous heuristics over the existing test problems for the OP. For the TOP, the worst-performing of the published heuristics for the OP, i.e., the

stochastic algorithm of Tsiligirides [176], was adapted for the multiple vehicle case. Not surprisingly, the new heuristic appears to improve upon this adapted heuristic.

Another extension of the OP that has been investigated is to apply restrictions on the time at which the customers may be serviced. These create time window constraints, which will be discussed in much more detail in Chapter 5. The problem with these time windows is known as the Orienteering Problem with Time Windows (OPTW), and it was introduced by Kantor and Rosenwein [91].

Summary of the OP

For the OP a number of heuristic solution methods have been used. Most of these methods involve creating an initial solution, improving upon the solution and then creating a new starting solution, through either repeating the construction phase or making alterations to the current solution. The approach of Golden et al. [73] is an example of this second updating method, as one of the criteria used to create the new solution, is the distance from a candidate point to the weighted centre of the current solution. The method of Ramesh and Brown [148], involves successively inserting customers into and deleting customers from the current solution. This process effectively creates new solutions through making small changes to the current solution.

The published heuristic for the OP which appears to be the most effective is that of Chao [25]. One of the reasons for its effectiveness is the advanced search structure employed, as the heuristic allows deterioration to the current solution in order to possibly obtain improved solutions at a later stage. A more advanced metaheuristic approach, e.g., Tabu Search is likely to further improve upon the solutions obtained, although with probable increase in the computational time required. The other major difference between this approach and the previous heuristics is that this method assigns all customers to vehicles, and then takes the vehicle routes with the highest reward values to be the solution. In this manner, the heuristic is able to deal with cases that the other methods can not, i.e., if there are whole clusters to be included in the solution, then this method will obtain the best solution regardless of the order in which the clusters are inserted, whereas the previous methods require the best cluster to be inserted first. We will use this technique, i.e., including all customers in feasible routes, in our heuristic methods. This technique enables a greater variety of routines to be considered, as, for example, the Cross exchange

routine can be used to swap subpaths between different routes. The customers that are not currently in the solution are able to be ordered in a effective manner, which leads to an implicit consideration of a greater number of solutions. The possibility also exists for improving a non-solution route sufficiently so this route becomes part of the solution.

One of the interesting routines used in the solution methods is the subgravity approach. In this the aggregate score for a customer is given as the weighted sum of the rewards of all customers, where the weighting for a reward is given by a negative exponential function of the distance. This approach of Golden et al. [74], was improved by Sokkappa [164], so that the score only takes into account the eligible customers, i.e., those that could be feasibly included in the current solution. This measure is therefore updated for each new solution and each new time limit. Sokkappa also proposed that the parameter for the exponential function be a function of the average distance in the problem instance, which enables the method to be sensibly applied to a variety of problem instances. We also will employ the subgravity approach, as it implicitly considers the customers that may be included following an insertion, which seems to be a sensible upgrade over myopically considering the effectiveness of the candidate customer.

The optimal algorithms use variations of Branch-and-Bound within a linear programming environment, to identify the best solution. The algorithms have been able to solve larger problems, due to the development of tighter bounds through using tighter inequalities. Thus, the algorithm of Fischetti et al. [51] was used to solve problem instances for the OP with Euclidean distances, containing up to 300 customers. The time limit in these cases was set equal to 5 hours, and even with this relatively long computational time, the methods could not solve all of the problem instances. Thus, while the state of the art in algorithms for the OP have been able to solve large problem instances, the development of fast heuristic methods and metaheuristics to obtain good solutions quickly, are still warranted. Examples of situations where faster computational times are required, occur in dynamic decision-making, which we consider further in Chapter 9.

3.1.2 Reward Constrained Subset Routing

We now consider problems where there is a lower bound on the total reward to be collected within the solution route, and the objective is to minimize the total cost

of a route. In this case the cost of the route includes the costs of travelling around the selected customers, as well as penalty costs for the customers that are left unscheduled. This problem was first introduced by Balas [8], which he called the Prize Collecting Travelling Salesman Problem (PCTSP). He described a number of facet-defining inequalities for the PCTSP, which may be used within optimization algorithms. In a follow-up paper (see [9]), Balas extended these results to deriving facet-defining inequalities from those for the asymmetric travelling salesman problem.

This problem can be formulated as previously for the OP, except we replace the objective function by the following:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} - \sum_{i=1}^{n} p_i x_{ii} \quad \text{(vii)}$$

where p_i is the penalty that is incurred if the customer is not serviced. We also replace the constraint (iv) by the new constraint:

$$\sum_{i=1}^{n} r_i(1 - x_{ii}) \geq R \quad \text{(viii)}$$

where R is the minimum reward that must be obtained.

Fischetti and Toth [53] described a method of generating tight bounds on the solution quality for the PCTSP. These involve determining a lower bound via a relaxation of the problem, as well as a bound on the difference between the bound and the actual solution. These residual bounds are added in order to obtain tighter bounds, enabling the branch-and-bound algorithm to work more efficiently. They were able to solve some problems with up to 200 customers with non-Euclidean, asymmetric distances.

Gomes, Diniz and Martinhorn [75] developed a GRASP metaheuristic method, with local search using variable neighbourhood structures for obtaining solutions to the PCTSP. The improvement routines involved 2- and 3-opt distance reduction, as well as methods that insert or delete single customers from the solution route. The delete methods allowed infeasibility of the reward constraint to be introduced, with insertion then used to restore feasibility. The metaheuristic was tested on some adapted standard TSP problem instances with up to 360 customers, with methods that create a number of initial solutions and improve upon the best obtained, taking less computational time and obtaining a similar solution quality to methods that begin with a completely random solution.

3.1.3 General Versions of Subset Routing

We have previously discussed papers that relate to two specific versions of subset routing: the OP and the PCTSP. We now consider other versions of subset routing and how these have been presented in the literature.

Beasley and Nascimento [12] developed a framework upon which to classify subset selection routing problems. They called their problem the Vehicle Routing-Allocation Problem (VRAP). In the VRAP, the objective is to minimize the total costs of servicing a set of customers. Each customer is either included in the service route, allocated to one of the customers in the route, or left isolated, and there are different costs for each situation. Under this framework, the OP can be modelled as a VRAP where the costs are the reward of the isolated customers, with a constraint on tour distance added. The PCTSP is a VRAP with routing and isolation costs, with the additional constraint on the minimum reward collected.

Mittenthal and Noon [125] described a general version of subset selection routing, called the Travelling Salesman Subset-Tour Problem with one additional constraint (TSSP+1). They formulated this as a problem where all customers are either included in the solution route or are left isolated, and the objective is to minimize the total cost. The addition of the single constraint may be used to obtain a number of different problems from the literature, e.g., if the constraint is on the maximum length of the solution tour and the costs are the negative of the rewards of a customer, the problem becomes the OP. They created a general insert/delete heuristic that may be used for any version of the TSSP+1, and tested it on a number of random problems that they have generated for the PCTSP.

Pekny, Miller and McRae [135] described a scheduling problem, in which the costs of changing from one job to another depend upon the jobs, there are benefits that accrue for carrying out each job and there are costs incurred for not carrying out each job. They formulated the problem and demonstrated how their general model may incorporate a number of other problems through varying the objective and/or through adding an extra constraint. They described a parallel branch and bound algorithm for solving the PCTSP version of the problem, and gave results for random, non-Euclidean test problems with up to 200 jobs. They also illustrated how the algorithm may be applied in order to obtain solutions for the OP, by iteratively modifying the constraint on the minimum reward to be obtained and attempting to obtain a solution that is feasible with respect to the total time.

Kubo and Kasugai [103] generated valid inequalities for what they call the Subtour Problem, which again is any single vehicle subset selection routing problem. They created several formulations for the problem and compared the bounds for the different formulations. These are then able to be used within an optimization algorithm for the particular application of the subtour problem, although no computational experience was given.

Awerbuch, Azar, Blum and Vempala [5] described the k-MST, which is the minimum spanning tree that visits k nodes. They devised approximation bounds for the k-MST problem and used these results to obtain bounds for the PCTSP of Balas as well as the OP where there is no depot and so there are no points that are required to be in the solution. They stated that the question of obtaining approximation bounds for the OP with fixed points, is still open. The results obtained are only applicable when the rewards are integers.

Malandraki and Daskin [121] devised problems where there are costs incurred for travelling on arcs of the network, rewards are received by either traversing the arcs or visiting the nodes and the objective is to maximize the net reward received. In this problem each customer may be visited as many times as required, although the reward received is a decreasing function of the number of previous visits. They formulated this problem as a minimum cost flow problem, with the rewards given as negative costs, and used branch and bound to solve it. This formulation is only relevant where there are no time limits on the total collection process.

Ladany [104] developed a model for the problem of maximizing the total attractiveness of a tourist bus tour, where there are given attractiveness scores which are gained for traversing the edges and for servicing the nodes. Here, there is a time that is spent in order to service the node, and it is possible to continue on to a succeeding edge without carrying out the service. There may also be costs on both the nodes and the edges, and the objective is to maximize the 'attractiveness' of a tour, subject to constraints on the route duration, the distance travelled and/or the cost of the tour. Ladany gave an integer programming formulation of the problem, from which the optimal solution is obtained, where the numerous subtour elimination constraints are introduced purely as subtours arise.

The article gives a practical test problem involving the generation of a bus route through some tourist sights in Israel. When solving this problem with only route duration constraints, a solution was identified that involves the tour bus

travelling over the same section of route twice. This was said to show a case where the optimal solution is different from that which would be obtained by route planners. We believe, however, that this case is one where the mathematical model is insufficient, as the likely disutility of the repeated section of route may outweigh the additional gains in attractiveness in other sections of the tour.

Deitch and Ladany [31] adapted the problem of Ladany [104], through removing the costs of visiting the nodes and eliminating the constraint on the cost of the tour. This problem therefore entails maximizing the total attractiveness, subject to a constraint on the route duration. They called this version of the problem the Bus Touring Problem (BTP) and found a solution to it by transforming it into an OP. This transformation involves duplicating each node (to account for actually servicing a site rather than just passing through it), and creating a node that relates to each edge, where visiting this node corresponds to travelling along that edge. A solution is found to this OP instance, and this solution is re-transformed to give a solution to the BTP. This transformation doesn't guarantee that a feasible solution to the BTP will be found, which limits its effectiveness. The method therefore takes a problem with n nodes and m edges, and creates a problem with $2n+m-1$ nodes and $2m+n-1$ edges. This seems to be an unnecessary increase in the size of the problem, and solution methods would therefore be better served to be applied to the original problem. Also questionable with this approach is the use of the stochastic algorithm of Tsiligirides [176] for solving the OP, when many other techniques have been found to be more effective.

Volgenant and Jonker [184] defined the Generalized Travelling Salesman Problem (GTSP), in which there is a penalty for not visiting a customer and the objective is to minimize the sum of the routing costs and the non-inclusion penalties. This problem is the same as that of Bienstock, Goemans, Simchi-Levi and Williamson [18], which they named the PCTSP, and it is the same as the PCTSP of Balas, except there are no rewards to collect. We call this the unconstrained-PCTSP as it drops the constraint on the minimum amount of reward that must be collected. Bienstock *et al.* determined that the worst case performance ratio of the linear relaxation for this problem is 2.5. Goemans and Williamson [70] developed a heuristic for this unconstrained-PCTSP with worst case performance ratio of $(2-\frac{1}{n-1})$.

Göthe-Lundgren, Jörnsten and Värbrand [76] solved a problem of allocating

the costs of a VRP to the customers involved. A sub-problem that they solved is a special case of the OP, where the objective is to maximize the net profit of the tour, which is the reward on the nodes less the costs of the arcs. The constraint on the tour length in this problem is that the total demand of the route can not exceed the capacity of the vehicle. This constraint therefore relates to the demand at the nodes, rather than on the arcs as is the case with the OP.

Diaby and Ramesh [36] described the Distribution Problem with Carrier Sequence (DPCS) as a subproblem of their overall vehicle routing problem. In the DPCS there is a capacitated vehicle, with fixed time limit, available for servicing a set of customers. Each customer has a load to be delivered and any customers not included in the service vehicle are serviced by an outside service company. These costs are constant for each customer and can be viewed as penalties for non-servicing. The objective is to minimize the total cost, which is the cost of the service route plus the sum of the penalty costs. The DPCS is therefore the unconstrained-PCTSP, with the added capacity constraint.

Another routing problem containing subset selection elements is the Covering Salesman Problem (CSP) of Current and Schilling [30], in which all customers are required to be within a given distance of one of the customers on the solution tour. Only a subset of customers need to be visited, and the objective is to minimize the total distance travelled in the solution route.

In the Generalized Travelling Salesman Problem (GTSP) described by Laporte and Nobert [109], the objective is to find a minimum distance tour such that at least one customer from each set of customers must be visited. This is therefore a subset selection routing problem, in which the selection decisions relate to which of each fixed set of customers to service. Laporte, Asef-Vaziri and Sriskandarajah [107] modelled a number of problems, including the CSP and arc routing problems, as examples of this GTSP. Laporte and Semet [110] transformed the GTSP into a TSP with double the number of vertices, which can then be solved by standard TSP techniques. The advantage of this approach is not clear, as the increase in the problem size, through the transformation, may make the problem intractable, and they found that the computational time required for 50 node GTSPs was much higher than for their solution method on a TSP with 100 nodes.

Rogerson [152] described another subset selection routing problem based on the TSP. His problem consists of minimizing the expected cost of purchasing an

item from one of a number of stores, where the price of the item at each store is independently and identically distributed. The cost incurred is the purchase price plus the travel costs. This problem reduces to finding the minimum length tour that contains a given number of customers, so is a special case of the PCTSP without penalties.

Verweij and Aardal [183] described the Merchant Subtour Problem (MSP), which is a subset selection routing problem in which the rewards received are a function of the size of the load that is delivered from an origin to a destination location. The objective is to minimize the net costs, which are the costs incurred by travelling less the rewards received. This problem consists of selecting a set of locations to visit, and determining the amount of load to carry between the locations, where the load size decision can be solved in polynomial time. Problem instances of the MSP containing up to 22 customers, were solved optimally using a branch-and-price algorithm, and a Tabu Search algorithm applied to the same instances obtained solutions which have an objective function value within 3% of the optimal value, using far less computational time. This problem introduces some forms of precedence constraints to the subset selection routing problems, and we will investigate techniques for efficiently dealing with these precedence constraints in Chapter 5. Another additional feature that has been included within subset selection routing problems, is reward functions which vary over time, and we will also explore the possibilities related to this phenomenon in later chapters.

3.2 Possible Extensions of the OP

The OP, as previously described, is only interested in the customers that are selected for service and the customers that are not included in the solution are ignored. This is appropriate for the score orienteering event upon which the problem is based, but for cases involving the service of customers other considerations may be required.

We define a *pure* subset selection routing problem, to be one where only the customers that are selected in the solution routes are considered in the objective function. Therefore the OP is a pure subset selection routing problem, as the objective of reward maximization is only concerned with the customers in the solution route, while the other customers are simply ignored.

We also define a *restricted* subset selection routing problem, to be one where some or all of the customers that are not included in the solution routes, are considered in the objective function. In order to apply this definition, we need to make the distinction between the customers in the solution and those that aren't, with these different sets of customers treated differently in the objective function.

We also need to differentiate between a restricted subset selection routing problem and a non-subset selection routing problem. With non-subset selection routing problems, all the customers must be included in the solution, and all customers are treated in the same manner by the objective function. For example with the TSP, all customers must be included within the solution route, and the objective considers the sum of the distances required to service each of these customers. An example of a restricted subset selection routing problem that has been mentioned previously is the version of the OP proposed by Gendreau et al. [64]. This problem involves some customers having to be serviced, and so the objective treats these customers differently to other customers. If there is a compulsory customer that is not serviced in the solution route, the solution obtained is said to be infeasible. This situation could also be modelled by using large penalties for the compulsory customers, and zero penalties for the other customers, where these penalties are incurred if the customer is not in the solution route. In this case the objective is to maximize the reward less the penalties incurred, where the objective function will become large and negative if any of the compulsory customers are not serviced.

We now consider the restricted subset selection routing problem in more detail. For cases where customers are being serviced, we may need to consider each customer in some manner, when creating the solution routes. If all customers are explicitly considered in the objective function and there are different effects, depending on whether the customer is in the solution routes or not, then we have a restricted subset selection routing problem. We consider different methods of modelling customer service as restricted subset selection routing problems.

One method for modelling non-service, would be to consider all the customers in the objective function, with some measure of dissatisfaction for the customers that aren't serviced. The objective becomes a maximization of the net reward, which is the reward received from the serviced customers less the costs incurred for the non-service of the remaining customers. For this purpose we propose the introduction of penalties for non-service and now consider some of the forms these

penalties may take.

One possibility is for the penalties to be known constants that relate to the loss of goodwill through non-servicing. The inclusion of constant penalties makes no difference to the implementation of the OP, apart from changing the values used in the calculations. If no customers are serviced, the value of the objective function is the negative of the sum of the penalties. The addition of each customer increases the value of the objective function by the reward plus the penalty. Therefore we may view this problem as simply the OP, but we are dealing with the "total value" of the customer, which is the sum of the reward and penalty. A difficulty with this form of penalties would be in how to make the rewards and penalties commensurable.

Constant penalties may be used where there are a given set of customers that must be in the solution; these customers are therefore assigned an infinite penalty that is incurred if they are excluded. This problem then becomes one of finding a feasible solution that contains all these compulsory customers, with the addition of optional customers used to improve upon the solution. The effect of finite penalties is to alter the relative priorities of the inclusion of the customers.

Another possibility with the restricted subset selection routing model is to require all customers to be serviced, with costs incurred for the service of the customers which are not serviced in the solution routes. In this case the objective is to maximize the net reward received, which is the reward for the customers in the solution routes less the costs of servicing the remaining customers. One such cost may be the cost of hiring additional vehicles in order to service the remaining customers. This cost may be a constant charge for each customer, in the case where each remaining customer is serviced by an individual vehicle. This approach is similar to that used by Madsen, Ravn and Rygaard [119] for their problem of transporting a number of customers at minimum cost. The customers who were unable to be serviced by the existing fleet of vehicles were serviced by the use of taxis, with there being a given fixed cost for this service. This form of penalty was also used by Diaby and Ramesh [36], where there is a known, fixed cost for the servicing of a customer by the outside company. These constant costs or penalties, may be dealt with in the same manner as the OP, and so this problem is not appreciably different from the OP.

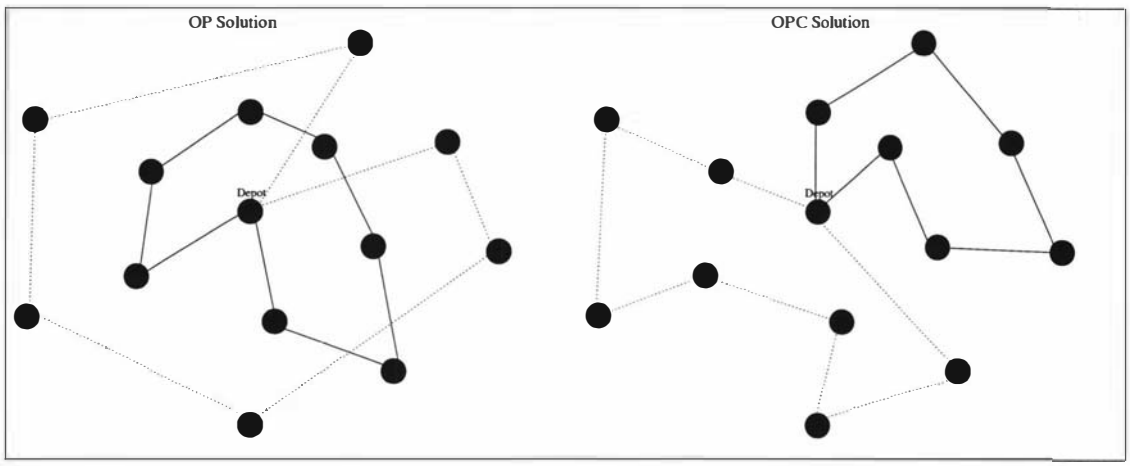

Figure 3.1: Possible Effects of the OPC Formulation

A different case may occur when the penalties relate to the total set of unserviced customers. We define the Orienteering Problem with Clean-up (OPC) for this situation. The OPC involves a number of customers to be serviced by a single vehicle which has a given time window of availability, with the remaining customers being serviced by a fleet of vehicles, each with a given time limit. The objective is to maximize the sum of rewards collected by the service vehicle, less the costs of the additional vehicles, with these costs relating to the total distance travelled. We call the additional vehicles the "clean-up" routes, and so the decisions to be made involve which customers to service and in which order, as well as devising efficient clean-up routes. With low penalties, the main priority is to maximize the reward received with little consideration for the penalties, but with high penalties the importance of the rewards becomes diminished as the minimization of the clean-up costs becomes more relevant.

An example of the OPC is shown in Figure 3.1, where we assume that all customers have the same reward and the cost of the clean-up tour is significant. For the OP solution, seven customers are serviced in a circular tour around the depot, given by the solid line. The minimum distance tour that includes the remaining customers is shown as a dotted line, although these customers are not explicitly considered in the objective function. With the OPC solution we consider all customers in the objective function, and so only six customers are serviced in the solution route. The difference between this solution and the one for the OP is that the unserviced customers are all in the same region, so greater concern is given for how to effectively deal with these customers.

A slightly different situation is where the existing service vehicles must service each of the customers. Reward is only received for those customers which are serviced within the time limit, and costs are incurred for the time by which the time limit constraint is exceeded. This problem also requires different priorities of the reward maximization and distance minimization to be considered.

3.2.1 Computational Results

We now create some test problems for the OPC and test some methods for solving these instances heuristically. The OPC is of interest, as it involves a logical extension to the OP, which creates a restricted subset selection routing problem. This problem may arise in home delivery systems where there is a time limit over which service is deemed acceptable (which corresponds to the service vehicle's route), with all customers needing to be serviced, and 'poor' service earning no reward.

The version of the OPC that we are considering has a single service vehicle with a given time limit, a single clean-up vehicle with no time limit, and the objective is to maximize the reward collected less the costs of the clean-up tour. This problem can be formulated as follows:

$$
\begin{aligned}
\max \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} r_i x_{ij} - P\sum_{i=0}^{n}\sum_{j=0}^{n} c_{ij} y_{ij} \\
\text{s.t.} \quad & \sum_{j=0}^{n}(x_{ij} + y_{ij}) = 1, \quad i = 1, \ldots, n \\
& \sum_{i=1}^{n}(x_{ij} + y_{ij}) = 1, \quad j = 1, \ldots, n \\
& \sum_{j=1}^{n}(x_{0j} + y_{0j}) = 2 \\
& \sum_{i=1}^{n}(x_{i0} + y_{i0}) = 2 \\
& \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij} x_{ij} \leq C \\
& 2\sum_{\nu_k \in S}\left(\sum_{j=0}^{n} x_{kj}\right) \leq |S|\left(\sum_{\substack{\nu_i \in S \\ \nu_j \notin S}} x_{ij} + \sum_{\substack{\nu_i \notin S \\ \nu_j \in S}} x_{ij}\right), \quad |S| \geq 3
\end{aligned}
$$

$$x_{ij} \in \{0,1\}, \quad i = 0,\ldots,n, \quad j = 0,\ldots,n$$

$$y_{ij} \in \{0,1\}, \quad i = 0,\ldots,n, \quad j = 0,\ldots,n$$

In this case, the x_{ij} variables correspond to the solution vehicle and the y_{ij} variables correspond to the clean-up vehicle. We ensure that each customer is included only once, except for the depot which is included in both routes.

We assume that the vehicles travel at unit speed, so the time limit represents the maximum distance that can be travelled in the solution route. If there was a time limit on the clean-up vehicle then the structure of the problem would change, as the number of vehicles used could become a decision variable or issues of feasibility of the allowed vehicle routes could become more important than the selection elements. The costs of the clean-up vehicle are given as a constant penalty factor, P, times the distance of the clean-up route. We randomly generated 30 layouts for the OPC with 50 customers in each and the depot being centrally located. Of these layouts, ten were generated from the uniform distribution, ten were with customers uniformly distributed within five even sized clusters, and ten were with 75% of the customers located within a central cluster and the remaining customers distributed as outliers. For each of our layouts we set the time limit at three levels, with these being 40%, 60% and 80% of our estimate of the length of the TSP tour through all the customers. The customer rewards were set equal to their distance from the origin, as this gives a circumstance where greater reward is received from the 'harder' customers. We intend solving the problem over the range of values of P for which the objective function is positive, so we start with $P = 0$ and increment it in steps of 0.1 until the objective value obtained becomes negative.

We describe our method, OPCB4B, for constructing solutions to this problem, in Algorithm 3.1. Our estimate of the TSP length was generated through the first phase of our solution method. We start by putting all the customers into a single clean-up route, C, using the GENIUS method, which has been shown to be an effective construction and improvement technique for the TSP [60]. We initialize our solution route, T, to just include the depot. We use a randomized bang-for-buck approach whereby we select one of the five customers with the highest ratio of reward change to distance change for the insertion, where the probabilities of selection are proportional to the size of the ratio. The reward change is given

Algorithm 3.1 function OPCB4B

```
// Create the clean-up route, C
Assign all customers to the clean-up route C, using GENIUS
numavail ← n
sizeroute ← 0, T ← {0}  // Initialize the solution route, T
for (i = 1 to n) do
    avail_i ← i
end
while (numavail > 0) do  // While there are still candidate customers for inserting into T
    for (i = 1 to 5) do  // Initialize where we store our improving moves
        change_i ← 0, which_i ← 0, place_i ← 0
    end
    for (i = 1 to numavail) do  // For each candidate for moving
        maxratio ← 0
        for (j = 1 to sizeroute + 1) do  // For each position to insert it in route T
            δ_ij ← the B4B ratio of inserting avail_i in position j of T
            if (δ_ij > maxratio) then
                best_j ← j, maxratio ← δ_ij
            end
        end
        if (no insertion of avail_i was feasible) then
            numavail ← numavail − 1
            for (j = i to numavail) do
                avail_j ← avail_{j+1}
            end
        else
            if ((maxratio is the j'th highest ratio found so far)&(j ≤ 5)) then
                change_j ← maxratio, which_j ← i, place_j ← best_j
                Update the subsequent ratios and quantities stored
            end
        end
    end
    for (j = 1 to 5) do
        sum ← sum + change_j
    end
    if (sum = 0) then  // No improving candidates so stop
        numavail ← 0
    else
        // Randomly select one of the candidate moves and implement this
        for (j = 1 to 5) do
            p_j ← change_j/sum
        end
        Choose customer k according to the selection probabilities p_j
        Insert customer avail_{which_k} in position place_k and update T and C
        numavail ← numavail − 1
        for (j = best_i to numavail) do
            avail_j ← avail_{j+1}
        end
    end
end

end
```

by the additional reward of the moved customer, in tour T, plus P multiplied by the reduction in the distance of C that is obtained through removing the customer. The improvement techniques we use involve the application of inter-route Move and Exchange and Cross path exchange, where the move that increases the objective function is carried out. We varied the method of evaluating the neighbourhoods of the local search techniques, and this lead to the development of the different methods which use in the testing.

Our solution method therefore consists of using a randomized construction technique with improvement to repetitively solve slightly different problem instances, i.e., problems with different values of P. For each instance we obtain a value of the reward collected in the service vehicle's route and the distance for the clean-up vehicle, and we store the non-inferior solution set over all possible values of the P. We are therefore using randomization, with repetition through perturbation of the problem instance, in order to generate a range of good solutions.

We used the above solution methods on each of the 30 test problems described above, with each of the three time limits. The construction method is the same for each of the four methods, but the improvement techniques are varied, and for each method we stop once no improving solution is found. For method 1, the entire neighbourhood obtained by inter-route Move and inter-route Exchange is evaluated, and the best improving perturbation is carried out. With method 2, we evaluate the entire neighbourhood obtained by the inter-route Exchange and Cross path exchange routines, and carry out the best improving perturbation obtained. For method 3, Move, Exchange and Cross are considered successively, with Exchange only considered when no improving Moves are feasible and Cross only considered when no improving Exchanges are feasible.

We illustrate the solution obtained through using the third improvement method for our first problem instance with random layouts and with the lowest time limit, and give the results in Table 3.1. We solve the problem instance with P initially set to zero, and we increment this penalty factor by 0.1, until five consecutive values obtain a negative value of the objective function. For our problem instance, the last positive value obtained was for $P = 1.8$. In the Table the first two columns give P and the best objective function value that we have found for this penalty factor, which we calculate from the non-inferior solution set we obtain. The final three columns give the objective function value that we obtain for this particular

P	Best Obj Val	Actual Obj Val	Reward	Distance
0.0	6.734837	6.219901	6.219901	3.758140
0.1	6.361391	6.268208	6.629894	3.616855
0.2	5.991441	5.901858	6.658464	3.783028
0.3	5.619742	5.578017	6.629894	3.506255
0.4	5.248044	5.145253	6.658464	3.783028
0.5	4.876767	4.876348	6.734837	3.716979
0.6	4.526141	4.504650	6.734837	3.716979
0.7	4.175515	4.026611	6.560331	3.619600
0.8	3.824890	3.394623	6.219901	3.531598
0.9	3.474265	3.133071	6.127259	3.326876
1.0	3.123639	2.514071	6.178533	3.664462
1.1	2.773013	2.503618	6.620710	3.742811
1.2	2.422388	2.327279	6.629894	3.585512
1.3	2.071762	1.854851	6.560331	3.619600
1.4	1.721137	1.469633	6.127259	3.326876
1.5	1.370511	1.124476	6.165987	3.361007
1.6	1.019886	0.019954	5.665549	3.528496
1.7	0.669260	0.096365	6.034832	3.493216
1.8	0.318635	0.025483	6.366137	3.522586

Table 3.1: An Example Solution for the OPC

heuristic, as well as the actual reward and actual distance of the solution obtained.

The noninferior set we obtain for this problem is the solution with solution reward of 6.734837 and clean-up distance of 3.716979, which is the best solution obtained for $0.0 \leq P \leq 0.4980$, and the solution with solution reward of 6.629894 and clean-up distance of 3.506255, which is the best solution up until the reward reaches zero. We give these solutions in Figures 3.2 and 3.3. The Figures show the large difference that occurs between having an objective that is focussed on reward maximization (as is the case for the lower penalty factors in Figure 3.2) and one that considers both the reward collected and the clean-up distance (as for Figure 3.3).

We see from Table 3.1 that our solution method is not very effective in obtaining the actual solutions for a particular value of P. This can be seen through comparing the columns Best Obj Val and Actual Obj Val, and noting that for none of the values of P did we obtain the same solution with our heuristic as the best solution from the non-inferior set. This is due to the randomness involved with our solution

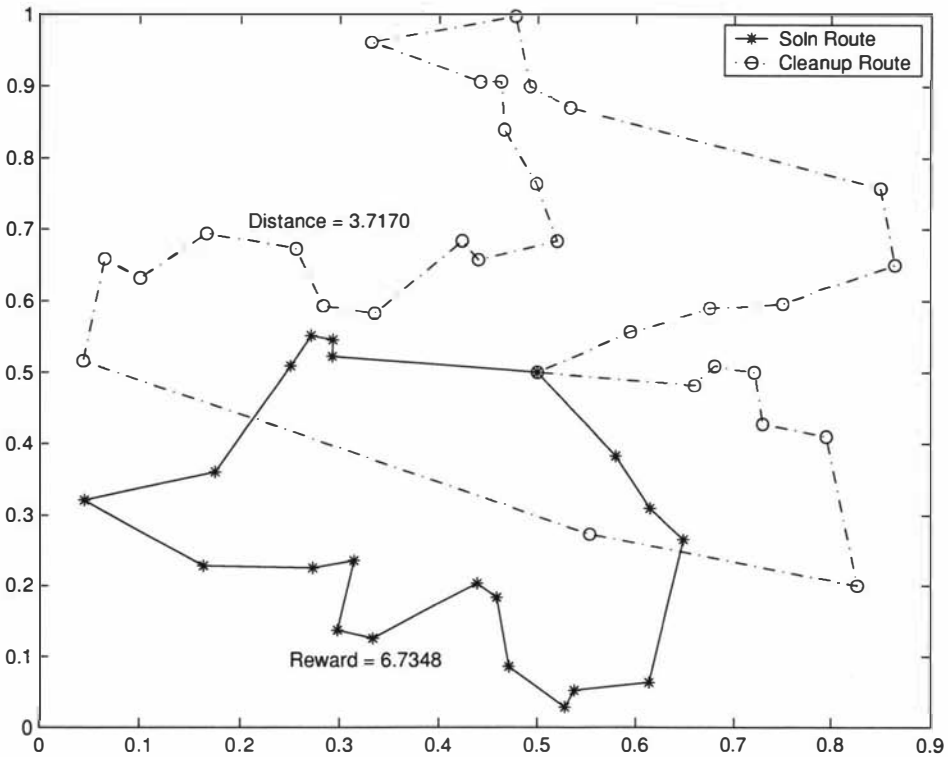

Figure 3.2: First Non-Inferior OPC Solution

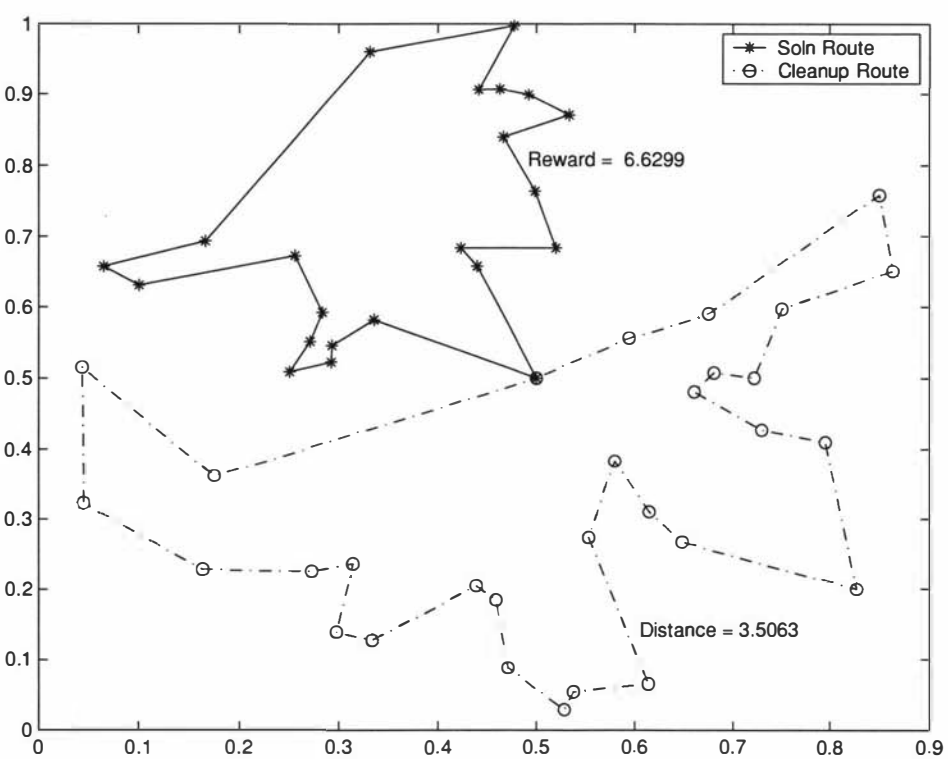

Figure 3.3: Second Non-Inferior OPC Solution

Type	Limit	Method 1	Method 2	Method 3
1	40	68.253739	71.002776	70.541427
	60	223.729410	239.835206	239.378625
	80	753.867385	802.956020	799.960848
2	40	67.175271	69.254534	68.744597
	60	213.200044	229.192801	227.825712
	80	628.520175	648.865542	637.474503
3	40	41.355650	43.767978	43.147109
	60	107.401714	120.475820	121.050098
	80	305.044061	353.726651	341.274873
Total Time (mm:ss)		04:13	17:46	10:17

Table 3.2: Collated Results for OPC Instances

technique, which explores different areas of the solution space, in order to obtain a reasonable solution to the current problem. This randomness is an important part of our solution technique, as we found that obtaining a greater number of reasonable solutions aided in finding improved solutions for the range of penalty factors considered. The unpredictability of the solution quality would invalidate the use of a single run of the solution routine for a particular problem instance, with either multiple repetitions on the particular problem instance or repeated solutions from the perturbed instances, being required to obtain effective solutions.

The results of the testing on the relative effectiveness of the methods are shown in Table 3.2. The values within the Table are the total score over the 10 repetitions, with the score for each instance given by the area under the curve defined by the non-inferior solution set over the range of values the reward remains positive. This enables us to obtain a single figure to summarise the range of solutions obtained for a particular problem instance. The times given are the running times for solving the 90 problem instances on a DEC 1100 (24MHz), and are included to indicate the relative running times of the methods.

From the Table we see that the solution quality improves as the solution time increases, with the inclusion of the Cross routine significantly increasing both the quality and the time. The most effective improvement method appears to be using the best improvement from the Move, Exchange, and Cross techniques. This significantly improves upon the use of the Exchange with Move, which is a subset of the Cross technique. A compromise in solution quality and solution time, is to only

use the Cross technique when the Move and Exchange techniques can not improve the solution further. We conclude from these problems, that the Cross technique appears to be quite effective although it significantly increases the running time over the use of Exchange and Move. If running time is of the most importance, then the restricted use of Cross, such as in method 2 or through restricting the length of paths considered for exchanging, may be required.

3.3 Conclusions

In this chapter we have explored previous subset selection problems from the literature, and have defined different forms of subset selection problems. The application to servicing customers leads to different ways of dealing with non-service, where penalties of various forms may be used to prevent the service company from completely disregarding the customers that it doesn't select to be serviced. We have developed a model for the OPC, which is a natural extension to the OP, with all customers counting toward the objective function value. In this problem there is a cleanup route that is used to service the unscheduled customers, and we have considered some basic techniques for obtaining solutions to this problem. Other definitions of penalties may require different techniques, particularly for evaluating the effectiveness of the solution produced.

The subset selection that relates to the OP will occur when it is not possible to service all of the available customers over the available scheduling horizon, as it would be more profitable to include all customers, if it were feasible. We consider in later chapters, cases where it may be beneficial to exclude some customers from service, even though it may be possible to service them. One occurrence of this type of subset selection problem, which we consider further, is the case where the reward functions vary over time, and forcing the service of a customer may lead to lower reward reward being obtained from the other customers.

Further situations where gains may be made by servicing fewer customers may occur if the objective is to maximize the net reward, which is the total reward collected less the cost of collection. In this case the time limit may be unnecessary as the collection will cease when the cost of further collection exceeds the potential reward. In cases with the time limit, it may be possible to extend the time of servicing or increase the rate of servicing, while incurring a higher cost. This

problem, where the time limit is effectively a decision variable, is similar to that of Keller and Goodchild [96], where the non-inferior set of reward collected for each time limit is computed. Using this set, we can identify the best time so that the net rewards are maximized.

This net reward problem definition can be used to incorporate most VRSP instances within the subset selection framework. For example, if there are a number of customers each with a given load that needs to be delivered, there is a fleet of vehicles available for servicing the customers and the reward for servicing each customer is very high compared to the cost of servicing, then we have a version of the VRP. Thus, all effective solutions will service all the available customers, but the best solution will be the one that minimizes the total distance or cost of the service routes.

CHAPTER 4

Time Concerns

In a number of vehicle routing applications the time at which a customer is serviced is an important consideration. For example, a customer may desire service as soon as possible for an urgent need or may require service during certain hours due to resource availability. In this chapter we consider how time considerations may arise in VRSPs and what impact they may have.

4.1 Time Windows

An important addition to vehicle routing problems has been the inclusion of constraints on the time at which a customer is allowed to be serviced. We define a *time window* to be a continuous period of time during which a customer may be serviced, where each customer may have one or more windows of availability. There have been many published applications of time windows for vehicle routing and scheduling problems.

The Travelling Salesman Problem with Time Windows (TSPTW) takes the traditional TSP and adds restrictions on the time at which each customer may be serviced. All customers need to be serviced within their given time window and the objective is to either minimize the total time taken or the total distance travelled. Baker [6] solved small problems exactly using branch-and-bound. He reduced the number of arcs being considered by removing arcs between pairs of nodes with incompatible time windows, e.g., remove arc (i, j) if $l_i + d_{ij} > u_j$ where the time window for a customer i is given by $[l_i, u_i]$ and t_{ij} is the travel time between

customers i and j. Different problem instances were created through varying the number of customers whose time window is 'active', or binding. These problem instances involved tight time windows, and he was able to solve problems with up to 50 customers with most of the time windows active.

Savelsbergh [154] showed that even finding a solution to the TSPTW is an $\mathcal{NP}$-complete problem, and he therefore proposed local search interchange heuristics. Tsitsiklis [177] developed further complexity results for some special cases involving the TSPTW on a line, and on a plane with a bounded number of locations.

Langevin, Desrochers, Desrosiers, Gelinas and Soumis [106] adapted a two-commodity flow formulation for the TSP to problems allowing time windows. They defined the TSPTW to be the problem where the objective is to minimize the total distance travelled, and the Makespan Problem with Time Windows (MPTW) involves the minimization of the total time taken. Their two-commodity formulation involves two commodities corresponding to time taken, with one being picked up and one being delivered at each node. They reduced the size of the network, optimally solved the linear relaxation of the minimum cost flow problem, and then used branch and bound on the violated integer constraints to obtain the optimal integer solution. The method was tested on problems containing up to 60 customers, although with 60 customers many of the problem instances were unable to be solved.

Dumas, Desrosiers, Gelinas and Solomon [41] developed a dynamic programming algorithm to optimally solve the TSPTW. They reduced the number of states that need to be considered, by only including customers whose insertion can allow a feasible solution to be obtained. They reported being able to optimally solve problems with up to 200 customers with narrow time windows, and with fewer customers for problems with wider time windows.

Mingozzi, Bianco and Ricciardelli [124] used dynamic programming for the TSPTW with precedence constraints also present. They reduced the state space through checking that appending a customer to the current sub-path won't prevent any of the other customers from being feasibly included next, and through creating bounds on the total extra cost to insert the remaining customers.

Gendreau, Hertz, Laporte and Stan [62] adapted GENI (the Generalized Insertion method of Gendreau, Hertz and Laporte [60]) for use on the TSPTW. This method involves sequentially inserting customers to the service vehicle's route,

through a generalized insertion that allows sub-paths of the current route to be reversed. A customer can only be inserted between points that are in the set of its nearest neighbours. For the GENI method for the TSP, customers are placed in the solution route in random order, but with the feasibility concerns for the TSPTW this is no longer applicable. In order to construct solutions to the TSPTW, customers were added in order of difficulty of insertion, which is measured by the width of the customer's time window. Further adaptations to the GENI method that are required for the TSPTW include the ability to backtrack by removing a customer from the current solution if it is not possible to obtain a feasible solution from the current route and a redefinition of the neighbourhoods, since the tour has a given orientation when time windows are present. The advantage of this method is its ability to find feasible solutions for a number of test problems, with problems with wide time windows being able to be solved. Comparisons were made with the results of Langevin et al. [106], with the GENI method obtaining greater numbers of feasible solutions, while obtaining optimal solutions for most of the problems with 20 and 40 customers.

Time windows have also been included frequently in the VRSP, with the problem created being called the VRSPTW. Solomon [166] introduced the VRSPTW and adapted existing heuristic methods for the VRSP: the savings method of Clarke and Wright [29], nearest-neighbour, Gillett and Miller's [67] sweep heuristic and insertion under different criteria, to obtain solutions for the VRSPTW. He developed a set of test problems and compared the effectiveness of the heuristics over these problems. The most effective heuristic was an insertion approach that took into account both the additional distance and additional time required for inserting the customer. The problem sets developed have been incorporated as the standard for the VRSPTW, with most subsequent papers comparing the effectiveness of their methods with other published results on these problems.

Solomon, Baker and Schaffer [167] described techniques for efficiently implementing local search methods for the VRSPTW. These techniques involve the preprocessing of customers, to create precedence constraints where the time window constraints will not allow a customer to be serviced before another in a feasible solution route. While these precedence constraints are not sufficient to determine whether a proposed move is feasible, they eliminate the consideration of a number of moves, thereby reducing the computational requirement. The preprocessing was

found to significantly reduce upon the computational time required to carry out local search on Solomon's test problems, when compared with a straightforward implementation.

Desrochers, Lenstra, Savelsbergh and Soumis [34] gave a survey on the techniques that had previously been applied to various forms of the VRSPTW. They described optimization methods which include dynamic programming and column generation, and approximation methods for constructing and improving feasible solutions. The survey concluded that the area of constrained routing is an important and very active area of research that warranted much additional investigation.

Van Landeghem [182] developed an insertion heuristic for solving the VRSPTW. To evaluate the cost of an insertion, the heuristic uses a weighted sum of the savings from adding the customer to a route compared with serving it separately, and the change in the flexibility for the route with the customer added. This is a tradeoff between the current improvements to the route, and the possibility of future improvements.

Desrochers, Desrosiers and Solomon [32] developed an optimal solution algorithm for the VRSPTW. This method involves column generation, with columns generated through using dynamic programming to solve a constrained shortest path problem. This algorithm was used to solve a subset of Solomon's benchmark problems with short planning horizon, where all problems containing 25 customers could be solved, but few instances containing 50 or more customers could be solved within reasonable computational time for any problems apart from those with clustered customer locations.

Thompson and Psaraftis [174] used a multi-vehicle inter-route improvement technique of transferring a subset of customers from one route to another, where the routes considered form a cycle. So, if the current cycle considered was 1—3—4—1, a set of customers is moved from vehicle 1 to vehicle 3, with another set moved from vehicle 3 to vehicle 4 and a further set moved from vehicle 4 to vehicle 1. They claimed that their heuristic is effective in improving initial solutions and is comparable with any published literature. The computational complexity of this method contributes to its effectiveness, but also seems to restrict its practicality due to the computational time required.

Savelsbergh and Sol [159] developed an optimization-based heuristic using set partitioning and column generation to obtain solutions to a large, real-life problem.

The problem involves pick-up and delivery, with time windows on each location and dynamic customer arrivals. The other time consideration is the timing of breaks for the vehicle, as the problem is continuous over many days, and the driver must be allocated appropriate breaks throughout the schedule. The dynamic nature of the problem, effectively reduces the size of the problem, but nevertheless, heuristic methods are required to supplement the optimization methods in order to obtain feasible solutions within acceptable computational time.

There has been a lot of recent research on the VRSPTW, focussing on creating metaheuristic solution methods. Potvin, Kervahut, Garcia and Rousseau [139] obtained solutions to Solomon's test problems using Tabu Search. The solutions found are very effective, and are comparable with the best heuristic solutions obtained, although with a higher computational time requirement. Chiang and Russell [28] also implemented a version of Tabu Search on Solomon's test problems, where the method is called 'reactive' Tabu Search as the size of the tabu list varies depending upon the current state of the search. The quality of the solutions obtained was said to be competitive with other heuristic methods, with the computational time requirement said to be small enough to allow solutions to larger problems to be obtained. Potvin and Bengio [138] used a Genetic Algorithm to also obtain solutions for Solomon's test problems, with the method being particularly effective for the problems involving clustered customers.

As previously mentioned, there has been one paper that involves time windows applied to the Orienteering Problem, creating the OPTW. This paper was by Kantor and Rosenwein [91], and since the OPTW involves subset selection with constraints on the customers, it is therefore very relevant to our overall study, so we consider the paper in more detail than the previous time window problems. This problem can be formulated using the objective function (i) and the previously defined constraints (ii)–(iv), as were previously used for the OP, but with the following constraints also required:

$$t_j \geq t_i + c_{ij} - Mx_{ij}, \quad i = 0, \ldots, n, \quad j = 0, \ldots, n \quad \text{(ix)}$$

$$l_i(1 - x_{ii}) \leq t_i \leq u_i + Mx_{ii}, \quad i = 0, \ldots, n \quad \text{(x)}$$

Constraints (ix) ensure that the time each customer is serviced, t_j, is after the time of its predecessor in the solution route plus the travelling time between these

customers. Constraints (x) ensure that each customer, i, in the solution route is serviced within its time window of availability, $[l_i, u_i]$.

Kantor and Rosenwein [91] claimed that the OPTW is a heavily constrained problem, so it is amenable to solving using a tree search. They used an abbreviated branch-and-bound search, with partial routes abandoned if they don't meet some heuristically determined criteria. They also used an insertion heuristic, wherein the customer with the highest ratio of additional reward to additional distance is inserted in its best position. The solution methods were tested on one set of data, with the time limit, T_{max}, being varied and the width of each time window being a randomly generated from a distribution that relates to T_{max}. The insertion heuristic used, is concerned only with the distance of a feasible insertion, and needs to also be concerned with the additional time requirements, so as to be applicable for problems where waiting times are required. The constraints in the tree search heuristic need to be selected carefully, in order to ensure that the method does not exclude too many promising solutions. This involves a significant trade-off, as if there is little restraint on the search, the method will be similar to an optimization method, with similar effectiveness but high computational time required. If the search is too restricted, the solutions will be obtained quickly, but their quality is likely to be poor.

Generalizations of the Time Windows

We now consider generalizations of time windows that are appropriate for the VRSP. One such generalization is to allow a customer to have multiple windows, e.g., at any time during working hours as long as it isn't during a given break. If a vehicle were to arrive between time windows, it would have to wait until the start of the next time window in order to service the customer. Previous references to multiple time windows have occurred in the context of the TSP (see [153]) and the VRP (see [57]).

Problems with deadlines or earliest allowable service times may be considered to have one-sided time windows. These are therefore special cases of the general time window problem. Thangiah, Osman, Vinayagamoorthy and Sun [172] develop three heuristics for the VRP with deadlines, with these heuristics based on the methods of sweep and insertion, and with a Genetic Algorithm used to create clusters of customers that are inserted into the solution routes.

Another generalization of the time windows is to allow some customers to be serviced outside their given time interval. We define a *soft time window* to be one that may be violated, thereby incurring an appropriate penalty. Analogously, our previous examples, where the customer must be serviced within the given time window, may be referred to as hard time window problems. In routing problems with soft time windows the objective is to minimize the total cost of the routing, i.e., the sum of the travel costs and the total penalties for time window violations. The hard time window case is a special case of soft time windows, where the penalty for violating a time window is infinite.

Koskosidis, Powell and Solomon [99] gave some advantages of modelling problems with soft time windows rather than hard time windows. These include:

- the increased generality, and the ability to encompass hard time window situations within the model;
- the practicality of the soft time windows, in that they are more likely to be used in real situations;
- the greater flexibility in the solution, as the penalties may be varied in order to alter the priorities of servicing certain customers, or alter the relative priority of the minimization of travel costs;
- the ability to always obtain a 'feasible' solution, whereas even finding feasible solutions to problems with hard time windows may be difficult.

Sexton and Choi [162] described a pick-up and delivery problem where there are soft time constraints on both the pick-up and delivery points. The penalties are given as linear functions of the deviation between the time window and the time of servicing. They solved this using Benders' decomposition to create a route, and optimally solved the scheduling subproblem as a network flow problem.

Koskosidis et al. [99] described the VRP with soft time windows (VRPSTW). Their penalty costs were linear functions of the total earliness and lateness deviations, where there was able to be different weightings for the earliness and lateness, although the weightings were the same for each customer. They solved this using an adaptation of Fisher and Jaikumar's [54] generalized assignment procedure to form clusters, and solved the TSP through the customers in the clusters using

either complete enumeration or the Lin and Kernighan [114] edge exchange algorithm. The problems were tested on the benchmark problems of Solomon, where the penalties were increased in order to promote service within the time windows.

Dumas, Soumis and Desrosiers [43] developed an algorithm for optimally solving the scheduling component of a soft time window problem, where the violation penalty function may take any general convex form. From an initial, fixed route, the optimal time to service each of the customers within the route is found. The algorithm proceeds by grouping together certain sequences of customers and finding the optimal service time for these sets. This procedure is then able to include waiting times, in order to reduce the total penalty over the vehicle route through delaying the time of servicing some of the customers. This method can, in practice, be used to identify the optimal waiting times for any problem for which the sum of a penalty functions is convex and optimal solution to this sum of functions can be found.

Balakrishnan [7] described the VRPSTW, where there are linear penalties for deviations from the time windows, with each customer having different penalty coefficients for earliness and lateness. He developed simple heuristic procedures, and demonstrated that the extra flexibility in using soft time windows could lead to improvements to hard time window solutions. Thus, by not requiring all customers to be serviced within their time window, the number of vehicles required and/or the cost of routing may be reduced.

Taillard et al. [171] described a version of the VRPSTW where the customer may not be served prior to its time window, and there are linear penalties for lateness. They developed a Tabu Search heuristic for obtaining solutions for the VRPSTW, and test it on Solomon's benchmark problems. Although these problems contain hard time windows, the heuristic was able to improve upon the best known solutions for some of these problems.

Other versions of soft time windows may be seen in the context of the Dial-a-Ride Problem (DARP). In the DARP, each customer comprises a pick-up and a delivery and the objective is to minimize the sum of routing costs and customer inconvenience costs. Sexton and Bodin [160, 161] formulated a DARP where the objective is to minimize the sum of inconvenience. Each customer i has a given deadline τ_i, and the inconvenience is the weighted sum of the deviation from the deadline and the excess time the customer is in the vehicle. The deadline can

be considered to be an infinitesimally small time window, where the penalties for violating the time window are given by:

$$P_i = \begin{cases} A(\tau_i - y_i) & \text{if } y_i \leq \tau_i \\ \infty & \text{otherwise} \end{cases}$$

where A is the weighting factor for the deadline deviation and y_i is the time at which customer i is serviced. Sexton and Bodin solved this problem using Benders' decomposition and separation of the problem into its routing and scheduling components. Psaraftis [144] described two heuristic methods for solving the DARP: one that decomposes the set of customers into groups and further into clusters, before carrying out the routing, and the other involving a sequential insertion algorithm. Each customer is assumed to have either a desired delivery time or a desired pick-up time, where these times may again be considered to be infinitesimally small time windows. The penalties for violating these windows were given as functions containing both linear and quadratic terms, which he suggested is a more general and a more appropriate form than linear functions. This allows the customer dissatisfaction to increase at a higher rate as the delays increase, rather than having a constant rate for all delays, as is the case with linear functions.

Another generalization of the time windows problem is to allow the time windows of certain sets of customers to be varied. An example of this type is the sliding time window problem of Ferland and Fortin [49]. This problem requires the customers to be partitioned into sets, where it is possible to slide the time windows of all customers within a given set by shifting the time windows by the same amount. The example they gave is with school bus scheduling, where the sets correspond to the schools that have the same ending time, and there are given time windows for when the buses must pick-up the students. This problem assumes it is possible to slightly alter the time at which all these schools are serviced, in order to reduce the fleet size or the total distance travelled. They obtained solutions to this problem by fixing the time windows of all schools and solving a VRPTW, then improving through looking to find pairs of schools where the solution would be improved if their time windows were altered to allow them to be serviced by the same vehicle.

A form of time restriction that doesn't explicitly restrict the service times is to restrict the position in the solution route that a customer may be serviced. For the DARP with dynamic arrivals, Psaraftis [141] defined Maximum Position Shift

(MPS), to be the maximum allowed deviation between the order the customer arrives for service and the order the service takes place. This is used in order to prevent any customer from having their service deferred indefinitely. Bianco, Mingozzi, Ricciardelli and Spadoni [17] also used MPS for their DARP. They created an initial feasible tour, and used η_i (the position of customer i in the solution) to simulate the order the customers arrive. The requirement for the solution is that the position of customer i must lie in the range $\{\eta_i - \text{MPS}, \eta_i + \text{MPS}\}$. This imposes some limit on the times of servicing the customers. The MPS is also used in the definition of these problems, in order to restrict the size of the state space so that the dynamic programming algorithm is able to solve these problems.

4.2 Time-Dependent Rewards

The Orienteering Problem (OP), as it was formulated by Tsiligirides [176], has rewards being constant for the duration of the available time period. We now look at instances where it may be more suitable for the rewards offered to vary over time. In these cases the reward received for the servicing of a customer is dependent on the time at which the service is initiated or completed. We consider, for the time being, that each customer consists of a single location to be visited, and so the reward received for servicing customer i to be given by the function $R(i, t_i)$ which is a function of t_i, the time at which the service is completed. We formulate this problem using the previous constraints (ii)–(vi) and (ix)–(x) but we use the following objective function:

$$\max \sum_{i=1}^{n} R(i, t_i) * (1 - x_{ii}) \qquad \text{(xi)}$$

The field of inventory routing involves problems where time-dependent rewards may arise. Bell et al. [13] described an application involving the delivery of gases to a number of customers, where the objective was to maximize the net reward of the routing, i.e., the rewards received for supplying customers minus the costs of routing. The reward received depends upon the quantity of gas that is delivered, where the demand for a customer depends upon the time since the customer was last serviced, and therefore increases over time. Restrictions on the time of servicing and the minimum inventory allowed at a customer before servicing must take place,

are also involved in this problem.

Explicit time-dependent rewards, in problems involving subset selection, were discussed in the PhD thesis of Keller [94]. He described a time-dependent vending problem (TDVP), where the reward received for visiting a particular location depends upon the demand at that location and the level of demand is dependent upon the time at which the location is visited. An example that was given of this type of demand function is in election campaigning, where the level of interest in a meeting will rise as the election approaches before dropping away immediately after the election is held. The base problem that was considered is the OP with time-dependent rewards, although this was considered to be a general multi-objective problem where the two conflicting objectives are to minimize the total cost of travel while maximizing the total reward collected. However, for any fixed time limit, the objective is to maximize the reward received.

The reward functions for these problems are given by

$$\mathrm{REX}_{O_i, A_{O_i}} = \alpha(A_{O_i}) R_{O_i}$$

where O_i is the ith location in the route and A_{O_i} is the time it is visited, $\alpha(t)$ is a time-dependent function of the time, t, at which the location is serviced and R_{O_i} is the maximum reward that is offered at location i. The time-dependency of the reward function gives the proportion of the maximum available reward that can be obtained at any time. It is assumed that the vehicle must remain at the location for a period of time given to be a linear function of the reward that is received. This models a sort of 'harvesting rate', whereby each unit of demand takes a set time to service. Three reward functions were considered: quadratic functions, oscillating functions based on cosine curves, and approximation of Gaussian curves. A common reward function, $\alpha(t)$, was applied to each of the locations.

Keller also discussed a number of possible extensions that could be applied to the TDVP. These include allowing the reward received to be non-linear over the length of the vehicle's stay at a location and allowing the vehicle to leave a location before all the demand is satisfied. An example where this may occur was given in a follow-up paper by Keller and Goodchild [96], in the case of a mobile lunch vendor. Here the demand will increase as a result of the vehicle's arrival at a location, but will peak and then tail off. Other extensions include allowing for different reward functions at each location, allowing for a vehicle to spend some time waiting in order to receive greater reward at a location rather than servicing the demand that

is currently available, and allowing the demand to replenish over time so it may be viable to return to a particular location to service the demand that has accrued since the location was last serviced. An interesting case here is, if the demand at a location replenishes quickly enough, it may be viable to remain at this location to set up shop instead of travelling to service demands at other locations.

Hill, Mabert and Montgomery [83] described an application relating to banking where the rewards are time-dependent. The problem involved a courier company visiting a number of branches of banks in order to collect cheques. The volume and value of cheques at each bank increase over time. The objective is to maximize the ratio of

$$\frac{\text{total value of cheques collected}}{\text{the time taken for the route}}$$

where this ratio is further penalized if the routing violates certain guidelines, i.e., deviation from the desired number of stops, exceeding the desired routing time and not carrying enough items. The reward in this case is the value of the cheques collected. In this problem it is possible to collect from a branch more than once each day, with the reward received depending on the time that has elapsed since a vehicle last visited the branch.

Another application of rewards varying over time is the Maximum Collection Problem with Time-Dependent Rewards (MCPTDR), which was defined by Brideau and Cavalier [20] and also described by Erkut and Zhang [45]. In the MCPTDR, the reward for each customer is a linearly decreasing function of the time at which the customer is serviced. The reward for customer i is given by:

$$\mathrm{R}(i, t_i) = R_i - b_i * t_i$$

where b_i is the rate of decline in the reward's value and t_i is the time at which customer i is serviced. This reward structure imposes implicit time windows on when the customer may be serviced, i.e., customer i will only be serviced while its reward is positive, so this provides the restriction that $t_i \leq \frac{R_i}{b_i}$. Examples given of where the MCPTDR may occur include rock bands touring in order to capitalize on the short-lived success following an album release and equipment repair, where the reward for servicing a customer is diminished over time as a result of the dissatisfaction of the customer due to being unable to use their equipment. A feature of this problem is that, since the rewards are decreasing, the servicing of the customer takes place as soon as it is possible. Therefore the decision making

process is to choose the order to visit the customers, with servicing occurring as soon as the vehicle arrives at the customer's location.

The MCPTDR is similar to a soft time window problem. The declining reward may be viewed as increasing penalties for deviation from the time window. The difference, though, is that the MCPTDR has been defined to be a subset selection problem, so it is possible to allow the penalties for customer inconvenience for non-selected customers to rise indefinitely, without affecting the objective. Therefore, only the customers that are currently the most profitable need to be considered, without concern for future costs. If all the customers must be serviced, as is standard for soft time window problems, the penalties for each customer become a more important concern.

The OPTW of Kantor and Rosenwein [91], may be considered to be a problem with time-dependent rewards. In this case, the reward at a customer i is given by:

$$\mathrm{R}(i, t_i) = \begin{cases} R_i & \text{if } e_i \leq t_i \leq l_i \\ 0 & \text{otherwise} \end{cases}$$

where R_i is the initial reward for customer i, t_i is the time at which customer i is serviced and e_i and l_i are respectively the earliest and latest times at which the service of customer i may commence. The rewards are therefore constant within the time window, and zero outside the window, and so servicing will be implicitly restricted to within the given time window.

Critique of Previous MCPTDR Solution Methods

We now study in more detail, the solution methods that have been proposed for the MCPTDR. This problem incorporates time-dependent rewards within a subset selection problem, and is therefore relevant to the combined problem we develop in Chapter 6, which is why these methods are of particular interest.

Both the heuristic solution methods considered in the papers for the MCPTDR are based upon the use of 2-step look-ahead to construct the initial solution. Thus instead of considering in isolation the effect of inserting a customer, the implications of one additional customer are considered. The viability of including a customer is determined by considering the customer and the best one to follow, and determining the 'attractiveness' of this pair.

In the method of Brideau and Cavalier [20], all feasible pairs of unrouted customers are considered for insertion at the end of the current route. The customer

that is added, is the first one from the pair whose inclusion will maximize the ratio of (total reward for the path)/(total time for the path). This seems odd, that the ratio involves the whole path and not just the additional effect of including the pair. This measure therefore includes the reward and time of the current path, which is not changed, and which has a large effect on the viability of the next moves. We suggest that an improvement in this method would be to evaluate each pair in terms of the ratio of (additional reward for the pair)/(additional time required), so only the incremental effect is considered. This approach looks at including the customers that add the most reward, and as such it is better suited for problems where few customers are included in the solution.

The heuristic method of Erkut and Zhang [45] considers also the effect of including pairs of customers at the end of the current path. For each unrouted customer, the minimum reward that will be lost by not including this customer next, is calculated. The customer with the maximum ratio of (minimum reward lost)/(additional time to include), is included next. This approach looks to minimize the total penalties in the problem, and is therefore suited to problem instances where most customers may be included.

Neither approach really utilizes full look-ahead, which has been found to be an important consideration in creating good solutions to the OP (see Sokkappa [164]). The only consideration made is to check what single point can be inserted after the candidate point has been included, whereas look-ahead involves greater consideration of the moves that may be made after the current move. These methods may easily be tripped up through including customers that are in positions that don't allow further insertions to be made. An example of this is shown in Figure 4.1. The rewards for each customer are given by:

$$\mathrm{R}(i) = \begin{cases} 35 - 2t_i & i = 1 \\ 10 - t_i & i = 2, \ldots, 8 \end{cases}$$

The distances are symmetric and given by

$$d_{ij} = \begin{cases} 1.5 & i = 0, j = 1, \ldots, 8 \\ 2.0 & i = 1, j = 2, \ldots, 8 \\ 0.2 & i, j = 2, \ldots, 8, i \neq j \end{cases}$$

If the time limit for the problem is 5.0, the optimal solution is to include all the customers from 2 to 8. This will take total time of 4.2 and will obtain total reward

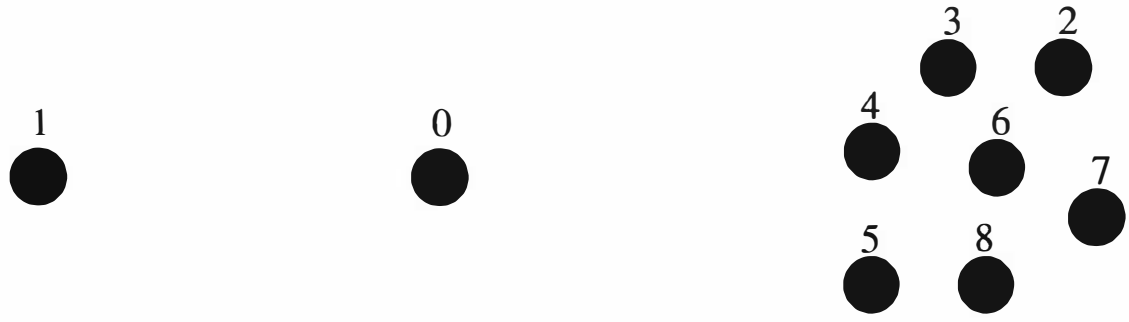

Figure 4.1: A Small Example of the MCPTDR

of 55.3. When using the heuristic of Brideau and Cavalier, the feasible pairs (i, j) have the following ratios:

$$R_{ij} = \begin{cases} \frac{32.0+6.5}{3.5} = 11.00 & i = 1, j = 2, \ldots, 8 \\ \frac{8.5+8.3}{1.7} = 9.88 & i, j = 2, \ldots, 8 \\ \frac{8.5+28.0}{3.5} = 10.43 & i = 2, \ldots, 8, j = 1 \end{cases}$$

Therefore the first insertion that will be made is to insert customer 1. From this solution, only one further insertion may be made, so the total reward obtained is 38.5 in total time 5.0. With the heuristic of Erkut and Zhang, the customers have the following ratios of minimum loss to additional time required:

$$\text{loss}_i = \begin{cases} \frac{2*2.0}{1.5} = 2.67 & i = 1 \\ \frac{1*0.2}{1.5} = 0.13 & i = 2, \ldots, 8 \end{cases}$$

Since the maximum loss by not including the customer next, is for customer 1, this will again be the first customer inserted. The total reward will again be 38.5 in total time 5.0. Both these heuristics insert customer 1 in the first position, although the optimal solution doesn't include this customer, and instead includes the cluster that includes customers 2 to 8. This is due to the fact that the heuristics only consider the next two possible insertions, whereas we suggest that using some form of look-ahead may be beneficial, particularly in instances where there are clusters and/or isolated customers.

Another feature of these solution methods is that they both use 2-opt to improve upon the solutions obtained by their construction method. This is carried out on the solution path where all customers are included, and where those customers whose reward will be zero are included at the end of the path. Neither method mentions how it assigns the customers with zero reward, but the use of Nearest Neighbour, i.e., assigning the unrouted customer that is closest to the end

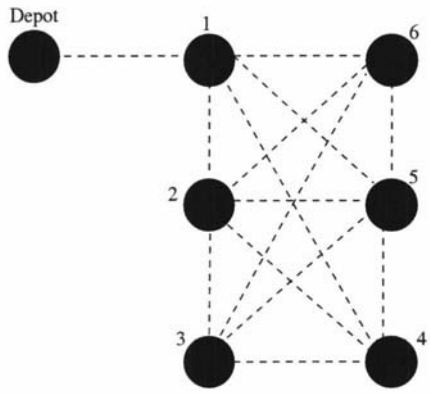

Figure 4.2: A Network for the MCPTDR

of the current path, seems to be consistent with the solution methods. Our major concern with this approach is that the 2-opt method has difficulties in accessing the customers with zero reward. 2-opt is able to re-order the solution route so as to increase the reward received, but it cannot pick out an individual customer to move earlier in the route to obtain positive reward. We illustrate this case through using the construction method of Brideau and Cavalier on the example network given in Figure 4.2. The distances between the customers are symmetric, and are given by:

$$d_{ij} = \begin{cases} 1.0 & i{=}0,j{=}1;\ i{=}1,j{=}2;\ i{=}1,j{=}6;\ i{=}3,j{=}4;\ i{=}5,j{=}6;\ i{=}2,j{=}5 \\ 0.8 & i{=}2,\ j{=}3;\ i{=}4,\ j{=}5 \\ 1.4 & i{=}1,j{=}5;\ i{=}2,j{=}6;\ i{=}3,j{=}5;\ i{=}2,j{=}4 \\ 1.8 & i{=}1,j{=}3;\ i{=}4,j{=}6 \\ 2.2 & i{=}3,j{=}6;\ i{=}1,j{=}4 \\ 1.0 + d_{1j} & i{=}0,j \neq 1 \end{cases}$$

The reward functions are:

$$R(i) = \begin{cases} 10 - t_i & i = 1 \\ 6 - t_i & i = 2 \\ 9 - 4t_i & i = 3 \\ 5 - 2t_i & i = 4 \\ 6 - 2t_i & i = 5 \\ 6 - 3t_i & i = 6 \end{cases}$$

In constructing an initial solution we initially insert customer 1, with a reward of 9.0 obtained. The next insertion is to include customer 2, and no further customers

can be included in a position where positive reward is obtained. We complete the initial solution through inserting the remaining customers via nearest neighbour insertion, which gives an initial route of 0-1-2-3-4-5-6, with total reward of 13.2. The optimal solution contains the sub-route 0-1-5-2, of customers with positive reward, which gives a total reward of 13.6.

Through using 2-opt, there is no way of including customer 5 in the position where it has positive reward. In order to include customer 5, we would need to reverse the path 2→5; this move doesn't allow positive reward for customer 2, and so this does not improve upon the current solution.

We suggest that an improvement technique that is able to move a customer or set of customers, i.e., Or-opt, may be useful for the MCPTDR, particularly when there is a large proportion of customers that have zero reward. In the above example, the optimal solution is obtained through using the 1-Or exchange to move customer 5 between customers 1 and 2.

4.3 Other Time Considerations

A further time consideration that has been used in the vehicle routing literature is the case where the travel times between various points differ depending on the time at which the travel occurs. An example of this type is the Time-Dependent Travelling Salesman Problem (TDTSP), where the objective is to minimize the total duration of a tour through a given set of points. Picard and Queyranne [136] described the TDTSP where the time of travel between two points i and j, depends upon the two points concerned as well as the position in the tour that this arc occurs. They use the TDTSP in the context of machine scheduling problems, where the time taken to change between the jobs depends on the jobs involved and the order of the tour so far. Vander Wiel and Sahinidis [181] developed lower bounding procedures for this form of TDTSP using Benders decomposition and they generate test problems with known optimal solutions. Malandraki and Daskin [120] described the TDTSP where the travel times depend on the absolute time at which the travel occurs, rather than the relative position in the tour. This case is relevant for road travel within cities, where the travel time varies according to the traffic congestion, which varies throughout the day.

Ahn and Shin [2] extended the VRPTW to cases where travel times vary during

the day according to the traffic congestion. They develop necessary conditions for determining the feasibility of a perturbation of the present route. These conditions rely upon a "non-passing" property of the congestion, whereby if two vehicles travel along the same arc, the vehicle that leaves first will always complete the arc before the second vehicle. This requirement is likely to be valid for most practical instances.

Time-dependent travel times have also been used in problems where the objective is to find the shortest path between two given nodes on a network. Hall [79] created such a shortest path problem through a network where the travel times along each arc are random, with the distribution of the travel times dependent on the time at which the travel takes place. This again models travel in traffic congestion, where in this case the effect of the congestion can be variable.

These time-dependent travel times could easily be extended to the OP, by varying the travel times over the course of the day. This would be valid for modelling the OP over urban settings. This would require greater consideration of the time at which the customer is serviced as, for example, the path may be required to have a given orientation, and it may no longer be able to be feasibly traversed in the reverse direction.

Another time consideration occurs in a problem known as the Delivery Man Problem (DMP) [52] and the Travelling Repairman Problem (TRP) [177, 186]. This problem involves a single service vehicle servicing a set of customers where the objective is to minimize the sum of the times at which the customers are serviced. The objective therefore takes into consideration the time at which each customer is serviced. This can be seen to be a special case of the TDTSP as devised by Picard and Queyranne [136], by writing the objective function of the DMP to be

$$\min \sum_{i=0}^{n} (n+1-i) d_{i,i+1}$$

where n is the number of customers to visit and $d_{i,i+1}$ is the distance between the ith and i+1th customers visited. Thus with the cost of moving from customer j to customer k in the ith position of the tour being given by $(n+1-i)d_{j,k}$, this is of the form of the TDTSP. Lucena [116] develops lower and upper bounds for the TDTSP and uses these to solve the DMP as a special case.

The DMP can in turn be used to incorporate other related problems. If we have an MCPTDR, in which all customers need to be serviced and the reward for each

customer is declining at the same linear rate, we have an example of the DMP. If we have a MCPTDR with all customers needing to be serviced and the reward for each customer declining at a different linear rate, then we have a version of the DMP where the objective is to minimize the weighted sum of the times of servicing.

4.4 Implications of the Time Constraints

We now consider the various time concerns and what effect they may have on our solution methods for subset selection routing problems. We are considering our solution methods to consist of construction techniques based on insertion, followed by intra-route and inter-route improvement techniques.

4.4.1 Implications of Time Windows

For problems involving time windows we are predominantly concerned with whether the current route remains temporally feasible when we alter the route in some manner. Some previous solution methods have allowed infeasible solutions to be generated within the construction phase of their heuristic search, e.g., van der Bruggen, Lenstra and Schuur [180] constructed solutions with penalties for violation of the time window constraints, and local search was applied in order to obtain feasible solutions. Infeasibility has also been incorporated within metaheuristic search routines, including Tabu Search (see [128]) and Genetic Algorithms (see [138] and [173]), to allow a larger neighbourhood of the current solution to be defined. With the inclusion of infeasibility, care must be taken to ensure that it is possible to return to feasibility at some later stage, and because there is no guarantee of being able to easily obtain feasible solutions, we preserve feasibility throughout our construction and search phases.

Our construction methods ensure that a feasible solution is maintained, by only inserting a new customer in a feasible position. Therefore, in order to check the feasibility of a given insertion, we ensure that the new customer is inserted within its time window, and the time of servicing each subsequent customer in the route, is within its time window. In practice, this may be carried out via storing the "headroom" of each customer, which is the maximum amount that the service time of the customer can be pushed back, without violating the time window feasibility of any subsequent customer. This is illustrated in Figure 4.3. There are seven

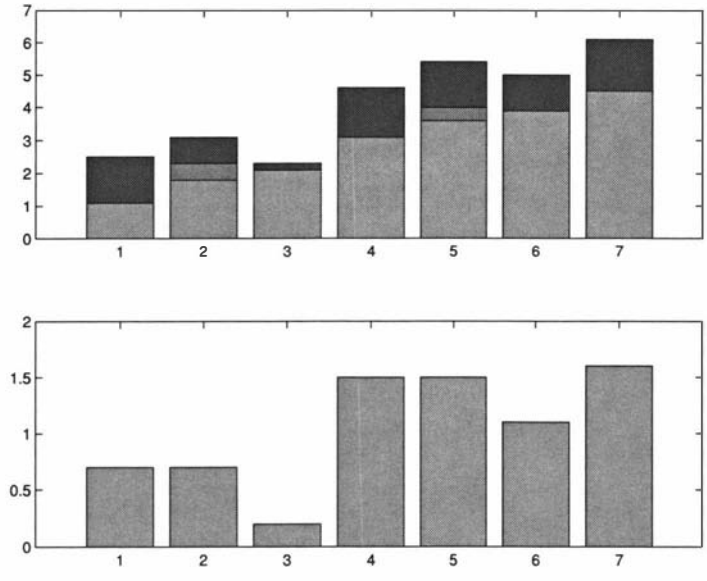

Figure 4.3: Headrooms for a Vehicle Route

customers in the vehicle route, with the top bar chart showing the arrival time at each customer, with waiting times before the commencement of the time window being included as a second bar for customers 2 and 5. The allowed delay at the customer until the end of the time window, is included on top of this bar and therefore the latest time at which the customer is allowed to be serviced is given by the total height of the bar. The second bar chart shows the headrooms for each customer, where these are calculated through using a backward iterative approach. Customer 7 can be delayed a time of 1.6 while still allowing the service to be carried out within the time limit. Customer 6 may be delayed by 1.1, so since this is less than the headroom at its successor, this is the headroom for this customer. For customer 5, the maximum delay in its service is 1.8, which consists of the waiting time of 0.4 plus the length of the time window. The maximum delay that allows the remaining customers to be serviced is however only 1.5, which is the waiting time at this customer plus the headroom of the following customer. The remaining headrooms are calculated in the same manner.

We may simply ascertain the feasibility of an insertion through making two checks. We firstly check that the new service time of the inserted customer is feasible, and then check that the time change of the successor in the route is less than its headroom.

For the improvement phase, Savelsbergh [154, 155, 156, 157] has developed efficient methods for carrying out local search techniques to reduce the time taken

for a route, in the presence of time window constraints. These methods are able to deal with both single and multiple time window cases, with feasibility checks carried out in constant time, and work by iteratively updating the maximum shift that is allowed.

For 2-exchanges, as shown in Figure 2.2, the lexicographic search is carried out, with three global variables used. These are: S^+, (the maximum change in the departure time at j–1 so no time windows along the path i+1$\longrightarrow$ j–1 are violated), W (the waiting time along the path i+1$\longrightarrow$ j–1) and T (the travel time along the path i+1$\longrightarrow$ j). For each iteration, S, (the change in departure time at j–1), is calculated. Our feasibility check is that $S \leq S^+$, i.e., the change at j–1 is less than the maximum allowable, and if this is violated we stop this current iteration, increment i and start again with $j = i$+2. Our check for profitability is that the new departure time at j+1 is less than the current departure time, where the new departure time is found as the departure time at $j + T + W + d_{i+1,j+1}$. Since we require that the new departure time at j+1 decreases, we are implicitly checking that the remaining customers from j+1 onwards are also feasible, as their departure time will be non-increasing. We can't guarantee that the improvements in the time at j+1 result in a decrease in the total time for the route, as the improvements may be absorbed through increased waiting times among the remaining customers.

The global variables can be updated in constant time, when we increment j: the travel time is simply increased by the length of the new arc in the path, the new waiting time is the maximum of zero, and the old waiting time minus the shift, i.e., $W = \max(W - S,0)$, and the new allowable shift is the minimum of the allowable change at j and the allowable shift less the current shift, i.e., $S^+ = \min$(the latest time at j - the new departure time at j, $S^+ - S$).

For Or-exchanges, as previously shown in Figure 2.3, different checks are required for shifting the points forwards or backwards in the route. The lexicographic search for a k-Or exchange, begins with the calculation of G (the gain in time of servicing the customer in position i+k, generated by removing the sub-path $i \longrightarrow i$+k–1). For each position j, into which to move the sub-path $i \longrightarrow i$+k–1, we calculate L (the loss in time for the customer currently in position j, when the sub-path is inserted in this position). This calculation may be carried out in $O(k)$ steps, through calculating the new arrival time for each of the customers on the moved subpath.

For shifting the sub-path forwards, we calculate the global variable S^+ (the allowable shift in the departure time of point j, so that the departure times along the sub-path $j \longrightarrow i$–1, will not violate the time windows). We also calculate the global variable W (the waiting time along the sub-path $j \longrightarrow i$–1). We can update our global variables in constant time, as the new S^+ is the minimum of the current S^+ and the allowable change at j plus the waiting time at j, with W incremented by this waiting time also. The check for feasibility and improvement involves checking whether L is less than the minimum of S^+ and G+W. If a move is infeasible, no further moves involving this sub-path will be feasible, so we increment i, and start again with $j = i$–1.

For shifting the sub-path backwards, we calculate the global variable S^- (the allowable shift in the departure time of point i+k, so that the departure times along the sub-path i+$k \longrightarrow j$–1 will not violate the time windows). We update our global variables in constant time as the new S^- is the minimum of the current S^- and the allowable forward change at j–1. The check for feasibility and improvement, involves checking whether L is less than the minimum of S^- and G+W. If a move is infeasible, no further moves with this sub-path will be feasible, so we increment i, and start again with $j = i$+k+1.

With inter-route Move and Exchange, we need to check that the new time of servicing the moved customer is feasible and that the change in time of servicing the succeeding point does not exceed its headroom. For the Cross exchange, we calculate the time of servicing each point on the moved sub-path and stop the current iteration if the servicing time of a point exceeds its time window. In practice, this calculation just requires finding the new time to service the last customer of the sub-path. If the time window of this customer is violated or if the time change at the succeeding point exceeds its headroom, then we stop this current iteration, and proceed through incrementing the appropriate search parameter.

4.4.2 Implications of Time-Dependent Rewards

Upon the inclusion of time-dependent rewards, the time at which each customer is serviced becomes a major concern. The evaluation of each proposed solution involves determining the time of servicing and calculating the reward that would be received for each customer.

Our construction phase is comprised of sequentially creating solution routes

through inserting single customers to partial routes, with some criteria used to determine which insertion to carry out at any stage. Our insertion is carried out as previously described and illustrated in Figure 2.1, where the new customer is inserted before the ith customer of the current route. The effect of such a move is to push back the time of servicing at each subsequent point in the route by $\Delta_{ins} = d_{i,new} + s_{new} + d_{new,i} - d_{i-1,i}$, where new is the inserted customer.

With constant rewards, intra-route improvement techniques can only be used to reduce the distance travelled, with techniques such as 2-opt and 3-opt having been used for this purpose for the OP(see [176], [148]). This distance reduction is used in order to create more available time in which to improve the current solution through the insertion of unassigned customers. With time-dependent rewards it is also possible to improve upon the reward received within the vehicle, through using intra-route improvement routines. For the TDVP, Keller [94] used a generalized Or-exchange procedure to improve the service vehicle's route. This routine removes successive sets of one through to $n/2$ points, where n is the number of points in the solution route. These points are repositioned in their best position within the route in either their current or their reverse orientation. The heuristics in the two papers on the MCPTDR [20, 45] involve constructing an initial solution and improving it using 2-exchanges. This procedure may also be used to remove some customers from the solution, by altering the customer's service time so their reward becomes zero (thus effectively excluding these customers). The evaluation of these intra-route improvement moves involves calculating the change in the rewards for each customer whose time of servicing is altered.

Our inter-route improvement techniques of node move and node and path exchange may be applied with these time-dependent rewards. The feasibility concern is to evaluate the distance change in the two affected routes to ensure neither exceeds the time limit. Again the main consideration is in the evaluation of the exchange.

Evaluation of Perturbations

We now consider how to evaluate the change to the total reward received in a vehicle during perturbation of the current route. We consider separately two cases for problems involving the time-dependent rewards. The first requires that a customer is serviced at the earliest allowable time after the vehicle arrives at its location; in

this case, the order of visiting the customers determines the time at which they are serviced. The second case allows the vehicle to wait at any stage of its route in order to delay the time of servicing subsequent customers; in this case, the time at which to commence servicing the customer is a decision variable.

When there are no idle times allowed in the solution, the time of servicing follows immediately from the specification of the route. The simplest case involves calculating the new times of servicing each customer, under the given perturbation, and calculating the new reward at each. The effect of the perturbation is the new total reward less the previous total reward for the route. It is possible to improve the computational efficiency by approximating this calculation. Taillard et al. [171] used linear interpolation to approximate the total cost change in their soft time window problem. For each customer in the solution, they selected three time shifts forward and three backward, and calculated the exact change to the total cost of the route if the time of servicing the customer is changed by each amount. For a given time shift at a customer, they approximate the change in total cost through linear interpolation between their given points. They took the best P of these, where P is a parameter of the solution procedure, and calculated the exact changes in order to identify the best move to make. They said that this approximation works well for the cost functions used in their problems, which are all piecewise linear. For more general functions, larger numbers of points would be required in order to create effective approximations and this reduces the benefit of the approximation technique.

When idle times are allowed, further considerations become necessary. In determining the optimal change for a given move, the times at all customers need to be considered. The current times of servicing prior customers within the route can't be assumed to be still optimal, as the idle times may need to be adjusted to improve the new route. Therefore it is necessary to calculate the times at all points of the new path. One method of calculating these, when the reward functions are concave, is to adapt the optimization algorithm of Dumas et al. [43].

This algorithm calculates the optimal time at which to service each customer in a fixed route, so as to minimize the total inconvenience costs. The algorithm starts by recursively calculating the time window $[a_i, b_i]$ of each customer i, such that the current route may be feasibly serviced, and calculating the optimal time, $\bar{x}_i$, to service this customer within their new time window. The algorithm reduces

the route to a number of sets of customers that need to be serviced consecutively, without waiting times. Initially all customers are linked to their successor in the route and, at each step, each linked pair, k and l, is considered. If $\bar{x}_k$ plus the travel time along the route to customer l exceeds $\bar{x}_l$, i.e., if k and l cannot both be serviced at their optimal time, then we link j with l, where j was previously linked to k, and remove the links j to k and k to l. We update b_l (which relates to b_k) and $\bar{x}_l$, which involves optimizing over the sum of functions along the sub-path $j+1 \rightarrow l$. Finally, we add customer k to a list of customers whose optimal time is directly dependent upon other times.

Once each customer has been processed, and there are no further pairs for which service cannot be carried out at their optimal times, we find the optimal times for all customers through scanning through the route in reverse order. If customer i is in the set of dependent customers, then its optimal time for servicing is equal to the time at its successor minus the direct travel time; otherwise, the optimal time for customer i is $\bar{x}_i$ as found earlier. The waiting time can then be determined to be the difference between the arrival time and the optimal time of servicing.

We illustrate this technique by means of an example. Suppose that there are four customers in a solution route, 0—1—2—3—4—0, where the distances between each pair of customers is equal 1.0. The time limit is 6.0 and each customer is available up until the time limit. The reward functions of the customers are given by:

$$\mathrm{R}(i) = \begin{cases} 10 - 3t_i & i = 1 \\ 5 + 3t_i & i = 2 \\ 9 - 2t_i & i = 3 \\ 6 + t_i & i = 4 \end{cases}$$

The objective is to maximize the reward received in a tour that returns to the depot (customer 0) by the end of the time limit. The initial time windows, so that the tour is feasible, are (0,1) for leaving the depot, (1,2), (2,3), (3,4) and (4,5) for the four customers in the tour respectively, and (5,6) for returning to the depot. Initially each customer is linked to its successor within the tour.

We begin by calculating the optimal time to service the customers. This is at the start of the time window for customers 1 and 3, i.e. $\bar{x}_1 = 1.0$ and $\bar{x}_3 = 3.0$, and at the end of the time windows for customers 2 and 4, i.e., $\bar{x}_2 = 3.0$ and $\bar{x}_4 = 5.0$. Considering the pair (1,2): $\bar{x}_1 + d_{1,2}$ $(= 1.0 + 1.0) \leq \bar{x}_2$ $(= 3.0)$, so these optimal times are compatible. For the pair (2,3): $\bar{x}_2 + d_{2,3}$ $(= 4.0) > \bar{x}_3$ $(= 3.0)$.

Therefore, customers 2 and 3 can not both be served at their optimal times, so we remove the links (1,2) and (2,3) and create the link (1,3). The new optimal time for customer 3 now considers servicing customer 2 directly before it, so $\bar{x}_3$ is now equal 4.0. Now we have $\bar{x}_3 + d_{3,4} \leq \bar{x}_4$ so we stop.

The final linkages are (1,3) and (3,4), with customer 2 serviced immediately prior to servicing customer 3. The times of servicing are therefore 5.0 for customer 4, 4.0 for customer 3 with customer 2 therefore serviced at 3.0, which results in a waiting time of 1.0 for customer 2, and customer 1 is serviced at 1.0. This has a total reward of 33.0, compared with 31.0 with no waiting times considered.

The main drawback of this technique is that it is only tractable for functions of a restricted form. The calculation of $\bar{x}_i$ requires the minimization over the sum of a number of functions. This sum is given by:

$$f_i^C(x_i) = \sum_{k=l_i}^{i} f_k(x_i - T_{k,i}), a_i \leq x_i \leq b_i^C$$

where the superscript C indicates that the present set of dependent customers is C, $T_{k,i}$ is the travel distance along the path from k to i and $f_i(x_i)$ is the function we are trying to minimize. We therefore require the minimum of the sum of these functions to be able to be calculated. Dumas et al. [43] show how this may be done with quadratic and linear functions, but more general functions will not be solved so easily. Another necessary requirement, which restricts the applicability of the algorithm, is that the cost function to be minimized be convex.

For our time-dependent reward problems, the objective is to maximize the total reward along the fixed route, and we now adapt this algorithm so as to minimize the deviation between the maximum and the actual reward received. This will identify the optimal time of servicing each customer within this fixed route.

4.5 Combining Time Constraints

We now consider the way that we can use the time considerations mentioned previously in this chapter to model different phenomena, and we provide a technique for dealing with these problems. The time concerns may be used to model the level of service that a customer is afforded. With hard time windows, the time of servicing is restricted to be within given time windows, where these are imposed by either

the server or the customer, and constitute a form of guarantee of the level of service given. Therefore the customer knows that service will take place within some given time boundaries, so this allows future plans to be made with some certainty.

With soft time windows, the time of servicing is promoted toward given times of highest desirability, with penalties for deviations from these times. These penalties model the customer dissatisfaction for service not being up to their most desired standard. The soft time windows are able to incorporate service guarantees, as there may be very large penalties for servicing outside of a given time window. Thus, if customer i has a desired time window of $[a_i, b_i]$ but there is also an acceptable window $[A_i, B_i]$ (for which $A_i \leq a_i$ and $B_i \geq b_i$), then the penalty for deviation from the customer's desired times may be given by:

$$\mathrm{P}(i, t_i) = \begin{cases} 0 & a_i \leq t_i \leq b_i \\ f_1(t_i - A_i) & \text{if } A_i \leq t_i < a_i \\ f_2(B_i - t_i) & \text{if } b_i < t_i \leq B_i \\ \infty & \text{otherwise} \end{cases}$$

where $f_1(\cdot)$ and $f_2(\cdot)$ are decreasing functions. This incorporates a hard time window as well as the soft time window, and so forces reasonable service while further promoting good service.

The concept of time-dependent rewards may be seen in a similar manner. The maximum reward values correspond to the most desired times of servicing, whereas a reward of zero indicates that service is unacceptable at this particular time. For the MCPTDR of Brideau and Cavalier [20] the rewards are all decreasing, which models the case where immediate service is desired but service is acceptable until the reward reaches zero.

By altering the form of the reward function we can vary the penalty for deviation from the desired time. For example, a polynomial function of order greater than 1.0 has increasing rates of penalties as the deviation increases, which emphasizes servicing before the deviations become too large. A polynomial of degree less than 1.0 will have higher rates of penalty for smaller deviations, which will emphasize the value of servicing on time.

With reward functions of general form, there may be other solution methods that are required. One technique that may be effective is to guide the overall search procedure towards more profitable regions. We propose now to create artificial time windows for each customer, where these windows ensure the customer is serviced

at a time when its reward is high.

We created some test problems for preliminary testing of the effectiveness of these artificial time windows. We generated 5 layouts, each involving 50 customers randomly generated on the Euclidean Plane $[0, 1]^2$. For each layout we generated 10 instances by varying the reward functions for each customer. The reward functions were all quadratic of the form:

$$\mathrm{R}(i, t_i) = \begin{cases} p_i(t_i - q_i)^2 + r_i & \text{if } e_i \leq t_i \leq l_i \\ 0 & \text{otherwise} \end{cases}$$

where $p_i < 0$. We randomly generated the q_i parameter in the range [0,LIMIT], with the time windows placed around this point having width proportional to the time limit. The r_i parameter was randomly generated in the range [10,100], with p_i generated so that the reward remained positive throughout the time window, i.e.,

$$-p_i = \min \left\{ \frac{r_i}{(q_i - e_i)^2}, \frac{r_i}{(l_i - q_i)^2} \right\}.$$

We varied the time limit at 5 different levels for each instance, where the limits were selected through experimentation so that approximately 20%, 40%, 60%, 80%, and 100% of the customers appeared in the solutions. We assume there is a single service vehicle available for servicing the customers, and this vehicle travels at unit speed.

Our solution method ARTIFINSERT is described in Algorithm 4.1. We constructed an initial solution, where at each step the customer that added the most reward to the current solution route was placed in its best position. If no customer can be added to the current route, we start a "new vehicle" and continue. In this manner, each customer is assigned to a "vehicle", although only the vehicle with the highest total reward actually provides service and is counted for the objective. The solution obtained was improved using intra-route Or-opt and 2-opt techniques and inter-route Move and Cross routines, with the effect of each move calculated exactly. Our improvement technique was steepest ascent where, for each routine, the move that increased the reward the most was carried out, until no further improvements could be made. Throughout the above stages we assumed that the time windows were hard and no idle time was allowed for the vehicle. As a final step we ran the optimization method of Dumas et al. [43] to include waiting times, where doing so may improve the total reward for the solution route.

Algorithm 4.1 function ARTIFINSERT

// Initialize the Time Windows
for ($i = 1$ to n) **do**
  $e_i \leftarrow$ the first time at which $k\%$ of the maximum reward is available
  $l_i \leftarrow$ the last time at which $k\%$ of the maximum reward is available
  $avail_i \leftarrow i$
end
// Initialize the first Vehicle Route
$route \leftarrow 1$, $tour_1 \leftarrow \{0\}$
$sizeroute_1 \leftarrow 0$, $numavail \leftarrow n$
while ($numavail > 0$) **do**
  $maxreward \leftarrow 0$
  for ($i = 1$ to $numavail$) **do**
    for ($j = 1$ to $sizeroute_{route} + 1$) **do**
      $\delta_{ij} \leftarrow$ extra reward from inserting $avail_i$ in position j of $route$
      if ($\delta_{ij} > maxreward$) **then**
        $best_i \leftarrow i$, $best_j \leftarrow j$, $maxreward \leftarrow \delta_{ij}$
      end
    end
  end
  if ($maxreward > 0$) **then**
    // Carry out the best improving move, if we have found one
    Insert customer $avail_{best_i}$ in position $best_j$ of $route$
    Update $current\_route$, e.g., $sizeroute_{route} = sizeroute_{route} + 1$
    $numavail \leftarrow numavail - 1$
    for ($j = best_i$ to $numavail$) **do**
      $avail_j \leftarrow avail_{j+1}$
    end
  else
    // Otherwise initialize the next Vehicle
    $route \leftarrow route + 1$, $route_{route} \leftarrow \{0\}$, $sizeroute_{route} \leftarrow 0$
  end
end
Apply intra- and inter-route Improvement routines to the solution
Apply the waiting time insertion algorithm of Dumas

end

	Approx. Proportion of Customers in Solution				
	0.2	0.4	0.6	0.8	1.0
Original Prior	555.98	1166.69	1664.77	2121.31	2606.02
Original Post	567.62	1175.63	1670.84	2126.71	2624.88
Artificial 0.5 Prior	579.22	1167.34	1666.27	2111.26	2611.28
Artificial 0.5 Post	603.42	1179.02	1673.45	2118.88	2628.65
Artif. 0.5 + Orig. Prior	607.62	1188.44	1683.34	2134.43	2630.72
Artif. 0.5 + Orig. Post	609.71	1190.84	1685.95	2136.59	2633.41
Artificial 0.7 Prior	590.11	1159.77	1671.41	2108.69	2616.96
Artificial 0.7 Post	611.68	1177.63	1681.85	2118.70	2635.26
Artif. 0.7 + Orig. Prior	621.99	1193.57	1697.88	2140.27	2638.06
Artif. 0.7 + Orig. Post	626.16	1197.76	1700.31	2142.97	2642.04
Artificial 0.9 Prior	568.50	1112.59	1581.27	2000.86	2615.13
Artificial 0.9 Post	583.98	1129.14	1593.64	2013.99	2631.18
Artif. 0.9 + Orig. Prior	601.58	1179.22	1677.50	2106.30	2638.92
Artif. 0.9 + Orig. Post	610.52	1185.36	1683.21	2112.02	2643.76

Table 4.1: Mean Reward Over 50 Instances of Time-Dependent Rewards for Each Method

We attempted to model the general time-dependent reward problem using artificial time windows. In this case we identified the maximum value of the reward function for each customer, and identified the time on each side of the maximum where the reward received was equal to $100\alpha\%$ of the maximum where $0 < \alpha < 1$. Our solutions are shown in Table 4.1. The rows correspond to the different time windows imposed on the solution, with prior and post relating to the solution being found before and after the application of the algorithm of Dumas et al. [43]. The problems were solved using the original time windows and with the artificial time windows incorporating respectively 0.5, 0.7 and 0.9 of the maximum reward. With the artificial time windows, the solution obtained was then further improved by reintroducing the original time windows, adjusting the current solution so it is feasible, and applying the improvement techniques again.

Table 4.1 indicates the possible improvements that may be made through including the artificial time windows. The most effective setting of the artificial time windows appears to be at a position incorporating 70% of the maximum reward although further testing could obtain better time windows to use to use. We see that in general the artificial time windows are able to create better solutions than the original time windows. The solution with artificial time windows at 70% of

the maximum obtains higher reward through the initialization and improvement phases than with the original time windows, for each of the time limits except with approximately 80% of the customers in the solution. This shows the effectiveness of the artificial time windows in themselves, in forcing the search procedure to only consider customers while their where reward is high, i.e., near their desired service time. Further improvements are obtained through reintroducing the original time windows, while starting from the solution obtained with the artificial time windows. The original time windows are required in order to fine-tune the solution by including further points that improve the solution, even though the reward received is well below the maximum available reward. Therefore the artificial time windows seem to be very useful in guiding the search to promising areas, that can be taken advantage of through carrying out local search from the solutions obtained. A possible application for artificial time windows could thus be in a meta-heuristic setting, as a means of guiding the search procedure.

We now look at more detail at the differences between our best artificial time windows and the original windows. Tables 4.2, 4.3 and 4.4 show respectively the average total waiting time, average number waiting and the average number in the solution for each set of problems. With the re-introduction of the original time windows to the problems with artificial time windows, the reward received increases due to the reduction in total waiting time and the inclusion of more customers in the solution. The case where the original time windows obtained a better solution than with the artificial ones is with approximately 80% of the customers in the solution. We see in this case that the total number of customers serviced is much lower and the waiting times much higher with the artificial windows. With this time limit there is the greatest flexibility in the selection of customers to include, and it appears that the ability to include more customers by allowing ones with low reward, as is the case with the original time windows, is important in obtaining good solutions. The re-introduction of the original time windows improves the solution over using the original time windows alone, which again shows that the artificial time windows may be beneficial.

	Approx. Prop. of Custs in Soln				
	0.2	0.4	0.6	0.8	1.0
Original Prior	0.0960	0.1060	0.1099	0.2994	3.9417
Original Post	0.1232	0.1511	0.1447	0.3564	4.1844
Artificial 0.7 Prior	0.2235	0.3434	0.3520	0.5657	4.6007
Artificial 0.7 Post	0.2580	0.3951	0.4143	0.6494	4.8299
Artif. 0.7 + Orig. Prior	0.1143	0.1823	0.1699	0.2172	3.7758
Artif. 0.7 + Orig. Post	0.1528	0.2255	0.2195	0.2788	4.0238

Table 4.2: Mean Waiting Times Over 50 Instances

	Approx. Prop. of Custs in Soln				
	0.2	0.4	0.6	0.8	1.0
Original Prior	1.16	0.98	0.92	0.86	3.04
Original Post	2.16	2.00	2.02	2.56	6.34
Artificial 0.7 Prior	2.30	2.44	1.74	1.98	3.96
Artificial 0.7 Post	2.94	3.44	3.22	3.32	7.44
Artif. 0.7 + Orig. Prior	1.16	1.14	0.60	0.72	2.82
Artif. 0.7 + Orig. Post	2.14	2.36	2.12	2.12	6.44

Table 4.3: Mean Number Waiting Over 50 Instances

	Approx. Prop. of Custs in Soln				
	0.2	0.4	0.6	0.8	1.0
Original	10.22	20.76	29.86	39.28	49.40
Artificial 0.7	10.06	19.96	29.04	37.60	49.20
Artif. 0.7 + Orig.	10.64	20.82	30.10	39.10	49.52

Table 4.4: Mean Number in the Solution Over 50 Instances

Conclusions

We have shown that there may be merit in using artificial time windows to promote the service of customers to intervals where their reward is highest. In a practical sense, this may involve service guarantees, whereby the customers that are serviced will be serviced to an acceptable level. These can therefore be used as tools to explicitly consider the level of service that is afforded to the customers. This may be beneficial on a purely financial level, if the gains through increased customer goodwill were to outweigh the additional reward received when greater numbers of customers are serviced, but to a lower standard. We have found that forcing the selected customers to be serviced within a well-selected artificial time window, gives promising results for problems with quadratic reward functions. The rewards received were superior to those found by a similar heuristic, that did not include the artificial time windows, for the majority of the random problems tested.

The imposition of these time windows is not sufficient for the reward maximization criteria, and further improvements may be made through relaxing the artificial time constraints and continuing to search. The improvements come about as a result of increased vehicle utilization, as greater numbers of customers are serviced, since a greater amount of time is spent travelling between customers, rather than waiting for the customer reward to reach a required level before the service can be commenced. In this case, the time windows are used as a guide, to promote the service to more profitable areas.

CHAPTER 5

General Tasks

5.1 Background

There are many examples of VRSPs where the vehicle needs to visit a single location in order to service a customer; the Travelling Salesman Problem (TSP) and most instances of the Vehicle Routing Problem (VRP) are classic examples. In this chapter we are interested in problems where servicing customers does not necessarily involve visiting just single locations.

We define *task routing problems* to be problems where a set of locations need to be visited in order to service a customer. An example of a task routing problem is the Dial-a-Ride Problem (DARP). In the DARP there is a fleet of vehicles and a set of customers, where each customer is serviced by picking up a load from a given origin and delivering it to a given destination. The objective is to minimize the total distance travelled in servicing the set of customers. The general context for the DARP is in transporting passengers from one location to another, so this is most relevant for carriers such as taxi companies.

Stein [168] introduced the DARP and gave asymptotic results for the expected length of the optimal DARP tour through $2n$ randomly generated points in a given area. The n customers were created through randomly assigning each point to be either an origin or destination of one of the customers. Asymptotic results were obtained, which extend those found by Beardwood, Halton and Hammersley [11] for the TSP. Therefore, with a single vehicle, the asymptotic length of a DARP

tour through n customers (L_n) is given by

$$\lim_{n\to\infty} \frac{L_n}{\sqrt{n}} = c\sqrt{a}$$

where $c = \frac{4}{3}\sqrt{2b}$, and b is the constant found by Beardwood et al. [11] for the TSP.

Psaraftis [141] used dynamic programming to solve the DARP in which customers arrive for servicing and request service immediately. He used both a static and a dynamic approach for obtaining a solution. With the static approach, the vehicle carries out the optimal route over the current set of available customers and, upon finishing this route, the optimal route over the customers that have arrived in the meantime is calculated. The dynamic approach involves reoptimizing the entire route each time a new customer arrives. The objective function given for this problem is to minimize the weighted sum of the total distance travelled by the service vehicle and the customer inconvenience costs (which are linear functions of the time between requesting service and being serviced, and the excess time the customer is in the vehicle).

Psaraftis [142] created a heuristic to obtain a solution to the immediate request Euclidean DARP for a fixed set of customers, where the objective is to minimize the total distance travelled. The heuristic begins by creating a minimum spanning tree (MST) for the points that comprise the set of customer locations. Each point is considered in turn to be the starting point of the tour, and the solution is obtained through traversing the MST in a certain direction, omitting points that can't be included in the solution, i.e., origins whose destinations haven't yet to be visited and vice versa. The procedure continues until all points have been included in the solution and then local search, consisting of interchanging the positions of two adjacent locations within the tour, is applied in order to obtain improved solutions. The heuristic was compared with the optimal solution for problems with up to eight customers, and heuristic results are obtained with up to 50 customers.

Jaw, Odoni, Psaraftis and Wilson [89] developed a heuristic for the DARP where each customer requests service in advance and they give desired times for servicing. The objective is to minimize the cost, which is a weighted sum of the travel costs and the customer inconveniences from excess riding time and deviation from desired times. The heuristic involves sequentially inserting customers into the solution route in the position that adds the least to the overall cost. They created solutions to simulated problems involving 250 customers arriving over a 9

hour period, with the number of vehicles used being a decision variable. They also adapted a real life problem from a German city, by assuming all customers were advanced request. A solution for this problem was then obtained for a problem involving 2617 customers over a 16 hour time period, where there were 28 vehicles available, with heterogeneous vehicle capacities. This solution was favourable when compared to the solution obtained by the actual method that is employed, although direct comparison is not valid as the actual method deals with immediate request customers, while the developed method treats these as all being advanced requests.

Kubo and Kasugai [101] developed simple heuristics for the DARP where the objective is to minimize the total distance travelled. They used various insertion methods, nearest neighbour, the MST heuristic of Psaraftis [142] and space-filling curves (see [10]) to construct the initial solutions, and apply 2-, 3- and Or-opt to improve the solutions. The best construction method was found to be cheapest insertion, where the pair of points comprising the customer are inserted in the positions that minimize the total change in distance.

Healy and Moll [82] created a new local search method for the DARP where the objective is to minimize the total distance travelled. They developed a meta-heuristic approach where the criterion for selecting the best move to make is given by the ratio $C(x)/|N(x)|$, where $C(x)$ is the distance for the new solution, x, and $|N(x)|$ is the number of solutions in the neighbourhood of x. Their approach was called 'sacrificing' as it sacrifices the solution quality of a move in order to carry out a move that also increases the possibilities for further searching. This approach was used in conjunction with 2-opt and was found to improve over simply applying 2-opt, although it wasn't as effective as 3-opt. This therefore gives a compromise between the solution quality and computational requirements of the 2-opt and 3-opt methods.

Madsen, Ravn and Rygaard [119] provided a heuristic that sequentially inserts customers for a DARP with time restrictions, dynamic customer arrivals and heterogeneous vehicles. The customers were dealt with in non-increasing order of their difficulty of scheduling and are inserted into the current solution in their best position. The objective may be varied to suit the requirements of the given situation. New customers may be placed in the solution in less than one second, enabling it to be used for dynamic situations.

Toth and Vigo [175] developed local search heuristics for the DARP with time

constraints, where the objective is to minimize the costs incurred through the distance travelled and time considerations. The improvement techniques involve moving the position of a customer within a vehicle route, and moving and exchanging customers between different routes. They obtained a solution for an actual data set for transportation in Bologna, where there are approximately 300 customers requiring service each day. There were a number of different service vehicles and a matching between the type of vehicle and the service required by the customer was necessary.

Some recent research has been carried out at ZIB in Berlin, regarding on-line versions of the DARP (see [77], [81], [80] and [100]). The application with which they were concerned is the elevator system of a manufacturing plant. The tasks consist of the transportation of products from one floor to another, where there may be some precedence constraints on the actual order in which the tasks themselves may be carried out, i.e., the task corresponding to the first product available at the elevator needs to be completed before any subsequent products may be moved. A number of theoretical results for these problems were provided.

The DARP is also known as the Stacker Crane Problem, which was first proposed by Frederickson, Hecht and Kim [56]. This application relates to the process of a crane moving loads from one position to another, with an objective of minimizing the total crane movement. This problem allows only one task to be undertaken at a time, as just a single load can be transported by the crane.

A similar problem with loads of general sizes is the Pick-up and Delivery Problem (PDP). Dumas, Desrosiers and Soumis [42] formulated a PDP with time constraints, where the objective is to minimize the total travel costs and the cost incurred depends upon the load carried along each arc. They solved problems with up to 55 customers using column generation with branch and bound used to create feasible integer solutions.

The above problems involve tasks consisting of a pick-up and a delivery. Cases involving other definitions of tasks occur within the context of the Precedence Constrained Travelling Salesman Problem (PCTSP). The PCTSP is the same as the TSP, except there are given orders in which the locations must be visited, i.e., certain of the locations must be visited before the visit to certain other locations is allowed. Bianco, Mingozzi, Riccardelli and Spadoni [17] solved the PCTSP with

general precedence constraints. Here, any location that has no immediate successor can be assumed to be the destination of a task, and the successive precedences leading to this destination can be construed as the components of the task. The asymmetric TSP with precedence constraints is also known as the Sequential Ordering Problem (see [46], [4]), and tasks for this problem may be defined in the same manner.

Savelsbergh and Sol [158] developed a general model for pick-up and delivery problems. Their model allows for general definitions of tasks, as each customer i consists of a set of pick-up locations, N_i^+, which need to be visited before visiting a set of delivery locations, N_i^-. Different side constraints and objectives are also allowed within this model. They classified a number of problems, previously mentioned in the literature, according to their general model and describe the solution methods that have been applied to these problems. These methods involve either optimization or construction and iterative improvements. For each of the problems described, all customers require a single pick-up and a delivery.

Savelsbergh and Sol [159] described a practical problem where each task, i.e., the service of each customer, consists of a pick-up and a number of deliveries. The deliveries need to be serviced in a prescribed order, so there is a fixed set of precedences defined for each customer. These problems were solved using a branch and price algorithm, based on column generation.

5.1.1 Definition of a Task

We formally define a *task* to be the set of routing requirements which are needed to be fulfilled in order to service a customer. We term the individual components of the task to be *sub-tasks*, where the sub-task may consist of a pick-up, a delivery or a location to be visited with no load transferral. Any sub-task that must appear after another sub-task is called its *successor*, while any sub-task that must appear before another sub-task is called its *predecessor*. The definition is very general and allows for a number of possibilities. Unlike the DARP and PDP, where the task is strictly defined to consist of a single origin and single destination, here our task can take a number of forms, as will be outlined in detail in Section 5.2.

Single locations may be considered to be tasks via this definition. Here the task relating to a delivery will consist of the depot and the location to be delivered to, while the task for a pick-up will be comprised of the pick-up location and the

depot. If, as is the case with the TSP, no load is required and the location just needs to be visited, then the task will consist of a single point.

5.1.2 Definition of the Distance between Tasks

Finding a sufficient measure for the distance between tasks has been described as being an open problem (see [158]). We now consider a possible definition and the implications of this definition. We define the distance, d_{ij}, between customers i and j to be the additional distance required to service customer j over and above the distance required to service customer i. By this definition, the distance between tasks is asymmetric. This definition requires us to find, or estimate, the shortest distance to service the two customers together.

In defining the distance, we are interested in being able to find customers which are 'close' to each other, so we may put close customers together within our solution routes. The main difficulty with this is that with the load requirements of a task, the distance between customers is very dependent upon the route that already exists. Therefore, although two tasks may have been found to be close together, the routing distance required may be large if the current route is incompatible with inserting the task in a reasonable manner. This is shown in Figure 5.1, where the nodes are given as (*task name, load change*) and the current solution route is shown. The capacity of the service vehicle is 5. The distance between A and C is small, but when attempting to insert task C into the current route, the additional distance is relatively large. This is due to the task being unable to be included in its best position, due to the capacity constraint. Thus although the customers are close with respect to our definition of distance, the insertion distance required to add the customer to the route is large.

This measure of distance is therefore only of use when the customers are not part of an existing route. When a route does exist, our only concern is with the insertion distance required to include the unscheduled customer in the route. The relevant distance in this case is route-dependent, and we believe that there is no reasonable measure of distance that is independent of the routing, particularly when the vehicle capacity constraint is binding.

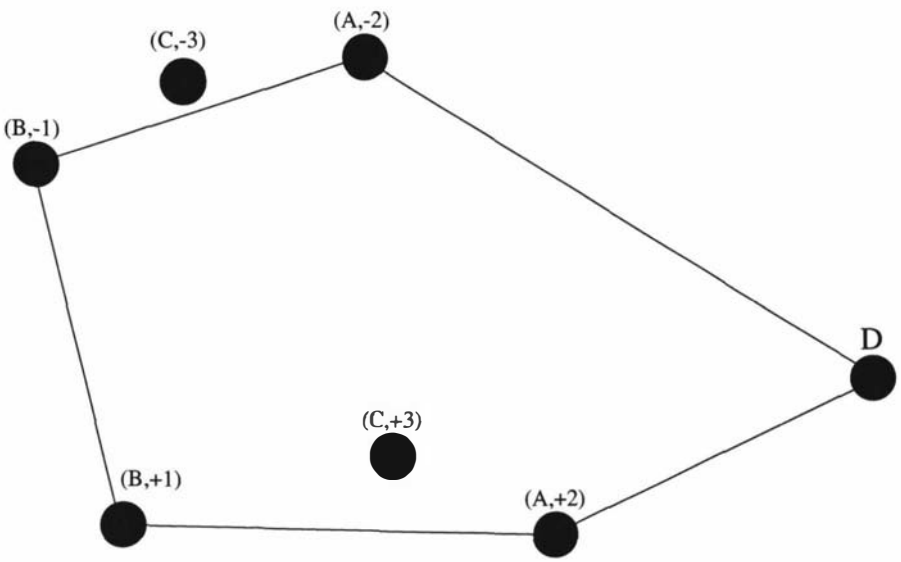

Figure 5.1: An Example of Incompatible Tasks

5.2 General Forms of Tasks

We now consider alternative possibilities to the type of task that have commonly been studied in the literature. These allow for greater flexibility, and thus expand the scope of what we can refer to as task problems.

5.2.1 Series of pick-ups and deliveries

A natural extension of the tasks being a pick-up and a delivery, as in the DARP and the PDP, is to allow the task to consist of a given sequence of pick-ups and deliveries. In order to service the customer's requirements, a number of locations need to be visited, with different sub-tasks carried out at each location. The precedences within the task are strict, as each of the routing requirements must be completed before the next one can be performed. As such, this type of task is not fundamentally different from our conventional task, as there are no decisions to be made as to how the customer is routed, just a decision of where to position the customer. An example task is shown in Figure 5.2, where the task i consists of $k+1$ locations to be visited.

Each location has associated with it a change in the load, according to the requirements for the task. For example we could have each location $i, i+1, \ldots i+k-1$ involving successive loads being picked up, with the final delivery occurring

Figure 5.2: An Example of the Basic Task

at location $i + k$. This form of task is relevant in cases involving deliveries of loads that need to be successively accumulated from a number of sources, or loads that, once picked up, are delivered to a number of destinations. Practical examples of where this type of task might arise occur in courier companies. Here a customer may require a succession of visits to be made in a prescribed order, e.g., visit a balloon shop, a chocolate shop and a florist in order to build up a present to be delivered. While we assume that the task has fixed precedences, it isn't necessary for the sub-tasks of a particular task to be performed sequentially, and so other tasks may be carried out while the loads for a task are in the service vehicle.

5.2.2 Routing Decisions within the Task

A relaxation of the above definition of a task that allows for greater flexibility, is to not completely fix the order in which the sub-tasks of a task are to be carried out. The sub-tasks that comprise the task are fixed, although there is some flexibility as to the order in which these can be performed. An example of this is where the task consists of a number of pick-ups and a single delivery. If, in the case of the courier company described above, the customer is not concerned with the order in which the components of the present are picked up, then we have a task of this form. As there is no prescribed order for the pick-ups, then it is possible to visit the pick-up locations in the order that best suits the schedule. The only constraint within the task is that the vehicle can't visit the delivery location until after all the pick-ups have been made.

With tasks of this type there is a routing component within the task to consider, when attempting to carry out the task. The selection of the order of carrying out the routing requirements adds a lot of extra flexibility within the routing process. These may be explicitly restricted due to the precedence constraints, or implicitly constrained through feasibility considerations. For example, the load of the task that is in the vehicle must always be positive, so no ordering that violates this constraint can be considered.

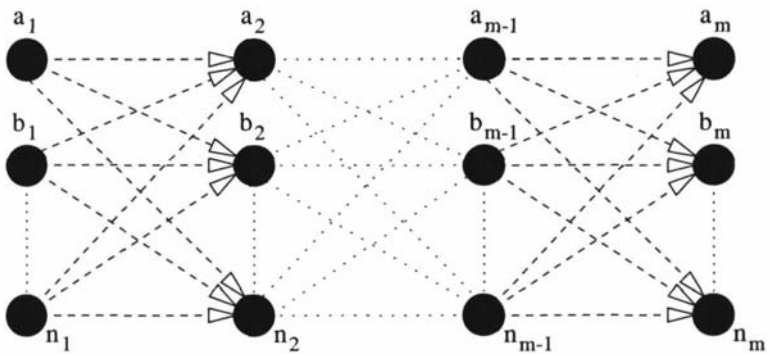

Figure 5.3: An Example of a Task with Selection Components

5.2.3 Subset Selection

A further relaxation of the definition of a task is where the sub-tasks within the task aren't required to be fixed. An example task of this type is shown in Figure 5.3. Here there are m requirements for the task and these are labelled $1, 2, \ldots m$. For each requirement i, there are n_i possible locations where these may be carried out, and these are labelled $a_i, b_i, \ldots, n_i$. The task is completed by visiting one location out of each of the m sets in order to carry out the sub-tasks. We assume that the sub-tasks are homogeneous within the sets, i.e., for the i'th sub-task each of the locations $a_i, b_i, \ldots, n_i$ is equivalent, although there may be different costs associated with the different locations. This type of task may occur again in the context of courier companies, where the mth location is the destination of a task and the $m-1$ previous routing requirements involve building up the overall load to be delivered. Consider in this case that the jth requirement is to pick up, say flowers, instead of having to pick up from a specific florist there is an option of n_j florists from which to choose. The selection element allows even more variety in the solutions, as the decision is to be made as to which is the best combination of eligible locations to include in the vehicle's schedule. This problem is further complicated by allowing the order of carrying out the subtasks to vary.

5.2.4 Split Tasks

One extension that has been found to be useful for the VRP is to assume that the load to be delivered to a customer is divisible and it can be split between different vehicles and/or depots. Dror and Trudeau [38, 39] showed that considerable improvements in distance travelled and number of vehicles used, may be made by

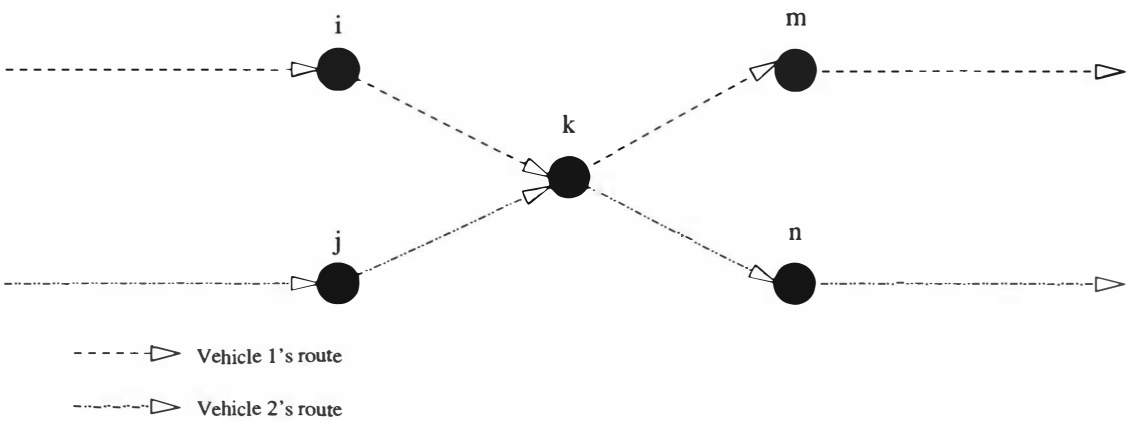

Figure 5.4: An Example of a Split Task

allowing these split deliveries. They created initial solutions to the VRP, and then made improvements by splitting some of the loads. Frizzell [57] developed a construction heuristic specifically for split deliveries and used standard improvement techniques of moving and exchanging customers between routes. A possible application he found is to solve VRPs where a customer's demand exceeds the capacity of each of the service vehicles (a case that is unable to be dealt with in the standard VRP). Mullaresil, Dror and Leung [127] modelled a system of distributing feed to cattle as a split-delivery problem. The customers are the pens of animals and these were assumed to be serviced by travelling along the road segments in front of the pens. Each road segment and even each pen may be serviced by more than one vehicle so the problem allows for split deliveries. The solution method used was similar to that of Dror and Trudeau [39].

We define a task to be *splittable* if the routing requirements of the task may be carried out by more than one vehicle. An example of a task whose service has been split is shown in Figure 5.4, where the task involves picking up loads from locations i and j and delivering this combined load to location k. In this instance we are splitting the task between vehicles 1 and 2.

Task splitting may be valid in a number of cases where the load is allowed to be carried in different vehicles. If the task consists of a number of pick-ups and a delivery, the successive pick-ups could be carried out by any of the vehicles, with all of these vehicles needing to visit the delivery location in order to complete the task. It is also possible for a co-ordination exercise to be required in order to complete a task, where the different vehicles being used to carry out the different components of the task need to meet to allow the task's completion. To illustrate this, we again

use the example in Figure 5.4, but now we assume that there is a task consisting of pick-ups from locations i, j and k and a delivery to location m. In this instance we assume that vehicle 2 makes the pick-up from location j and drops the load off at location k. Vehicle 1 picks the load up from location i, travels to location k where it picks up the load that was originally there, together with the load left by vehicle 2, and delivers this composite load to location m. One important temporal consideration involved with this example is to ensure that vehicle 1 does not leave location k before vehicle 2 has delivered its load there.

5.2.5 Preemptive Tasks

A further extension of tasks is to allow the vehicle to drop off the customer's load at an intermediate point, instead of having to carry it in the vehicle until the delivery is made. This approach has been considered by Guan [78] and Frederickson and Guan [55], who gave complexity results for routing on a restricted network, i.e., it is possible to find the shortest path in linear time when the network consists of a path and all customers are preemptive. The preemptive task concept may be extended further in the case of multiple vehicles, to allow for redistribution of loads en route, where the load may be dropped off by one vehicle and delivered by a separate vehicle. Some applications involving this multiple case, were discussed by Rivers [150].

An example of where preemptive tasks may enable improved solutions to be found, is given in Figure 5.5. The situation on the left shows the solution that is obtained for the capacitated DARP, where the vehicle capacity is equal 10. When we allow preemptive tasks, we obtain the solution on the right. Here the pick-up for task B is taken from its original location, to the location of the pick-up of task C. After completing tasks A and C, the service vehicle returns to pick up the load for task B and delivers it to its required location. The benefit from preemptive tasks in this case is due to the fact that the vehicle is operating near to its capacity, and there are distant clusters of points to which we would like to limit the number of visits. The other major benefit for this approach would be in the case of networks, where vertices may need to be passed through a number of times in the course of a service route. The additional routing flexibility obtained through leaving a load at a vertex that will be visited later (thereby increasing the effective capacity of the vehicle) may be very beneficial.

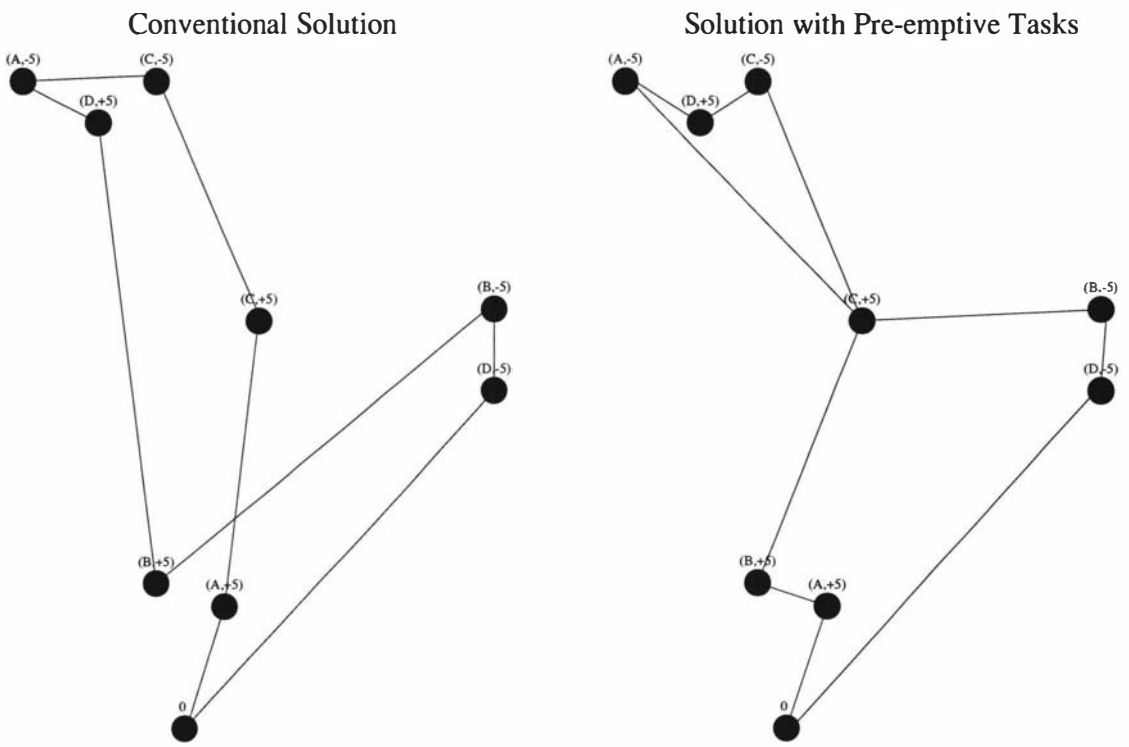

Figure 5.5: An Example of Preemptive Tasks

5.3 Consequences of Type of Task

We now look at the implications arising from the different forms of task that may be included within our problem definition. The additional considerations include ensuring the feasibility of the routes with regard to vehicle capacity and precedences within the tasks, and ensuring that complete tasks are carried out, rather than having partial service.

Traditional heuristic methods for solving vehicle routing problems involve construction methods for creating initial feasible solutions, followed by local search improvement techniques. In local search, small perturbations of the current solution are made in order to create an improved solution. In this section we consider the construction methods, and we consider separately the cases of intra-route and inter-route improvement techniques. We are initially concerned with tasks comprising sub-tasks with fixed precedences, and extend these to the more general tasks as defined previously.

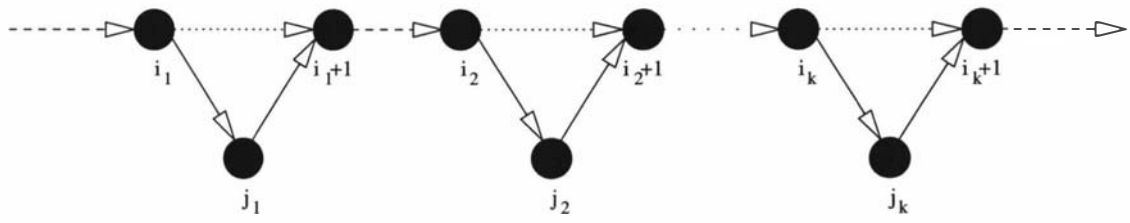

Figure 5.6: An Example of a Task Insertion

5.3.1 Construction Methods

We are considering the feasibility concerns of methods for creating solutions to our problem, where these methods consist of the use of insertion to sequentially build up vehicle routes. Thus, we start with an empty vehicle and add tasks in their best feasible position in a vehicle's route. An illustration of how this insertion operates, is given in Figure 5.6, where customer j, consisting of k sub-tasks, is inserted in its best position in the vehicle route. To ensure feasibility, we need to check that the capacity of the vehicle is not violated at the points visited between the first and last added sub-tasks for the inserted task. We also check that the new vehicle's total route duration does not exceed the maximum allowable duration.

5.3.2 Intra-Route Improvement Techniques

We now consider the improvement techniques that are used to improve a single vehicle route. Some standard techniques used for these purposes are k-exchanges and Or-exchanges (see [129]). In this section we will restrict our attention to just 2-exchanges, which were previously shown in Figure 2.2, and Or-exchanges, as previously shown in Figure 2.3. We consider a general k-Or exchange, which is the movement of a sequence of k nodes to its best position in the tour. This is a restricted version of a 3-exchange, where the arcs are repositioned so that no sub-path needs to be reversed.

For these local search techniques, we see that particular considerations are required in order to account for the precedence constraints that are present. In a 2-exchange, the path $i{+}1 \longrightarrow j$ is reversed, and so the precedence constraints are violated if there is a sub-task within the reversed path whose successor is also on that path. With the Or-exchanges, if $j > i$, the sub-path $i \longrightarrow i{+}k{-}1$ is placed after the sub-path $i{+}k \longrightarrow j$, and so the precedence constraints are violated if a sub-task in the moved sub-path has a successor in the sub-path that it is moved

after. If $j < i$, the sub-path $i \longrightarrow i+k-1$ is placed before the sub-path $j \longrightarrow i-1$, and so the precedence constraints are violated if a sub-task in the moved sub-path has a predecessor in the sub-path that it is moved before. Another consideration with task routing, is that we need to check whether the capacity constraints on the vehicle are still valid for the proposed new route. Simple checks of feasibility would require searching for each move, for each moved sub-task, whether the precedence and capacity constraints are still met in the new position, with this search process requiring $O(n)$ calculations.

Savelsbergh [155, 156] developed a search procedure that exploits a certain structure of the search process to reduce the computational effort required in each iteration to a constant number of calculations. This approach is similar to that used previously described for handling time windows, and can be used when there are time constraints as well as precedences. The process (valid for both 2-exchanges and Or-exchanges) increments the edges in the exchange and updates global variables in each iteration. The only feasibility check required, is to check the feasibility of the one point that is changed in moving into the current iteration. To demonstrate this how this works for a 2-exchange we will again refer to Figure 2.2.

Each iteration consists of fixing location i as the initial point of the exchange. We start with $j = i+2$ and successively increment j. For each new value of j, if the sub-task in position j has a successor, we "mark" the position in the tour of this successor. The exchange is feasible with respect to the precedence constraints if the new j, i.e., the head of the reversed path, is not "marked", i.e., if the sub-task in position j does not have its predecessor in the reversed path. With Or-exchanges we need to deal with the cases where the sub-path is moved forward separately from when it is moved backward in the route. When we move the sub-path $i \longrightarrow i+k-1$ forward, we mark the positions of the predecessors of each of the sub-tasks in the moved path. We start with $j = i - 1$ and continue to decrement j as long as the point j is not "marked". With moving the sub-path backward, we mark the successors of the moved sub-tasks, and we set $j = i+k+1$, and increment j as long as point j is not marked.

The capacity constraints are dealt with by recording the maximum change in the capacity over the moved sub-path, as well as the minimum allowable change over the sub-path between the current location and the proposed location of this moved sub-path. A move is feasible if the maximum capacity change is less than

the minimum allowable change. The allowable change is easily updated through taking the minimum of the allowable change at the newly affected customer and the allowable change of the previously affected sub-path.

The method of Savelsbergh for the local search techniques mentioned is designed for dealing with routing problems with fixed precedence constraints. With the more general definition of the task that we are using, refinements to the method are necessary in the following situations:

- series of pick-ups and deliveries – since the precedences are still fixed, this case may be dealt with in the same manner as with a single pick-up and delivery.

- routing decisions within the task – with routing decisions being allowed within the task, the precedence structure is altered significantly. Instead of a sub-task having either a single predecessor or a single successor or both, here we are able to have a number of predecessors or successors for each sub-task. When "marking" in the local search phase, instead of "marking" a single position as is the case with fixed precedences, we would need to "mark" the position of each successor or predecessor of the appropriate sub-task.

- subset selection – when the routing decision allows for the locations used for a particular sub-task to be interchangeable, it is possible to use this extra freedom within the search process. A generalized form of 2-exchange was proposed by Fischetti, Salàzar and Toth [50] for the Generalized TSP (GTSP). The GTSP is the TSP except instead of there being a number of points to visit, there are a number of sets of points, of which at least one point from each set must be visited. For their 2-exchange, Fischetti *et al.* treat each vehicle's route as a sequence of sets. Their 2-exchange is used to change the order of visiting the sets and this is shown in Figure 5.7. The arcs between sets i and $i+1$ and j and $j+1$ are replaced by arcs from i to j and $i+1$ to $j+1$, with the path from set $i+1$ to set j reversed. The procedure also selects the best point to use from each of the sets i, $i+1$, j and $j+1$, while the selected point from each of the other sets is fixed. Fischetti et al. [50] defined a procedure that moves the position of a set within a vehicle route. In this case there are five sets from which it is possible to vary the selected point: those immediately before and after the moved set, both prior to and after the

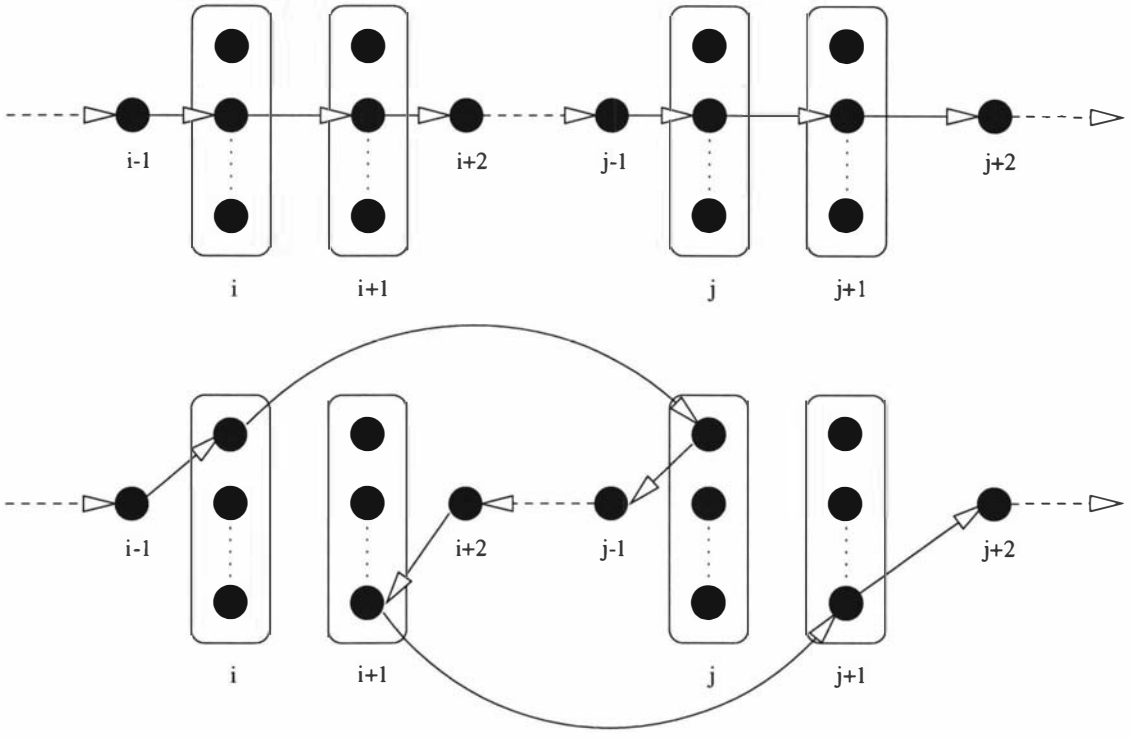

Figure 5.7: Generalized 2-Exchange

move, and the moved set itself. This procedure is a generalized form of the 1-Or exchange which can be directly applied to our task problem, where the sets consist of the interchangeable sub-tasks. The precedence and capacity checks would still be carried out in the same manner as previously.

- split tasks – split tasks further complicate the local search procedures mentioned previously, as more than one vehicle may be involved in the servicing of a task. The evaluation and feasibility checks for a move may need to take into account the temporal aspects of the routing. If the task involves more than one vehicle visiting the same location in order to transfer loads, the effect to the route caused by the change in the arrival time at a location must consider the other vehicles involved. Specifically, if coordination of routes is required, the times at which the routes intersect must be explicitly considered, so this requires the incorporation of techniques for handling time constraints, as was discussed in Chapter 4.

- preemptive tasks – with preemptive tasks, we need to ensure that the coordination points are still valid after applying an intra-route move. This would create many difficulties in terms of evaluating the feasibility and effectiveness

of a move. The most efficient manner of employing this technique would be to restrict the local search, to prevent the section of route containing the preemptive tasks from being affected. This would then allow the standard techniques to be applied to the unaffected sections of the route.

5.3.3 Inter-Route Improvement Techniques

We now turn to improvement techniques involving more than one vehicle. The standard techniques that we will consider involve the moving and exchanging of customers or sub-paths between pairs of vehicles.

The inter-route Move routine, of moving a customer from one vehicle to another, was previously shown in Figure 2.4 for the non-task case. The Move routine consists of selecting a customer to move, and finding the best position to place it in another vehicle. This technique can be used for task problems by moving a complete task. The process is to select which task to move and to which vehicle, and to determine the best feasible positions to place the sub-tasks in the new route. The feasibility concerns are to ensure the precedence and capacity constraints hold in the new vehicle. Ensuring that a route is feasible can be easily achieved by sequentially inserting the sub-tasks. Thus we increment the position of the last sub-task that can still be moved, and when we can no longer increment it, we return to the previous sub-task, increment its position, and start with the following sub-tasks serviced consecutively. This process ensures the precedence constraints hold, and the checking of the capacity constraints can be updated in constant time by comparing the load change within the task with the allowed change in the route.

The next technique we consider is the inter-route Exchange, which involves the exchange of customers between two vehicles. This process involves provisionally removing a customer from each of two vehicles and then inserting each removed customer in their best position in the other route. The non-task case was shown in Figure 2.5. Again this technique can be used with tasks, by moving whole tasks and finding the best positions for these tasks in the new vehicle routes. We carry out the same insertion process as with the Move routine.

The exchange of paths, as is carried out by the Cross procedure of Taillard et al. [171], was illustrated in Figure 2.6. This procedure can also be applied to task problems by ensuring that only complete tasks are moved. We define a *stand-alone sub-path* to be a sub-path consisting of just complete tasks, and so the Cross

exchange is valid if the sub-paths $i{+}1 \longrightarrow k$ from the first route, and $j{+}1 \longrightarrow m$ from the second route, are both stand-alone sub-paths.

By utilizing a lexicographical search procedure, we are able to carry out the feasibility checks in constant time. With our search procedure, we start with no points being moved and increment the number of points in the moved sub-paths. The capacity constraints are checked through calculating the maximum load change over the moved sub-path and comparing that with the maximum allowable change for its new position. The maximum load change can be updated in constant time and, if the capacity constraint is violated, we increment the starting point of the sub-path and begin again with an empty sub-path.

The checking of the stand-alone nature of the sub-path can also be implemented in constant time, by updating the minimum length of sub-path that could be stand-alone when incrementing the endpoint of the sub-path. This involves finding the latest position of the successor of one of the sub-tasks within the moved sub-path, and no sub-path that doesn't contain this point can be considered for being moved. A necessary requirement is that the first point of the sub-path be the first point of a task.

We now consider how the above moves can be applied to the problems where the tasks are of the more general form that we are allowing:

- series of pick-ups and deliveries – the inter-route moves may be carried out exactly as described above.

- routing within task – the inter-route improvement techniques may be directly applied to the case where tasks involve routing within the subtasks, but it is also possible to incorporate the extra freedom in the routing. The insertion phases in the customer Move and Exchange routines may be altered so that the sub-tasks may be placed in their best positions, with this decision also including the decision as to what is the best order to service the particular sub-tasks. The Cross exchange could be similarly altered, so that the best ordering of the moved tasks is placed in the other vehicle. These processes would quickly become too large to handle if the routing is very flexible, in which case only a subset of the available moves should be considered.

- subset selection – it is possible to introduce some generalized versions of the above improvement techniques in order to accommodate the subset selection

elements. The Move and Exchange routines could be revised in order to allow the insertion to alter the sub-tasks selected for each task. The Cross exchange routine could also be changed to allow the sub-tasks within the moved path to be altered. Again, this would only be possible for small cases, with restrictions necessary for larger problems.

- split tasks – with split tasks it is possible to shift parts of tasks, rather than necessarily moving complete tasks. For the customer Move and Exchange routines, we would transfer all the sub-tasks from a particular task to the new vehicle. A consequence of this process would be that, if we move a partial task to a vehicle that already contains some parts of this task, the process would automatically need to recombine the parts of the task into a single load. This would require removing one of the copies of the sub-task that is in both vehicles. The Cross exchange routine could also deal with split tasks, by requiring that, if any tasks are moved, all sub-tasks from that task within the vehicle must be moved. In moving parts of tasks, consideration is again needed as to the time at which the service can occur.

- preemptive tasks – special consideration of the feasibility and effectiveness of moves are again required for preemptive tasks. We would recommend that inter-route moves retain the structure of the preemptive tasks, so as to not create too many additional computational requirements.

5.3.4 Specific Moves for the Task Problems

With the more general definitions of tasks that we are considering, there are a number of additional moves that may be considered. These allow us to exploit the additional flexibility that is permitted with customers of these forms.

The techniques above are sufficient for dealing with tasks consisting of fixed sets of customers that have to be serviced by the same vehicle. For problems involving subset selection, a possible additional intra-route improvement move is a substitution move, in which a sub-task (or set of sub-tasks) can be replaced by an equivalent sub-task (or set of sub-tasks), if this leads to an improved solution. This therefore affects the sub-tasks that are serviced in order to complete the service of a task.

For task splitting problems, appropriate inter-route improvement moves include

splitting the tasks. This involves replicating an appropriate sub-task and moving the partial task to another vehicle, unsplitting the tasks by moving a partial task into an appropriate other vehicle, removing a replicated sub-task and recalculating the load.

With preemptive tasks, we may create intra-route improvements that alter the manner in which a customer is serviced. This could involve dropping off the load of a customer at an intermediate location and delivering this load at some later stage. If we have the multiple vehicle preemptive case, an inter-route improvement technique may be to drop off a customer's load at a location, where another vehicle may pick it up and finish the customer's service.

5.4 Computational Results

We now consider one of the more general definitions of a task described previously, and explore the potential gains that may be made through exploiting the additional flexibility that this task definition allows. We randomly generate 90 points within $[0,1]^2$, and position a depot at the centre of this region. From these points we create 30 tasks, by randomly assigning three points to each task, with the first two being assigned to be pick-up sub-tasks and the third being a delivery. There are two service vehicles, each having capacity of 5 units, and we randomly generate the loads for each pick-up sub-task so that the total load for a particular task is not allowed to exceed the vehicle's capacity, i.e., the load l_{i1} for the first point of task i is an integer in the range [1,4] while the load of the second pick-up is an integer in the range $[1, 5 - l_{i1}]$.

For the problem considered, the objective is to maximize the total reward collected within the time limit. This problem therefore incorporates subset selection, as we assume that not all tasks need to be serviced by the service vehicles. The vehicles travel at unit speed and must return to the depot before the end of the time limit. We assume here that there is no time required to actually service the customer and so each location need just be visited, without needing to remain there while making the pick-up or delivery.

We create four different types of problems through varying the type of reward function and the time restrictions for servicing the customers. In problem sets **A** and **B**, the rewards received for servicing a customer are linearly decreasing

functions of the time at which the delivery is made to the customer, while in problems **C** and **D**, the rewards decrease according to an exponential function of the time of servicing. For problem sets **A** and **C** there is no restriction on the time at which each customer may be serviced, while for problem sets **B** and **D**, the earliest time at which a customer's service may be completed is randomly generated within the time limit.

The initial reward for each customer was randomly generated in the range $[10, 50]$. For linearly decreasing rewards, we randomly generate the slopes so that the time the reward would reach zero is randomly distributed in the range $[a_i + 0.5 * (LIMIT - a_i), a_i + 1.5 * (LIMIT - a_i)]$, where a_i is the earliest time at which the service of customer i can be completed. For exponentially decreasing rewards, we set the exponential parameter for customer i equal to $1 + \frac{LIMIT}{2 * u_i * LIMIT}$, where u_i is a uniformly distributed variable in the range $[0, 1]$. The time limit is set equal to 10.0.

For each of the problem sets defined above we create 10 problem instances. We assume that each task may be split, in which case the two pick-ups are made by different vehicles and the reward is received at the time that the second delivery is made to the destination. We assume that the delivery may be made separately for each vehicle, so the vehicles need not meet at the delivery location. In order for the task splitting to be applicable, we relax the assumption of fixed precedences, thereby enabling the pick-ups to be made in any order.

Our solution method consists of constructing an initial solution by randomly selecting the next customer to insert in its best position in the current vehicle, and starting a new vehicle route if no improving insertion exists. This solution is improved using intra-route 2-exchange and Or-exchange techniques, and inter-route Move and Exchange. After applying these techniques, we carry out the best feasible split if splitting a task will improve the solution, and, if so, we perform these intra-route improvement techniques again. Since our construction heuristic uses random insertion, we solve each problem instance 10 times. We created 10 instances for each of the problem set, giving a total of 100 solutions for each of these sets.

We show the solution found for one instance in Tables 5.1 and 5.2. In the Tables, the column Pt refers to the number of the point that is visited, with X and Y the X- and Y-coordinates of its location. The column C gives the customer being

Vehicle 1 : Reward = 54.0167							Vehicle 2 : Reward = 47.1038						
Pt	X	Y	C	P	Time	L	Pt	X	Y	C	P	Time	L
0	0.5000	0.5000	0	0	0.0000	0	0	0.5000	0.5000	0	0	0.0000	0
19	0.3290	0.2497	7	+	0.4152	1	8	0.1132	0.4282	1	+	0.3818	4
9	0.9169	0.0852	7	+	0.4787	4	26	0.9328	0.0261	1	+	0.4416	5
2	0.6668	0.3478	5	+	0.6920	5	5	0.3280	0.0298	1	-	1.2415	0
6	0.3647	0.1132	7	-	0.8278	1	3	0.2164	0.3280	9	+	1.3257	3
27	0.7381	0.2214	5	+	0.9124	5	4	0.3478	0.3647	9	+	2.2477	4
12	0.4103	0.9372	5	-	1.1595	0	14	0.9372	0.7657	9	-	2.9486	0
1	0.5000	0.2164	10	+	1.3268	3	0	0.5000	0.5000	0	0	3.4983	0
10	0.4282	0.4103	10	+	1.6075	4							
30	0.1635	0.1952	10	-	1.7088	0							
16	0.7657	0.1345	8	+	2.3363	3							
25	0.3799	0.7381	8	+	2.3822	4							
20	0.3881	0.9381	8	-	2.4944	0							
0	0.5000	0.5000	0	0	3.0191	0							

Table 5.1: Initial Solutions for a Problem Instance

visited, with P referring to the status of the subtask: a + in this column means the subtask is a pickup, while a − is used when the subtask is a delivery. The column *Time* gives the time of servicing the particular subtask, and the column L gives the load in the vehicle upon the completion of each subtask. The instance that is solved has linearly decreasing rewards and no restrictions on the time of servicing. It is generated in the same manner as our other instances except, for clarity purposes, we have restricted the number of customers to 10, and the time limit to 5.0. The initial solution has total reward 101.1205 while the final solution has total reward 102.0045. The splitting removes the pick-up point 10 (from customer 10) from vehicle 1 and puts it into vehicle 2, along with the delivery point, 30. The delivery for customer 10 is made at time 1.6439 by vehicle 1 and at time 1.6480 by vehicle 2, so the reward for this task is attributed to the second vehicle. One interesting feature to note in these solutions is the lack of local optimality in the problem, which leads to different orderings being required when task splitting occurs. We see that in vehicle 1 prior to splitting, the best order is 0—19—9—2—6—27—12—1, while after the splitting the best order is 0—19—9—6—27—2—12—1. The latter solution visits the node 1 later in time, but the savings due to the removal of part of task 10 make this particular ordering better for this route.

Our computational results are shown in Table 5.3, where the *initial mean* is the

Vehicle 1 : Reward = 50.1851							Vehicle 2 : Reward = 51.8193						
Pt	X	Y	C	P	Time	L	Pt	X	Y	C	P	Time	L
0	0.5000	0.5000	0	0	0.0000	0	0	0.5000	0.5000	0	0	0.0000	0
19	0.3290	0.2497	7	+	0.4152	1	8	0.1132	0.4282	1	+	0.3818	4
9	0.9169	0.0852	7	+	0.4787	4	26	0.9328	0.0261	1	+	0.4416	5
6	0.3647	0.1132	7	-	0.7802	0	5	0.3280	0.0298	1	-	1.2415	0
27	0.7381	0.2214	5	+	0.8647	4	3	0.2164	0.3280	9	+	1.3257	3
2	0.6668	0.3478	5	+	0.9290	5	10	0.4282	0.4103	10	+	1.5468	4
12	0.4103	0.9372	5	-	1.2173	0	30	0.1635	0.1952	10	-	1.6480	3
1	0.5000	0.2164	10	+	1.3845	3	4	0.3478	0.3647	9	+	2.3117	4
30	0.1635	0.1952	10	-	1.6439	0	14	0.9372	0.7657	9	-	3.0126	0
16	0.7657	0.1345	8	+	2.2714	3	0	0.5000	0.5000	0	0	3.5623	0
25	0.3799	0.7381	8	+	2.3173	4							
20	0.3881	0.9381	8	-	2.4296	0							
0	0.5000	0.5000	0	0	2.9543	0							

Table 5.2: Post-Splitting Solutions for a Problem Instance

Problem Set	Initial Mean	Split Mean	Mean % Increase
A	273.708434	283.239912	3.4634
B	264.457178	276.147841	4.4928
C	30.223151	32.292320	7.3791
D	84.749187	93.211529	10.6164

Table 5.3: Table of Results for Split Loads

mean of the improved solutions prior to splitting, the *split mean* is the mean of the improved solutions after splitting, and the *mean % increase* is the mean of the percentage increases between the final solutions prior to and post splitting.

Table 5.3 shows that, over the problems generated, improvements are possible through allowing tasks to be split. For the problems with the exponentially declining rewards, the improvement through splitting is more dramatic; this is due to being able to use parallel servicing of a customer. In problems with decreasing rewards, the ability to split the servicing of a customer in order to complete the service sooner, may be beneficial for creating improved solutions. With exponentially declining rewards, the gain through the earlier servicing of a customer, is more dramatic, because of a higher rate at which the reward is decreasing, especially in the early stages of delay.

The computational results demonstrate that potential gains may be made by allowing tasks to be split and serviced by more than one vehicle. We believe that the splitting of tasks is a beneficial area of research due to the naturalness of such an approach. In vehicle routing problems with split deliveries, the loads to be delivered are split and are delivered by separate vehicles, usually involving some arbitrary division of the loads in order to fit the vehicle's capacity. With task routing problems, as we have defined them, there can be a natural division in the loads due to the different roles of the sub-tasks. Thus, with tasks consisting of a number of independent pick-ups to constitute a load, the different pick-ups provide natural divisions to the overall load. We have shown that, in certain problems, it is possible to make substantial improvements to the solution quality by allowing tasks to be split. This is therefore an area where further research appears to be warranted.

CHAPTER 6

The Combined Problem

We now combine the elements discussed in Chapters 3 through 5 to obtain a general routing problem, which we call the Maximum Collection Problem (MCP). We look at what are the important features of the problems, and we consider a generic solution method for any type of problem. Our underlying problem is the Orienteering Problem (OP), to which we add various additional features.

We define the MCP to have the following properties:

- there are a fixed number of vehicles available, each with a given capacity, a given origin location, and a time period over which it can be used
- there are a number of customers available for servicing. Each customer has an associated reward that is received for servicing the customer, where the value of the reward received is dependent upon certain attributes of the time at which the customer is serviced, e.g., the time at which service is started and/or finished.
- each customer may be comprised of a number of points to be visited with loads to be transported between the points, and there may be a time window (or time windows) during which each customer is available for servicing.
- there are penalties for non-service, which may relate to the individual customers that are not serviced or to the total set of unserviced customers.
- the objective is to service a given subset of the customers such that the net

reward received is maximized, where the net reward is the reward collected less the penalties incurred and the costs of servicing.

The MCP, as we have defined it, consists of the components of Chapters 3, 4 and 5, i.e., subset selection, precedences and capacities, time windows and time-dependent rewards. Although the time window problem may be viewed as a special case of time-dependent rewards, we deal with these separately, as the time windows cause feasibility concerns while the time-dependent rewards influence the viability of the routing.

6.1 A Classification Scheme for the MCP

We now consider the features that comprise the MCP, and use these to create a classification scheme for subset selection problems. The classification scheme we are creating is based upon one that was used by Desrochers, Lenstra and Savelsbergh [35] to describe vehicle routing and scheduling problems. This scheme was later used by Desrochers, Jones, Lenstra, Savelsbergh and Stougie [33] within a decision support system which attempted to use appropriate solution techniques for a problem instance of unknown type, through comparing the problem's representation, via this classification scheme, with that for a known problem for which solution methods are available.

The scheme gives a classification of the form:

⟨classification⟩ ::=

⟨addresses⟩

⟨vehicle attributes⟩

⟨problem characteristics⟩

⟨objectives⟩

A fifth category may also be added, which gives some additional information about the practical aspects of the particular problem instance, and we will consider this category in more detail later. For the other categories, which are known as *fields*, the scheme breaks these into *subfields* which correspond to the attributes being considered. For our notation, we surround each field and subfield by angular brackets ⟨, ⟩. The subfields are comprised of *tokens*, which are tokens that stand for the attribute in the problem instance. We use the symbol ∨ (the exclusive or) to separate the different possible values of the tokens, and we use the symbol ∘ for the default

attribute. The representation of the classification scheme involves showing each of the relevant tokens from each of the subfields, where we separate tokens from the same field by commas, and we separate the different fields through using the vertical bar, | . A subfield being empty indicates that the default token is present in this problem.

We will now consider each of the fields, using the subfields from the original paper and noting the changes that we use for our classification scheme, where required. We adapt the scheme so that different attributes are only considered if they invoke different behaviour from the solution methods.

6.1.1 Addresses

The addresses field refers to the customer and depot attributes and these are given as:

⟨addresses⟩ ::=
- ⟨number of depots⟩
- ⟨type of demand⟩
- ⟨address scheduling constraints⟩
- ⟨address selection constraints⟩
- ⟨customer rewards⟩
- ⟨non-service⟩

⟨number of depots⟩ ::= $o \vee l$
- o [one depot]
- l [specified number of depots]

⟨type of demand⟩ ::= $\langle\alpha_1\rangle\langle\alpha_2\rangle$

$\langle\alpha_1\rangle$ ::= $o \vee$ EDGE $\vee$ MIXED $\vee T_{fix} \vee T_{rtg} \vee T_{sub} \vee T_{dec} \vee T_{pre}$

o	[node routing]
EDGE	[edge routing]
MIXED	[mixed node and edge routing]
T_{fix}	[task routing; tasks have a fixed order]
T_{rtg}	[task routing; routing allowed within tasks]
T_{sub}	[task routing; subset of addresses within each task must be visited]
T_{dec}	[task routing; decomposable tasks]
T_{pre}	[task routing; preemptive tasks]

$\langle \alpha_2 \rangle ::= \circ \vee \sim$

- $\circ$ [deterministic demand]
- $\sim$ [stochastic demand]

⟨address scheduling constraints⟩ ::= $\circ \vee fs_j \vee tw_j \vee mw_j$

- $\circ$ [no time window constraints]
- fs_j [fixed schedule]
- tw_j [single time window for each customer]
- mw_j [multiple time windows for each customer]

⟨address selection constraints⟩ ::= $\circ \vee subset \vee choice \vee period$

- $\circ$ [single plan; all addresses must be visited]
- *subset* [single plan; only a given subset of addresses needs to be visited]
- *choice* [single plan; at least one address in each subset of a given partition must be visited]
- *period* [a number of plans over a given time period is to be made]

⟨customer rewards⟩ ::= $\circ \vee K \vee TDR_{inc} \vee TDR_{dec}$

- $\circ$ [no rewards]
- K [constant rewards]
- TDR_{inc} [time-dependent rewards, may increase]
- TDR_{dec} [time-dependent rewards, non-increasing]

⟨non-service⟩ ::= $\circ \vee NP \vee \infty \vee M$

- $\circ$ [non-service is not allowed or there are no penalties for non-service]
- *pen* [there are finite, constant penalties for non-service]
- ∞ [some penalties for non-service are infinite]
- M [all customers must be serviced in some manner; costs are incurred for those customers that are not in the solution route]

For the address attributes we have added two subfields to those given in the original classification scheme; customer rewards and non-service. We consider the case of time-dependent rewards that may increase over time to be different from the strictly non-increasing case, as there may be advantages in delaying the time of servicing a customer with increasing rewards. For non-service, we consider different cases for where there are finite penalties for non-inclusion (which affects the

attractiveness of the customers), infinite penalties (which need to be explicitly considered in order to obtain feasible solutions), and where the customers that are not considered to be in the solution route may contribute to the costs of the system.

We have also adapted the 'type of demand' subfield, in order to allow for the more general cases of tasks that we have considered previously. The case T_{dec} refers to decomposable tasks, where this incorporates the idea of split tasks, and also for cases where tasks may be partially serviced.

6.1.2 Vehicles

The next field considered relates to the vehicles used in the problem. The attributes considered here are very similar to those used in [35], so we cover them quite briefly.

⟨vehicles⟩ ::=
- ⟨number of vehicles⟩
- ⟨capacity constraints⟩
- ⟨commodity constraints⟩
- ⟨vehicle scheduling constraints⟩

⟨number of vehicles⟩ ::= $c \vee m$
- c [there are a constant number of vehicles, c that are available for servicing]
- m [multiple vehicles are available, with the number used being a decision variable]

⟨capacity constraints⟩ ::= $\circ \vee cap \vee mcap \vee cap_i \vee mcap_i$
- $\circ$ [no capacity constraints]
- cap [common capacity for all vehicles]
- $mcap$ [common, multiple capacities for all vehicles]
- cap_i [different capacities for the different vehicles]
- $mcap_i$ [different multiple, capacities for the different vehicles]

⟨commodity constraints⟩ ::= $\circ \vee sep \vee ded$
- $\circ$ [no capacity constraints]
- sep [vehicles have interchangeable compartments]
- ded [vehicles have dedicated compartments]

⟨vehicle scheduling constraints⟩ ::= $\circ \vee T \vee T_i$
- $\circ$ [no scheduling constraints]
- T [a common time of availability for all vehicles]
- T_i [a separate time of availability for each customer]

The differences we are making to the classification scheme are to not consider the case where all available vehicles have to be used, and to not consider there to be a distinction between time windows of availability and route duration constraints. The 'commodity constraints' subfield, enables different commodities to be handled in different manners. The inclusion of dedicated compartments would imply that there are multiple capacities of the vehicle, which gives additional features to be concerned with. Because of this possible occurrence, we introduce the additional tokens within the 'capacity constraint' field to deal with multiple capacity vehicles. For example, in the handicapped transportation application considered by Toth and Vigo [175], the vehicles considered are able to carry various numbers of passengers with differing care requirements, and so the mix of the passengers assigned is an important feasibility consideration.

6.1.3 Problem Characteristics

The field on the problem characteristics contains a number of features that enable the classification scheme to be used for general routing problems. These allow for restrictions on the assignment of customers to vehicles, of customers to depots and vehicles to depots, and they also consider the restrictions of customers so that incompatible customers can not be carried together. We acknowledge that these type of restrictions may be useful in the classification of general routing problems, but since these are not relevant for the types of problems we are currently considering, we do not include these features within our scheme. The classification, in terms of problem characteristics, that we consider is then:

⟨problem characteristics⟩ ::=
⟨restrictions⟩
⟨type of network⟩
⟨type of strategy⟩
⟨costs incurred⟩

⟨restrictions⟩ ::= $\circ \vee rew$
$\circ$ [no target reward restriction on the routing]
rew [a given target reward must be received]

⟨type of network⟩ ::= $\langle\gamma_1\rangle \vee \langle\gamma_2\rangle$

$\langle \gamma_1 \rangle ::= \circ \vee g$

$\circ$ [Euclidean distances]

cap [general distances]

$\langle \gamma_2 \rangle ::= \circ \vee dir$

$\circ$ [undirected network]

dir [directed network]

$\langle$type of strategy$\rangle ::= \langle \delta_1 \rangle \vee \langle \delta_2 \rangle$

$\langle \delta_1 \rangle ::= \circ \vee back \vee full$

$\circ$ [no restrictions on the service order of the loads carried]

$back$ [node routing; only backhauling allowed]

$full$ [task routing; only one load at a time is allowed in the vehicle]

$\langle \delta_2 \rangle ::= \circ \vee MD \vee path$

$\circ$ [each route starts and ends at the same location]

MD [multiple-depot routes are allowed]

$path$ [each route is a path starting at a given depot]

We create a new subfield which considers the constraints on the problem. These are the resource limitations that may be used within the subset selection problem to restrict the routing or to ensure that certain goodness of routing properties are met. We are also allowing different forms of the solution, through enabling the solution routes to consist of open paths. This case involves the service of customers being the important feature, with service allowed until customers are no longer available. The final location of the vehicle is assumed to be a far less important consideration than the customer service, in this situation.

6.1.4 Objectives

The field containing the objectives, needs to be adapted from the original classification scheme in order to handle subset selection problems. The original objectives considered only the minimization of costs, where either the minimum of the sum of costs or the minimum of the maximum cost of the vehicles was sought. For subset selection problems, the standard objective is either to maximize the net rewards or to minimize the costs of servicing a subset of the available customers. We also consider different cases concerning which customers affect the objective function, as we make the distinction between pure subset selection (where only the customers in

the solution route will affect the objective function) and restricted subset selection (where some or all of the customers will affect the objective function even if they are not included in the solution).

This field is therefore given as:

⟨objectives⟩ ::=

⟨objective⟩ ∨ ⟨objective⟩⟨objectives⟩

⟨consideration⟩

⟨objective⟩ ::= ⟨operator⟩ ∨ ⟨function⟩

⟨operator⟩ ::= max ∨ min ∨ minmax

max	[maximize the sum of the values]
min	[minimize the sum of the values]
minmax	[minimize the maximum value]

⟨function⟩ ::= $dur_i \vee NV \vee R_i \pm C_i \pm C_j \pm NS \pm P_i$(⟨vehicle constraints⟩) $\pm P_j$(⟨address constraints⟩)

dur_i	[route duration]	NV	[number of vehicles employed]
R_i	[reward of vehicle]	C_i	[vehicle cost]
C_j	[address costs]	NS	[costs of non-service]
P_i(⟨vehicle constraints⟩)			[vehicle penalty]
P_j(⟨address constraints⟩)			[address penalty]

⟨vehicle constraints⟩ ::= ⟨vehicle constraint⟩ ∨ ⟨vehicle constraint⟩⟨vehicle constraints⟩

⟨vehicle constraint⟩ ::= $cap \vee cap_i \vee mcap \vee mcap_i \vee T \vee T_i$

⟨address constraints⟩ ::= $tw_j \vee mw_j$

⟨consideration⟩ ::= ∘ ∨ *sub* ∨ *part*

∘	[all customers are considered equally in the objective]
sub	[only the customers in the solution routes are considered]
part	[all customers are considered in the objective, with those in the solution contributing differently from those who aren't]

The objectives field therefore describes the possible objectives of the problem, where we drop the vehicle subscripts if we only have a single vehicle. Of note here is that we have included arithmetic operations within the 'function' subfield, in order to deal with cases where there are a number of factors in the objective function. In this case the original classification scheme lists all the components of

the objective function in decreasing order of importance, but this approach seems more appropriate for multiple objective problems than for composite objective functions. Our notation for a multiple objective problem is to list all the objectives, with these separated by colons if the functions are dealt with in an hierarchical fashion, or separated by semi-colons if all the objectives are considered.

6.1.5 Examples

We now demonstrate how the above classification scheme may be used to describe a wide variety of routing problems, particularly with regard to subset selection problems.

1. $|1||\min\ dur$ — this is the travelling salesman problem, where there is one vehicle and the objective is to minimize the total route duration.

2. $|m, cap||\min\ C_i$ — this is the standard vehicle routing problem, where there are a number of available vehicles and the objective is to minimize the costs of routing. In most examples in the literature (see [165, 166]) an hierarchical objective is considered, with the first objective being to minimize the number of vehicles required, and the 'objectives' field in this case would be given as 'min NV; min C_i'.

3. $|m, cap||\min\ (P_j(tw_j) + C_i)$ — this is the vehicle routing problem with soft time windows, where there are a number of available vehicles and the objective is to minimize the costs of routing, which are incurred for the costs of operation as well as for the penalties for deviation from the customer's desired service time.

4. $subset, K, T|1||sub, \max\ R$ — this is the Orienteering Problem [73], where the objective is to maximize the total reward received in a tour which has a given limit on route duration.

5. $subset, TDR_{dec}|1||sub, \max\ R$ — this is the Maximum Collection Problem with Time-Dependent Rewards [20], where the objective is to maximize the total reward collected, where the rewards decrease over time.

6. $subset, K, M, T|1||sub, \max\ (R - NS)$ — this is the form of the Orienteering Problem with Cleanup, that we considered previously in Chapter 3.

7. $T_{dec}, subset, TDR_{dec}, T|1||sub, \max\ R$ — this is the case of the task routing problem with split tasks and decreasing time-dependent rewards, as considered in Chapter 5.

8. $subset, K, pen|1|rew|\min\ (C + NS)$ — this is the Prize Collecting Travelling Salesman Problem (PCTSP) of Balas[8].

9. $subset, pen|1||\min\ (C+NS)$ — this is the unconstrained PCTSP of Bienstock, Goemans, Simchi-Levi and Williamson [18].

6.1.6 Additional Features

The additional fifth field, while not actually being formally defined, is where information on the problem instance may be stored in order to help in devising appropriate techniques for the problem type, and we now discuss how we might use this information for the MCP. We consider that finding solutions for the MCP consists of three types of decisions: routing, scheduling and selection ; we consider how these decisions may dominate particular problem types.

Routing decisions relate primarily to the spatial aspects of including a customer in the solution route. We regard the routing decisions of the MCP to involve the inclusion of customers purely with regard to the insertion distance in the solution routes. We believe that when the time limit is high enough that most of the customers may be included in the solution, there is little variability in the customer rewards and the time windows of availability are wide, then the routing decisions will become more important, and so solution methods that focus on the distances of inclusion should be applied.

Scheduling decisions relate primarily to the temporal aspect of the time at which each customer is serviced. This will be of primary importance if there are tight time windows over which the customers are available to be serviced and/or there is a high variability in the reward that is available at different times for each customer. If either of these cases exist, the time of servicing the customers will be of most importance, and methods that focus on customer scheduling will be important.

The selection decisions involve the decisions of which customers to service. These decisions become more important as the variability in the available reward for each customer increases, and as the number of customers able to be serviced,

and the penalties for non-service, decrease. Another important factor is the variability in the time required for inserting each customer, since with low variability the selection becomes more important as there is little to distinguish between the viability of the customers according to the times. In these cases the main consideration is which customers to include, and so methods that effectively select the customers are relevant.

Therefore we would suggest that the fifth field contain some information on the relative priorities of the routing, scheduling and selection components of the problem, as these can promote the use of appropriate methods. Another important factor may be the form of the solution that may be obtained, which we consider now with reference to task problems. If the size of the load that is carried within each task is high, then a service vehicle may only be able to carry one customer at a time. In this case, the task problem becomes similar to a node routing problem, except there is an entry and exit point for each task and there is a duration to consider. Methods based on node routing will likely be relevant for these problems, and so storing the information regarding the form of problems within the fifth field would again aid in matching appropriate solution methods to the problems.

The information that is described above is somewhat subjective, summarizing what we feel are the features of a particular problem instance. We now consider a more objective method of describing an instance of the MCP, which considers the actual impact of the constraints present.

We may know that the problem contains tasks, but if there are very few tasks and customers mostly consist of single nodes to be visited, then the problem may behave differently from one where all the customers consist of tasks, in which case different solution methods may be required. Therefore we now give a scheme for determining the actual impact of the constraints that are included in the problem. We describe any instance of the MCP by the quadruple (*subset, task, TW, TDR*), where:

- *subset* is a value that indicates the proportion of customers which may be freely included in the solution, i.e., if $subset = 0$, then all customers must be accounted for in the solution
- *task* is the proportion of customers with precedence constraints and/or loads
- *TW* is a measure of the average tightness of the time window constraints,

which is given by: $1 - \frac{\text{average length of time windows}}{\text{time horizon of problem instance}}$. If there are no time windows present then $TW = 0$, while $TW = 1$ if all customers have a single point of time at which they are available.

- *TDR* is the average proportion in variability in a customer's reward over the course of its horizon of availability, due to the time-dependent rewards. This is $\frac{1}{n}\sum_{i=1}^{n} \frac{\max_i - \min_i}{\max_i}$, where $\max_i$ and $\min_i$ are respectively the maximum and minimum values that the reward for customer i may take within the time it is available for servicing. If all rewards are constant, then $TDR = 0$, while $TDR = 1$ if all rewards may go to zero within the time of availability.

Each of these terms fall in the range [0,1]. A number of different problems may be defined using this methodology, and some examples are shown in Table 6.1. We consider that problems without any subset selection elements form one type of problem which cannot be usefully described by this terminology, as the objective of such problems are different from the ones we consider for the MCP. The one useful version of the MCP with *subset* = 0, that fits within the context we are currently considering, involves the maximization of time-dependent reward functions.

The range of values of *subset* determines the priorities in the problem; higher values indicate that the selection element is the most important, while lower values require more serious consideration of the feasibility of including all the required customers. The *TW* term indicates the extent to which the time windows may impact upon the routing, through restricting the time of servicing the customers, while the *task* term indicates whether the precedence constraints of the tasks will have a significant impact on the problem or if the customers can mostly be considered as single points. The *TDR* term indicates whether or not the time of servicing a customer will greatly affect the reward that is available.

This form of problem identification enables greater information about the impact of the constraints to be found. This again does not give complete information about general versions of the MCP as, for example, we haven't included the impacts of penalties or costs of routing in this terminology. Also, other exploitable information such as the variability in rewards and the load sizes cannot be included without a very large classification scheme being given. The very general definition of the MCP leads to these issues, which we seek to resolve to some degree, through refining the versions of the problem that we intend to study in more detail.

Description	Problem Described
(1,0,0,0)	OP
(1,0,0,1)	MCPTDR
(1,0,0.9,0)	OPTW with tight time windows
(1,0,0.1,0)	OPTW with loose time windows
(0,0,0,1)	scheduling problem of reward maximization
(0.25,0,0,1)	OP with 3/4 of the customers fixed
(1,1,0.9,1)	the MCP with tight time windows

Table 6.1: Classification of Types of MCP

6.2 Individual Components of the MCP

We have defined the MCP to consist of a general routing problem that features subset selection, tasks and time considerations. We now consider the practical implications of how these components may be incorporated into the model, and then define the particular cases that we choose to investigate further.

The first feature we consider is how to deal with customers that are not serviced. We observe that most previous subset selection routing problems, e.g., the OP and the MCPTDR, simply ignore these customers and focus on the customers that are serviced. Also, we observe that for a problem where there are constant penalties for non-service, we can ignore these penalties through adding the penalty to the reward for each customer. We discussed other methods for dealing with non-service in Chapter 3, and considered one special case where the cost of servicing the remaining customers in a single tour is included in the objective function. This additional feature has the ability to make dramatic changes to the actual problem being considered, as a greater focus on the whole set of available customers is required.

We believe that the consideration of non-serviced customers may be an important feature for certain practical applications of subset routing problems. We leave this as an area for future research, as the consideration of all customers will take away from the focus on the subset selection elements, which is the area we are most interested in obtaining practical solution methods for. Therefore the problem instances we consider further will allow for customers to be excluded from the solution with no associated impact on the objective value. We now look in greater detail at relevant task and time considerations.

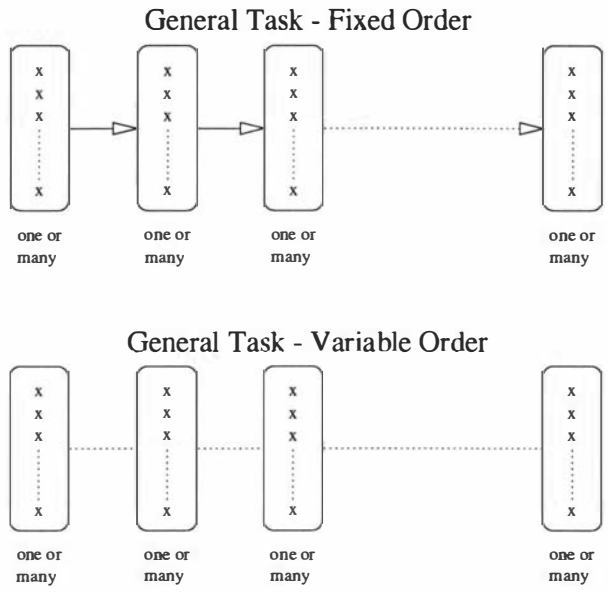

Figure 6.1: Our General Form of Tasks

6.2.1 General Forms of Tasks

As discussed in Chapter 5, there are many possible definitions of tasks that we may use within the MCP. The general form of the tasks is shown in Figure 6.1. For each customer there are a number of sets of points, of which we must choose a given number of points to visit, in order to complete the customer's service requirements. We may service the customers within each set in any order. The two different forms of tasks in this case are where we have a fixed order and a variable order for servicing the sets.

Even when we are allowing a variable order of servicing the subtasks of a customer, there are some practical issues that restrict the routing that is allowed. For example, if a task consists of a number of pick-ups and deliveries that may be carried out in any order, we still require that the load carried is always non-negative, i.e., we must have picked up a load before we attempt to deliver it. This creates some restrictions which are situation-dependent, rather than being fixed constraints, i.e., we can't place a subtask in a position where the delivery that is made is less than the load for this task that has previously been picked up.

We note that the form of task that has received the most attention is the case with one single pick-up and one single delivery for each customer. This is the case studied in the DARP (see [141, 142]) and the PDP (see [180, 42]). The distinction between the DARP and the PDP that is generally used is that, with the DARP the

load carried for each customer is one unit, i.e., it consists of transporting one person, whereas with the PDP, general loads may be carried. Savelsbergh and Sol [158] described the General Pick-Up and Delivery Problem (GPDP) where there are a number of pick-ups and a number of deliveries, and all pick-ups are made before any deliveries. The only paper that we are aware of, with tasks consisting of sets of customers to be visited is that of Savelsbergh and Sol [159]. They described a practical problem where each task consists of a single pick-up followed by a number of deliveries that must be carried out in a pre-specified order.

One form of task we will study, is where tasks consist of a fixed order of points to be visited. We assume that for each task there is a positive load in the vehicle for the duration of the task. This requires the first subtask visited to be a pick-up and the last to be a delivery, with the intermediate subtasks consisting of either pick-ups or deliveries. This form may include the cases considered by Savelsbergh and Sol [159], but can also include more general sequences of subtasks.

We consider the case where a customer consists of a delivery to a location i to be a task consisting of two subtasks, where the first subtask occurs at the depot. The case where customers consist of a pick-up to be taken to the depot, may similarly be considered as tasks, where the second and final subtask occurs at the depot. Therefore problems such as the Vehicle Routing Problem (VRP) and its extensions involving linehauls (considered to be node routing problems), may be incorporated within this task terminology.

We are also interested in the effect of using more general task definitions. For this purpose we consider the case where the customer consists of a number of pick-ups that may be carried out in any order, followed by a single delivery. These are cases where there is a divisible load that is to be collected from various sources, and the desired delivery consists of the total load. This case is applicable with taxi companies, where a number of customers are picked up from different locations, to be taken to a common destination.

Since there has been very little research into problems involving tasks consisting of more than two components, and none where there is flexibility in the order in which the components of the task may be carried out, we feel that there is much to be gained through the additional requirements we mention above. We restrict attention in our problem instances to cases where there are at most three subtasks in each task, which enables the extensions to be applicable and allows methodologies

for dealing with larger cases to be introduced, without superfluously increasing the data requirements for the problem.

With more general cases of pick-ups and deliveries that may be made in any order, an important consideration is to ensure that the load is feasible at all times. This requires some additional tracking of the tasks themselves, and the cumulative loads carried at any stage. The additional requirements of this tracking are left for future work. We are also only interested in focusing on cases with precedences and loads, so we do not consider tasks where all the subtasks may be visited in any order. Other cases involving selection of which subtasks will be used to complete the tasks also bring additional tracking requirements, and there is also much extra flexibility with these problems that requires further improvement routines to take advantage of the possibilities that are present. Decomposable tasks require keeping track of a number of vehicles, in order to allow coordination of customer service, and this creates even further possibilities for routing. In order to gain the most out of these problems, much work that is beyond the scope of our current study, would be required.

Implication of the Task Definition

In order to include more general definitions of tasks, we need to adapt the data structures that we use for our solution methods. When the problem consists of n tasks where each task is an (origin,destination) pair, the simplest approach is to label the origin of task i as i and the destination as $i+n$. Therefore, in scanning through a vehicle's route, we can easily identify the task that a point comes from and the role it has within the task. For more general tasks, we store the $num\_pts(i)$ points that comprise task i in a matrix, where $task(i, j)$ is the point in the j'th position of task i. We need an identifier which shows the role the point has within the task, and so we use a variable $point\_to\_task(i)$ to indicate the role of point i. If k is the i'th point of task j, then we set $point\_to\_task(k) = N * j + i$ where N is a number that is larger than the greatest number of points within any task. Therefore when scanning through a vehicle's route, we can identify the task a point k is from, as it is $\lfloor point\_to\_task(k)/N \rfloor$ and the position within this task is $point\_to\_task(k) - N * \lfloor point\_to\_task(k)/N \rfloor$.

With precedences consisting of pairs, we assign labels to indicate the position of the predecessor or successor of the point in a particular position. This has been

used by Savelsbergh[155], who labelled the precedences as:

$$prec(v) = \begin{cases} u & \text{if vertex } v \text{ must precede vertex } u \\ -u & \text{if vertex } u \text{ must precede vertex } v \\ 0 & \text{if vertex } u \text{ has no precedence relationship with any other vertices} \end{cases}$$

When we have tasks consisting of more than two points, this form of labelling is no longer valid. For these problems we create a two point precedence label, which enables us to create forward and backward precedences. We set $prec(v, 0)$ to be the position that vertex v must be placed after. If there is no predecessor for v, we set $prec(v, 0) = 0$. Similarly, we define $prec(v, 1)$ to be the position that vertex v must be placed before. If v has no successor, we set $prec(v, 1) = place + 1$, where $place$ is the number of vertices in the current vehicle route.

We also need to consider the precedences for the more general case we investigate, where there are a number of pick-ups and a delivery. We consider a solution route where there are three subtasks in a task, where the first two are pickups and these are placed in positions i and j, and the third is a delivery, placed in position k in the current solution. Here the pick-up subtasks have no predecessor, i.e., $prec(i, 0)$ and $prec(j, 0)$ are both equal zero, and they have a common successor, which is the position of the delivery subtask, i.e., $prec(i, 1)$ and $prec(j, 1)$ are both assigned the value k. The delivery subtask has no successor, so we set $prec(k, 1)$ to be equal to $place + 1$ and its predecessor is the position of the second of the pickup subtasks, i.e, $prec(k, 0)$ is set equal to j.

Other forms of the tasks may require the precedences to be defined differently. For example, if the subtasks can be serviced in any order, then there may be no fixed precedence constraints, with the only restrictions being situation-dependent, and the restrictions on the order a customer is serviced are just used to ensure that the load remains feasible.

6.2.2 Time-Dependent Rewards

With our definition of the MCP, we allow there to be any general reward function. We now look at the practical implications of this, with respect to the way our customers are defined, and we will define some appropriate reward functions to investigate further. We initially consider only the case where each customer consists of a single subtask that needs to be visited in order to service the customer. In

this case, the time-dependent reward is a function of the time at which the service of the customer is completed. Later in this section, we consider the implications of the use of time-dependent reward functions with tasks.

As noted previously, the only reward functions that have been considered in the literature are rewards decreasing linearly at different rates (as is the case for the MCPTDR), and the functions of Keller [94], where there is a common reward function for each customer, and this function is either sinusoidal, quadratic or Gaussian. Sinusoidal functions model seasonal demand for services or possibly multiple time window problems, where the windows are evenly spaced and there is a penalty that relates to the deviation from the nearest window. Gaussian functions involve the reward always remaining positive, with steep declines in the reward near its maximum and lower declines further from the maximum. This models sales where there is always some demand, and these increase around the time of a special event. These functions were used in the computational examples of Keller [94], with the main advantage being that all customers may be serviced as the reward function always remains positive. Quadratic functions have low declines near the maximum and increasing rates of decline the further away from the maximum; they can therefore model customer dissatisfaction where the rate of dissatisfaction is increasing the further from the desired time the customer are serviced. A generalized quadratic function, where the rates of decline are different for earliness and tardiness, may increase the applicability of the function.

Our concerns are mostly with modelling customer dissatisfaction, where the dissatisfaction is a function of the difference between the customer's desired service and the time the actual service is carried out. The functions we deem to be most appropriate in these cases are ones where the rate at which the reward declines is non-decreasing in the deviation from the desired time. Some of the models that fit these requirements involve rewards decreasing as a polynomial function of the absolute value of the deviation, with the polynomial of degree at least 1. We are interested in our research being comparable with previous work, and with these purposes in mind we propose to model the case where the reward received is a linearly decreasing function of the deviation from the desired service time.

We define the Piecewise Linear Maximum Collection Problem (PLMCP) to be a form of the MCP where the reward functions for each customer are restricted to be of the form shown in Figure 6.2. This reward function assumes that there is

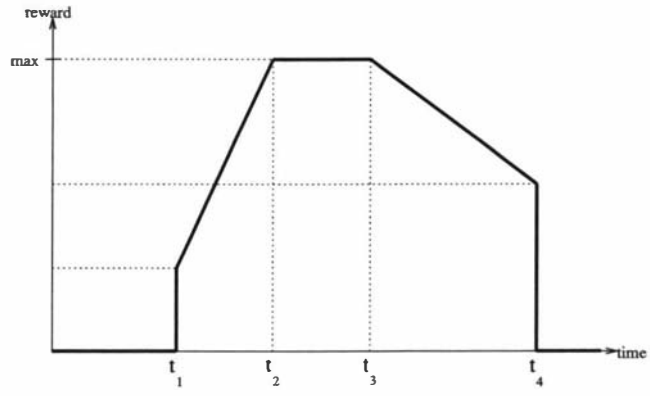

Figure 6.2: Our Piecewise Linear Reward Function

a time window, $[t_1, t_4]$, over which it is possible to service the customer. There is a desired interval $[t_2, t_3]$, over which there is a constant reward for servicing. The customer may be serviced before this desired interval and there is an associated linear earliness penalty, which has a rate of p_1. There is also a linear lateness penalty with rate p_2 for servicing after the desired interval. This is analogous to the problem of soft time windows, where the costs are linearly increasing for deviations from the time window for servicing. By using different penalties p_1 and p_2, we are able to give different priorities for serving the customer earlier or later than their desired time.

The piecewise linear reward function is appropriate with customers consisting of single points, with the reward received being a function of the time at which the customer's service is completed. This reward function is able to incorporate many possibilities for the form of the reward, including the reward function of the OPTW (if $t_1 = t_2$ and $t_3 = t_4$) and the reward function of the MCPTDR (where $t_1 = t_2 = t_3 = 0$). Therefore the OPTW and the MCPTDR are special cases of the PLMCP, with all customers consisting of points. The reward function is able to incorporate cases with constant rewards (where $t_1 = t_2 = 0$ and $t_3 = t_4 =$ the time limit for the problem), which means that the TRCP can be considered to be a special case of the PLMCP, with constant rewards, but with customers consisting of tasks. The OP is a special case of each of these problems, where all customers consist of points and the reward for each customer is a constant over the whole time horizon. We give a hierarchy of problems in Figure 6.3, where each tier indicates one additional level of constraint from the previous tier. This is used to show how the reward function fits among problems that have already been defined.

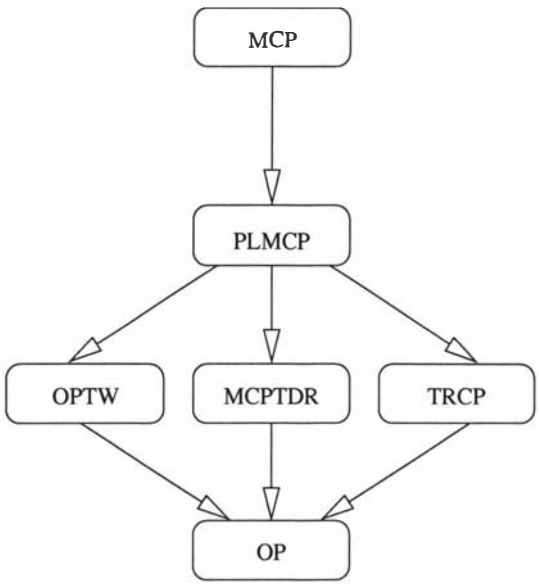

Figure 6.3: A Hierarchy of Problems

For problems with customers consisting of tasks, there are a number of additional possibilities that may be considered. The time at which a customer is serviced is no longer just the time of completing the task, but there are times at which each component of the task is serviced, that may be incorporated into the time-dependent reward function.

If the reward depends only on the time of completing the task, this single time is considered in the reward function. The reward functions for our point problems, including the piecewise linear reward function of Figure 6.2, are again appropriate in this case. This case may arise when the customer is only concerned with the time at which their requirements are met, as may be the case with the transportation of non-perishable goods. The time of picking up the loads may be unimportant, as the only concern is with when the load arrives at the destination location.

We also consider the case where the reward received depends upon the time of servicing some or all of the subtasks, which may involve an additive or multiplicative effect. In these cases we assume that there is a desired time of servicing each component, and there is a penalty for deviation from this time. One way we may use this is with multiplicative rewards, where we may create a hard time window on a subtask, by giving zero reward for servicing this subtask outside the window. Thus the total reward for the task will be zero if this time window is violated. We may make the time of servicing certain of the subtasks to be more or less important, by altering the relative penalties for these subtasks. An example of this

type of reward may be for a task consisting of a number of pick-ups and deliveries, where the customer is only concerned with the time that the load is taken from its location. In this case, we may have a multiplicative reward function, where penalties are only incurred for deviations from the desired pick-up times. This case may arise with warehouses, where the penalties are used to promote manageable levels of inventory, by penalizing the deliverer for delays in picking up the goods, with delays being costly due to additional inventory being accrued.

One possibility that arises with an additive reward function, may be the ability to service part of a customer's requirements with a portion of the available reward received for this partial service. This type of service might occur within the context of some home delivery systems, where there is still some benefit from delivering part of the overall load, although this is less than if the total load was made available. For this case, we could create a number of tasks for each decomposable task. Therefore instead of there being a task consisting of p_1—p_2—p_3—d_1, where the p_i are pick-ups and d_1 is the delivery, we could create the tasks p_1—p_3—d_1 and p_1—p_2—p_3—d_1, through assuming that p_2 is not essential for the service. The rewards for the new task may then reflect the relative benefit that is gained from the partial service, but in order for solution methods to be able to handle this case, we would require a constraint that prevents these two tasks from both being carried out. We are not considering any of these task compatibility constraints in the version of the MCP that we are solving, so we leave this situation as a theoretical occurrence (nevertheless worthy of future exploration).

Another form of time-dependent reward, may be to only consider the time at which the task is started and the time it is completed. This allows us to consider the *riding time*, which is the length of time the customer's load is in the service vehicle. This then becomes analogous to some of the objective functions used for the DARP. The objective considered by Jaw, Odoni, Psaraftis and Wilson [89] was to minimize the distance travelled plus the customer disutility. For their problems there is a desired time for either the pick-up or delivery, and the disutility is the sum of quadratic functions on the time deviation from this desired time and the excess riding time. Sexton and Bodin [160, 161] similarly defined disutility, except they used linear functions.

We propose a time-dependent reward function for task problems that has linear penalties for the deviation from the desired time of completing a task and for the

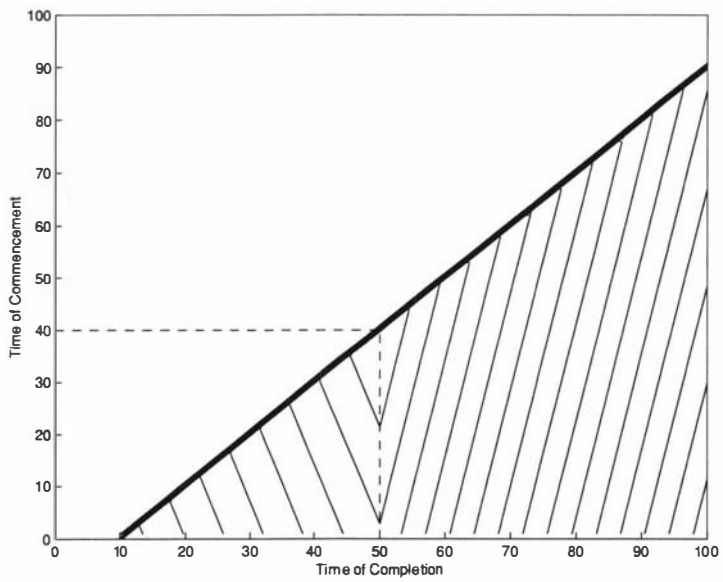

Figure 6.4: Contour Plot of the TASKTDR Function

excess riding time, and we call this reward function TASKTDR. We define $RT(i)$ to be the minimum riding time required for task i and $DT(i)$ to be the desired completion time. This reward function is given by

$$R(i) = R_i - a_i * |f(i) - DT(i)| - b_i * (r(i) - RT(i))$$

where R_i is the maximum reward for customer i, a_i and b_i are the penalties for the deviation from the desired completion time and minimum riding time respectively, and $f(i)$ and $r(i)$ are the actual completion time and riding time respectively. A typical contour plot of rewards for this function, with desired completion time of 50 and direct travel distance of 10, is shown in Figure 6.4. In this case, the maximum reward received is when the task is completed at its desired completion time and the subtasks are carried out consecutively. We may incorporate time windows of desired or allowable times into this model, and we may alter the relative priorities of keeping to the desired time and keeping the duration that the task is in transit, to an acceptable level.

We use this form of reward function as a flexible means of considering the quality of service a customer receives. The riding time penalties can be used to encourage immediate service, where required, or to allow very little consideration of the time that a customer's load is in transit. Therefore high penalties can be incurred for excessive riding time for loads that may deteriorate through being carried longer, as may be the case with the transportation of perishable goods.

Explicit Consideration of Riding Times

In the previous section we considered the use of time dependent rewards with tasks, to promote the favourable service of a customer through restricting the excess time that a customer's load is in the service vehicle. We consider now the effect of including explicit restrictions on the riding time for a customer. These may arise with delivery problems where the load carried is perishable, and there is a limit on the total time that the load may be in a vehicle before it starts to spoil.

We show that, when there are constraints on the riding time for a customer, we may use a form of local search on a vehicle route with the feasibility checks being carried out in constant time. This is an extension of the previous work of Savelsbergh [154, 155, 156, 157], which we believe to be new. We consider the lexicographical search for the 2-exchange as previously discussed in Chapter 2, where we remove the arcs $(i,i+1)$ and $(j,j+1)$ and replace these with (i,j) and $(i+1,j+1)$. For our analysis, we assume that there is an equivalence between the distance and time travelled, so that distance and time can be used interchangeably.

We begin an iteration with $j = i+2$ and proceed through incrementing j. Any exchange will affect the riding time of all customers whose first point is before position i of the vehicle and whose last point is after position i, and all tasks whose first point is after position i and before position j. If a task starts prior to position i and finishes after position j, then the change in the riding time will be the change in the distance of the vehicle, i.e., $\Delta = d_{i,j} + d_{i+1,j+1} - d_{i,i+1} - d_{j,j+1}$. Since we require Δ to be negative for the 2-exchange to create improvements, this allows us to ignore these customers, as their riding time constraints cannot be violated because their riding times must decrease for any improving move. The cases where the riding times may increase, are therefore simply the ones that start or end in the affected path, $i+1 \rightarrow j$. We define the following global variables for use in the solution method:

$$\begin{aligned} AF &= \text{allowable shift in the first point of a task} \\ AL &= \text{allowable shift in the last point of a task} \\ T &= \text{the travel along the path } i{+}1 \rightarrow j \end{aligned}$$

We define R_i to be the current riding time of the task in position i of the vehicle, while MR_i is the limit on the riding time this task is allowed. Let F be the set of first points of tasks, and L be the set of final points. Our search procedure is given

in Algorithm 6.1. We set the allowable changes to be some large number initially. For the point in position $i+1$, we calculate the new riding time, R'_{i+1}, with the allowable change in riding time being $MR_{i+1} - R'_{i+1}$. If $i+1 \in F$ we calculate the allowable change to this riding time, and set AF to be the minimum of the current value for AF and the calculated allowable change. Similar steps are carried out if $i+1 \in L$, with AL being updated in this case.

For each iteration, where we increment j, we begin by calculating the new travel time along the path $i+1 \rightarrow j$, which we use for calculating the change in riding time for the customer in position j. Any first point along the path $i+1 \rightarrow j$ will have its current riding time altered by $\Delta_F = d_{i+1,j+1} - d_{i+1,j}$, while the current riding time of any last point will be altered by $\Delta_L = d_{i,j} - d_{j,j-1} - d_{i,j-1}$. Our requirement, for the precedence constraints to hold, is that if $j \notin F$, then none of its predecessors will be in the path $i+1 \rightarrow j-1$.

We calculate R'_j, the new riding time for the point in position j, which will be different depending on whether $j \in F$ or $j \in L$. If $j \in F$, we find the minimum of the allowable change for point j, i.e., $MR_j - R'_j$, and the new allowable change for first points along the path $i+1 \rightarrow j-1$, i.e., $AF - \Delta_F$. This minimum is the new allowable change to first points along the path $i+1 \rightarrow j$, without violating the riding time constraints. We also update AL, and it will be the old allowable change less the current change, i.e., $AL - \Delta_L$. The updates of AF and AL are similarly carried out if $j \in L$.

Therefore the updates of feasibility checks, require the calculation of four quantities (T, Δ_L, Δ_F and R'_j), and evaluating two quantities (the new values of AL and AF) consisting of a minimization of two numbers and a subtraction. This process can thus be carried out in constant time.

This process is unable to be repeated for Or-opt routines, due to the order preservation properties of the search procedure. Suppose we have the current iteration of the Or-exchange procedure as in the top of Figure 6.5, where we move customer i into position j, and currently we have $j = m$. We need to check that the riding times for each customer of the original path $i \rightarrow j-1$ are still feasible. The problem with our constant time updates arises when we increment j. Suppose that with $j = m$, the minimum riding time leeway is for the customer in position k where $i < k < j$, and $k \in F$. If the last point of the task that k is from is in position m, when we increment j, our current minimum leeway is no longer valid,

Algorithm 6.1 function RIDECONSTANT2EX

```
// Initialization
j ← i + 2, T ← d_{i+1,j}
AL ← LARGE, AF ← LARGE, Best_Change ← 0
if (i + 1 ∈ F) then
    R'_{i+1} ← R_{i+1} + d_{i+1,j+1} − d_{i+1,j} − d_{j,j+1}
    if (MR_{i+1} − R'_{i+1} < AF) then
        AF ← MR_{i+1} − R'_{i+1}
    end
end
else if (i + 1 ∈ L) then
    R'_{i+1} ← R_{i+1} + d_{i,j} + d_{i+1,j} − d_{i,i+1}
    if (MR_{i+1} − R'_{i+1} < AL) then
        AL ← MR_{i+1} − R'_{i+1}
    end
end
if (j ∈ F) then
    R'_j ← R_j + d_{j,i+1} + d_{i+1,j+1} − d_{j,j+1}
    if (MR_j − R'_j < AF) then
        AF ← MR_j − R'_j
    end
end
else if (j ∈ L) then
    R'_j ← R_j + d_{i,j} − d_{i,i+1} − d_{i+1,j}
    if (MR_j − R'_j < AL) then
        AL ← MR_j − R'_j
    end
end
if (AF > 0 and AL > 0) then
    Δ ← d_{i,j} + d_{i+1,j+1} − d_{i,i+1} − d_{j,j+1}
    if (Δ < Best_Change) then   // Store best solution
        Best_Change ← Δ
    end
end
while (AF > 0 and AL > 0) do   // while no riding time constraints have been violated

    j ← j + 1
    T ← T + d_{j−1,j}  Δ_L ← d_{i,j} − d_{j,j−1} − d_{i,j−1}, Δ_F ← d_{i+1,j+1} − d_{i+1,j}
    if (j ∈ F) then
        R'_j ← R_j + T + d_{i+1,j+1} − d_{j,j+1}
    end
    else if (j ∈ L) then
        R'_j ← R_j + d_{i,j} − d_{i,i+1} − T
    end
    if (j ∈ F) then   // Update global variables
        AF ← min(AF − Δ_F, MR_j − R'_j)
        AL ← AL − Δ_L
    end
    else if (j ∈ L) then
        AL ← min(AL − Δ_L, MR_j − R'_j)
        AF ← AF − Δ_F
    end
    if (AF > 0 and AL > 0) then   // Store best solution
        Δ ← d_{i,j} + d_{i+1,j+1} − d_{i,i+1} − d_{j,j+1}
        if (Δ < Best_Change) then
            Best_Change ← Δ
        end
    end
end

end
```

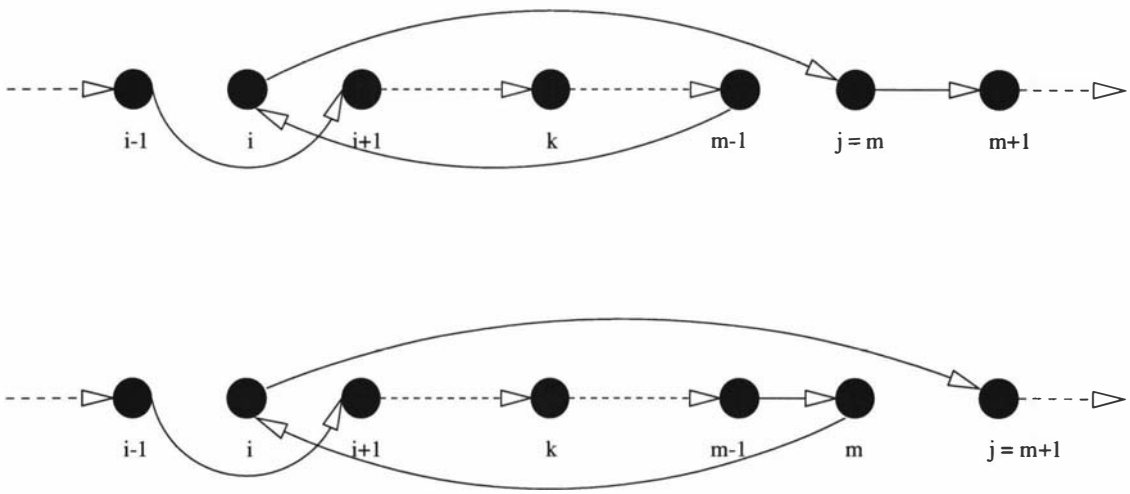

Figure 6.5: An Example Iteration of Or-Exchange

because the riding time for this task will be the same as in the original solution. In this case, we need to revert to the customer with the next lowest leeway, and so in the worst case, we need to preserve and update the leeways for each customer. This means that with this method, the riding time updates cannot be carried out in constant time for the Or-opt exchange. The path reversal featured in the 2-exchange procedure does not allow whole customers to be considered within the same section of route without violating the precedence constraints, and this fact allows the 2-exchange constant time updates to be possible.

Due to the inability to handle these riding time constraints explicitly, we do not consider riding times within our model. We use the implicit constraints of the TASKTDR reward function to restrict the times of servicing, as these will prevent any customer from being serviced when their reward is negative. The riding time constraints would be more relevant for problems where there are large penalties for non-service, as in this case there may be an advantage to servicing a customer at a level below their requirements, since this might prevent the incurrence of the penalty. The explicit constraint on the riding time, which considers the service quality, may then be needed to ensure adequate service is provided.

6.3 Solution Methods

We now consider generic approaches that may be used for the MCP, and we consider the appropriateness of the methods. We use these to create generic solution

methods for the MCP, which we will test on the selected problems in Chapter 8.

6.3.1 Construction Methods

The methods we use for constructing solutions for the MCP, are based upon insertion and cluster-first, route second methods. For insertion methods, as we described in the method INSERTION from Algorithm 2.1, we sequentially build up solutions by adding customers one at a time to the solution. We use various criteria for evaluating the insertion, in order to create solutions that are best suited to particular problem classes. For example, one method we consider is to create solutions through adding the customer that adds the least to the time travelled for a route. This particular method is best suited to problems where the routing is of primary importance, so would appear to best suited to problems with large numbers of customers included, low reward variability and few time restrictions.

One restricted form of INSERTION that we consider, is to append the best customer to the end of the current route. This method reduces the computational effort required for the insertion, as each customer is only considered for inclusion in one position, rather than trying each feasible position. One method that is most suited to subset selection, particularly for appending customers, is the subgravity approach of Golden, Liu and Wang [73]. This approach assesses the attractiveness of a customer through considering all customers that may be included after the customer, with the affect of each other customer diminished according to its distance from the candidate customer. An improved version of this method, which is the one we apply, was proposed by Sokkappa [164]; this method only includes the customers that can be feasibly included after the candidate customer, rather than all unscheduled customers. We consider the insertion time rather than the insertion distance, as this allows us to apply the method appropriately for problems where there are time restrictions and therefore the insertion distance may be much different from the insertion time.

The subgravity measure is most relevant when appending customers to the current tour, as the remaining customers are included after the candidate customer, and therefore the time to include the next customer can be calculated directly. If a customer is added to the interior of a route, then the time to the next customer is more difficult to assess directly. In this case the insertion time should be considered, and the impact of the inclusion of a customer would relate to the change in the best

insertion distance. This approach could potentially become untidy and difficult to implement.

Since the subgravity method considers adding a customer to the end of a vehicle route, for the insertion of tasks, we assume that the customer will be added consecutively and the time to the next customer is the time from the conclusion of the candidate customer to the proposed conclusion of the new customer. Since the new customer is assumed to be serviced consecutively, this method will be less appropriate for task problems where customers are able to be interspersed, as the solution obtained through appending may be far different from preferable solutions which allow the customer service to be mixed.

For our cluster-first, route-second methods, an important phase is the creation of appropriate clusters, and this involves the grouping of customers that are likely to be routed together. Often clustering is carried out through putting customers that are close together, in the same cluster. This approach will create what we call "completely connected" clusters, in which each customer within the cluster is close to each other. One method for creating clusters of this type is to ensure that the furthest distance between customers within a cluster is less than the distance to the nearest customer from another cluster.

Another form of cluster, which we call a "minimally connected" cluster, is one where a customer need only be close to one other customer within the cluster. Johnston [90] defined "stringy" clusters in this manner, where the strings are just paths of customers. Our approach for creating clusters is based upon this approach, as we create sub-routes of customers that can be directly included in a solution. This approach enables the clusters to be of practical use as, with side constraints, having a set of customers that are close to each other will not guarantee the cluster will be of any use, as the constraints may lead to the customers being incompatible when considered together. For example, we previously showed in Figure 5.1 how customers consisting of tasks that are close together may not be close together in a route, as the distance between customers is dependent upon the route involved. We ensure that the clusters created are feasible for the problem instance, by ensuring that the relevant constraints, including route duration, may be met by the cluster considered.

One additional requirement for methods for the MCP may be relevant when there are infinite penalties for non-service. In this case the most important feature

of the problem is to ensure that a solution is obtained that is able to service each of the customers that have these infinite penalties. The first phase of the solution method should then involve the insertion of these customers, with just the routing considered. Methods that are relevant for the TSP (and appropriately constrained versions of the TSP) could then be applied to obtain a feasible solution initially, with other methods that are suited to the MCP then able to be applied.

6.3.2 Improvement Methods

There are a number of improvement techniques that have been applied to vehicle routing problems, and we now consider which of these are appropriate for the MCP. We consider intra-route and inter-route techniques separately. Since the MCP contains time-dependent rewards, our intra-route improvement techniques are able to be used to increase the reward of a route. If the rewards are all constant, intra-route improvement can only be used to reduce the route duration and/or distance.

Intra-Route Improvement

For vehicle routing problems, one technique that has commonly been used to reduce the distance travelled is the k-opt method of Lin [113] and Lin and Kernighan [114]. A k-exchange involves k arcs being removed from a solution, and replaced by k others so as to reconnect the tour. Any solution that can not be improved by use of a k-exchange is said to be k-optimal, and the process of k-opt involves using k-exchanges to create a solution that is k-optimal. Because of the computational complexity involved, most researchers have restricted the use of k-opt to having $k \leq 3$, with many only using the case of $k = 2$. With $k = 2$ we have the situation previously shown in Figure 2.2, which involves the removal of arcs $i \rightarrow i+1$ and $j \rightarrow j+1$, with these replaced by $i \rightarrow j$ and $i+1 \rightarrow j+1$ and with the sub-path from $i+1 \longrightarrow j$ being reversed. The major issue with the applicability of this technique to the MCP is with the sub-path reversal, and the effect this has on problems containing precedences. Whenever there is a subtask and its successor on this sub-path, the 2-exchange will be infeasible. Thus if each task is defined so that the subtasks must be serviced consecutively, then the 2-exchange technique will be unable to identify any feasible exchanges. As shown earlier in Chapters 4 and 5, the techniques of Savelsbergh [154, 155, 157] may be used to carry out the feasibility checks on precedence and time window constraints in constant time. If

we have time-dependent rewards, we are unable to fully utilize these computational savings as, in order to evaluate the effect of an exchange, we need to calculate the time at which each customer is serviced in the new route. Thus the feasibility checks only save us from evaluating an infeasible solution. We use 2-exchanges in our intra-route improvement techniques, as they have been so commonly used and found to be quite effective for a number of vehicle routing applications, including subset selection (see [164] and [20]).

Because of the limited applicability of 2-exchanges with certain forms of problems, it is necessary to explore 3-exchanges. With the removal of three arcs, there are eight ways to feasibly reconnect the tour. One of these involves maintaining the same path and three others involve moves that may be carried out using a 2-exchange. Of the remaining four possibilities, two involve the reversal of one sub-path, another involves the reversal of two sub-paths and the fourth involves preserving the orientation of the sub-paths but moving the position of a sub-path within the route. The improvement technique Or-opt [129] involves the moving of a sub-path consisting of successively 1, 2 or 3 points, until no more moves of this type may be made (a k-Or exchange involves the moving of k consecutive points with $k = 1$, 2 or 3). Or-opt therefore be seen to be a restricted version of a 3-exchange, involving just the case where the orientations of the affected sub-paths are preserved. This technique is likely to be more appropriate than the 2-exchange for problems involving precedence constraints. There are certain problems, e.g., for problems where all tasks consist of more than three subtasks and these must be serviced consecutively, where the Or-opt technique will not be applicable, in which case we propose the use of a Generalized Or-exchange, whereby a sub-path containing 4 or more points is moved within the route. The constant time update techniques of Savelsbergh [154, 155, 157] may again be used to efficiently carry out feasibility checks, with moving the sub-paths forward and backward requiring different feasibility checks. Since we are only considering tasks of length up to 3, we only use these standard Or-exchanges in our intra-route improvement techniques, although with larger tasks we would recommend the use of a Generalized Or-exchange.

k-exchanges with $k \geq 4$ are not commonly used in vehicle routing research, due to the large computational effort required. Byrne [24] used a restricted version of a 4-exchange, where two sub-paths are exchanged within the same vehicle. This

is a potentially useful technique for precedence constrained problems, but we do not use it in our solution method, due to the computational expense involved (this method is $O(n^4)$).

The Generalized Insertion procedure (GENI) of Gendreau, Hertz and Laporte [60] is another routine that may be used to improve upon solutions obtained. The major advantage of this routine is the ability to carry out improvements to a current route while inserting customers into this route. These improvements rely heavily upon the use of path reversals, but these make it difficult to apply GENI to many constrained problems. Gendreau, Hertz, Laporte and Stan [62] adapted GENI for problems with time windows, to create a routine known as GENITW. GENITW adapts GENI by including the customers in the order of their difficulty, rather than in random order as with GENI, and by adapting the neighbourhoods to include both temporal and spatial closeness. The solution can also only be considered in the one orientation, rather than being in either direction.

We feel that the techniques of GENI and US are inappropriate for problems that contain precedences, as the path reversals required are unlikely to allow feasible moves to be made. The insertion process employed is unable to take advantage of the constant time updates, for the precedences and also the time windows, as is the case with the lexicographical search and so the feasibility checks will increase the computational requirements quite significantly. Also we feel that the wide-ranging effects of the insertion on the time of servicing a number of customers affects the solution (thus the time required for evaluating the solution) too much to be useful for problems with time-dependent rewards. We therefore do not employ GENI or US in our solution methods for the MCP.

For each of the above techniques that we implement, we carry out the feasibility checks differently depending on the constraints present. For example, for problems involving just time constraints, we only check that the time windows aren't violated, whereas we only consider the precedence and load constraints when we have a task problem with no time constraints. We also implement the different routines for the cases where we are just reducing the distance travelled and the cases where we are attempting to improve the reward received, and this enables increased efficiency in our routines.

An important consideration in the implementation of the selected routines is how the form of the problem affects the manner in which our solution techniques

are applied. We previously showed that, with time window constraints, we are able to carry out the updates of feasibility checks for 2-exchange and Or-opt in constant time. One of the assumptions that is used to implicitly check the constraints for a number of customers, is that the specified move will not increase the time of servicing any of the subsequent customers of the route. This is because any improving move must decrease the time at which the last of the affected points is serviced. However, with time-dependent rewards and intra-route improvement, we do not require that the time of the solution route is reduced. We allow moves that increase the time for the route as long as the overall reward for the vehicle is increased. This means that when we have time-dependent rewards and time windows we can not use the constant time updates. In this case, we use the reward functions to check feasibility, as we set the reward to be a large negative number for service outside the customer's time window. We stop an iteration of Or-exchange if one of the moved customers is placed in a position where its time window is violated, and we stop 2-exchange if one of the customers along the path that is reversed is placed after its time window. In these cases it isn't possible to return to feasibility in further iterations, so we restart through looking to move a different set of customers. A Tabu Search framework could possibly be developed to cope with the consequences of allowing infeasibility.

Inter-Route Improvement

With our inter-route improvement techniques we are interested in the trade-off between the effectiveness of the techniques and the computational effort required. One of the standard techniques that is often used, is the inter-route Move routine, which moves a customer from one vehicle to another, as we previously illustrated in Figure 2.4. We move the whole of a task, which ensures that each customer is serviced by the one vehicle, as is required when the tasks are prohibited from being split.

Our search process for the Move routine involves finding the best positions in each other vehicle, in which to place the subtasks of the moved customer, and we carry out the move that makes the best change to the solution. Our method of finding the best positions to insert the customer, involves incrementing the position of the last subtask that can be increased and, because of this lexicographical insertion, we are able to check the feasibility of the insertion in constant time. This

Move routine is potentially able to be very effective for the MCP, as it involves servicing an additional customer within the affected route. As it is the simplest routine that alters the composition of the routes, we see Move as an important routine for any improvement heuristic. Move is ineffective when the routes of the vehicles are full, with respect to the capacity or time constraints, and in these cases other inter-route improvement techniques are required.

Toth and Vigo [175] used another customer move routine, which they call Double Insertion, for improving solutions for the DARP. This routine involves removing a customer, i, from its route and placing it in its best positions in another route. A customer is removed from a third route and is placed in its best positions in the route that customer i was originally in. This routine has two moves being carried out at the same time, and it is able to create solutions that would not be found through carrying out the move routine separately, due to the intermediate solution being unacceptable. Double Insertion was found to be effective for the DARP, and it also appears to be suitable for the MCP.

One important situation that Double Insertion can be used for is shown in Figure 6.6. Here the vehicle capacity is 5, and the task definition is placed next to the point and is given as (*task*, *load*). We assume that there are constant rewards which are the same for each customer (we arbitrarily set these equal to 1), with the best two vehicle's rewards counting in the objective function. With the solution on the left, there is no improving inter-route Move possible, so the total reward obtained is 3. By employing Double Insertion, we can move customer B into the route containing customer C and then move customer D into the route that customer B came from. This reduces the number of vehicle routes required, and we enables all the customers to be included within the solution routes, giving a total reward of 4. This routine is one that appears to be quite useful for problems where the loads in the service vehicles are close to the available capacity.

Another technique that is frequently used is the inter-route Exchange routine, which exchanges customers between two vehicles. This routine was shown previously in Figure 2.5 and, in order to carry out this routine, we provisionally remove the two customers and find the best insertion positions within the provisional routes created through removing the customers. The identification of insertion positions is carried out in the same manner as with our move routine, so again we can check feasibility in constant time. Exchange is an important method for cases where the

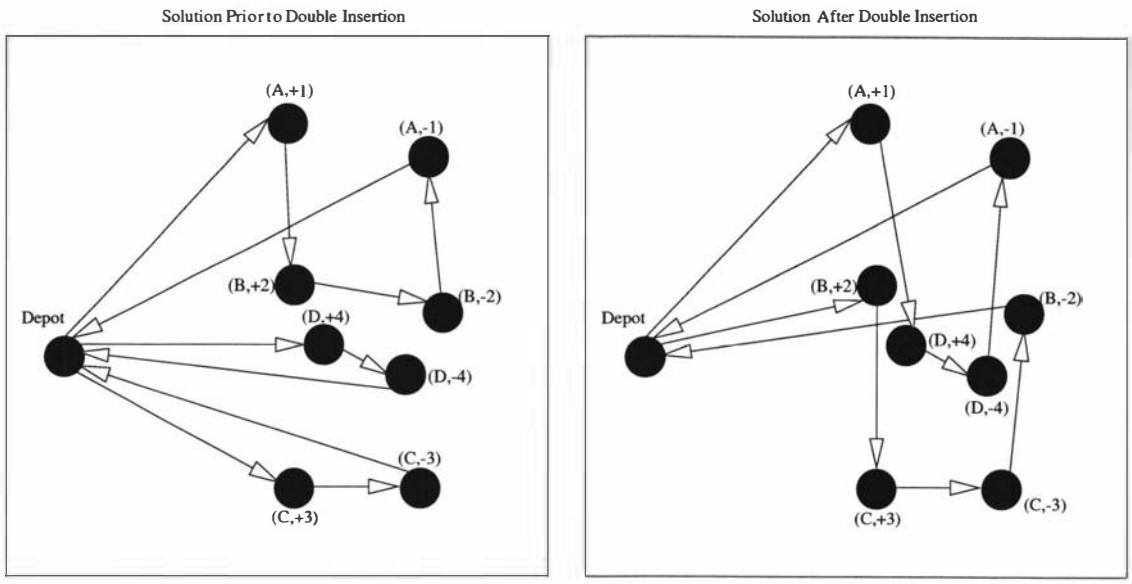

Figure 6.6: An Improvement found through Double Insert

vehicles are 'full' with respect to capacity or route duration constraints (in which case inter-route Move will be ineffective), as it removes a customer before attempting to add another. Therefore a method that involves exchanging customers seems to be a necessary part of a solution method, particularly where feasibility constraints play an important role.

One new technique that may be useful, then, is to combine the Double Insert and Exchange routines. We remove a customer, move a customer from another route and then reinsert the removed customer in its best positions in any vehicle. We call this routine Generalized Exchange, as it has Exchange as a special case, i.e., where the removed customer is placed in the route the moved customer was from. A case where this may be useful is shown for the single vehicle problem of Figure 6.7, where the task definition is placed above the point and is given as (*task, load*). Here, customer C is unable to be serviced in the solution vehicle, and so we would place it in its own route, with the customer unable to be inserted elsewhere due to the capacity constraints. The solution on the right is one that has been improved by this Generalized Exchange, where customer B is removed from the service vehicle, customer C is inserted into the vehicle and then customer B is repositioned within the route. Since this routine incorporates both Exchange and Double Insert, which are useful techniques, we will use it instead of explicitly carrying out these techniques separately, within our solution routine for the MCP. The additional choice of which vehicle to position the moved customer will increase

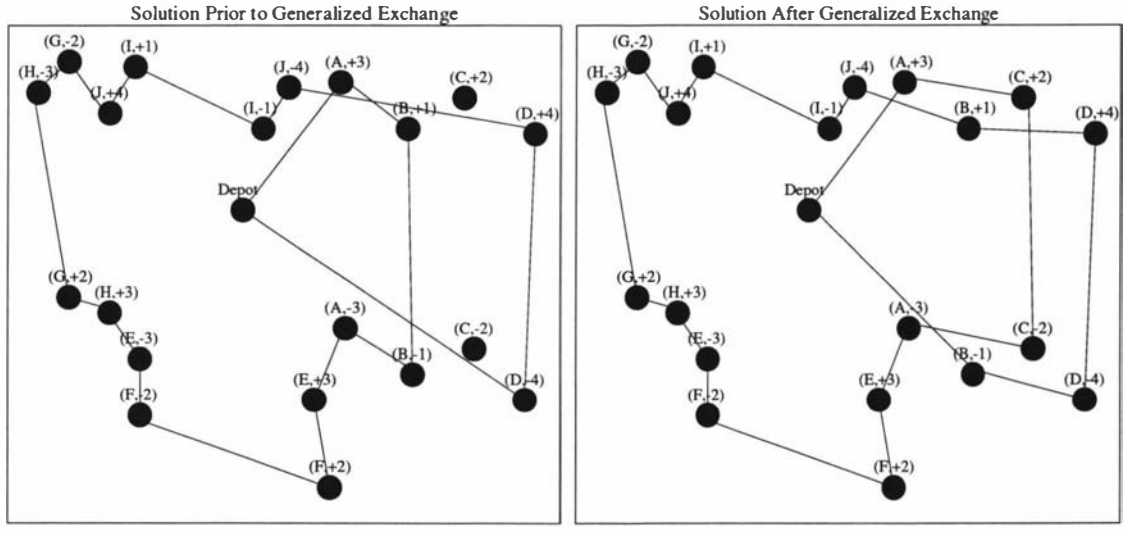

Figure 6.7: A Generalized Double Insert

the computational time required, but we believe this is acceptable, given the extra versatility of the routine.

The Generalized Exchange routine seems to be particularly well suited to problems involving subset selection routing. It enables a customer to be added to a current solution route, while carrying out a local improvement step (removing a customer and reassessing the best positions for insertion, in the presence of the newly inserted customer). The three route exchange enables a customer to be inserted into a route, with the removed customer able to be inserted in a different route, and this enables a swap move to be performed even though the removed customer is incompatible with the route that the inserted customer was from.

Solution methods that swap sub-paths between vehicles are also popular inter-route improvement techniques for vehicle routing problems. One method that has been found to be effective, particularly for time window problems is the 2-opt* method of Potvin and Rousseau [87]. This is a generalization of 2-opt, as the arcs that are removed are replaced by arcs that link customers from different routes. An example of the moves used in 2-opt* is shown in Figure 6.8. The illustration shows two routes with the depot, denoted by D, given twice. This routine preserves the orientation of the sub-paths and adds the end of the first route onto the end of the second, while putting the end of the second route at the end of the first route. This is particularly relevant in problems where there are time windows, since this routine preserves the approximate times of servicing.

A more general sub-path exchange heuristic is the Cross exchange of Taillard,

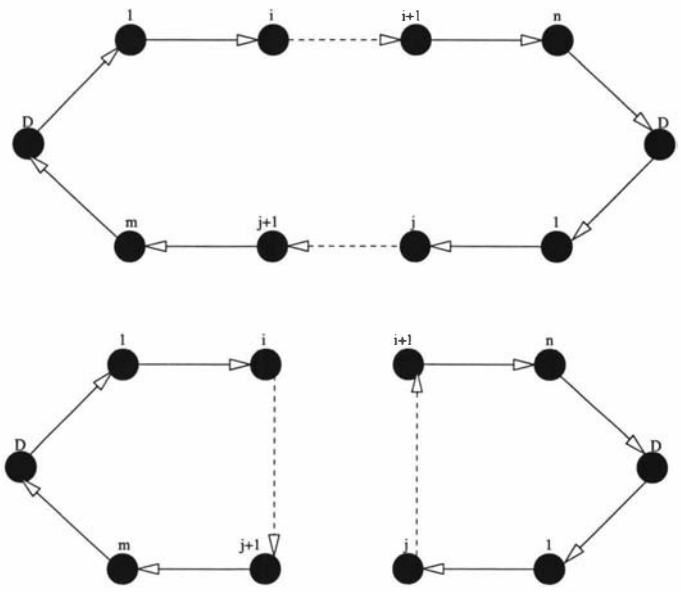

Figure 6.8: An example of a Move Used by 2-opt*

Badeau, Gendreau, Guertin and Potvin [171], which was previously illustrated in Figure 2.6. The Cross exchange is a very general technique, that takes sub-paths of various lengths from different vehicles and exchanges them. To exchange two sub-paths, Cross removes four arcs from within the two affected routes, and replaces them with four other arcs, which link the two routes, and so Cross is a restricted version of a 4-exchange. Another possible Cross exchange is to move a sub-path from one route to the other, where three arcs are removed from within the routes and are replaced by three others, with two of these linking the routes. If we restrict the number of points in a sub-path that may be moved to at most l, then the complexity of the routine is $O(n^2l^2)$, and without this restriction, the worst case complexity is $O(n^4)$.

The Cross exchange incorporates a number of other inter-route improvement techniques. If the customers consist of single points, then the inter-route Move routine is a subset of Cross, where the sub-path moved consists of a single point (inter-route Exchange is not a subset of cross as we place the customers in their best positions, rather than swapping the positions directly as is the case with Cross). Cross can also incorporate a generalized inter-route Or-exchange, where a path is moved from one route to another. The 2-opt* routine is also a special case of Cross, where the last points of the moved sub-paths are the last points of the routes.

Because of the wide-ranging scope of the Cross routine, we choose to use it as part of our improvement routine for the MCP. In most cases, we use Cross

instead of the routines that it incorporates, i.e., inter-route point Move and 2-opt*, although there are some shortcomings to the routine. The computational effort required for Cross can become excessive, compared with the Move and Exchange routines, particularly for problems where the customers consist of just points, due to the number of feasible cross exchanges. For task problems, it may be difficult to find any stand-alone sub-paths, particularly for solutions where there is a large separation in the positions of the consecutive subtasks of a task, and this will render the Cross routine ineffective.

A more general path exchange for vehicle routing problems is the method of Fahrion and Wrede [47]. They determine the position to insert a chain containing a given number of points, such that the distance change is minimized. Two chains are moved into their best insertion positions, if this creates an improved solution. This process involves similar changes to the solution as Cross, except the sub-paths are placed in their best position, which involves additional computational effort. We therefore do not include this routine in our methods.

An even more general approach for exchanging customers between routes is the λ-exchange of Osman [130]. This routine involves removing subsets of at most λ customers from two different routes, and placing them in the other affected route. This is a generalization of the Cross routine, where the moved customers are not required to be currently placed consecutively within the route, nor do they need to be placed consecutively in the new solution. This routine incorporates all the above inter-route routines that affect two vehicles, but due to the computational complexity, we choose to restrict ourselves to exchanging consecutive paths, as is the case with Cross.

Another very general inter-route improvement routine is the cyclic transfer algorithm of Thompson and Psaraftis [174], where sets of customers are transferred between the vehicle routes according to a cyclical pattern. This method is not even restricted to affecting two solution routes, and we again omit this method from our improvement routines due to its increased difficulty of implementation and computational complexity involved.

6.3.3 Including Non-Compulsory Waiting Times

One important consideration with the MCP is how to improve solutions where the reward may be increased through waiting to service a customer. This case

involves developing a schedule where the time of servicing is a decision variable. One technique that has been developed for this type of problem is the method of Dumas, Soumis and Desrosiers [43], which obtains the optimal time for servicing each customer within a fixed route. This method requires us to be able to calculate the time, within limits, to minimize the sum of a number of convex functions. Their paper describes how this optimization may be carried out with linear and quadratic functions. Our problem involves the maximization of a sum of reward functions that may be concave, and we have previously used this method for quadratic reward functions in Chapter 4.

We consider different reward functions from those used by Dumas et al. [43], and adapt their method so it can remain effective. For problems where the reward function is piecewise linear, as in Figure 6.2, the sum of the functions is also piecewise linear. In order to find the optimal solution over the sum of these functions (an important step of the optimization algorithm), we solve through searching the corner points of this function. We need to consider in this case that the optimal time of servicing a customer may be an interval, rather than a single time. Therefore we calculate the optimal interval during which to service each customer, and in our calculations we define $chosen_i$ to be the time at which we select to service customer i. This time is the earliest time within the optimal interval at which we may service the customer. We recalculate the optimal times, and thus the selected times, if the chosen time plus the travel time to the next customer exceeds the latest time of the optimal time interval.

If our customers consist of tasks and the reward function is dependent only upon the time at which the task is completed, then this technique is easily adapted, as we demonstrate for the solution shown in Figure 6.9. The original path consists of three customers, each consisting of an origin and a destination, e.g., in the figure, task A is given by the pair (A^+, A^-). If the are no time constraints apart from at the destination point of the task, then we reduce the path to consisting only of the destinations of each task, as the other subtasks can be serviced as soon as their location is visited, without affecting the objective function, and so their time of servicing can be obtained directly from the time of servicing the previous points. The travel times between these destinations are taken to be the travel times along the original path, i.e., travel time from 0 to B^- consists of the sum of the travel times along the arcs $(0, A^+)$, (A^+, B^+) and (B^+, B^-). With this reduced path and

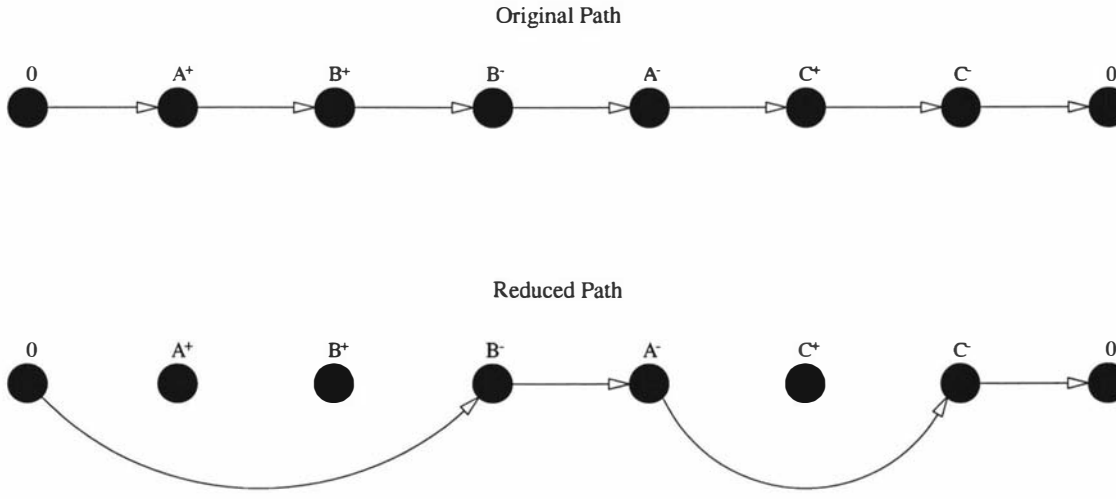

Figure 6.9: Path Reduction Employed for Waiting Time Inclusion

the new travel times, we calculate the optimal service times as before.

If there are time windows on some of the points of a task, then we can not carry out the above reduction, as the time of servicing each of the subtasks needs to be explicitly considered. In this case we need to consider the time of servicing any subtask that has time restrictions, although the optimal time of servicing needs to be calculated only for the last point of each task.

For our TASKTDR time dependent reward function, we have adapted the technique of Dumas et al. [43] so that this technique can be used to find the optimal times to service the customers. Our first step is to identify the positions in which it is potentially beneficial to delay the customer's service. Since we are penalizing for additional riding times, we consider that a destination which is included at the end of a sequence of non-destination points cannot be delayed, as we would be better served to delay the time of servicing the preceding points. Therefore, we consider the service times for any points that immediately succeed the final point of a customer, within the current route. We illustrate this in Figure 6.10, where our path consists of four (origin,destination) pairs, with task i given by the pair (i^+, i^-). The eligible points at which we may include waiting time are circled. The cases where customers C^- and B^- are eligible to include non-compulsory waiting time are when their time windows are not compatible with the selected service time of the previous point, and when the increased reward for delaying the service exceeds the additional penalty for the excessive riding times.

The first difference between this approach and that previously used, is that

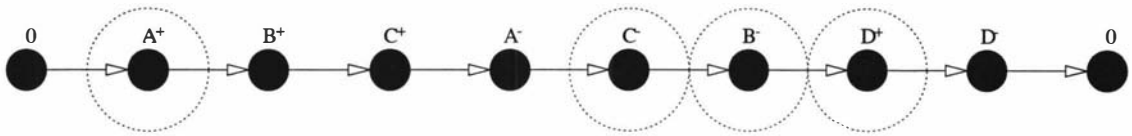

Figure 6.10: Eligible Positions for Waiting Time Insertion

we are calculating the service time at a point which need not be the destination of a task, and we are using this to find the optimal time to service the following customer (or customers). Therefore we need to keep track of the problem on two different levels: the first being the level of the candidate points at which the service time is a decision variable, and the second being the points which are needed in order to evaluate the effect of the service times. We need to store the travel times on each of these restricted problems. Referring again to Figure 6.10, we create a restricted problem, containing the route $0—A^+—A^-—C^-—B^-—D^+—D^-—0$, for evaluation purposes with the route $0—A^+—C^-—B^-—D^+—0$ including the candidate customers.

The other major difference between this and the previous case, is in the identification of the optimal times of servicing. Previously, we just needed to identify the optimal time within the available time window. In our current problem we need to find the trade-off between additional reward and additional riding times. For this purpose we store a value $open_i$, which is the sum of the riding time penalties of any customer that has had its service partially completed, when we service the i'th candidate point. We calculate the optimal time for servicing the point by maximizing the reward less the penalties for excessive riding time, which is the additional delay multiplied by $open_i$. We illustrate this technique with an example, which again refers to Figure 6.10. Suppose that the travel times along this route are: 1.0, 0.5, 0.8, 0.4, 0.6, 0.4, 0.3, 0.5 and 0.8, while the direct travelling times for tasks A, B, C and D are 1.1, 1.4, 0.8 and 0.5 respectively. There is a time window on the time of arrival, Arr_i, at the final point of each task i, and the reward functions are given by:

$$R(A) = 10 - 3 * |2.8 - Arr_A| - 1 * (ride_A - 1.1), 2.3 \leq Arr_A \leq 3.2$$

$$R(B) = 12 - 9 * |3.9 - Arr_B| - 2 * (ride_B - 1.4), 3.6 \leq Arr_B \leq 4.2$$

$$R(C) = 15 - 5 * |3.4 - Arr_C| - 6 * (ride_C - 0.8), 3.1 \leq Arr_C \leq 3.6$$

$$R(D) = 11 - 8 * |5.1 - Arr_D| - 4 * (ride_D - 0.5), 4.7 \leq Arr_D \leq 5.5$$

where $ride_i$ is the time that customer i is in the vehicle. The time limit is equal to 6.5. Our initial solution has all customers serviced as soon as possible, so we have the schedule:

$$A^+(1.0)-B^+(1.5)-C^+(2.3)-A^-(2.7)-C^-(3.3)-B^-(3.7)-D^+(4.0)-D^-(4.5,.2)$$

where the arrival times and the waiting times, where required, are given. The total reward for this solution is 38.8. The reduced network for evaluation is 0—A^+—A^-—C^-—B^-—D^+—D^-—0, which has travel times of (1.0, 1.7, 0.6, 0.4, 0.3, 0.5, 0.8). The reduced route for our service time calculations is 0—A^+—C^-—B^-—D^+—0, with travel times of (1.0, 2.3, 0.4, 0.3, 1.3), and the time windows of these points are (0.0,0.7), (1.0,1.7), (3.3,3.6), (3.7,4.2), (4.0,5.5) and (5.5,6.3) respectively.

The initial optimal times for servicing the points are 1.1 for A^+ (so that A^- will be served when its maximum reward is obtained), 3.3 for C^- (in order to limit the riding time penalties, since the riding time penalties exceed the slope of the reward for C), 3.9 for B^- (so its maximum reward will be obtained) and 4.6 for D^+ (so D^- can be served where its maximum reward is obtained).

Our search then finds that the optimal time for A^+ (1.1) plus the travel from A^+ to C^- (2.3) exceeds the optimal time for C^- (3.3), so we link A^+ to C^-. The new optimal time for A^+, considering the times of the points up to C^-, is 1.1. We then check and find that this is our optimal solution, so the schedule obtained is:

$$A^+(1.0,.1)-B^+(1.6)-C^+(2.4)-A^-(2.8)-C^-(3.4)-B^-(3.8,.1)-D^+(4.2,.4)-D^-(5.1)$$

and this has a total reward of 44.4.

This method in its current form, is only applicable for problems with linear functions for the rewards and the riding time penalties. In order to incorporate non-linear penalties for excessive riding time, we would need to keep track of the exact time of servicing. With linear functions, the rate of change is constant regardless of the time at which the service takes place, so our current calculations just identify the change in the riding times with the change in penalties being given by this change multiplied by the sum of the penalties. With non-linear rewards, the effect of the riding time penalties depends upon the length of the riding time, so we would need to store the proposed starting time for each customer, in order to correctly evaluate the impact.

Solving for the optimal service times within a given route would be much simpler if we did not allow there to be any waiting time while a customer is in the vehicle.

With this restriction, we would just have to consider the time of servicing the first point of a block of customers. We would still need to consider the time of servicing the final point of each of the customers within the block, when calculating the optimal times.

For more general functions which are less tractable, a possible approach is to estimate the best waiting times to include. To do this, we identify the maximum waiting time that may be included at a particular point, and calculate the effect of inserting feasible waiting times of various lengths, at this point. Our calculation involves pushing back the time of service at a customer, and identifying the flow-on effect through the route. Thus we service each subsequent customer as soon after the current service time as we can. This is an estimate, as we are only inserting the waiting time at one customer at a time, with any changes at subsequent customers being as a result of this change. We include the waiting time that increases the reward the most, until no further waiting times will increase the total reward of the vehicle's route. The advantage of this technique is that it is able to be used with any reward function, although with less smooth functions, and to obtain better results, we would require a greater number of estimates.

We implement this waiting time insertion as a post-improvement step. Therefore, once we have obtained a solution through our generation and improvement steps, we use this waiting time insertion method to improve the solution. We only carry out this step on routes that haven't previously had this waiting time applied to them, and we store the best solution, including waiting times, for each route. If we are continuing to use the solution, we remove the additional waiting times and continue with the solution obtained.

The insertion of waiting time introduces another issue, that of the form of the solution. With problems where each customer is serviced as soon as feasibly possible within the route, there is no benefit from including non-compulsory waiting times, and therefore the identification of the schedule is trivial, for a given route. When we can obtain improved solutions through delaying the service of some of the customers, we need to store both the route and the times of servicing, as the specification of the route alone is insufficient for describing the solution.

Algorithm 6.2 function STEEPEST ASCENT

```
finished ← 0
while (finished = 0) do
    go_on ← 0
    if (all rewards are constant) then
        Distance Reduction
    else
        Intra Route Improvement
    end

    Carry out Inter-Route Move
    if (an improved solution is found) then
        go_on ← 1
    end
    Carry out Inter-Route Generalized Exchange
    if (an improved solution is found) then
        go_on ← 1
    end
    Carry out Cross exchange
    if (an improved solution is found) then
        go_on ← 1
    end
    finished ← 1 − go_on
end
Carry out optimal waiting time inclusion

end
```

6.3.4 Implementing the Improvements

We use two methods of implementing the improvements, with these based on a steepest ascent approach and Tabu Search. With steepest ascent we carry out the move of a particular type that makes the largest improvement to the current solution. We use the routines that we selected in Sections 6.3.2 and 6.3.2, and create an overall improvement routine that we call STEEPEST ASCENT, which is outlined in Algorithm 6.2.

We carry out the intra-route improvements in the order 3-Or opt, 2-Or opt, 1-Or opt, 2-opt, as this proved to be the most effective in our preliminary investigation of task problems. The improvement routine of van der Bruggen, Lenstra and Schuur [180], involved the routines being applied in the reverse order of what we

Algorithm 6.3 function RANDOMCONSTRUCT

```
    while (There are still customers unplaced) do
        Randomly select an unplaced customer
        Place selected customer in best position in first feasible route
        if (no route exists) then
            Place selected customer in a new route
        end
    end

end
```

are doing, as they claimed that this order would reduce the computational effort required as the most effective routines would be called the most frequently. We found that with task problems, more effective solutions were found if we moved the larger sections of route earlier in our search procedure, and then use the more fine-tuning moves later. This enabled more of the structure of the solutions to be preserved, as we are able to move sections of the route that we designated to be placed together in the construction phase, rather than having the movement of single points eliminating the original structure. At the end of our improvement routine, we apply our routine that finds the optimal waiting time to include in the solution. This routine is only considered when the solution is amenable to such improvements, i.e., when there are time-dependent rewards that are not strictly non-increasing.

During the investigation of our solution methods, we consider different forms of the STEEPEST ASCENT routine, through varying the inter-route improvement routines that we use. This enables us to obtain routines that are more relevant to particular problem types, rather than the generic solution method that we have described here.

Our Tabu Search heuristic is based on the routine of Taillard, Badeau, Gendreau, Guertin and Potvin[171]. Their method starts by randomly creating solutions, and applying the search, using Cross only. Intensification in the form of within route improvements and diversification through penalizing frequently executed exchanges were also used. The improved solutions are placed into an adaptive memory, and the heuristic is restarted through generating new solutions from the routes that have been stored in the memory.

In order to apply our Tabu Search algorithm, we need to first define our randomized construction method, RANDOMCONSTRUCT, which we describe in Algorithm 6.3. In this method, which we use for a number of different problem types, we randomly select the order for inserting the customers and place them in their best position in the first route in which the insertion may be feasible. In this manner, we are able to obtain many different starting solutions, which may not necessarily be of good quality.

The Tabu Search paradigm that we are using is given in Algorithm 6.4. The TABUIMP method continues through the application of selected improvement routines, as variations of the STEEPEST ASCENT method, to these solutions. The solutions obtained are stored in our memory, for using later to start our search. We randomly select the routes to create a new solution, with the probability of selecting route i being proportional to R_i, the reward for the route, divided by $f_i + 1$, where f_i is the frequency with which the route has previously been used to start our search phase. We ensure all customers are included in the solution, through randomly inserting any customers left over once all candidate routes have been included.

After initializing our adaptive memory we carry out Tabu Search through randomly selecting which routine (Move, Generalized Exchange or Cross) to use, according to the previous success we have had with this routine on this particular problem. This step enables us to have a number of routines available (as we have seen that none of the routines by themselves are always applicable for each problem type), and to select the routines that are most likely to allow us to improve the solution further. The tabu status we consider relates to the complete solution obtained, and we ensure that we don't repeat the same complete solution within the search. We do this by storing $\sum_i R_i d_i$, where d_i is the distance of the route. Solutions with the exact sum of the product of reward and distance are made tabu for the duration of the search, thus preventing us returning to a solution we have found previously. The diversification that we use is our restart mechanism, while we intensify the solution through carrying out Steepest Ascent any time we find a solution with reward that exceeds the current best.

In order to calculate the tabu status of each proposed solution, we calculate the new reward and distance of the provisional solution routes. We observe that our exchange routines involve removing both of the selected customers from their

Algorithm 6.4 function TABUIMP

```
// Initialize the solution
overall_best_solution ← −∞
for initial_solutions = 1...5 do
    Create initial solution using RANDOMCONSTRUCT
    Improve solution using STEEPEST ASCENT
    // Intensify the solution through reducing the number of routes
    Carry out Move, allowing changes that don't affect the solution
    for each solution route do
        if (the solution route is not in the adaptive memory) then
            Add solution route to our adaptive memory
        end
    end
end
// Create a new solution through combining previous solutions
for search_solutions = 1...10 do
    current_best_solution ← −∞
    Create a solution through selecting a route from the adaptive memory,
    with probability proportional to R(i)/f_i
    // Carry out our Tabu Search Phase
    for num_search_iterations = 1...50 do
        Randomly select an improvement routine,
        with probability proportional to previous success
        Carry out best non-tabu move for the routine
        if (solution_reward > current_best_solution) then
            Carry out STEEPEST ASCENT to improve the solution further
            num_search_iterations ← 0
        end
    end
    if (a new solution route is found) then
        Add solution route to our adaptive memory
    end

end
return(overall_best_solution)

end
```

routes, inserting the second customer into its best positions within the first route and then inserting the first customer into the desired route. Thus, we fix the first route, after removal and reinsertion, before attempting to place the first customer, so the evaluation occurs sequentially, and may not permit some feasible exchanges. We do not allow the insertion positions of the second customer to be varied, so the tabu status may prevent some valid cases, e.g., where the best positions for the second customer are the postions the customer has previously occupied during the search. If we were to not carry out the search sequentially, then there would be a large increase in the computational time requirement of the exchange routines, as we would be calculating the best insertion positions for the first customer for each feasible insertion positions found for the second customer. A better method may be to store the previous positions that a customer has held, and preventing the reinsertion in these positions, although this would require much additional storage and need to be updated as the routes are updated throughout the search. We do not include such a feature in our Tabu Search, but we just note that there are some restrictions that need to be placed upon the exchange routines in order to maintain the computational requirements at manageable levels.

6.4 Revisiting Models for Non-Service

We now reconsider the models of subset selection proposed in Chapter 3, and consider how these may be modelled within the complete MCP that we have defined. We consider some cases of restricted subset selection problems, where we defined a restricted subset selection problem to be one where the customers that are not included in the solution routes are treated in a different manner within the objective function. We create examples of restricted subset selection problems through incorporating time-dependent rewards into our problems.

We propose an instance where there are a number of customers who require service in the following time period, where service consists of visiting a single location. Each customer has a reward that is received for it being serviced effectively, and only those customers that are serviced to their requirements are considered to be included in the solution routes. We assume that there is a time period during which the service may take place, and there is an additional time period during which the remaining customers may be serviced.

This situation creates a form of time-dependent reward, with a reward function of the form shown in Figure 6.4. The customer's reward is able to be collected if service is completed prior to $L1$, with no reward received after this time. If service takes place after time $L2$, then a large penalty is incurred. This problem involves subset selection, but all customers must be considered when we create the solution route, and so it is an example of a restricted subset selection problem. If $L2$ is sufficiently large, this problem becomes a path version of the OP (or TOP in the multiple vehicle case), as the location of the service vehicle at the end of the time period over which reward may be collected, i.e., at $L1$, is not fixed. If $L2$ is not large enough to enable all customers to be serviced, then we have a multi-objective routing problem, where the principal objective is to maximize the number of customers that are serviced prior to $L2$, and the secondary objective is to maximize the total reward received up to $L1$. By varying the value of $L2$, we obtain problems where different priorities of routing and selection are present.

Also possible within this approach, is consideration of the situation where the time limit $L2$ is large, but there is a cost incurred which is an increasing function of the time required to service all the customers. The objective in this case would be to maximize the net reward of servicing, where reward is only received until $L1$. The priorities of routing and selection can be varied, in this case, through altering the costs of the time required by the service vehicle.

An extension of this model is for each customer to have a desired time window for servicing, where this time window is a subset of the time period $[0, L1]$. The reward for the customer is only received if service takes place during this time window. This case may create a similar problem as with no time windows, except with restrictions on the time of servicing each customer. Another possible case is where we create the model SUBSETOPTW, which occurs where $L1 = L2$ and all customers are required to be serviced in a tour that must be finished prior to $L1$.

With this model, our predominant concern is with the case where all customers are able to be serviced within the overall time limit (otherwise we have a multiobjective problem, where the main objective consists of solving a version of the OP where the objective is to maximize the number of customers that are able to be serviced up until the time limit). With all customers able to be serviced, every customer is required to be included in the solution route, with the subset selection component of the problem relating to which customers to schedule within their

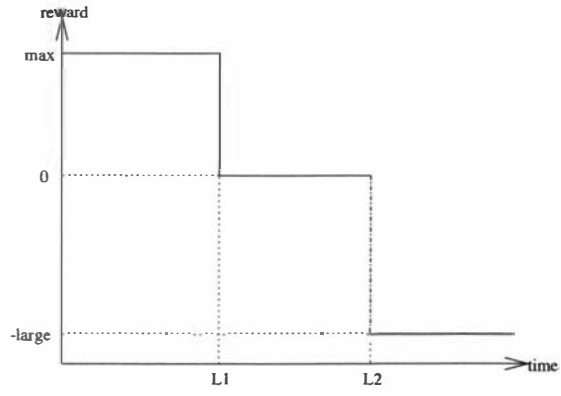

Figure 6.11: Subset Selection Modelled as a TDR Function

time window. This guarantees service of some kind for all customers, with only customers serviced within their desired windows being beneficial.

This model may also incorporate time-dependent rewards, where all customers are required to be serviced, although not all can be serviced while their reward is positive (the minimum reward is assumed to be equal to zero). This is the situation for which the solution methods of Erkut and Zhang [45] and Brideau and Cavalier [20] were designed, except that they assume there is an infinite time horizon over which the servicing may take place. We consider the case where the time limit is binding, and so we are able to restrict the flexibility of the routing through altering the time limit. With a short time limit, the focus is on including all the customers and thus devising tours that are efficient, i.e., routing concerns, whereas the objective of reward maximization becomes more prominent as the time limit increases, i.e., there is a greater influence of selection concerns.

The effect of this model is to require all customers be included in the solution route. This creates subset selection problems that are more similar to conventional vehicle routing problems, which means that conventional solution and problem generation methods may be applied. We are interested in the differences that occur when we force all customers to be included, as opposed to where we simply allow any customers to be included.

We now provide a worst case situation for our model SUBSETOPTW. We first consider the network of Figure 6.4, as defined by the solid lines, where the distances are given adjacent to the arcs and the customer identifier and the customer's time window are given on the nodes. We assume that each customer has the same reward, which without loss of generality we set equal 1. For this problem, with

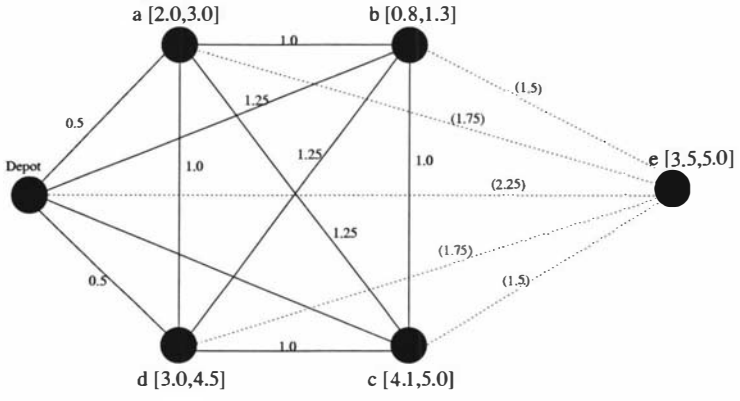

Figure 6.12: Small Network for OPTW

$L1 = L2 = 5.5$, it is possible to service the four customers by following the route $Depot - b - a - d - c - Depot$. The duration of this route is 5.5 and the reward received, which is the optimal reward, is 4. If we include the customer e, then its time window data implies that the only feasible solution routes are $Depot - a - b - e - c - d - Depot$ and the reverse of this route, each of which has the duration of 5.5. In neither case can we service any of the customers within their time window, and so the optimal reward in this case is 0. For our traditional subset selection model, we would just ignore the new customer e, and so the optimal reward would still be equal 4. Thus the worst case performance of reward received for our new model as opposed to the traditional model, is 0.

We also consider a case of the MCPTDR on the whole network of Figure 6.4, where again we have the time limit of 5.5. In this case we set the time window for each customer to be [0,5.5]. The reward functions of the customers are $100 - 20 * t$ for a, b, c and d, and $100 - 30 * t$ for customer e, where t is the time that the customer is serviced. For our traditional model for the MCPTDR, the optimal solution is to follow either the route $Depot - a - b - c - d - Depot$ or the reverse of this, and the total reward received is 240. If we require all customers to be serviced, then again the only feasible solutions are the routes $Depot - a - b - e - c - d - Depot$, and the reverse of this route, and the solution received, and thus the optimal solution is 197.5. The restrictions of forcing all customers to be serviced, results in a lower reward than with full subset selection.

Obviously we can create some trivial problems, where the worst case performance of forcing all customers to be served is zero. We again consider the complete graph of the Depot and customers a, b, c and d of Figure 6.4. In this case,

we set the time limit to be 4, the rewards for customers a and d to be 0 and the rewards for customers b and c to be $\max\{80 - 60 * t, 0\}$. The optimal subset selection route is to just serve either customer b or c, which yields a total reward of 5. If we have to service each of the customers, then the only feasible routes are $Depot - a - b - c - d - Depot$ and its reverse, and the reward for each of these is zero.

Testing a Model

We create problem instances for one form of the model described above, and test some solution methods for these problems. The model we are investigating is where there is a single vehicle available for servicing a set of customers, where all customers need to be serviced, and there may be time-dependent reward functions, where the reward is only received up until a given time limit. There is a linear cost of the total distance travelled in the solution route, and the objective is to maximize the net reward received. This problem is not actually a subset selection problem, as all customers need to be serviced and so all customers need to be considered when creating solution routes. We entitle this problem NET REWARD, and it is given as $T_{inc}|1||\max(R - C)$, by our classification scheme.

We created 50 layouts, by randomly assigning 60 points in a Euclidean square $[0, 1]^2$, with the depot placed at the middle of this square. For each layout we create 27 problem instances through varying three factors: the type of reward function, the cost of the distance and the length of time over which reward may be collected, at three different levels. Thus we obtain a total of 1350 problem instances upon which to test our solution methods. We set the maximum reward value for each customer to be equal to 20 times the distance from the depot to the customer location, as this obtains problems where the 'harder' customers have higher potential benefits. For the reward function the three cases we consider are with all rewards increasing from zero and reaching the maximum reward value at the end of the time horizon for collection, all rewards decreasing from the maximum value and reaching zero at the end of the collection horizon, and all rewards constant over the course of the collection period. For the costs of the distance, we test where the distance is not considered, i.e., cost = 0; where the cost plays some part, which we have set to have cost = 20; and where the cost is the most important factor, i.e., cost = 1000. For the tour lengths, we use the results of the asymptotic expected distance

of a TSP tour through a given number of customers, as proposed by Beardwood et al. [11], to obtain an estimate of 5.62 for the expected distance to be able to service all 60 customers in a TSP tour. We obtain three different lengths, through setting the limit of collection to be 0.9, 0.6 and 0.3 of this distance. These tour lengths then vary the number of customers that can be serviced in order to obtain reward, and so alter the priorities of the selection.

We create three construction methods for these test problems, with these being variations on the INSERTION Algorithm of Algorithm 2.1. These methods are:

- CHEAPEST — we add the customer that adds the least distance to the current solution. This method ignores the rewards completely and focuses on the routing aspects of the problem.

- BESTCHANGE — we add the customer that makes the best change in the objective function value, where this is given by the change in reward less the change in distance cost for the insertion. This method takes into account both the reward and costs and so should be applicable to any form of the problem.

- APPEND — we use the subgravity approach to append the customer that obtains the highest ratio of aggregate score to additional distance, when it is added to the end of the current route. If no customer obtains a positive aggregate score, i.e., if the time period when rewards may be collected has elapsed, then we append the remaining customers in the order of the additional distance. This method focuses on the reward collected, and is therefore more suited to problems where the costs of the distance travelled are low.

Each solution that is obtained is improved via application of the intra-route improvement routine as is used within the STEEPEST ASCENT routine. The change in objective function is always given by the change in reward less the change in distance cost for a move. The results of the testing are shown in Table 6.2, where the column 'Class' uses the notation (R,C,T) to indicate the reward type, cost and tour length respectively. The methods are given as CINIT and CFIN, which are the unimproved and improved methods using CHEAPEST, BINIT and BFIN, which are the unimproved and improved methods using BESTCHANGE and AINIT and AFIN, which are the unimproved and improved methods using APPEND.

Class	Solution Methods					
(R,C,T)	CINIT	CFIN	BINIT	BFIN	AINIT	AFIN
(1,1,1)	175.06	218.22	84.73	206.17	169.78	216.82
(1,1,2)	116.98	157.93	74.75	154.08	129.09	159.11
(1,1,3)	57.40	92.57	53.65	91.55	73.72	94.01
(1,2,1)	28.86	78.72	-17.63	70.73	-4.73	73.25
(1,2,2)	-29.22	16.47	-52.54	19.05	-37.74	16.52
(1,2,3)	-88.80	-43.61	-88.41	-44.41	-84.20	-43.53
(1,3,1)	-7134.98	-6298.89	-7129.16	-6316.68	-8555.61	-6204.11
(1,3,2)	-7193.06	-6386.93	-7194.34	-6390.24	-8212.42	-6289.45
(1,3,3)	-7252.64	-6451.72	-7268.97	-6450.00	-7822.57	-6324.43
(2,1,1)	155.84	212.21	193.27	212.58	193.37	214.40
(2,1,2)	97.36	143.06	125.00	143.47	131.83	145.75
(2,1,3)	38.71	69.74	57.37	69.92	61.96	70.36
(2,2,1)	9.64	74.90	24.43	74.72	29.03	78.76
(2,2,2)	-48.84	6.13	-33.49	6.76	-28.86	10.04
(2,2,3)	-107.49	-64.77	-92.50	-65.57	-92.04	-62.89
(2,3,1)	-7154.19	-6335.82	-7154.55	-6337.77	-8024.04	-6217.38
(2,3,2)	-7212.68	-6401.20	-7217.50	-6393.73	-7902.89	-6278.55
(2,3,3)	-7271.33	-6477.39	-7262.65	-6457.94	-7637.72	-6332.38
(3,1,1)	330.90	409.94	185.11	411.91	362.94	412.70
(3,1,2)	214.34	287.45	160.02	293.87	261.89	297.58
(3,1,3)	96.11	151.56	96.76	154.78	134.86	155.15
(3,2,1)	184.70	272.34	190.56	269.52	195.65	279.65
(3,2,2)	68.14	149.48	81.05	149.81	100.37	160.56
(3,2,3)	-50.09	18.61	-31.81	15.70	-19.12	21.39
(3,3,1)	-6979.13	-6138.24	-6980.10	-6131.45	-8001.20	-6018.09
(3,3,2)	-7095.70	-6263.68	-7094.80	-6250.54	-7814.13	-6147.54
(3,3,3)	-7213.93	-6399.79	-7229.99	-6394.35	-7564.08	-6299.70
Total	-2342.89	-2033.43	-2352.66	-2032.89	-2590.99	-1993.04

Table 6.2: Results of Solution Methods for NET REWARD

Table 6.2 shows that the APPEND method obtains the worst overall mean objective value, when we compare the unimproved solutions, and this is mostly due to the poor solutions that are obtained with the highest levels of cost (C = 3). In this case, the inclusion of customers due to the available rewards will result in the solution being longer than the minimum distance tour, and thus the distance cost incurred will be high. We see that with C = 3, the APPEND method obtains higher values as the tour length decreases (that is, as L increases), as the process spends a shorter amount of time collecting rewards and longer time applying the nearest neighbour method, i.e., appending the nearest available customer to the end of the current tour. The other methods, with C = 3, obtain lower objective values as L increases, as less total reward is collected, due to the shorter time period where the rewards are positive. The CHEAPEST and BESTCHANGE methods are very similar with C = 3, as they both focus on inserting the customers in their best position, according to the shortest insertion distance. With the lower values of the cost, the APPEND method is the most effective in obtaining good, unimproved solutions, particularly with constant rewards (R = 3).

When we consider the improved solutions, the APPEND method becomes the one with the highest overall mean objective value, and this method obtains the highest mean for 24 of the 27 problem classes. This method becomes the most effective with C = 3, so the improvement routines appear to be most effective with the APPEND method. We attribute this to the nearest neighbour approach being compatible with the improvement routines, as short sections of routes that are close to each other may be repositioned within the route. In fact we find that using the nearest neighbour method (adding the customer that is nearest to the end point of the current path) to create the whole solution routes with C = 3, the mean unimproved and improved objective values obtained are respectively -7258.645 and -6196.446, whereas for CHEAPEST, BESTCHANGE and APPEND these pairs are (-7167.515,-6350.406), (-7170.230,-6346.966) and (-7948.297,-6234.625). Thus the nearest neighbour method obtains the highest objective value after improvements, although it is less effective than CHEAPEST and BESTCHANGE when no improvements are considered. This nearest neighbour method is not effective when the rewards are an important concern, so it is not a method that can be used in general.

We also carried out a further implementation of the above methods, with the

addition of a general path move routine, which enables a path containing at least four customers to be moved to another position within the solution route. This routine was only implemented when the other routines were no longer able to obtain improvements, and this reduced the number of times we call this computationally expensive routine. The computational times that follow, are from implementing the routines on a Pentium II 400 MHz (128 MB RAM). With the new general path move routine, the new mean objective value for CHEAPEST was -2022.40 with a total computational time of 5 hours and 59 minutes (an average of approximately 16 seconds per problem instance), for BESTCHANGE the mean was -2021.63 and the computational time was 6 hours and 56 minutes (approximately 18.5 seconds per problem instance), while the mean for APPEND was -1986.93, with total computational time of 4 hours and 5 minutes (approximately 11 seconds per problem instance). The total computational time to obtain the improved solution for CHEAPEST was 2 hours and 12 minutes (approximately 6 seconds per problem instance), for BESTCHANGE it was 2 hours and 57 minutes (approximately 8 seconds per problem instance) and for APPEND it was 2 hours and 15 minutes (approximately 6 seconds per problem instance). Therefore with the additional routine, the computational time is increased dramatically, with relatively small gains in the objective value. The mean objective value for CHEAPEST and BESTCHANGE are still higher than those obtained by APPEND without the path move routine, so we conclude that the small improvements obtained by this additional routine need to be weighed against the large increases in the computational time required.

This NET REWARD problem is one where non-service is modelled by the rewards being equal to zero after a given time limit. We solved instances with three different levels of costs of the distance travelled within this model, where if the cost is zero we have a case of the MCPTDR, and with large costs the objective is basically to minimize the distance travelled, so this is very similar to the TSP. With moderate costs of the distance, the trade-off between reward received and distance travelled needs to be considered. We found with our testing, that the APPEND method, which appends the best customer to the end of the tour according to its subgravity score and adds customers in a nearest neighbour fashion when all rewards are zero, was an effective method when it was improved via our intra-route improvement routines. This method was reasonably effective for all the problem classes we considered, although with high distance cost, a method that ignored the rewards

and added the customer that was nearest the end of the current solution path, found improved results.

CHAPTER 7

Problem Generation Methods

In order to test the effectiveness of a solution method, there needs to be a set of problems available for testing purposes. When the application being modelled is a real situation, the real data may be used to determine the performance of the solution methods in practice. When there is no actual data present, or for checking robustness and predicting future behaviour in real situations, then it is necessary to generate the required data. An advantage of generating the data is that it is possible to vary the types of problems considered, in order to find out how the methods respond to different circumstances, and these results may be used for cataloguing problem types. We will study in this chapter, how researchers have previously generated problem instances that are related to the MCP and develop our own methods for problem generation for our particular problem.

7.1 Previous Problem Generation Methods

We now consider methods that have been previously used to generate problem instances for related problems. Our overall problem involves time windows, precedence constraints, reward functions, time limits and subset selection, so we will refer to problems containing some or all of these elements.

7.1.1 Time Window Generation

The Travelling Salesman Problem

The Travelling Salesman Problem with Time Windows (TSPTW) involves finding an efficient tour that services a set of customers, with each customer subject to constraints on the time at which they may be serviced. Savelsbergh [154] has shown that finding a feasible solution to the TSPTW is an $\mathcal{NP}$-complete problem, so the initial aim of the generation methods is to ensure that a feasible solution exists.

For the TSPTW, Baker [6] designed a set of problems based on data given for the Vehicle Routing Problem (VRP) by Eilon, Watson-Gandy and Christofides [44]; the locations of the customers were taken directly from these data sets. In order to create the time windows, a nearest neighbour tour through the customers was created, with the time windows determined so that the nearest neighbour tour was feasible and none of the time windows overlapped. This problem generation method ensures that there is at least one feasible route possible, as the nearest neighbour tour is feasible. Different problem instances are created through varying the number of customers whose time windows are active. The time windows create an explicit order in which the customers may be serviced; hence, with all the time windows being active there is only one feasible route. Identifying the optimal route is therefore trivial, as a simple ordering of the time windows will obtain the only solution, so this problem instance may be easily solved. With the time windows of some customers being inactive, the problem becomes more flexible, in that there may be a number of feasible solutions from which to choose. The active time windows still ensure a partial ordering of the customers, so this restricts the effective size of the problem. This problem generation method is therefore effective in creating problem instances that have a small state space, which makes it a useful method for creating problems that may be solved optimally, e.g., Baker uses branch and bound to optimally solve problems with up to 50 customers.

Langevin, Desrochers, Desrosiers, Gelinas and Soumis [106] created test problems for the TSPTW with distances either randomly generated (to obtain asymmetric or symmetric random problems) or with customer locations randomly generated (creating Euclidean problems). A second nearest neighbour tour was created through the customers, and the time windows were placed around the time each customer is serviced in this tour. Therefore, if customer i is serviced at time τ_i

in the second nearest neighbour tour, then the time window for customer i is uniformly random in the range $[\tau_i - width, \tau_i]$ and $[\tau_i, \tau_i + width]$, where $width$ is a user-defined parameter. This creates problems where there is a guaranteed feasible solution but the feasible tour is likely to be much different from the optimal tour. Dumas, Desrosiers, Gelinas and Solomon [41] used the method of Langevin et al. [106] to create test problems with larger numbers of customers. They also tested the effect of the density of customers, by increasing the number of customers and the size of the service area by the same proportion. They claimed that, with fairly wide time windows, only about 20% of arcs that are present in the second nearest neighbour tour are the same as in the optimal tour, and this percentage appears to remain constant as the number of customers increases. This indicates that the initial tour may be significantly improved upon.

Mingozzi, Bianco and Ricciardelli [124] created test problems for the TSPTW, with precedence constraints also present. They claimed that the test problems generated by Langevin et al. [106] and Dumas et al. [41] have tight time windows and that the initial tour the time windows are based on, i.e., the second nearest neighbour tour, is only about 10 percent longer than the optimal tour, which led to these problems being easily solved. Their method of generating test problems begins by randomly creating an initial tour, $(1, i_1, i_2, \ldots, i_{n+1})$, with τ_{i_k} being the time at which the kth customer in this tour is visited. The time window of customer i is given by $[e_i, l_i]$ with the time window of customer 1, the depot, given by: $e_1 = 0$ and $l_1 = \tau_{i_{n+1}}$. The time window of the k'th customer, i_k, is given by:

$$e_{i_k} = \begin{cases} \max[d_{1,i_k}, \tau_{i_{k-w}}] & \text{if } k - w \geq 1, \\ d_{1,i_k} & \text{if } k - w < 1, \end{cases}$$

$$l_{i_k} = \begin{cases} \max[l_1 - d_{i_k,1}, \tau_{i_{k+w}}] & \text{if } k + w \leq n + 1, \\ l_1 - d_{i_k,1} & \text{if } k + w > n + 1, \end{cases}$$

where w is a user-defined parameter which may be altered in order to vary the time window width. Therefore the time window of a customer depends upon the times at which the customers w positions before and after it in the tour are serviced. Precedence constraints were also generated so that the initial tour created is feasible in the presence of these given side constraints. The authors claimed that the test problems created by this method for the TSPTW have an initial solution, i.e., the randomly generated tour, that is on average more than 80% longer than the optimal solution and that the time windows are much wider than those found by previous

methods. Thus the initial solution is not a good bound on the optimal solution and there is a larger number of feasible solutions available, so these problems are more difficult to solve.

The Vehicle Routing Problem

The major difference between the generation of test problems for the TSPTW and for the Vehicle Routing Problem with Time Windows (VRPTW), is that for the TSPTW all customers must be serviced by the same vehicle, whereas with the VRPTW the fleet size is usually a decision variable. Therefore a feasible solution to the VRPTW will always exist, by using as many vehicles as are required to service all the customers. The feasibility concerns of the TSPTW aren't relevant to the VRPTW, so different generation methods may be used.

For the Vehicle Routing Problem with Time Windows (VRPTW), the standard problem set that has been used in many papers was created by Solomon [166]. He generated the customer locations in three ways: all customers randomly distributed, all customers distributed amongst clusters and semiclustered problems (in which customers are either clustered or randomly distributed). For each of these distributions of locations two problem sets were created, one with short time horizon and with service vehicles having low capacity, and the other with a longer horizon and higher capacities. These different time and capacity constraints allow more or fewer customers to be able to be included in a vehicle's route. Therefore there are six problem sets created, each with different location characteristics and different numbers of customers that can be included in a route.

For each of the problem sets, different instances were created by removing the time window constraint from certain customers, thereby altering the number of customers whose time window is active. With the random and semiclustered problem sets, the different instances were created by generating the width of the time window (either fixed width for all customers or different widths for the customers) and altering the proportion of customers whose time windows are active. For clustered problems, a 3-opt heuristic was used to generate a route through the points of the cluster, with the middle of the customer's time window occurring at the time the customer is serviced in this route. The proportion of customers with time windows and the width of the windows were varied to create the different instances. The use of the 3-opt heuristic to generate the centre of the time windows creates

a near-optimal solution within each cluster, which can be obtained by replicating the 3-opt method.

The two clustered data sets are shown in Figure 7.1. The first clustered data set, C1, clearly contains 10 clusters. Each cluster has been generated in such a manner that all the customers within the cluster may be serviced by the same vehicle. Therefore the time windows within the clusters allow a vehicle to serve all the customers with no waiting time required, and the total load for each cluster is less than, although close to, the capacity of the vehicle. This is designed so that there is a known benchmark solution, with as many vehicles as there are clusters, so any reasonable solution must attain this standard. This problem is not very flexible in terms of the number and "character" of feasible solutions allowed, as all reasonable solutions require each customer within a cluster to be serviced by the same vehicle. The different solutions are therefore created through varying the order of serving the relatively small number of customers in each cluster. The second clustered problem, C2, includes many of the same points, although some points are moved, and the resulting problem has the clusters less well-defined. In order to place the time windows, a solution route through the clusters had to be created, so the points were partitioned into three clusters, and, thus, all reasonable solutions are required to only use three vehicles. This problem is more flexible than the first clustered problem, as the time and capacity constraints aren't as restrictive in preventing a vehicle from being able to service customers from more than one of the clusters present, so the solution space is effectively larger.

An interesting feature of the problem instances is the dominance of the service times over the travel times. For the clustered set, C1, Desrochers, Desrosiers and Solomon [32] found the optimal solution for the problem for which all time windows are active. This solution required a total time of 9827.3 units; since the time of servicing each of the 100 customers is 90 units, the optimal solution involves 827.3 units of travelling time and 9000 units of time spent servicing the customers. Therefore, with the time windows being relatively short compared with the service times, the allowed flexibility in the routing is reduced.

For the Vehicle Routing Problem with Soft Time Windows (VRPSTW), Koskosidis, Powell and Solomon [99] randomly generated five problem sets. The important characteristics they identified include the number of customers to be serviced, the size of the area in which the customers may be located, the size of the loads to

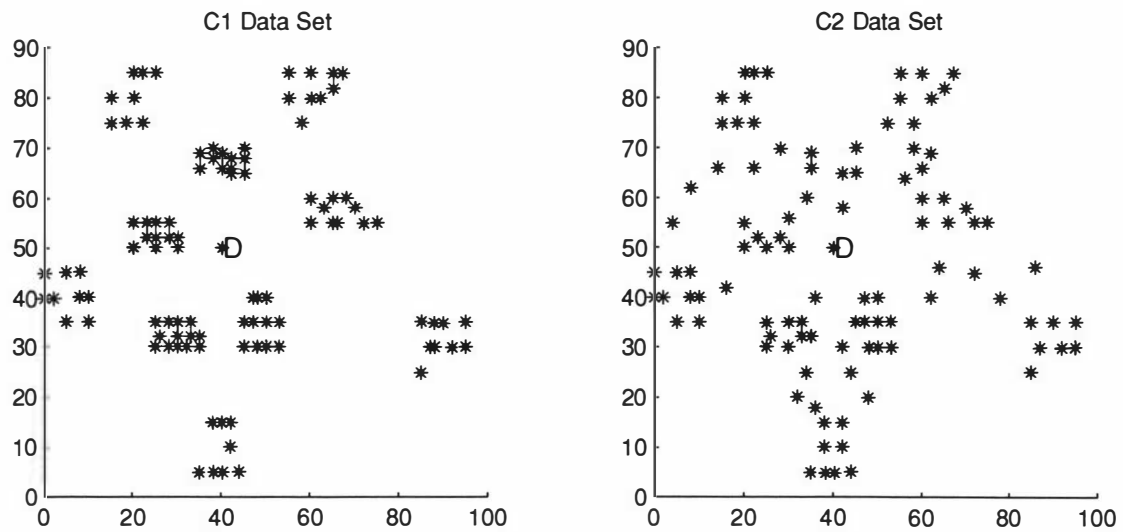

Figure 7.1: Solomon's Clustered Data Sets

be delivered, the planning horizon, the due date for a delivery and the length of the time windows. The number of customers, size of the service area, maximum load size and the planning horizon were used as inputs, and the other data was generated according to these values. The five problem sets involve a short time horizon with tight time windows, a medium horizon with three levels of time window strictness, and a long horizon with loose time windows. They also used Solomon's hard time window data in order to check the effectiveness of their solution method on these problems, by adjusting the penalties in order to reduce the number of customers serviced outside their time windows.

Van Landeghem [182] independently created a set of test problems for the VRPTW. He classified problems in terms of Ω, which is the average length of the window of availability for each customer, as a proportion of the total scheduling horizon. The problem classes he identified are shown in Table 7.1. The most difficult problems identified have time windows between 10 and 40 percent of the scheduling horizon, and solution of these problems requires both scheduling and routing considerations. He generated 20 problem instances from a real-life problem with 31 customers, with the time windows randomly generated so as to vary the value of Ω. Two other problem instances were generated from a random, Euclidean problem containing 50 customers, and two more problems were adapted from an existing data set. Again the time windows were randomly generated with known length, thus, these problem instances control for one characteristic of the problem,

Ω-range	Time windows	Hardness	Behaviour
$\Omega > 50\%$	very loose	moderate	routing problem
$30 < \Omega < 40\%$	loose	hard	mixed
$10 < \Omega < 20\%$	tight	hard	mixed
$\Omega < 8\%$	very tight	moderate	scheduling problem

Table 7.1: van Landeghem's Classification of VRPTW Problems

and can therefore be used to study how the solution methods are affected by this characteristic.

7.1.2 Subset Selection Problem Generation

We have seen that the major difference between the generation of problems for variants of the VRP and the TSP is in the manner that feasibility is ensured. With most subset selection routing problems there is no concern with feasibility, as a solution route that contains no customers is a feasible solution. This is the case for the Orienteering Problem (OP), as the inclusion of additional customers is beneficial for the goodness of a solution, although servicing no customers is a feasible option.

The Orienteering Problem

For the OP, the first test problems were created by Tsiligirides [176]. These problems have been used by subsequent authors for testing their solution methods (see [73], [74]). There are three problem sets, involving respectively 30, 19 and 31 customers available for servicing. For each set there is also an origin node, from which the solution route begins, and a separate depot where the route finishes. The customer locations appear to be randomly generated with the reward values being integer multiples of 5, which generally increase as the distances from the origin and destination nodes increase. The customers are therefore quite interchangeable, which creates multiple solutions of similar quality. In the case of the OP, many problem instances may be created from a given set of locations through varying the time limit, and Tsiligirides created 49 test problems from these three data sets.

Keller and Goodchild [96] developed test problems for the OP using real geographical data. The customers are 25 selected cities from the former West Germany,

with their rewards represented by the total population in the cities and the inter-customer distances being the road distances between the cities. This problem set therefore comes from an existing data set, which is assumed to be relevant for the OP.

Laporte and Martello [108] developed random, non-Euclidean test problems. The direct distance between each pair of points was taken to be a random integer in the range [1,100], with the actual distances used being the shortest path distances between the points. The rewards were also randomly generated in the range [1,100], so this problem is purely random.

Sokkappa [164] developed a number of test problems for the OP. She identified a number of important characteristics of test problems and varied these in order to test the effectiveness of her solution methods under different circumstances. The characteristics that were varied are: the form of the distances between customers, i.e., Euclidean or non-Euclidean, the distribution of the customer locations, i.e., random, clustered or with outliers, and the distribution of the rewards for the customers, i.e., all the same, uniform between various limits or exponentially distributed. The time limit was varied as a proportion of the approximate minimum distance required to service all the customers. She generated 18 classes of problems, each with 20 customers for testing her solution methods, and some larger problems (containing up to 50 customers) were generated for certain classes.

Kantor and Rosenwein [91] used the data of Tsiligirides for their version of the OP, which also contains time window constraints. The earliest time that customer i can be serviced, e_i, was randomly generated over the interval $[0, T_{max}/2]$ with the latest time, d_i, randomly generated over the interval $[e_i, \min(e_i + 2/3T_{max}, T_{max})]$. Therefore the width of the time windows varied as the time limit varies, so these needed to be generated for each time limit. These time windows are wide, with expected width of $\frac{23}{72}T_{max}$, or approximately 32% of the time limit. Therefore this paper only generates one type of problem, where the locations are those in Tsiligirides' data set and the time windows are quite wide.

Chao [25] used the data of Tsiligirides for testing his heuristics for the OP and the Team Orienteering Problem (TOP). He also created two new test problem layouts for the OP, with the customer locations and rewards set out according to a systematic pattern, and higher rewards for customers further from the start and end points. These problems contained respectively 62 and 64 customers. For the

TOP he took the test problems for the OP and divided the time limits by the number of service vehicles available. He also created locations for two new test problems with 100 customers, one with random customer locations and customer rewards, and the other an adaptation of a problem set used for a different vehicle routing application.

Fischetti, González and Toth [51] tested their optimization algorithm on a number of test problems from the literature. They defined four classes of problems; the first contained the data sets of Tsiligirides, as well as a number of Vehicle Routing Problem (VRP) instances from various sources, with the demand given in these problems being used as the customer rewards. The second class of problems involved a number of published instances of the Travelling Salesman Problem (TSP), with OP instances being created by generating rewards in three ways: the same reward for each customer, randomly generated reward and reward given by a function of the distance from the depot. They state that the problems with the latter rewards are the most difficult to solve, and this is because there is the most equitability in the attractiveness of the customers. In the third class of problems, the data was generated in the same manner as Laporte and Martello [108], i.e., with random rewards and random, non-Euclidean distances. The final class of problems involved customer locations being randomly distributed over a square, with Euclidean distances between the locations and the customer rewards randomly distributed. The authors claimed that this problem class appeared to be the most difficult to solve, where difficulty is measured by the computational time for their algorithm. This is because, for problems containing Euclidean distances, the paths between points are disjoint, so there are a greater number of feasible subpaths in these problems.

Gendreau, Laporte and Semet [64] randomly generated problem instances for the OP. They created instances containing Euclidean distances, in which customer locations were randomly generated, and non-Euclidean distances, in which the distances were randomly generated, and the reward values were also randomly generated. The time limit used was a proportion of the optimal TSP tour length around all the customers, for problems for which this length could be found, and an estimate of the TSP length was obtained via a heuristic method, for large instances. For their problem instances that required certain customers to be in the solution, the compulsory customers were randomly selected.

Other Subset Selection Routing Problems

The only previous problems generated for the MCPTDR have involved randomly generated problem instances. Brideau and Cavalier [20] generated examples with randomly generated distances between points and with the initial reward, the rate of decline of the reward and the service time at each point, being uniformly distributed. They stated that the range of values were such that some of the rewards became zero before all the customers were able to be be serviced, but they didn't give any more information about the parameter values used. Erkut and Zhang [45] randomly generated Euclidean locations for their points, with the other parameters uniformly distributed between given parameters. They tested the effect of increasing the variability in the rate of decline of the rewards, and used this to speculate as to the effectiveness of their different heuristics under different conditions.

We believe that the problem instances that have been generated for the MCPTDR have only considered one scenario of the problem. All the problem instances are of similar form, with most of the customers being serviced in good solutions. We believe that other problem instances, which include lower time limits and different relationships between the initial reward and the rate of decline, are also worthy of investigation. We also believe that a more general form of the problem, which allows other forms of the reward function, e.g., reward not necessarily just decreasing, should be investigated, with a further set of problem instances being required to enable this investigation to take place.

For the Prize Collecting Travelling Salesman Problem (PCTSP), most of the published research has involved proving theoretical results, without requiring any data for testing (see [8, 9, 18, 70]). The problem instances which have been used for solving, have all been randomly generated. Fischetti and Toth [53] generated test problems for the PCTSP by randomly generating the prize values and using three different forms of distances: general asymmetric, triangularized asymmetric and Euclidean. For Euclidean distances, customers were randomly positioned, whereas the distances themselves were randomly generated in the asymmetric cases. There were no penalties for non-service, and different problem classes were obtained by varying the proportion of the available reward that needed to be collected. Pekny and Miller [135] also generated test problems for the PCTSP, by randomly generating general, asymmetric costs of travelling, c_{ij} between each pair of customers, $i, j = 0, 1, \ldots, n$, with the cost c_{ii} giving the penalty for excluding the customer

from the tour. Gomes, Diniz and Martinhon [75] took layouts that had previously been used as TSP instances and added randomly generated rewards and penalties, in order to create their test problems for the PCTSP.

7.1.3 Generation of Task Problems

The published problem instances for precedence constrained problems fall mostly into two categories: random Euclidean instances, with locations uniformly distributed over a square region of the Euclidean plane, and real data, adapted for the variation of the problem being studied in the article.

Euclidean examples have been generated for the Dial-a-Ride Problem (DARP), for many papers including those by Psaraftis [142, 143], Jaw, Odoni, Psaraftis and Wilson [89], Kubo and Kasugai [101, 102], Bianco, Mingozzi, Ricciardelli and Spadoni [17] and Healy and Moll [82]. In each of these instances, n customers are generated by generating $2n$ nodes, with their locations coming from the uniform distribution, and randomly assigning one of these nodes to be the origin and another node to be destination for each customer. Bianco et al. [17] also allowed for general precedences, by generating a random tour through the nodes. Starting with the final node and working forwards, each unassigned node was assigned to be the final subtask of a customer, and a random number of preceding nodes in this tour were assigned to be the other subtasks for the customer. This approach is similar to that used for the TSPTW (see [106], [124]) in which a base route was created, with the constraints applied to this route, so that the route consists of a candidate solution to the problem.

Jaw et al. [89] used actual data for the DARP, from a city in West Germany. This problem instance included the additional considerations of heterogeneous vehicles, non-symmetric distances and time windows for customer servicing. Sexton and Choi [162] used real data from a Baltimore dial-a-ride system for their problem involving soft time windows. The locations and loads were taken from this data set, with the time windows generated by hand so as to make interesting problem instances. Sexton and Choi [162, pg390] stated that

> "The time windows were selected long enough so that several route sequences would appear attractive but short enough to make meeting all the windows a nontrivial task. Positioning the pickup windows and

> setting the window lengths involved some judgment on the part of the authors."

Therefore this problem was developed using intuition to create a problem with desired properties, which relate to its 'difficulty'. Toth and Vigo [175] obtained their test problems for the DARP with time windows from an actual problem situation in Bologna. The additional features of their problem included multiple vehicle types and the ability to increase the fleet size, by hiring additional vehicles at a higher cost. Madsen, Ravn and Rygaard [119] solved an actual DARP from Copenhagen, containing multiple vehicle types, and with different requirements for the different customers.

A new problem generation technique was used by Nanry and Barnes [128] to create test problems for the Pickup and Delivery Problem with Time Windows (PDPTW). They took the optimal solution to the benchmark problems of Solomon [166], and applied the precedence constraints so that these optimal solutions remain feasible. Therefore the optimal solutions to these PDPTW instances are the same as the optimal solutions to the VRPTW. They dealt only with the problems with the short time horizons, for which fewer customers are serviced within each vehicle. For instances where the optimal VRPTW solution contains an odd number of customers, a dummy customer was added to the route so the pick-up and delivery occur at the same location at the same time. They created a number of problem instances in which the first 25 nodes were used to create the customers. Fewer problem instances that contained the first 50 nodes were generated, and for problems containing the full 100 nodes, the problems generated were simply based upon Solomon's clustered problem instances. These 100 node problems involve 10 clusters, with the optimal policy being to send a single service vehicle to each cluster. The clustered 25 node problems contain three clusters and either 13 or 14 customers in total. With the precedence constraints on top of the time window constraints, there is a tightly constrained problem with very little possibility of deviation between feasible and optimal solutions.

7.1.4 Assessment of Previous Approaches

The generation of test problems methods have involved various degrees of investigation into the structure of the problems being studied. One well-studied problem, which is relevant problem to the MCP, is the OP, for which a large number of test

problems have been generated. The structure of this problem has been investigated a number of times, e.g., by Sokkappa [164] and Fischetti et al. [51], although the standard benchmark problems of Tsiligirides [176] are somewhat less definitive.

In order to test the effectiveness of methods applied to the VRPTW, the test problems of Solomon [166] have been used by many researchers (see [28, 140, 139, 151]). These test problems have even been extended to create larger problems, containing up to 1000 customers (see [84]). The test problems have also been used as a basis for testing methods for other types of problems, including the VRPSTW (see [7]), the PDPTW (see [128]) and dynamic routing problems (see [59, 86]).

The use of benchmark problems may play an important role in being able to compare the effectiveness of different techniques on the same sets of problems. A weakness of this approach is that only methods that achieve good results on the benchmark problems will be reported, and Hooker [85, pg35] describes this situation by stating that the

> "tail wags the dog as problems begin to design algorithms."

Thus, methods are, perhaps unintentionally, designed to deal with the specific properties of the standard problem instances, rather than to deal with general cases that may arise.

We are interested in problem design as a means of developing insight into the problems that we are solving, through creating a variety of problem instances over which to test the effectiveness of our methods. We seek to identify particular strengths and weaknesses of our methods, in order to develop additional features of the methods that may be used to overcome the identified shortcomings. Therefore, our problem generation approach is to identify the features of the problems that we wish to investigate, and to develop appropriate methods for generating problems with desired behavioural properties. Since our definition of the MCP contains a number of aspects that may be included within this problem, we seek to identify some features of the structure of the problems, in order to help characterize the general types of problems that might arise.

One potentially fruitful area for this investigation is in the generation of tasks, as previous research on problems involving tasks has eschewed the classification of the problem types considered. Randomly generated problems create one type of problem, where tasks are of various lengths, although the effect of the interaction between tasks is not considered. Gaboune, Laporte and Soumis [58] showed that

the expected distance between two randomly distributed points within a square is $\approx 0.5214d$, where d is the length of the sides of the square, and this result can be used to show that the creation of the mean task length in problems for which the tasks consist of two randomly generated points within $[0, 1]^2$ is equal to 0.5214. The new problems of Nanry and Barnes [128], involve tasks being created from within a solution route, so these take into account some of the features of the layouts, e.g., the clustered problems have all customers located wholly within the cluster, so involve relatively short task lengths. These problems are of a different form to those which are randomly generated, but we are interested in developing further features of the structure of task problems, by investigating how different forms of tasks elicit different behaviour from our solution methods.

7.2 Generating Test Problems

We now look to create our own test problems for the MCP, and for some of the special case problem areas that are subsets of the MCP. We determine that the MCP is a complex problem comprising three layers. These are:

1. locations of the customers
2. side constraints on the customers
3. reward functions

The base level of the problem is the layout and for any problem with only the layout given, i.e., without any constraints added, any candidate solution will be the shortest tour around the customers. The side constraints e.g., time windows will override the locations and bring a different form of solution, while the reward function has the ability to override both the locations and the constraints. This is shown for the layout given in Figure 7.2, where the shortest tour to visit all customers is 0—1—2—3—4—0, which has distance 4.4. If we introduce the time windows [0.5,3.2] for 1, [0.5,2.0] for 2, [4.0,4.6] for 3 and [2.0,3.2] for 4, then the shortest feasible path is 0—1—2—4—3—0, of length 4.8. If, additionally, we introduce the reward functions: R(1) = 100–10t, R(2) = 100–50t, R(3) = 100–t and R(4) = 100–30t, then the optimal solution, in terms of an objective function that relates to reward maximization, would be the path 0—2—4—1—3—0, which has total reward of 266.5 and takes total time of 5.2. Therefore the addition of the

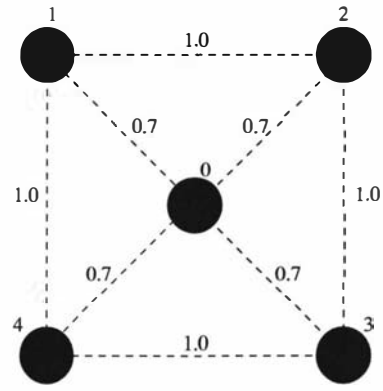

Figure 7.2: A Simple Cluster

extra layers to the problem, being the time windows and the time-dependency of the rewards, together with the different objective functions that are appropriate for the problems considered, greatly affect the form of the solution obtained. We propose an hierarchical approach to generating the problems, with firstly the side constraints and then the reward functions being applied to the layouts.

In order to develop techniques for effectively generating our problem instances, we study the range of options of consideration within problem generation methods. For fully randomly generated problems, each customer is generated independently of each other. When creating the constraints for a customer, each customer is treated individually and therefore there is no known interaction between the customers. The other end of the scale was demonstrated within the methods that were applied in order to ensure feasibility for time window problems. Here, all customers are placed within routes and constraints are applied which ensure the route remains feasible. This generation method involves implicitly considering all customers on a global level, where the application of constraints to each customer, takes into consideration the constraints on each other customer.

We consider a compromise between these approaches, where we consider a subset of other customers when generating the properties for a customer. Within a subset selection framework, we are concerned with the attributes of a customer, which include rewards as well as the constraints that we have previously considered. With our approach, the generation of the attributes of a customer depends on the attributes of some of the other customers and we create different classes of problems through varying the size and definition of grouping that we consider. We illustrate these approaches in Figure 7.2. At the individual level, we are concerned with

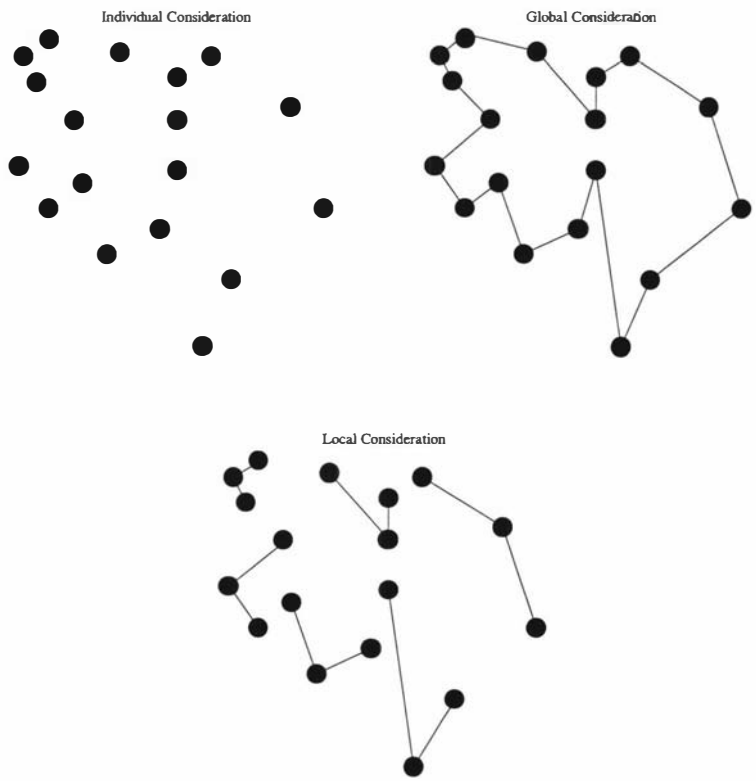

Figure 7.3: Levels of Consideration for Generation Methods

each customer in isolation, and we apply the constraints for a customer, without concern for any other customers. On the global level, we create a whole route and apply constraints so that this route is feasible in the presence of the constraints. We therefore are implicitly taking into account the constraints for every other customer when we create these constraints for a customer. At the local consideration, we create sets of customers, and apply the attributes to the sets to ensure that they are feasible. In the case illustrated, the sets are created by taking the route used on the global level, and creating sets of three customers by partitioning the route.

In this manner we are able to create groups of customers that have given properties and we feel that this approach is well suited to subset problems, because not all customers need to be included in the solution. Therefore we are generating candidate portions of routes that may be used as part of the solution routes.

As we vary the number of customers that are to be serviced within the problem instances, we vary the length of the sections of route to which the constraints are applied. We also vary the method used for obtaining the sections of route, in order to create different problems which involve different geometric properties.

When most of the customers are serviced, we base our constraints on tours through all of the customers, and we are able to alter the time limit of the problem or the compatibility of the constraints with the tour, in order to further constrain the proportion of customers that are able to be included within the solution route. We create the tours in three different ways:

- best insertion — this creates a TSP tour for which the total distance travelled is quite low. At each step we add the customer whose insertion adds the least total distance to the current tour.

- random append — we create purely random tours, by randomly selecting which customer to include next, and appending this customer to the current path (which is made into a tour by revisiting the origin once all the customers have been included). This method creates long tours, and since no consideration of the other customers is taken when selecting the next customer to include, the adjacent customers in the tour are independent of each other.

- randomized nearest neighbour — we sequentially build up a path by adding a customer to the end of the path, with the probability of candidate customer i being included next being proportional to $\frac{1}{d_i^2}$,where d_i is the distance to the current endpoint of the path. This creates tours with total distance that is a compromise between the distance found by the two previous techniques, and so creates reasonable length tours which do not strictly adhere to the geometrical properties of the layouts.

We illustrate the effect of the different methods for generating the base tours in Table 7.2, which contains the mean length of tour obtained by each of the methods over 100 randomly generated layouts, for each of the selected number of points that are to be included within the tour. We see from this table that the best insertion method, BESTINS, does obtain solutions which have a much shorter distance than those obtained by the other two methods. The random append method, RANDAPP, obtains tours whose mean length appears to increase linearly as the number of customers increase, while the randomized nearest neighbour method, RANDNN, obtain tours whose lengths are a compromise between those obtained by the other two methods.

The BESTINS method therefore obtains base paths which strongly take account of the geometry present within the layout, and this method therefore promotes

Num	BESTINS	RANDNN	RANDAPP
40	6.053	11.152	21.000
60	7.323	14.961	31.625
80	8.290	18.900	41.957
100	9.235	22.013	52.610
Total	7.725	16.756	36.798

Table 7.2: Average Lengths of TSP Tours Obtained

the use of solution methods which also focus on this geometry. The RANDAPP method obtains tours a the other end of the scale, and so these tours are somewhat structureless, as the points are independently placed. The tours obtained by this method are only of merit when the constraints applied to the tour are restrictive, as otherwise these tours can be easily improved upon by any solution methods. The RANDNN method does take into account some of the structure of the layouts, so the routes obtained do preserve some of the relationships between the locations, although not to the extent of the BESTINS method. RANDNN therefore obtains reasonable base tours, which, unlike the BESTINS method, can not be easily found by applying specific techniques to the problem, e.g., minimum distance insertion methods can be used to find the base tours in the problems generated by the BESTINS method.

For classes with few customers serviced, in order to create spatially close sections of route, we heuristically create what we call a k-matching. We define a k-matching to be the minimum cost partition of a set of points so that each subset contains k points which are linked together in a path. The criterion we often use to create the matching is to minimize the maximum length of the matchings, and this length can then be used as a time limit for the problem, thereby ensuring that each of the matchings form candidate routes. To create other small sections of route, which involve greater spatial separation, we create tours through all of the customers, using either the RANDNN or the RANDAPP method described above, and we partition these tours to obtain the required sections.

7.2.1 Generating OPTW Instances

We use a number of different techniques to generate prototypical problems of different classes for the OPTW. We create sections of route containing different numbers

of customers, and apply the time windows to these sections, so that the sections are feasible candidate routes, with no waiting time within a section. Different classes of problems are generated through creating time windows of different widths, thereby allowing more or less flexibility in the routing, i.e., with tighter time windows, servicing the entire sections become more attractive, as there are fewer options available. We also create different classes of problems by varying the length of the sections of route considered, so as to alter the number of customers that can be serviced within the solution route, which alters the priorities from being related to the selection of customers to being more related to the scheduling of the customers. The other factor we vary, to create different priorities within the OPTW, is the distribution of the reward values that are able to be collected from the customers. At its simplest level we vary the variability of the rewards, which alters the relative attractiveness of the customers.

The specialized technique we use to generate problems for the OPTW involves generating small sections of route, which are combined together to create the overall problems. We create small clusters of customers, and use different techniques to set the time windows so that certain desired properties of the routes are obtained. For example, we may require there to be a restricted number of routes available that service each of the customers, and this property may then be used to create problems in which it is harder to schedule all of the customers.

The technique we consider for generating the time windows involves randomly generating the initial time windows, checking some desired property of the solution obtained and appropriately altering the time windows if our desired property is not met. This method works recursively, by adapting the problem instance according to the current status of the problem, so involves a form of local search during the problem generation phase. Specifically, if we seek to have a given number of feasible routes through the customers, then the local search we apply is to increase the width of some of the time windows if too few feasible routes exist, and to reduce the widths if too many feasible routes exist. We also allow a small change to the start of the time windows, while preserving the same time window widths, although to allow this we need to set a limit on the range of the values of the time window that we allow within the problem instance. Other properties may be used within the search, and one which we employ is to ensure that the shortest path through the customers is not feasible with respect to the time windows, and

we prevent this from occurring by adding a large penalty if the shortest path is feasible. This penalty may be used to ensure that the feasible paths do not follow geometric shortest distance properties, and so prevents the generation of routes that may be easily replicated by applying these techniques.

We illustrate this process with a simple example, shown in Figure 7.4. In this example we have four customers, and we seek to obtain an allocation of the time windows which leads to there being exactly one feasible path through the customers, with the selected path not allowed to be the shortest path for the problem. In the Figure, the values given for each location represent the current time window, while any paths shown are the feasible paths at the current iteration. We start with the initial allocation of the time windows, shown in the top left plot, and there is no feasible solution with this set of time windows. The time windows are increased over the next two iterations, until after the third iteration, we obtain the problem shown in the top right hand plot, for which there are two feasible paths. To obtain the problem in the bottom left plot, three of the time windows are slightly altered, and this results in there again being no feasible paths. After four more iterations in which there are no feasible paths, we finally obtain the feasible solution shown in the bottom right hand plot. This solution begins at the customer located at $(0.0482, 0.0241)$, and proceeds around the path before ending up at the customer that is located by the first customer serviced. There is just the one feasible path that services each of the customers in this set, so this exact solution needs to be found in order to be able to completely service these customers. This method of generating clusters which contain a given number of feasible paths may be used for larger clusters, although in order to be able to solve large problems, some additional techniques which ensure the rapid convergence of the search would need to be employed.

We independently generate a number of clusters of customers by this method, and combine the clusters to create test problems for the OPTW, with different methods of combining the tasks resulting in different types of problems obtained. By only allowing one cluster at a time to be available, we generate problems that require the solution method to be able to find the feasible path through each cluster, so this type of problem focuses on the scheduling aspects of the problem. When there are a number of clusters available at a time, but with the clusters not overlapping, we generate test problems which require the effective scheduling of the

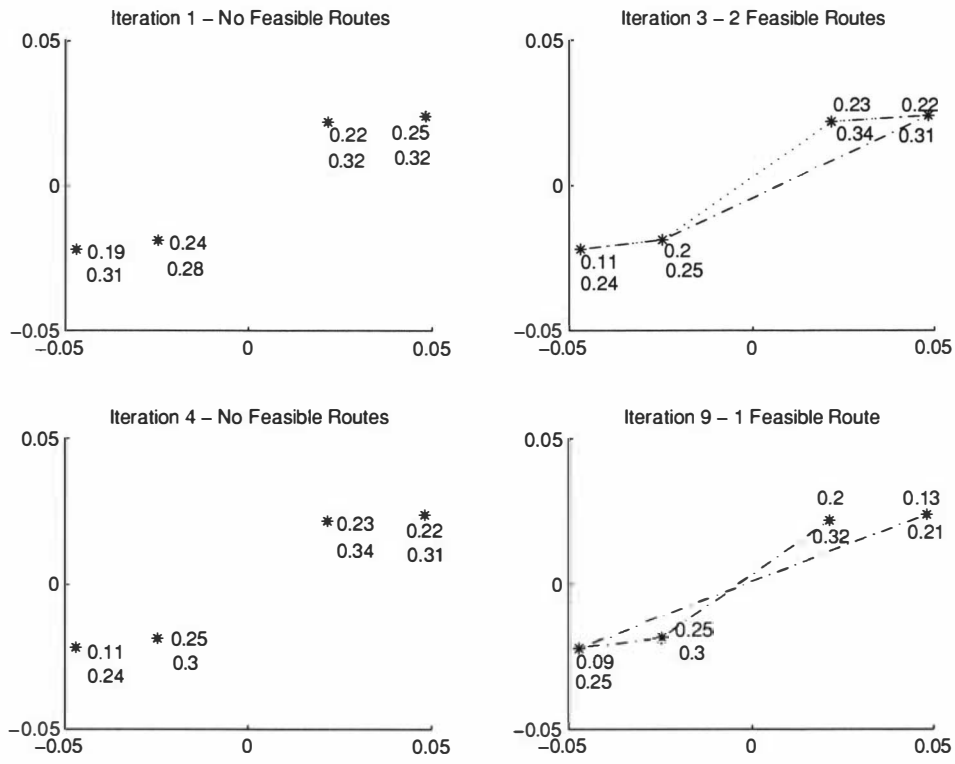

Figure 7.4: Generation Process for a Small OPTW Example

customers within the clusters, and also require the selection of the best cluster to service. With overlap between the clusters we obtain more general problems which allow whole clusters to be serviced, but also allow the clusters to be partially serviced, in order to obtain higher rewards from other clusters.

7.2.2 Generating TRCP Instances

In order to generate tasks we use the same approach described previously, by creating sections of route and applying the task constraints so these sections remain feasible. Our methods use different techniques for generating the sections of route to apply the constraints to, and we use a recursive method to apply the constraints. By assigning all customers to sections of route we create candidate solution routes, which may be used as a lower bound on the solution quality, and we are also able to introduce interaction between the tasks, by creating strings of customers which are interspersed or which involve spatially close customers.

We generate the task sizes and the loads associated with each task, according to the distribution that is used for the problem instance, and then carry out a

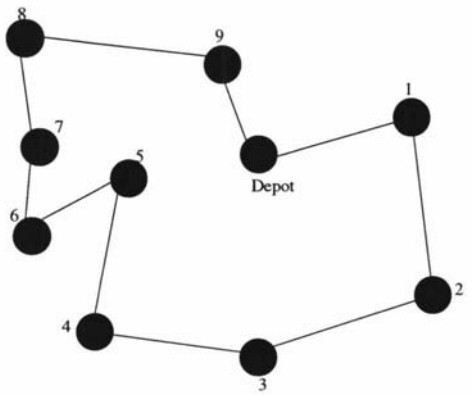

Figure 7.5: Base Route for Assigning Task Constraints

forward recursion through the candidate routes. For each new point encountered, we determine whether the point can be the first point of a new customer or if it has to be the next point of one of the tasks that we have already commenced but not yet completed. We do this by calculating the load that has currently been assigned to the vehicle, and the maximum load that is required to be in the vehicle at any time in order to service the next task. We prevent the next point from being the start of a new task if there is potential for the capacity constraint to be violated in servicing the next task. This precludes the possibility of the capacity constraint being violated, although, in doing so, it also prevents some valid routes from being considered. We felt the additional requirements of generating and storing all feasible paths, was unnecessary for the problem generation phase, so we instead remove all possible sources of infeasibility.

We illustrate this generation process on the fixed route shown in Figure 7.5. We assume that we have generated three tasks, each consisting of two pick-ups and a delivery; customer 1 requires two pick-ups of 2 units each (thus, a delivery of 4 units), and customers 2 and 3 require a pick-up of 1 unit followed by another pick-up of 3 units. The capacity of the vehicle is equal to 6 units. We begin by designating the first location of the tour (location 1) to be the first pick-up from customer 1. For location 2, we calculate the maximum load that is required for customer 2, which is equal to 4 units and occurs after the second pick-up. Therefore, since the current load (2 units) plus the maximum load for the next customer (4 units) does not exceed the vehicle capacity (6 units), we allow location 2 to be considered as a potential first pick-up for customer 2. We randomly determine whether the location will be assigned to be the first pick-up of customer 2 or the second pick-up

of customer 1, and we assume here that we allocate this location to be the pick-up from customer 2. The current load in the vehicle will be 3 units, and since the maximum load for customer 3 is 4 units, we do not consider location 3 to be a possible first pick-up from this next customer. We see that there is a potentially feasible tour that has location 3 being the first pick-up from customer 3, and this requires locations 4 and 5 to both be from customer 1; however, as mentioned earlier, we exclude this possibility. Therefore location 3 is randomly allocated as the second pick-up from customer 2 or 3 (which we assume is to customer 2), and consequently location 4 must be assigned to be the delivery for customer 2. This process continues by randomly assigning the remaining locations to customers, and we assume that location 5 is assigned to be the first pick-up from customer 3, location 6 is the second pick-up from customer 1, and then the only possibility for the remaining route is for location 7 to be the delivery for customer 1 and locations 8 and 9 being the remaining subtasks for customer 3.

Another technique we use to generate tasks, is to generate the sections of route in the same manner as described previously, but to then assign the tasks to consist of consecutive sections from the route. Thus, for the route shown in Figure 7.5, the three tasks that we would create would consist of locations 1, 2 and 3, locations 4, 5 and 6, and locations 7, 8 and 9. The main advantage of this approach is that there is greater control over the form of task that is created, as the tasks generated are directly dependent upon the method used for generating the tours. When allowing the tasks to be interspersed within the base routes, some of the structure of the routes becomes overwritten and the desired properties of the tasks may be lost. The disadvantage of this approach is that it generates tasks that do not overlap, so the method of generation and the way these tasks are actually included in the solution may be very different, in which case the base routes may be very loose bounds on the solution quality.

7.2.3 Generating MCPTDR Instances

As identified earlier, we believe that a weakness of the previous problem instances generated for the MCPTDR is the similarity of problem type. We consider that the problems with most of the customers able to be serviced are of a similar type and promote the use of certain techniques, but other types of problems would require different priorities to be taken into account. Therefore, one of the important

features of the problems we seek to investigate, is the number of customers that may be serviced at a time for which their reward is positive. This may easily be carried out, through varying the time limit available for servicing or the time period over which positive reward may be received. When the time limit is low and, thus, few customers may be serviced in a solution route, the main priority is in the selection of customers, while with higher time limits the routing and scheduling aspects become more prevalent.

We observe that for the MCPTDR there is an extra level of flexibility, as we are able to set the time limits of the problem to be much higher than the time that is required to service the customers, without creating a trivial problem, as would occur in the OP or the TRCP. The variation in the reward over time leads to different times being more attractive for servicing the customers, so a customer may be returned to at a later stage, rather than including the customer at the time when its insertion time is lowest. This therefore enables us to create an additional type of problem, in which all the customers may be easily serviced, but with the actual time of servicing becoming the crucial factor in determining the effectiveness of the solution obtained.

Another consideration is the variability in the rewards within each customer. If the reward available varies little over time, then the geometry of the problem is more important, as customers that are positioned close to each other will be included together in the solution. With larger variabilities in the rewards, the geometry present may be over-ridden, as it becomes more important to service the customers at the 'best' time. We will therefore investigate the effect of the variabilities of the rewards, by creating problems with different variabilities.

One of the techniques we develop for generating problems for the MCPTDR involves setting the 'best' time to service each customer. For the piecewise linear reward function that we described previously in Chapter 6, there is a given time period over which the maximum reward for a customer is available. This time period may be dealt with in exactly the same manner as for the OPTW, with the generated time windows referring to the time that the customer's reward is at its maximum, rather than the time the reward is available. By varying the width of the outer time windows and the slopes of the decline in the rewards, we are able to obtain problems with different characteristics; if the outer time window is wide then there are potentially greater numbers of feasible routes, and the identification

of the best times to service the customers may be more difficult to determine.

7.2.4 Generating MCP Instances

Our methods for generating problem instances for versions of the combined MCP containing time windows, time-dependent rewards and task constraints are based upon the methods that we use to create the test problems for the component problems of the MCP. We create sections of routes, apply the task and time window constraints to these sections and then assign the rewards to the created customers. The additional feature that we include within the combined MCP is the form of the reward function, as we enable the reward for a customer to relate solely to the time at which the customer's service is completed or to the time at which the service is started and completed. We generate the reward functions for the time of completing the task in the same manner as for the MCPTDR, as we are considering time-dependent reward functions of the same form that we used for the MCPTDR. For problems for which the reward received is penalized for excess riding time, we randomly generate the penalties within certain limits, and alter these limits to create the different problem classes. We set the penalties as a function of the minimum riding time for each customer, and this creates problems in which we can restrict the maximum length of a time that a customer is allowed to be in the service vehicle, to be a multiple of the minimum riding time.

CHAPTER 8

Computational Results

We now use the solution techniques discussed in Chapter 6 on a number of test problems, generated by using the techniques described in Chapter 7, in order to test the effectiveness of these methods. We investigate the MCP by firstly considering the component problems that comprise the MCP. Since the OP has been well-studied, the first of these problems we study involves adding time window constraints to the OP to obtain the OPTW, and adding task constraints to the OP to obtain a problem which we call the Task Routing Collection Problem (TRCP). We then consider the effect of adding time-dependent rewards to the OP, to create the MCPTDR, and we finally investigate a combined version of the MCP, which contains task constraints and time-dependent rewards.

To obtain the results in this chapter, our methods were implemented under Linux 6.2, on a Pentium II 400 MHz (128 MB RAM). The results obtained in our preliminary phase were analyzed using the non-parametric statistical method, the Kruskal-Wallis Test, while in our more specific testing, we used the Wilcoxon Signed-Rank Test for direct comparison between pairs of methods. Descriptions of the methods used can be found in Appendix B.

8.1 Preliminary Testing

8.1.1 The OPTW

We begin our computational experimentation with the Orienteering Problem with Time Windows (OPTW), which is classified as $subset, tw_j, K|1||sub, \max R$ according to the scheme we gave in Chapter 6. Recall (see pages 23 and 55) that the OPTW can be defined by the following integer programming formulation.

$$\max \quad \sum_{i=1}^{n} r_i(1 - x_{ii}) \tag{1}$$

$$\text{s.t.} \quad \sum_{j=0}^{n} x_{ij} = 1, \quad i = 0, \ldots, n \tag{2}$$

$$\sum_{i=0}^{n} x_{ij} = 1, \quad j = 0, \ldots, n \tag{3}$$

$$\sum_{i=0}^{n} \sum_{j=0}^{n} c_{ij} x_{ij} \leq C \tag{4}$$

$$2 \sum_{\nu_k \in S} (1 - x_{kk}) \leq |S| \left(\sum_{\substack{\nu_i \in S \\ \nu_j \notin S}} x_{ij} + \sum_{\substack{\nu_i \notin S \\ \nu_j \in S}} x_{ij} \right), \quad |S| \geq 3 \tag{5}$$

$$x_{ij} \in \{0, 1\}, \quad i = 0, ..., n, \quad j = 0, \ldots, n, \tag{6}$$

$$t_j \geq t_i + c_{ij} - M x_{ij}, \quad i = 0, \ldots, n, \quad j = 0, \ldots, n \tag{9}$$

$$l_i(1 - x_{ii}) \leq t_i \leq u_i + M x_{ii}, \quad i = 0, \ldots, n \tag{10}$$

As mentioned previously, there has only been one published paper on the OPTW, that of Kantor and Rosenwein [91]. This paper involves two heuristics being tested on one data set, where the time windows are given as a proportion of the time limit, and the time limit (and consequently the time windows), are varied in order to create different instances. This creates a number of problems which are of a similar type, and therefore the results obtained are only relevant for problems of this type. Our first concern with the OPTW is therefore to investigate a number of different types of problems, in order to find solution techniques that are suited to these different problem types.

We initially identify three important features of the OPTW and use these to create a classification of these problems. The parameters we use for these problems, and the features they describe are:

- N, the number of customers that may be serviced within a solution route
- T, the strictness of the time windows, and
- R, the variability in the rewards that are available

We create 27 classes of problems, by using three levels of the parameters for each of these three features. Our levels for the number of customers served are such that the time limit will allow either a small number (N = 1), approximately half (N = 2), or almost all of the customers to be serviced (N = 3). The time window strictness is varied so the windows are very tight (T = 1), i.e., all time window widths are approximately zero, medium tightness (T = 2), i.e., time window widths are between 20 and 40% of the time limit, and loose (T = 3), i.e.,time window widths are between 60 and 80% of the time limit. The levels of variability in the rewards have all rewards the same (R = 1), low variability in the rewards (R = 2), i.e., rewards are uniformly distributed between 10 and 20, and high variability (R = 3), i.e., rewards are uniformly distributed between 10 and 100.

The number of customers served determines the priorities of the solution methods for the problem, as the selection of which customers to service is more important with fewer customers serviced, whereas the routing is more important when greater numbers may be serviced. The time window strictness affects the flexibility in the routing that is available, and the variability of the rewards affects the priorities in the selection of which customers to service. Thus with few customers serviced, tight time windows and no variability in rewards, we have a selection problem where we need to determine a route that maximizes the total number of customers serviced. With high variability in rewards, the customers with higher rewards become more attractive and we are explicitly concerned with the total reward collected. By this classification scheme, we see that the problems of Kantor and Rosenwein [91] all involve low variability in the reward values, medium width time windows, and the varying of the time limit varies the number of customers serviced.

We isolate the effects of these features and randomize out the other features of the problem, such as the type of layout and the manner in which the time windows are applied. The types of layouts we consider are uniform layouts (with

a small, random component included for each location), clustered, semi-clustered and random. The semi-clustered layouts involve half the customers being placed in clusters, with the remaining customers being placed as in our uniform layout. We generate a number of layouts of each type, each containing 60 customers, which we feel is large enough to enable us to create a variety of problems for which trends can be found, without being so large as to be computationally prohibitive. We place the customers within $[0,1]^2$, with the depot placed in the middle of this square, and the distances and travel times between points being given by the Euclidean distances.

For N = 1 we apply the time windows to sections of route which are generated in three ways, with these based on short, medium and long distances between the consecutive customers in the sections. For short distances, we heuristically create matchings, each containing five customers, and apply the time windows so that these matchings are all feasible. For medium distances we create TSP tours by using the randomized nearest neighbour method, while for long distances we create TSP tours by using the random append method. We break these tours into sections, each containing five points, and apply the time windows so that these sections of route are feasible. The time limit was initially set to be a small time greater than the time at which the last of the sections could be fully serviced. With this method, it was found that the variability in the lengths of the sections led to the time limit often allowing too many customers to be serviced, which reduced the distinguishability between the cases for which N = 1 and N = 2, so we reduced the time limit to a level where each customer is feasible if serviced by itself. With N = 2 we create TSP tours, using the best insertion, randomized nearest neighbour and random append methods, so that half the customers are included in the tour, and we place the remaining customers in tours of this length or less. The time windows are applied so that each of these routes we obtain is a candidate, feasible solution. For N = 3, we create TSP tours through all the customers, again using the same three methods, and we set the time windows, so that the times of servicing within the TSP tour are feasible. In order to make the current tour to not be feasible, we set the time limit to be a small time less than the time required for the TSP tour and we also randomly select a small number of customers (between two and five) and perturb their time windows, so that these time windows no longer coincide with the time at which they were serviced in the TSP tour.

We are seeking a compromise in the number of problems considered, so as to be able to create conclusions about each of the problem classes, but to also be able to compute the results in a reasonable time. We therefore chose to generate 120 problems for each of the 27 problem classes, giving us 3240 instances in total. We generated 120 layouts, 30 of each type, and use these as required by the generation process. We then ran a number of heuristics on these problem instances in order to determine some trends that are present. Our aim was to tailor specific solution techniques and general approaches to particular aspects of the test problems, in order to be able to test whether the effective techniques are also effective in the presence of other constraints and different problem definitions that may occur within the framework of the MCP.

Our preliminary testing involved testing the effectiveness of a number of construction heuristics, with our standard improvement routine, STEEPEST ASCENT (which was previously described in Algorithm 6.2) applied to each solution obtained. For problems with customers consisting of single points to be visited, as is the case with the OPTW, customer Move is a subset of the Cross exchange routine, but we carry these routines out separately in order to expose the search process further.

The construction techniques we considered were all variants of the INSERTION Algorithm, as given in Algorithm 2.1, and they are:

- APPEND — for each customer we create an aggregate score for appending the customer to the current solution route. This score $R(i)$ is given by the function

$$R(i) = \sum_{j \in S} r_j * e^{-\mu t_{ij}}$$

where S is the set of customers that can be added after i to the current route, r_j is the reward of customer j, μ is a weighting parameter and t_{ij} is the additional time required to service customer j after i. We include the customer that has the highest ratio of score to additional time for inclusion. This method is an adaptation of the subgravity measure of Sokkappa [164], which takes into account the insertion time rather than just the distance involved. This is a general method which considers the times of insertion as well as the reward of the customer and the rewards of the customers that may be included after the candidate customer. The most effective value for μ was found to be approximately 10, and this is the setting we use to obtain

the results. This setting is such that only close customers are considered in the subgravity measure. If no customer can be added to the current route, a new route is initiated.

- KANTORB4B — this is the insertion method of Kantor and Rosenwein [91]. It consists of bang-for-buck, where the customer that has the highest ratio of reward to additional time for the insertion, is added. We extend Kantor's method, in order to place all customers in routes, by starting a new route if no customer may be added to the current route.

- CHEAP — the customer with the minimum additional distance is added to the first available route, in the position that adds the least time to the route. This technique is similar to that for the TSP, and it is therefore expected to be effective for problems where the routing is of major importance.

- GREEDY — the customer with the highest reward is added to the current route in the position that adds the least total time. If there is no feasible position to insert the customer, it is added to its own route. This technique is expected to be most effective for problems where the selection of the customers is the important consideration.

The results of these heuristics for each of the problem types are given in Table 8.1, with the computational times given in the form (h:mm:ss). Our testing in this case, is whether one of our methods is more effective than the others on each of the problem classes. Our first observation from this Table is that the GREEDY method is particularly ineffective. This method has the lowest overall mean score, and takes the longest computational time in order to obtain the improved solution. The method, as we have implemented it, obtains vastly inferior initial solutions compared with the other methods, albeit in much less computational time, and the poorness of the initial solutions leads to the excessive time that is required to obtain the improved solution. The GREEDY method is not even very effective for the problems for which it was devised, i.e., those with few customers serviced, particularly with a large variability in the rewards. Therefore we conclude that the GREEDY method is ineffective with these problems for the OPTW, and exclude this method from further consideration.

The KANTORB4B heuristic obtains the highest mean reward in the construction phase, which we found to be mainly due to it having the highest mean with

Problem Types	Methods			
(N,T,R)	APPEND	KANTORB4B	CHEAP	GREEDY
(1,1,1)	14.78	14.47	14.72	13.19
(1,1,2)	140.25	138.33	138.43	118.91
(1,1,3)	554.05	543.93	536.53	495.02
(2,1,1)	52.10	52.14	52.32	49.35
(2,1,2)	471.45	475.45	476.03	435.86
(2,1,3)	1728.71	1729.87	1733.42	1596.73
(3,1,1)	96.87	96.87	96.86	96.82
(3,1,2)	869.54	869.71	869.66	869.61
(3,1,3)	3175.01	3175.51	3175.52	3175.20
(1,2,1)	18.67	18.28	18.07	17.06
(1,2,2)	174.47	171.56	170.42	146.72
(1,2,3)	680.42	672.22	664.74	606.04
(2,2,1)	58.46	55.29	56.63	53.90
(2,2,2)	529.97	512.51	517.06	485.85
(2,2,3)	2024.53	1953.14	1944.12	1884.79
(3,2,1)	95.33	94.42	95.54	94.07
(3,2,2)	858.01	858.66	867.23	860.87
(3,2,3)	3169.55	3152.18	3172.21	3168.30
(1,3,1)	24.36	22.79	23.19	21.49
(1,3,2)	224.11	217.81	214.84	194.25
(1,3,3)	870.21	855.31	845.55	739.98
(2,3,1)	54.62	51.68	53.47	48.85
(2,3,2)	505.68	488.16	498.48	450.17
(2,3,3)	1911.75	1834.49	1857.77	1732.38
(3,3,1)	90.97	93.50	**97.85**	92.40
(3,3,2)	830.18	848.35	**884.12**	845.39
(3,3,3)	3134.74	3156.42	**3281.82**	3184.63
Overall Mean	828.10	820.48	828.02	795.48
Computational Time	0:34:45	0:28:39	0:26:41	1:33:32
Unimproved Mean	733.89	759.25	758.14	450.95
Unimproved Time	0:02:55	0:02:31	0:04:32	0:00:32

Table 8.1: Results from Preliminary Testing for OPTW Instances

tight time windows. Since there appears to be little differentiation between the final solutions obtained by the different methods with tight time windows, the advantage obtained in the construction phase is negated. When carrying out the Kruskal-Wallis test at a 5% level, we find that KANTORB4B is superior to APPEND for two problem classes, both with wide time windows and most customers served, while APPEND is superior for one of the classes. Again using the Kruskal-Wallis test at a 5% level, we find that KANTORB4B is not superior to CHEAP for any of the problem classes while CHEAP is superior for three. The KANTORB4B method appears to be a reasonable, general method, although its performance is dominated by the other methods that we consider.

We now compare the two heuristics, CHEAP and APPEND, that obtain the highest overall rewards. By the Kruskal-Wallis test at a 5% level, we find that the CHEAP heuristic is superior for the three classes with wide time windows, which is no real surprise, since for the problems with wide time windows, the time windows were based upon the time of servicing the customers within tours obtained by a method based on cheapest insertion. The rewards obtained by this method should therefore be close to the best attainable for problems with high numbers served, as the selection of the customers to service is less important than the routing for these problems. The CHEAP heuristic therefore obtains 'benchmark' solutions, with which to compare the effectiveness of the other methods. The APPEND heuristic appears to be more effective when there are fewer customers served, as this method's primary concern is with the selection decisions, although this method was not statistically significantly superior to the CHEAP heuristic.

We conclude that, of the four heuristics considered, the CHEAP heuristic is the most effective with wide time windows and most of the customers serviced, and we attribute this to the method we used to generate the problem instances. These cases resemble the TSP, as the routing decisions dominate the selection decisions, which leads to the method's effectiveness. The APPEND heuristic appears to be the most effective overall, although it is is relatively ineffective for the problems where most customers may be included. This is due to the method's lack of concern with placing a selected customer in the best position according to time of insertion criteria, as it instead focuses more on the selection of customers. In testing, we discovered that we could obtain higher mean rewards for the APPEND method with high numbers served, by decreasing the penalty factor used (which leads to

a greater consideration of the ratio of reward to insertion of the next candidate customer), although this leads to a reduction in the overall effectiveness of the method.

Our next concern is with the individual components of our improvement routines, to identify the routines that have the greatest effect on the solution quality and the computational time requirement. We apply the APPEND heuristic, and vary the improvement routines used within STEEPEST ASCENT; and the results of this testing are given in Table 8.2. The mean reward for each problem class is given in the Table, with the overall mean and computational time required for each method, given in the bottom rows. The ASCENT technique involves the use of STEEPEST ASCENT, and so the results are identical to those for APPEND in Table 8.1.

The method MOVE involves using only distance reduction and inter-route Move, and this method obtains solutions very quickly, although the quality of the solutions is quite poor. By removing the Move routine, and just having the Generalized Exchange and Cross routines, we create our method NOMV, which substantially increases the computational time required. This is because without the Move routine, more improvements are made using the more computationally expensive routines of Cross and, especially, Generalized Exchange. When applying the Kruskal-Wallis test at a 5% level, we find that there is no significant difference between the results obtained by the NOMV and the ASCENT techniques, and thus with the substantial increase in computational expense, we wouldn't consider the NOMV method to be a viable option.

The next two trials we carry out test the effect of the Cross routine. The first method, CROSS, involves using just the Cross routine, thus not applying the Generalized Exchange nor the Move routines. We see from this that just using the Cross routine reduces the computational time required by 52.4%, which indicates that Cross is not the major contributor to the time required. Using the Kruskal-Wallis test at a 5% level, we find that the CROSS heuristic is inferior to ASCENT for one problem class, although it has a much lower overall mean. For NOCR, which is the routine without Cross, we see that there is an 18.1% decrease in computational time compared with the full improvement routine. There is, however, quite a significant drop off in solution quality, which suggests that with the APPEND construction

Probs	Improvement Routines						
(N,T,R)	ASCENT	MOVE	NOMV	CROSS	NOCR	EXCH	OTHERS
(1,1,1)	14.78	14.42	14.78	14.79	14.42	14.78	14.94
(1,1,2)	140.25	133.48	140.25	139.79	135.50	139.79	141.24
(1,1,3)	554.05	530.13	554.67	551.20	538.23	551.19	556.48
(2,1,1)	52.10	50.53	52.08	52.49	50.53	52.10	52.40
(2,1,2)	471.45	457.04	471.45	473.43	458.38	471.10	475.95
(2,1,3)	1728.71	1666.96	1729.09	1734.91	1675.68	1726.99	1750.74
(3,1,1)	96.88	92.26	96.88	96.88	92.26	96.88	96.88
(3,1,2)	869.54	799.41	869.54	869.54	799.71	869.54	869.71
(3,1,3)	3175.01	2755.89	3175.01	3175.01	2755.99	3175.01	3175.51
(1,2,1)	18.67	18.39	18.68	18.60	18.57	18.54	18.74
(1,2,2)	174.47	168.13	174.50	171.00	171.43	172.42	173.57
(1,2,3)	680.42	654.01	680.75	666.92	672.15	673.36	680.63
(2,2,1)	58.46	57.13	58.49	57.86	57.96	57.61	58.31
(2,2,2)	529.97	512.20	529.26	523.91	525.30	527.07	532.39
(2,2,3)	2024.53	1972.87	2026.39	1979.52	2014.05	2006.03	2025.53
(3,2,1)	95.33	94.07	95.33	94.42	95.25	94.33	95.39
(3,2,2)	858.01	846.82	859.74	853.49	857.48	855.27	857.80
(3,2,3)	3169.55	3124.91	3175.36	3142.86	3162.33	3148.12	3171.94
(1,3,1)	24.36	23.81	24.29	24.14	24.01	24.17	24.46
(1,3,2)	224.11	216.72	224.20	220.42	222.08	221.53	224.99
(1,3,3)	870.21	844.44	872.20	851.99	864.64	860.07	873.47
(2,3,1)	54.63	53.72	54.50	54.04	54.15	54.14	54.58
(2,3,2)	505.68	490.23	505.79	494.51	503.04	501.24	505.05
(2,3,3)	1911.75	1843.12	1913.43	1863.13	1897.81	1894.60	1903.97
(3,3,1)	90.97	89.06	90.90	89.62	90.75	89.21	90.97
(3,3,2)	830.18	804.25	831.44	811.86	828.18	823.51	829.77
(3,3,3)	3134.74	3051.40	3140.34	3075.08	3132.47	3107.25	3134.74
Mean	828.10	791.31	828.86	818.57	804.16	823.18	829.27
Time	34:45	4:45	61:33	16:32	28:28	24:36	41:52

Table 8.2: Testing Improvement Routines for OPTW Instances

technique, the Cross routine is an important component in an effective improvement routine. Of particular note is that the CROSS heuristic has a higher overall mean than the NOCROSS heuristic. Using the Kruskal-Wallis test at a 5% level, we find that the CROSS heuristic is superior to the NOCR heuristic for four of the classes, while the NOCR heuristic is not superior for any. We note that each of the classes for which the CROSS heuristic is superior has tight time windows, with the huge difference between the means (3175.007 vs 2755.985) with tight time windows but high reward variability and high numbers served, accounting for much of the difference in the overall means. The Cross routine appears to be crucial with tight time windows, as we need to be able to exchange paths that may have been poorly placed in the construction phase.

Our next consideration is to check the effect of our Generalized Exchange routine, which enables customers to be moved between three vehicles, and the traditional Exchange routine, where two vehicles are used. By using Exchange instead of Generalized Exchange (technique EXCH), we see that the computational time required is reduced by 29.2% which suggests that the Generalized Exchange routine is a significant contributor to the computational time. The inclusion of Generalized Exchange, rather than the Exchange, does not obtain statistically significant differences in the solution quality, although the overall mean values are much higher.

Our last consideration is in the way we evaluate potential moves for our improvement routines. Our routines discussed thus far involve us only carrying out a move if the reward in the solution is increased and we now consider the case where we are able to make moves that don't necessarily affect the solution directly. Our implementation here is to allow a move to be carried out if the highest reward of the affected vehicles is increased; this allows us to increase the reward of the vehicles not currently in the solution, which may be included in the solution after a later move. For the routine OTHERS, the computational time required is increased to 41 minutes and 52 seconds, but there is no statistically significant difference between these results and those obtained by ASCENT.

From the results of Table 8.2 we conclude that the inter-route routines we include within STEEPEST ASCENT are all important contributors to the implementability of the method. The inter-route Move routine is an important contributor in reducing the computational time required to improve the solutions. The Generalized Exchange routine is the greatest contributor to the computational

time, but it also appears to have a positive affect on the effectiveness of the routines. The Cross routine is required in order to obtain improvements to problems where the time windows are tight, with this contribution enhanced by performing improvements on all the routes, rather than just the solution route.

We next look at the absolute quality of our heuristic methods, by testing a number of solution routines that explore the solution space, and the results obtained are shown in Table 8.3. The column BEST gives the mean reward obtained by taking the highest reward obtained by each problem instance, from the methods we have tried thus far, including the other techniques whose results are given in this Table. We consider three different implementations which create 10 solutions using the RANDOMCONSTRUCT construction technique of Algorithm 6.3, and improve upon each of these using the STEEPEST ASCENT routine. R+IMP uses the full STEEPEST ASCENT routine, whereas the only inter-route improvement routines considered within R+MV and R+CR are the Move and Cross routines, respectively. The routine R+IMP, which proved to be the best of the strategies considered, required approximately 16 hours of computational time, or an average of approximately 18 seconds per problem. Since the initial solutions are randomly generated, these are quite poor, and, thus, longer computational time is required in the improvement phase. R+MV and R+CR took approximately 37 minutes, and 3 hours and 42 minutes respectively, but the comparatively poor quality of the solutions they obtained makes these methods unsuitable for providing effective solutions to the OPTW.

Also within Table 8.3, we have the results from three different implementations of Tabu Search. TIMP, is our full implementation of TABUIMP (which was described in Algorithm 6.4), in which we use the inter-route improvement routines Move, Generalized Exchange and Cross. During our Tabu Search phase, we randomly select one of these three routines to carry out, with the probability of selection being proportional to the previous effectiveness of the routine on the current problem and we perform the best non-tabu move that we find. For TMOVE and TCROSS, we use only the routines Move and Cross respectively. Thus we improve the randomly created solution by just using either Move or Cross, and then for Tabu Search we simply apply the best non-tabu perturbation found by using the selected routine. The total time required for TIMP was approximately 40 hours, while 7 hours and 15 minutes and 10 hours and 52 minutes were required for

Classes	Improvement Strategies						
(N,T,R)	BEST	R+IMP	R+MV	R+CR	TIMP	TMOVE	TCROSS
(1,1,1)	15.33	14.58	13.67	14.10	15.29	14.94	15.31
(1,1,2)	144.45	141.18	130.75	137.64	144.25	141.36	144.17
(1,1,3)	570.58	561.24	536.80	551.70	570.25	559.84	570.13
(2,1,1)	53.21	52.86	51.28	52.63	53.22	52.68	53.19
(2,1,2)	482.30	480.13	468.23	477.14	482.27	474.16	481.98
(2,1,3)	1770.71	1766.34	1722.50	1764.08	1769.19	1752.08	1769.34
(3,1,1)	96.88	96.88	93.83	96.88	96.88	96.03	96.88
(3,1,2)	869.71	869.71	850.24	869.71	869.71	862.22	869.71
(3,1,3)	3175.52	3175.52	3114.80	3175.52	3175.52	3145.12	3175.52
(1,2,1)	19.92	18.69	16.68	16.94	19.72	18.99	18.78
(1,2,2)	183.48	178.53	156.89	168.47	182.47	174.29	175.99
(1,2,3)	714.92	703.86	644.70	663.15	709.66	687.90	691.91
(2,2,1)	61.40	57.50	48.03	49.40	60.96	55.25	53.35
(2,2,2)	556.88	542.35	452.98	478.39	554.18	502.70	499.02
(2,2,3)	2118.40	2083.65	1770.54	1835.97	2100.83	1942.51	1912.72
(3,2,1)	97.46	96.88	90.17	88.17	97.24	95.26	88.85
(3,2,2)	882.03	879.19	822.31	818.76	881.03	862.28	825.49
(3,2,3)	3229.05	3222.51	3056.39	3031.69	3226.13	3181.30	3049.70
(1,3,1)	25.67	23.67	21.00	20.78	25.32	24.38	23.31
(1,3,2)	237.29	230.10	204.55	202.96	235.67	221.69	219.05
(1,3,3)	921.11	900.28	808.41	813.67	911.60	874.36	847.13
(2,3,1)	58.71	53.49	44.90	44.79	**58.13**	50.79	49.92
(2,3,2)	542.19	524.32	430.58	442.31	534.76	464.39	470.18
(2,3,3)	2068.86	2022.47	1666.32	1708.85	2046.09	1828.91	1803.15
(3,3,1)	99.56	98.78	96.58	95.17	98.78	97.07	93.17
(3,3,2)	893.06	890.23	869.56	854.25	888.78	876.07	834.80
(3,3,3)	3308.30	3302.10	3245.47	3190.15	3297.34	3258.55	3150.50
Mean	859.15	851.37	793.64	802.34	855.75	826.49	814.19
Time		≈16 hrs	0:37:04	3:42:40	≈40 hrs	7:15:20	10:52:07

Table 8.3: Testing Search Strategies for OPTW Instances

TMOVE and TCROSS respectively. The TCROSS method appears to be effective only when the time windows are tight and otherwise it shouldn't be considered. TMOVE is a better alternative, with less computational time required, for problems with wide time windows. However, the mean reward obtained by this method is less than that obtained by the best heuristic method from Table 8.1, i.e, APPEND, thus, when comparing the effectiveness of this method, and the computational time required, with that for our basic construction techniques with improvement, we see that the TMOVE heuristic is relatively ineffective. It seems that the ability of the Generalized Exchange routine to create improved solutions, exceeds the ability of Move and Cross to improve the solution (apart from with narrow time windows), even when they are applied via more advanced search techniques such as Tabu Search.

The TIMP method appears to be the most successful, and its quality is not too dissimilar from the best solution found over all our routines. If we compare the results from TIMP and R+IMP, by the Kruskal-Wallis Test at a 5% level, we find that TIMP is statistically significantly superior for just the one problem class, although it appears to be more effective in general. Our last concern for the OPTW is with the difference between the solution quality of the TIMP method and the quality of our basic heuristics. We consider the four heuristics from Table 8.1, and the total time to run these is 3 hours and 4 minutes, although this time includes repeating a number of steps, e.g., inputting the data. The overall mean reward for these problems is 849.444 compared to 854.892 for TIMP. When applying the Kruskal-Wallis Test at a 5% level, we find that the best of the heuristics is superior for one of the problem classes, with this being with high numbers served, medium time windows and high reward variability. This again suggests that there is some merit in obtaining a number of solutions, preferably using some good initialization method, and improving upon these, in order to create good solutions for these loosely constrained problems. The TIMP heuristic is significantly more effective than the best of the other heuristic solutions for six of the problem classes, and this shows the overall effectiveness of the method.

8.1.2 The TRCP

Since there has been no published research on subset selection versions of problems involving tasks, we need to create new methods and new problem instances in order to make progress in this area. The TRCP, that we consider here, is classified as $subset, T_{fix}, K|1||sub, \max\ R$ according to the scheme we gave in Chapter 6. Therefore this is an extension of the OP to include customers consisting of tasks. This problem contains no time constraints, apart from an overall time limit, and there is a constant reward that is received upon the completion of a customer's service requirements.

The TRCP can then be modelled by the following integer programming formulation. Let N be the number of tasks. Each task i is composed of a sequence of n_i subtasks, each subtask k of task i being *either* a pickup of some load (positive ℓ_{ik}) or delivery of some load (negative ℓ_{ik}). Task i has fixed reward r_i upon completion of that task. If a task is selected then we must visit its subtasks in the correct order. Task 0 represents the initial depot (with one subtask). There is a single capacitated vehicle of capacity L_{max} and an overall time limit of the route of T_{max}. The objective is to maximize the total value in rewards collected.

For each task i, define the binary decision variable y_i which takes the value 1 if and only if the task is selected to service.

Consider task i and task j. Task i has n_i subtasks $(1, 2, \ldots, n_i)$ and task j has n_j subtasks $(1, 2, \ldots, n_j)$. Now consider the k-th subtask of task i and the m-th subtask of task j, and define the binary decision variable x_{ikjm} which takes the value 1 if and only if subtask m of task j immediately follows subtask k of task i on the single vehicle route. The time to travel between these subtasks are given by the variables c_{ikjm}.

To enforce the precedence amongst the subtasks of a task, we must keep track of the time at which the vehicle visits each subtask. Hence, define the continuous decision variable t_{ik} to be the time at which the vehicle visits subtask k of task i.

To observe the capacity constraints on the vehicle, define the continuous decision variable L_{jm} to be the load in the vehicle immediately following the vehicle servicing subtask m of task j.

$$\max \quad \sum_{i=1}^{N} r_i y_i \tag{11}$$

$$s.t. \quad y_j \leq \sum_{i=0}^{N} \sum_{k=1}^{n_i} x_{ikjm} \leq 1 \qquad \forall\ j = 1, \ldots, N \text{ and } m = 1, \ldots, n_j \tag{12}$$

$$\sum_{j=1}^{N} \sum_{m=1}^{n_j} x_{ikjm} \leq y_i \qquad \forall\ i = 1, \ldots, N \text{ and } k = 1, \ldots, n_i \tag{13}$$

$$\sum_{j=1}^{N} \sum_{m=1}^{n_j} x_{01jm} \leq 1 \tag{14}$$

Subtour Breaking Constraints (15)

$$y_i \in \{0, 1\} \qquad \forall\ i = 1, \ldots, N \tag{16}$$

$$x_{ikjm} \in \{0, 1\} \qquad \forall\ i, j = 1, \ldots, N \text{ and } k = 1, \ldots, n_i \text{ and } m = 1, \ldots, n_j \tag{17}$$

$$\sum_{i=0}^{N} \sum_{k=1}^{n_i} \sum_{j=0}^{N} \sum_{m=1}^{n_j} c_{ikjm} x_{ikjm} \leq T_{max} \tag{18}$$

$$0 \leq t_{i1} \leq t_{i2} \leq \ldots \leq t_{in_i} \leq T_{max} \qquad \forall\ i = 1, \ldots, N \tag{19}$$

$$t_{01} = 0 \tag{20}$$

$$t_{jm} \geq t_{ik} + c_{ikjm} - M x_{ikjm} \qquad \forall\ i, j = 0, \ldots, N \text{ and } k = 1, \ldots, n_i \text{ and } m = 1, \ldots, n_j \tag{21}$$

$$L_{max} \geq L_{jm} \geq L_{ik} + \ell_{jm} - M x_{ikjm} \qquad \forall\ i, j = 1, \ldots, N \text{ and } k = 1, \ldots, n_i \text{ and } m = 1, \ldots, n_j \tag{22}$$

$$L_{ik} \geq 0 \qquad \forall\ i = 1, \ldots, N \text{ and } k = 1, \ldots, n_i \tag{23}$$

$$L_{01} = 0 \tag{24}$$

In the formulation above, the objective function (11) measures the total reward collected. Constraints (12)–(14) are the route connectivity constraints that ensure that if a task is selected to be serviced, then every subtask must have an incoming and and outgoing arc on the route. Constraint (15) represents any suitable set of constraints that prevent subtours in the solution. Constraints (16)–(17) are the binary decision variables described above. Constraints (18)–(21) enforce the precedence constraints amongst the subtasks of each selected task (for some suitably large number M) and also the overall time deadline. Constraints (22)–(24) ensure that the vehicle is always capacity feasible.

We, again, look to classify the behaviour of heuristics according to their effectiveness on different classes of problems. Our initial classification of problems for the TRCP involves three features and these are:

- N, the number of customers that may be serviced within a solution route
- L, the size of the load that is to be carried, and
- R, the variability in the rewards that are available

We again create 27 classes of problems, by creating three levels for each of the three variables. For the numbers of customers served, we set the time limit so that respectively few (N = 1), approximately half (N = 2), and almost all of the customers (N = 3) may be serviced. We define our tasks to be a fixed ordering of locations which need to be visited, with given loads to be carried between the locations. Our load size variable, L, allows respectively one (L = 1), two or three (L = 2), and as many tasks as desired (L = 3), to be in the vehicle at one time, where the number of customers available relates to the relationship between the loads and the capacity constraint on the vehicle. If only one customer may be serviced at a time, then we have a problem where tasks must be carried out consecutively, and this is therefore similar to the OP, but with a known starting and ending point and a duration for each customer. When loads of many customers are able to be included in the solution, the tasks act as precedence constraints on the order that a path may be executed. Thus the load size affects the amount of flexibility in the routing that is available. The variability of the rewards is the same as for the OPTW, with the rewards being, respectively, all the same, having low variability and having high variability.

In order to generate the tasks, we create sub-paths, appropriately sized for the setting of N considered, and assign the tasks to these sub-paths so that the sub-path is a feasible section of route. For problem classes with N = 3 we create a feasible route through all the points that comprise the customers and set the time limit to be equal to 95% of the duration of this path. We implement an additional step that involves reversing the order in which the subtasks of a small number (between one and four) of the tasks are to be visited, and this further prevents the base route from being able to be followed exactly. For the classes with N = 2, we create two routes, each containing half of the points, and set the time limit to be the minimum of the two route durations. For the classes with N = 1 we create small sections of route, each containing either six or ten points, and set the time limit to be the maximum of the maximum time required to service a task and the minimum time required to service a section. This ensures that all of the tasks we generate are able to be serviced.

We generated a total of 3240 problem instances, with each instance containing 60 points that comprise the components of the tasks. Thus, we created 120 problem instances for each problem class, by randomizing out a number of other features of the problem, including the distribution of the locations of the points which comprise the tasks, the distribution of the inter-task distances and the number of subtasks in each task. We use the 20 distributions of locations that we used for the OPTW, so the points are placed within the Euclidean $[0, 1]^2$ and for each of these layouts we create six problem instances for each problem class. For three of these instances we include two points in each task, and for the other three instances approximately half of the customers contain two points and the remaining customers contain three points. We also vary the methods of generating the sections to which the task constraints are applied. The first method we employ involves creating sections that contain points which are geographically close to each other, and we do so by constructing TSP tours through all the customers or half the customers via our best insertion method, in the cases where N = 2 or 3. For classes with N = 1, we heuristically create matchings, containing 10 points, where the objective used in creating the matchings is to minimize the maximum length of the created matchings. The second method we employ involves creating appropriately sized sections of route, by using a randomized nearest neighbour construction method, which we found to obtain significantly longer sections than the

TSP tour construction methods of our first generation method. The third method we employ involves taking fully random tours and partitioning these appropriately to create the sections. The average length of task obtained by the three methods was 0.1859, 0.3176 and 0.5823, respectively, so these generation methods obtain tasks with varying geometric properties.

We found that the number of problem instances we generated was small enough to enable us to test our methods on the full problem set, whilst being large enough to enable statistically significant results to be obtained. Computational time was a large consideration in setting the problem size, as the time required to implement our improvement routines was much higher than the time required for the OPTW.

We carry out a number of construction heuristics on the problems, with these then improved by applying our STEEPEST ASCENT method. The routines used within STEEPEST ASCENT were adapted in order to be able to efficiently deal with the capacity and precedence constraints present. Due to the time required to implement the heuristics, we initially tested over a subset of the problems, in order to tune any of the required parameters.

The construction techniques we used that were variants of the INSERTION Algorithm, as given in Algorithm 2.1, are:

- B4BTASK – this is bang-for-buck, where we add the customer that has the highest ratio of additional reward to additional time when it is added in its best position to the current route. If no customer can be added to the current route we start a new route.

- AGGAPP – we append customers so they are served consecutively at the end of the current route. We use the subgravity approach in which the customer that has the highest ratio of aggregate score to additional time required, is appended to the current route., and the parameter we used for the aggregate score is again equal to 10. Since we add the subtasks of a customer so they are serviced consecutively, this method is particularly suited for problems where the large loads ensure that only one customer may be carried at a time. If no customer can be added to the current route, a new route is initiated.

- CHEAP – we add the customer that adds the least total distance when added to our current route. If no customer may be added, we create a new route.

- GREEDY – we add the customers in decreasing order of their reward value.

Algorithm 8.1 function CLUSTERING

```
// Begin with all customers in individual clusters
while (Improvement is still possible) do
    for each pair of clusters a and b do
        Calculate the change when inserting cluster a into cluster b,
        and store the change if it exceeds the best change found so far
    end
    if (Best change is 'good enough') then
        Combine best clusters into a single cluster
    end
end
end
```

The customer is placed in its best position in the current route, if possible, otherwise a new route is created.

Our other techniques we employed involved the insertion of clusters of customers, rather than individual customers. We create our clusters using the CLUSTERING process described in Algorithm 8.1, with the criteria for assessing the change from combining the clusters and for determining if the change is sufficient, being critical factors in defining the actual techniques used. Once the clusters have been created, the two clustering methods we use operate similarly to INSERTION, except we insert whole clusters in their best positions, rather than individual customers. The criteria for insertion which we use, is the ratio of additional reward to additional time, as was used for the B4BTASK method. The two clustering methods are distinguished by the method of cluster creation, and these are therefore:

- CLUSDIST: distance-based clustering, where we combine clusters if the ratio of the new distance for the combined cluster, to the distance of the two previous clusters is less than a certain threshold ratio. We found the threshold of about 0.7 to be the most effective.

- CLUSREW: reward-based clustering, where we combine clusters if the ratio of the new reward of the combined cluster to the new distance is greater than the highest of the ratios for the two previous clusters.

The results from applying these construction techniques, together with STEEPEST ASCENT, are shown in Table 8.4. In the body of the Table we give the mean reward that is obtained by each method for each problem class. The rows 'Mean' and 'Time', respectively, give the overall overall mean reward obtained and the total computational time (in the form h:mm:ss) required to implement the heuristic followed by STEEPEST ASCENT. The rows 'Un Mean' and 'Un Time' give the mean reward obtained and the computational time required by the heuristics to obtain the unimproved solutions.

Our first observation is that, under our current implementation, the insertion routines for these problems are computationally expensive. The computational time shows that the two methods which look to insert the best customer in its best position, i.e., B4B and CHEAP, take much longer for the construction phase, than the other methods. This indicates how methods that use best insertion for their construction or improvement phases, may not be applicable if computational efficiency is an important issue. The method AGGAPP, which adds the subtasks comprising a customer consecutively, takes a particularly short time to construct, although the time taken to improve upon the solutions obtained is greater than that for the other methods. The CLUSDIST method has the highest overall mean, so we will compare the other methods against this method, in order to determine for which types of problems this method is the most effective.

The lowest overall reward is obtained by the GREEDY method, whose relative performance is strongest for problems where most of the customers are in the solution and where there is a large variability between the rewards of the customers. In this case, the method attempts to place as many of the highest reward customers in their best position in the solution. This method obtains the lowest mean reward for 22 of the 27 problem classes, and so we don't consider it further.

The next lowest overall mean reward is for our method CLUSREW. This method seems to be very ineffective in constructing initial solutions, with the improvement routines being unable to make up for the deficiencies. The method of generating the clusters is quite strict, which leads to small clusters and many customers being left in clusters by themselves; thus the customers are inserted consecutively, rather than in combination with other customers. Although there is no statistically significant difference between this method and the CLUSDIST, this method appears to, in general, be inferior to the other clustering method, we won't consider it further.

Classes	Construction Techniques					
(N,L,R)	B4BTask	AggApp	ClusDist	ClusRew	Cheap	Greedy
(1,1,1)	20.88	20.86	21.00	19.60	20.88	17.86
(1,1,2)	87.24	86.75	87.61	83.12	87.25	80.82
(1,1,3)	345.39	344.85	341.89	329.22	344.64	327.27
(2,1,1)	59.97	59.88	60.08	57.45	59.97	55.87
(2,1,2)	245.87	244.49	246.38	242.65	245.66	237.31
(2,1,3)	966.97	963.06	965.53	956.06	952.34	940.35
(3,1,1)	95.49	95.52	95.86	95.99	95.49	95.14
(3,1,2)	382.82	382.77	383.95	382.86	383.62	381.42
(3,1,3)	1428.51	1424.87	1428.84	1424.81	1427.67	1429.55
(1,2,1)	21.96	21.62	21.91	20.28	21.96	18.79
(1,2,2)	93.38	93.80	93.20	90.44	93.32	85.19
(1,2,3)	368.51	364.45	363.83	349.39	363.66	334.98
(2,2,1)	58.20	58.13	58.67	56.72	58.24	50.85
(2,2,2)	239.75	237.78	242.50	239.75	240.37	220.27
(2,2,3)	909.32	927.67	934.98	920.96	909.10	880.05
(3,2,1)	94.78	94.73	95.66	95.84	94.84	93.14
(3,2,2)	380.19	379.17	384.52	383.08	382.07	376.21
(3,2,3)	1401.92	1405.39	1410.33	1406.76	1403.96	1402.25
(1,3,1)	24.14	23.53	23.85	22.00	24.14	20.34
(1,3,2)	101.16	98.66	100.53	95.00	101.32	92.18
(1,3,3)	399.64	396.25	395.04	373.90	395.38	375.15
(2,3,1)	59.66	57.79	58.89	53.96	59.66	52.48
(2,3,2)	242.84	241.31	243.15	235.29	243.95	230.44
(2,3,3)	957.59	949.84	964.26	944.93	942.69	911.51
(3,3,1)	93.06	93.30	93.72	93.65	93.06	94.32
(3,3,2)	368.42	370.85	370.47	370.30	367.28	368.06
(3,3,3)	1390.44	1399.81	1397.34	1397.79	1380.63	1406.34
Mean	401.41	401.38	403.11	397.84	399.75	391.78
Time	4:25:53	7:42:44	6:09:48	7:28:15	5:29:02	4:28:31
Un Mean	381.35	314.77	344.92	255.78	359.95	321.90
Un Time	0:29:57	0:00:27	0:05:47	0:11:23	0:29:54	0:04:21

Table 8.4: Results from Preliminary Testing for TRCP Instances

The CHEAP heuristic obtains reasonable quality initial solutions, although it does take a long time in the construction phase. The improved results appear to be dominated by the other solution methods and, since this method seems to obtain average solutions for all problems, without obtaining the highest mean reward, we do not consider it for further investigation.

The AGGAPP heuristic also obtains reasonable solutions for most of the problem classes, without appearing to be particularly effective for any of the types of problems. This method was particularly devised for problems where the tasks need to be carried out consecutively, i.e., with L = 1, but its overall mean is lower than that obtained by CLUSDIST over the problems with these loads. The B4BTASK heuristic obtains the best quality initial solutions, but with the longest time required. The time required in the improvement phase is the least with this method, since the least improvement is found. The mean obtained by this method is highest for problems with few customers serviced, and this method appears best-suited to these problems.

The most effective of the methods, overall, appears to be the CLUSDIST method. This method combines customers that are 'close' to each other into a subpath, and these subpaths are inserted into routes according to the bang-for-buck ratio. Therefore the distance is an important consideration in creating the clusters, and the reward is only considered when we evaluate the cluster insertion. This method obtains the highest mean reward for 11 of the problem classes, and seems to be the most effective method, particularly with N = 2 and/or L = 2. Therefore the most effective of the methods considered, involves creating clusters of customers that are close together geographically, and using these clusters as the basis of the solution routes. This cluster-first, route-second method takes, on average, under 0.11 seconds to create an initial solution, with the average time to create the improved solutions being approximately 6.85 seconds per instance. Due to the large contribution of the improvement phase on the computational time, we now consider the effect of the different improvement routines, in order to be able to create efficient improvement methods.

The results of this experimentation are shown in Table A.1 of Appendix A. We adapt the STEEPEST ASCENT method of Algorithm 6.2, by using only a subset of the inter-route improvement techniques. The methods given in the Table are: SA (which is our full implementation of STEEPEST ASCENT), MOVE (where only

the inter-route Move routine is carried out), NoMv (where the inter-route Move routine is not implemented), Cross (where only the Cross routine is applied), NoCr (where the Cross routine is not used), Exch (which is Steepest Ascent but with Exchange rather than Generalized Exchange), NoEx (which is Move and Cross), and Oth, which allows improvements to routes that do not affect the solution.

The Move method results in a significant saving in the required computational time (43 minutes rather than approximately 6 hours and 10 minutes), but with a large decrease in the solution quality. The NoMv method requires much longer computational time, as the far more computationally expensive Generalized Exchange routine is applied to solutions that require greater improvement. The quality of the solutions is approximately the same as for the standard routine, although NoMv appears to be slightly superior with large variability in the rewards. The additional time required in order to obtain the solution leads us to not recommend this routine as a viable option.

The routine Cross is very quick and very ineffective. The solution quality it obtains appears to be quite reasonable with few customers included and with high loads, but it appears that, in general, the Cross exchange technique is not effective for these problems. This conclusion is borne out further by the small difference between the solution quality, and time required, for SA and the NoCr routine. The Cross exchange routine swaps stand-alone paths from two different routes, and the effectiveness of this routine is reduced as longer sections of path need to be exchanged, due to the increased intertwining of the tasks.

We see that, when compared to the SA routine, the Exch routine results in a large decrease in solution time, without a large decrease in solution quality. This would suggest that the Generalized Exchange routine may not result in large enough improvement, compared to the Exchange routine, to warrant the additional computational time for these problems.

We see the actual impact of the Generalized Exchange routine when we consider the routine NoEx. The mean reward here is decreased quite significantly, compared with the SA routine, with the time requirement also dramatically decreased. This demonstrates the effectiveness of the Generalized Exchange routine, although it comes at a significant cost in terms of the computational time required.

The last improvement routine we consider, is OTH, which allows the improvement of non-solution routes, by executing the move that results in the largest increase in the maximum reward of the affected routes. This is of similar effectiveness to the SA routine and it manages to identify some improved solutions, e.g., OTH is significantly more effective than SA for the problem class with N = 1, L = 3 and R = 1. The computational time required is actually less than that for the SA routine, as the improving moves enable more far-reaching moves to be carried out later.

Our conclusions from this section are that the Cross routine is very ineffective and the Generalized Exchange routine is a huge contributor to the computational time requirements for the TRCP. Further, by using Exchange instead of Generalized Exchange, we may make make significant decreases in the computational time, without unduly affecting the solution quality.

We now consider how to obtain a good quality solution within a reasonable amount of computational time. We want to use an effective form of Tabu Search, but for the number of test problems that we consider, the computational time becomes prohibitive, with the major contributor to the computational time being the Generalized Exchange routine. The time requirements may be lessened considerably by employing the standard Exchange technique and so this is the compromise we employ in order to reduce the computational requirements.

The results of this testing are shown in Table 8.5. BEST refers to the best solution over all the methods we have employed previously, including the other methods in the Table. BESTH is the best of the solutions found by the heuristics from Table 8.4. R+IMP refers to the generation of 10 random solutions using the RANDOMCONSTRUCT construction technique of Algorithm 6.3, and improving these using the STEEPEST ASCENT method as before, whereas R+MC refers to creating 10 solutions by RANDOMCONSTRUCT and using Move and Cross to improve upon them. Our Tabu Search routines, TABUC and TABUM, are TABUIMP, except we restrict the inter-route moves considered within the search and within the STEEPEST ASCENT to just Cross and Move respectively. With TABUE we made a few minor alterations to TABUIMP in order to control the computational time requirement. We choose to use Exchange instead of Generalized Exchange, as this was previously found to reduce the computational requirement significantly. We reduced the number of non-improving moves that we allow from 50 to 25 and

we also only apply STEEPEST ASCENT when we obtain a new overall best solution, rather than a best solution for this particular iteration. We found the computational time was still prohibitive, with approximately 150 hours required to obtain the solutions.

Considering the quality of the solutions, we see that the lowest overall reward is with TABUC. The relative quality of the solutions obtained by this method decreases as the problems become less constrained, i.e, as the number of customers in the solution increases and as the load carried decreases. In these cases, longer strings of customers must be considered in order to find stand-alone paths that may be used by the Cross routine. The number of feasible exchanges and the likelihood of finding improvements decrease as the length of the stand-alone paths increase and this prevents the Cross routine from being effective, even when it is applied within a more advanced search routine, such as Tabu Search.

The TABUM method is far more effective than TABUC with our current implementation of Tabu Search. We find, by the Kruskal-Wallis Test at a 5% level, TABUM method is significantly more effective than TABUCROSS for five of the problem classes, whereas TABUC is not significantly more effective for any. Our other Tabu Search implementation, TABUE, with restricted search used, still takes a comparatively long time to obtain its solutions. This method does not appear to be effective enough to justify the large computational time, but more testing is required to determine whether or not this method is worth pursuing further. We are interested in testing the effectiveness of a full Tabu Search metaheuristic, and we test a full version on a smaller set of problems in Section 8.2.5.

The R+MC method obtains a higher mean than each of the heuristics from Table 8.4, and the computational time is quite comparable. The construction of a number of solutions, and improving each of these without using the computationally expensive Exchange routines, therefore seems to be an improved approach when the computational time available is very restricted. The R+IMP method seems to obtain reasonable results for the set of problems given. The effectiveness of this method appears to increase as the problem becomes less constrained, which suggests that the random generation of a number of solutions, followed by the effective application of improvement routines, is an effective approach for solving loosely constrained versions of these problems. Counteracting this effectiveness is

Probs	Search Routines						
(N,L,R)	BEST	BESTH	R+IMP	R+MC	TABUE	TABUC	TABUM
(1,1,1)	21.33	21.26	21.06	21.00	21.24	21.29	21.29
(1,1,2)	88.86	88.60	88.42	87.94	88.70	88.48	88.19
(1,1,3)	351.39	350.33	349.62	348.98	350.91	350.26	350.39
(2,1,1)	61.90	61.39	59.43	59.40	61.00	61.34	59.83
(2,1,2)	252.94	250.88	248.31	247.05	250.11	250.64	240.93
(2,1,3)	993.20	987.90	983.93	974.85	985.95	985.31	961.33
(3,1,1)	97.32	96.87	97.04	96.97	96.80	96.53	96.73
(3,1,2)	387.63	386.04	386.67	386.39	386.31	385.71	386.06
(3,1,3)	1439.06	1437.51	1438.51	1437.00	1437.02	1433.97	1434.89
(1,2,1)	23.20	22.91	22.36	22.26	23.04	22.10	23.08
(1,2,2)	97.78	96.85	96.79	95.31	96.71	93.91	96.00
(1,2,3)	380.23	378.63	377.51	372.89	376.47	365.74	376.31
(2,2,1)	62.20	61.15	57.68	56.97	59.38	56.17	59.55
(2,2,2)	254.06	251.51	249.06	236.25	246.34	229.30	238.66
(2,2,3)	981.24	967.09	966.84	902.26	962.93	866.06	935.11
(3,2,1)	98.07	97.52	97.53	97.33	96.93	94.54	97.17
(3,2,2)	389.73	388.57	388.50	386.40	386.79	378.73	386.52
(3,2,3)	1427.31	1424.02	1424.82	1420.21	1420.87	1394.56	1420.95
(1,3,1)	25.40	25.08	24.57	24.54	24.93	23.68	25.21
(1,3,2)	104.67	104.08	103.58	102.24	104.04	99.10	103.99
(1,3,3)	416.64	414.41	413.57	409.56	414.80	387.56	413.98
(2,3,1)	64.42	62.86	61.40	61.23	62.77	53.58	63.20
(2,3,2)	263.74	259.25	258.82	247.62	255.06	217.58	251.20
(2,3,3)	1026.23	1008.03	1014.22	958.17	1008.45	838.81	988.94
(3,3,1)	98.28	96.83	97.83	97.83	96.92	93.61	97.46
(3,3,2)	383.60	380.75	382.59	381.87	379.30	363.55	380.14
(3,3,3)	1427.95	1422.17	1426.03	1420.74	1421.70	1362.93	1421.62
Mean	415.50	412.68	412.47	405.68	411.68	393.15	408.10
Time		≈36 hrs	≈60 hrs	4h 45m	≈150 hrs	≈10 hrs	≈50 hrs

Table 8.5: Testing Search Routines for TRCP Instances

the increased computational time required for these problems, as the neighbourhoods of local search are much larger for loosely constrained problems, so the time taken in the improvement phase is increased.

The most effective, and seemingly the most efficient use of computational time, appears to be for BESTH. This would suggest that taking a number of reasonable, but varied, initial solutions and improving these, using effective search operators, is more effective than using more advanced search techniques with less effective operators. We will test this hypothesis later in Section 8.2.5.

8.1.3 The MCPTDR

The MCPTDR has previously been described by Brideau and Cavalier [20], and by Erkut and Zhang [45]. The test problems used in these papers were all of a similar form, with no time limit and most of the customers being able to be included in the solution. The only other explicit reference to problems with non-constant rewards, is in the thesis of Keller [94]. The rewards that were tested in this thesis were Gaussian functions, and therefore had all rewards being positive for the duration of the time horizon.

The two versions of the MCPTDR that we consider here can be classified as $subset, tw_j, TDR_{inc}|1||sub$, max R and $subset, tw_j, TDR_{dec}|1||sub$, max R according to the scheme we gave in Chapter 6. Recall that the MCPTDR can be defined by contraints (2)–(6) and (9)–(10) in Section 8.1.1 but with the objective function

$$\max \sum_{i=1}^{n} (1 - x_{ii}) R_i(t_i)$$

where for each customer i, the reward is a function $R_i(t_i)$ of the time at which customer i is visited by the service vehicle.

We test our methods for this problem, by creating a number of classes of problems, similar to those for the OPTW. Again we are using 60 customers in each problem. We differentiate between the problem classes according to the number of customers that can be serviced in the solution (given by the parameter N), the tightness of the time windows of availability (T), the variability in the value of the maximum reward (R) and the form of the reward functions. We set a time limit, over which the service vehicle may be operated, and create time windows within this limit, corresponding to the time for which the reward of each customer is positive. Our goal was to create a number of different types of test problems,

where we were particularly concerned with creating problems for which the time of servicing the customers is a critical concern.

For problems where few customers can be serviced (N = 1), we create sections of five customers, place the time windows around these, and set the time limit so that each of these sections are feasible, i.e., the limit is based upon the maximum time to service the sections. In this manner, we are able to restrict the number of customers that are able to be included in any solution route, so this type of problem accentuates the selection elements of the problem. For problems for which N = 2, we set the time limit to be approximately equal to the time taken for a TSP tour through each of the customers. Thus, it is possible to service each of the customers, although the form of the reward functions will determine the viability of their inclusion. With N = 3, we set the time limit to be equal to twice the TSP distance; this allows all the customers to be included, and so finding the 'best' time to service the customers is the most important concern.

We set the widths of the time windows to be respectively, between 10 and 30% of the time limit (T = 1), between 40 and 60% of the time limit (T = 2) and between 70 and 90% of the time limit (T = 3). With N = 1, we place the time windows so that the time at which the customer is served on the base section of route appears within its time window. Similarly, with N = 2, we position the time windows around the time of servicing within the TSP tour, while with N = 3 we randomly assign the time windows.

We set the variability in the maximum reward in the same manner as with the OPTW and TRCP. Thus we have all maximum rewards the same (R = 1), all maximum rewards between 10 and 20 (R = 2), i.e., low variability, and all maximum rewards between 10 and 100 (R = 3), i.e., high variability. We also generate two different forms of the time-dependent reward functions. The first form we consider is with the piecewise linear reward function, as previously shown in Figure 6.2, for which we randomly generate the time at which the customer's reward reaches its peak to be in the first half of the time window, while the time at which the reward starts to decline is randomly positioned within the second half of the time window. We set the initial reward value to be zero, and the reward at the end of the time window to also be zero, and the slopes can be calculated directly from this information. Our second form of reward function is with all rewards decreasing, and for this function we set the available reward to be equal to its maximum value

at the start of the time window, and we again set the slope so that the reward reaches zero at the completion of the time window.

We therefore have three levels of N, T and R, and two forms of reward function, which gives us 54 classes of problems. We create 60 different layouts for each class of problem, which gives us 3240 test problems in total.

We create a number of construction methods and test the effectiveness of these methods on the test problems. The construction methods that are variations on the INSERTION Algorithm of Algorithm 2.1 are:

- B4B — this is bang-for-buck where we evaluate, for each customer, the ratio of additional reward to additional time required when inserting the customer in the current vehicle. The customer with the highest ratio of reward to time is inserted in its best position. If no customer may be added to the current solution, without decreasing the reward, we start a new vehicle. This method takes into account both the reward and time considerations of any insertion and so may be used with any type of problem.

- BRIDPOT — this is an adaptation of Brideau and Cavalier's 2-step greedy method (see [20]). For each customer, we calculate the bang-for-buck ratio for including this customer next and the best other eligible customer following it. We include the customer with the highest ratio, and start a new vehicle if no customer can be added without decreasing the reward of the current vehicle. This method includes a simple form of look-ahead, as it considers the next customer after including the current customer. We adapted this method for our piecewise linear problems by evaluating the bang-for-buck ratio to account for the best ratio obtained by either servicing the customer as soon as we arrive at its location, or waiting until the reward reaches its maximum. We found that this implementation, taking into account the potential rewards, gave us improved results over just serving the customer immediately.

- REWAPP — we evaluate, for each customer, the aggregate score for appending this customer to the end of the current vehicle route. The aggregate score, $R(i)$ is given by the function

$$R(i) = \sum_{j \in S} \tau_{ij} * e^{-\mu t_{ij}}$$

where S is the set of customers that can be added after i to the current route, τ_{ij} is the reward of customer j when included after customer i, μ is a

weighting parameter and t_{ij} is the additional time to insert customer j after i. We include the customer that has the highest ratio of score to additional time for inclusion. If no customer may be added, we start a new vehicle. In our implementation, we set the parameter μ to be equal to 10.0, which placed a relatively high priority on the reward of the inserted customer in comparison to the subsequent customers.

- B4BApp — we add customers to the end of the current route, where the ratio of additional reward to additional time required is maximized. We implement this method in the same manner as BridPot, except we are only concerned with the effect on the candidate customer, rather than considering two customers.

- Cheap — we add the customer that adds the least additional time, when added to the current route in its best position. If no customer can be feasibly added to the current route, we start a new route.

An additional method we employ is called Artif, and it is an adaptation of the method ArtifInsert of Algorithm 4.1. We create artificial time windows to ensure that each customer is serviced such that a minimum proportion, k, of the customer's maximum reward is received. We use the method RewApp to construct initial solutions (which we then improve by using Steepest Ascent), and then reset the time windows to their original values and adapt the solution so as to eliminate non-compulsory waiting time. We discovered, in preliminary testing, that the best results were obtained by setting the time windows so that at least half of the maximum reward is received for each customer.

We are again using the Steepest Ascent routine, as was used for the OPTW and the TRCP, to improve upon the solutions constructed by these methods. With the MCPTDR, we are potentially able to increase the reward of a vehicle by making intra-route improvements. Therefore we use the routines of 2-opt and Or-opt (previously used for distance reduction) as methods for increasing the reward within a vehicle. The other intra-route improvement routine that we use, is the adaption of the algorithm of Dumas, Soumis and Desrosiers [43]. This method, as previously described in Chapter 4, enables us to optimally identify the best time to service the customers when we have a given vehicle route. Thus, we use this method as an improvement step, to insert waiting time to our final solution. We only apply this

routine to routes that have been changed since we last attempted to include waiting time and where the insertion of waiting time may lead us to obtain an improved solution, i.e., we consider including waiting time when the sum of the maximum rewards of the customers in the vehicle exceeds the reward of the solution vehicle with the lowest reward.

We test the effectiveness of the construction techniques on our test problems, and the results of these are given in Tables 8.6 and 8.7. The total computational time required for each of the methods is given in the form h:mm:ss in Table 8.8. We see that the improvement phase contributes the majority of the computational time for all of the methods, whereas the waiting time routine takes only just over 10 seconds, and so is not a large contributor to the time required. The quickest method is ARTIF, which is counter-intuitive, since this method involves STEEPEST ASCENT being applied to the solutions twice, whereas the other methods only use the improvement routine once. In this case, the gains that are made by ensuring the customers are initially serviced at times where their reward is high will lead to shorter time being required in the improvement phase, when the original time windows are again used, and these savings result in the shorter computational time requirement.

The least effective method is CHEAP. This method does not take into account the rewards of any of the customers, and as such does not seem to be suitable for problems with time-dependent rewards. With the MCPTDR, the time at which servicing takes place is of major importance, and since the CHEAP method is not concerned with the service time, we discard this method.

The B4B method is the next least effective method overall, although it obtains the highest unimproved rewards with Type I reward functions. The rewards obtained are only minorly increased by the insertion of waiting times, whereas the other methods' solution values are significantly enhanced, and as a result, this method is overtaken by three of the other methods. This behaviour is due to B4B being concerned only with the actual reward received when arriving at the customer, whereas the other methods take into account the potential reward for each customer. The relative ineffectiveness of the final, improved solutions, leads to us discard this method from future consideration.

B4BAPP also obtains quite poor initial solutions, but this method is greatly improved (from 526.62 to 641.13 for Type I rewards) by the inclusion of waiting

Classes	Construction Techniques					
(T,N,R)	B4B	BRIDPOT	REWAPP	B4BAPP	CHEAP	ARTIF
(1,1,1)	17.77	18.69	19.08	18.62	17.75	19.17
(1,1,2)	171.20	176.30	177.55	170.14	163.08	176.49
(1,1,3)	648.70	684.94	690.48	660.51	634.85	696.15
(1,2,1)	46.26	46.29	47.17	46.05	44.06	47.82
(1,2,2)	417.65	429.72	428.15	421.38	397.13	433.45
(1,2,3)	1636.19	1657.57	1657.24	1623.55	1550.23	1676.42
(1,3,1)	58.28	60.36	60.37	59.08	57.04	61.06
(1,3,2)	531.88	546.48	545.48	538.40	522.47	554.26
(1,3,3)	2054.82	2108.69	2095.06	2094.99	1985.51	2125.75
(2,1,1)	22.94	24.03	24.11	23.37	22.26	24.15
(2,1,2)	206.82	211.42	216.73	207.97	203.35	218.15
(2,1,3)	803.51	832.66	837.78	823.69	789.69	836.64
(2,2,1)	54.29	55.92	55.99	55.06	52.50	57.18
(2,2,2)	499.02	510.81	508.74	495.83	479.53	517.35
(2,2,3)	1911.25	1944.39	1933.14	1922.33	1809.85	1967.43
(2,3,1)	77.67	78.79	78.65	77.84	76.12	79.28
(2,3,2)	693.41	711.47	712.00	708.98	692.71	720.29
(2,3,3)	2592.32	2657.16	2632.44	2639.40	2575.11	2660.83
(3,1,1)	26.54	27.49	27.95	26.78	25.78	27.82
(3,1,2)	242.69	247.69	247.73	244.96	233.95	248.07
(3,1,3)	945.46	958.35	953.95	933.84	903.56	971.60
(3,2,1)	57.48	58.44	58.62	59.08	56.36	60.47
(3,2,2)	526.69	547.52	547.93	526.93	519.81	551.11
(3,2,3)	1965.26	2013.55	2016.66	2015.69	1879.81	2065.66
(3,3,1)	88.36	89.07	89.11	88.74	88.32	89.81
(3,3,2)	802.57	799.12	801.02	800.87	798.06	816.12
(3,3,3)	2927.80	2925.29	2933.43	2950.52	2945.62	2987.16
Mean	741.74	756.38	755.43	749.43	723.13	766.29
Unimp Mean	622.57	555.52	541.84	526.62	455.84	N/A
Wait Mean	631.78	643.25	643.62	641.13	489.30	N/A

Table 8.6: Results from Testing MCPTDR Instances – Type I

Classes	Construction Techniques					
(T,N,R)	B4B	BRIDPOT	REWAPP	B4BAPP	CHEAP	ARTIF
(1,1,1)	16.71	16.80	16.79	16.96	16.25	16.73
(1,1,2)	152.41	154.27	152.64	154.65	147.83	153.51
(1,1,3)	599.76	610.16	611.70	610.11	585.90	607.24
(1,2,1)	46.39	46.86	47.01	46.01	46.02	46.65
(1,2,2)	421.58	427.38	428.17	422.74	416.53	428.39
(1,2,3)	1601.23	1614.97	1621.64	1611.35	1557.53	1617.10
(1,3,1)	59.84	60.26	60.30	59.76	58.11	60.54
(1,3,2)	545.21	547.90	549.54	543.45	531.46	550.21
(1,3,3)	2086.31	2111.45	2113.88	2099.72	2045.01	2121.68
(2,1,1)	18.77	19.17	19.55	18.72	18.24	18.98
(2,1,2)	169.09	175.42	176.52	171.08	166.92	172.87
(2,1,3)	660.64	673.13	678.48	658.85	647.95	679.61
(2,2,1)	51.92	52.63	53.15	51.95	51.38	52.95
(2,2,2)	471.73	477.93	480.22	470.88	462.09	481.77
(2,2,3)	1778.98	1814.58	1808.69	1790.44	1733.02	1812.94
(2,3,1)	71.45	72.46	72.98	71.37	69.57	73.31
(2,3,2)	653.12	654.84	663.18	650.86	638.09	663.30
(2,3,3)	2433.60	2461.29	2475.42	2447.26	2370.70	2487.62
(3,1,1)	21.57	21.88	22.46	21.68	20.70	22.25
(3,1,2)	198.51	200.58	205.45	198.95	189.59	201.38
(3,1,3)	766.00	785.86	791.89	772.94	733.36	782.16
(3,2,1)	52.88	54.07	55.07	52.90	52.23	54.90
(3,2,2)	479.80	487.43	493.86	483.17	475.58	492.78
(3,2,3)	1863.04	1878.40	1905.35	1859.52	1808.81	1912.05
(3,3,1)	78.92	79.33	79.54	78.82	78.56	79.68
(3,3,2)	708.25	709.69	711.51	707.92	706.40	712.15
(3,3,3)	2638.34	2651.79	2658.39	2643.82	2628.56	2656.51
Mean	690.59	698.54	701.98	693.18	676.16	702.19
Unimp Mean	640.37	642.50	649.85	613.58	467.97	N/A
Wait Mean	640.37	642.50	649.85	613.58	467.97	N/A

Table 8.7: Results from Testing MCPTDR Instances – Type II

times. This is due to the myopic insertion considered, and the localized impact of the waiting times. We consider the provisional best time to add a customer to the end of the current route, and add following customers at the best allowable service time after its provisional time. Therefore we are able to improve the solution dramatically by including the waiting time that we have factored in during the construction phase, without affecting the following customers.

RewApp and BridPot are the most effective of our methods that simply involve construction and improvement, and their effectiveness is similar, i.e., the overall mean for RewApp is 728.702, while it is 727.458 for BridPot. The two methods are quite similar, with each adding a customer to the end of the current route, and employing some form of look-ahead: BridPot considers the reward for the best point that can be inserted next, whereas RewApp considers the weighted sum of the rewards of all customers that can be inserted next. Both of these techniques take into account the potential reward for a customer, which results in the initial solutions being dramatically improved by the subsequent inclusion of waiting times.

The last method, Artif, appears to be the most effective of the methods considered. It requires the least computational time, and appears to be particularly effective on the problems with type I reward functions. Since the construction phase for this method involves the application of Steepest Ascent to the solution obtained on a restricted form of the problem, the results obtained in this construction phase are not directly comparable to those from the construction phase for the other methods, hence, the omission of results from construction in Tables 8.6, 8.7 and 8.8. The Artif method doesn't appear to be well-suited to problems with all rewards decreasing, although it can be effective when all customers may be serviced so that a high reward is obtained. For general problems where rewards are decreasing, we would recommend the use of the RewApp method, but for other problems, the Artif method appears to be the most effective of the methods we have considered.

We tested the effectiveness of the method of Erkut and Zhang [45] on our test problems with Type II rewards. This yielded an average reward of 682.253, which is well below the reward found by the other methods. The form of the test problems considered, i.e., having the time-dependent rewards placed within time windows, does not allow it to work effectively, and so this method appears to not be adaptable to problems of this type.

	Construction Techniques					
Phase	B4B	BridPot	RewApp	B4BApp	Cheap	Artif
Initial	0:18:33	0:07:54	0:06:37	0:00:33	0:06:43	N/A
Wait	0:18:43	0:08:04	0:06:54	0:00:46	0:06:56	N/A
Final	3:31:56	3:40:14	3:40:03	4:04:35	3:47:16	2:25:54

Table 8.8: Overall Computational Requirements of MCPTDR Heuristics

We also concern ourselves with the effectiveness of the different routines that comprise the Steepest Ascent routine. We use the BridPot heuristic to construct our initial solutions, as this method appeared to create effective solution routes to the problems. This method obtained mean rewards that were similar to those obtained by the RewApp method, with these two methods appearing to be the most effective of our methods that simply construct solutions and apply Steepest Ascent, but we arbitrarily select the results obtained by the BridPot heuristic to explore further.

The results from this testing are shown in Tables A.2 and A.3 in Appendix A. The routines considered are: Ascent (which is our full Steepest Ascent routine), NoIntra (which is Steepest Ascent without the intra-route improvement) NoMove, NoExch and NoCross (which are the Steepest Ascent routine without the Move, Generalized Exchange and Cross routines respectively), and IntExch (which is the Steepest Ascent routine without either Move or Cross). The computational time required for these routines are respectively, 3 hours and 40 minutes, 2 hours and 41 minutes, 4 hours and 40 minutes, 2 hours and 9 minutes, 3 hours and 5 minutes, and 4 hours and 4 minutes.

The results show that the intra-route improvement routine is a large contributor to both the computational time and the solution quality. The overall mean over the two problem types is reduced from about 728.702 to 696.485 by the removal of the intra-route improvement. Of particular interest are the significant improvements that are made using the intra-route improvement with our least constrained problems, i.e., with wide time windows and with customers being easily served. In these problems, most of the customers can be placed in the initial solution and the intra-route improvement routines are the only ones that can be effective once all customers are in the solution route. In fact, with rewards just decreasing, the only routine that is effective is the intra-route improvement, as all other combinations

of the inter-route improvement routines result in the same solution being obtained. Therefore we conclude that the intra-route improvement routine is essential for creating good solutions, particularly when most of the customers are included in the solution.

We can see that the Generalized Exchange routine is again relatively computationally expensive. The routines in which Generalized Exchange is the first inter-route improvement routine that is used, i.e., NoMove and IntExch, result in higher computational times, as the Generalized Exchange routine is called to improve upon the unimproved solutions. The fact that there is a slight improvement in the overall solution with NoMove shows the ineffectiveness of the Move routine, compared with the Generalized Exchange routine. The absence of the Generalized Exchange routine significantly decreases the computational time, and reduces the overall mean over the two problem sets from 728.702 to 717.633.

Our next concern is with the effectiveness of our methods with regard to the time of servicing the customers. We are interested in problems where the solution may be improved by inserting waiting time, and how effective our methods are for these problems. We created simple construction methods, based upon Insertion, and used Steepest Ascent to improve upon the constructed solutions. The difference in these methods, is that in the evaluation of a potential perturbation of a solution, instead of evaluating the reward as it would be if the customer were serviced immediately, we consider the best time at which the customer would be serviced in the new route. For this purpose, we use the method of Dumas et al. [43] to find the best times at which to service each customer for our candidate perturbation. This approach is only relevant for problems where there is potential for increasing the reward by not servicing the customer immediately, so we only apply these methods to problems where the rewards are not strictly non-increasing, i.e, with type I reward functions.

Our construction methods are GreedyWait, in which the customer that adds the most to the reward of the current route is added in its most lucrative position, and B4BWait, in which we evaluate the ratio of additional reward to additional time for each potential insertion and we perform the insertion with the highest ratio. With each of these methods we create a new route if no customer can be added to the current route. When we carried out these construction techniques on our full set of test problems, i.e., the 1620 problem instances containing Type I

rewards, the mean reward obtained and computational time required is 697.726 in 7 hours and 13 minutes for GREEDYWAIT and 664.432 in 21 hours and 19 minutes for B4BWAIT. Therefore the time required to construct the initial solutions exceeded the total time to create the improved solutions via the previous heuristics.

The increased computational time requirement for the exact evaluation, led us to attempt to reduce the computational time required, whilst still carrying out essentially the same evaluation; the modification we attempted was to estimate the reward that would be received for a particular perturbation. This estimation involved identifying the allowable time window during which each customer of the current route could be serviced, and assuming that each customer is serviced at the time that gives the maximum reward within this interval. This gives an upper bound on the actual reward received, calculable more efficiently than the full calculation. We exactly evaluate the perturbation with the highest estimated reward change, and carry out this perturbation if the solution is improved. We applied this estimation with our new method GWIMP, which consists of GREEDYWAIT followed by STEEPEST ASCENT, where each evaluation involves the potential reward by including waiting times. The total computational time required for GWIMP was 26 hours and 7 minutes. We compare the results from this method compared with ARTIF, which was the best method we found for our Type I problems, and the results are shown in Table 8.9.

From this Table, we see that the overall mean for GWIMP is 750.226, compared with 766.285 for ARTIF, and so our approximation method is less effective than that found using the standard approach. The problems for which we would expect waiting times to be the most important,are the ones with high slopes and, with our problem sets, these are the problem classes with narrow time windows. What is of some promise is that, for these tight time window problems, the waiting method is superior; using the Kruskal-Wallis Test at a 5% level, we find that of the 27 problem classes we tested, GWIMP was superior to ARTIF for two classes, with both of these having tight time windows.

Because of the potential shown in the above testing, we tested GWIMP for all of the problem classes with tight time windows, with the exact changes evaluated for each potential step. The comparison for GWIMP with ARTIF for these problems is given in Table 8.10. The total computational time required for GWIMP was 19 hours and 26 minutes. We see from this Table that the GWIMP method is

Classes	R=1		R=2		R=3	
(T,N)	GWIMP	ARTIF	GWIMP	ARTIF	GWIMP	ARTIF
(1 1)	18.349	19.172	171.368	176.492	683.607	696.152
(1 2)	46.442	47.822	427.825	433.447	1671.498	1676.425
(1 3)	61.571	61.061	562.504	554.262	2171.236	2125.749
(2 1)	21.882	24.154	201.872	218.145	796.838	836.642
(2 2)	52.791	57.184	491.765	517.346	1893.673	1967.432
(2 3)	79.248	79.278	719.124	720.287	2664.100	2660.825
(3 1)	24.562	27.824	228.289	248.069	916.022	971.595
(3 2)	56.599	60.472	523.247	551.115	1959.119	2065.657
(3 3)	89.589	89.811	804.921	816.121	2918.070	2987.165
Mean	50.115	51.864	458.991	470.587	1741.574	1776.405

Table 8.9: Mean Rewards with Estimated Evaluation

Classes	R=1		R=2		R=3	
(T,N)	GWIMP	ARTIF	GWIMP	ARTIF	GWIMP	ARTIF
(1 1)	19.456	19.172	179.143	176.492	709.902	696.152
(1 2)	49.551	47.822	452.984	433.447	1756.496	1676.425
(1 3)	62.358	61.061	566.791	554.262	2189.251	2125.749
Mean	43.789	42.685	399.639	388.067	1551.883	1499.442

Table 8.10: Mean Rewards with Exact Evaluation

superior to the ARTIF method for the problems with tight time windows, as it obtains a far higher overall mean (665.104 compared with 643.398). When we test the significance of this difference via the Kruskal-Wallis Test at a 5% level, we find that GWIMP is superior for seven of the problem classes. Since ARTIF was the best of our conventional methods, this result shows that these methods are not able to deal as effectively as specialized techniques with the reward functions that evaluate the exact impact of increasing reward functions.

Our final concern with the MCPTDR, is with the absolute quality of the solutions obtained. To test these, we carried out a number of search strategies, with greater computational time allowed. The results from this testing are shown in Tables 8.11 and 8.12. The column BEST, includes the results from each of the methods we consider, and is used as an estimate of the optimal solution. BESTH refers to the best score obtained by the six heuristics considered in Tables 8.6 and 8.7. TABUMV

and TABUEX are versions of TABUIMP, with TABUMV only involving the Move routine, whereas TABUEX includes the Exchange routine rather than Generalized Exchange. The one alteration to this routine that we use, is to include waiting time in the solution after we have found a new best solution or if we have made 10 moves without improving the solution, and this enables a periodical examination of the exact reward that can be obtained within the current route. RAND10 involves finding 10 random solutions (using RANDOMCONSTRUCT) and using STEEPEST ASCENT to improve upon these initial solutions. Due to the potential shown by the GWIMP method, we also implement this method, although the computational time required was quite excessive, even though we only use this method with Type I reward functions.

The TABUMOVE routine takes a total of 19 hours and 24 minutes of computational time and it is clearly the least effective of the methods. This shows that, with our current implementation of Tabu Search, the more complex search routine is unable to compensate for the relative ineffectiveness of the Move routine, so other improvement routines are needed in order for our methods to be effective.

The GWIMP routine, is very computationally expensive, as it takes approximately 120 hours in order to obtain solutions for the 1620 problems with type I rewards. This method provides the second least overall mean reward, although the particular strengths of the method are that it performs well when there are steep slopes and much to be gained from waiting (as is the case with the problems we have created with tight time windows), and when all the customers may be easily placed in the solution, in which case, the greatest improvements are made by selecting better times at which to service the customers.

The total computational time required to achieve the column BESTH is 20 hours and 22 minutes, which shows that it is an effective use of the computational time. The overall means for BESTH and RAND10 are higher than for TABUEXCH for the problems with type II reward functions, which we believe is due to the importance of the Generalized Exchange routine for these rewards. We didn't carry out the Tabu Search using Generalized Exchange, due to the prohibitive nature of the computational time for the large number of problem instances, but we believe that a full Tabu Search heuristic, including Generalized Exchange, will be superior to these other methods. If computational time is of major importance, we would recommend the use of a number of varied construction techniques, combining these

Classes	Methods					
(T,N,R)	BEST	BESTH	TABUMV	TABUEX	RAND10	GWIMP
(1,1,1)	20.33	19.64	18.94	**19.93**	19.36	19.46
(1,1,2)	187.41	183.30	174.77	183.71	176.29	179.14
(1,1,3)	737.39	721.59	683.27	716.74	696.72	709.90
(1,2,1)	50.43	48.73	43.80	49.11	48.03	49.55
(1,2,2)	458.71	444.96	401.92	449.24	437.49	452.98
(1,2,3)	1769.55	1714.97	1534.32	1720.94	1692.02	1756.50
(1,3,1)	63.70	62.16	56.71	62.98	61.68	62.36
(1,3,2)	577.06	564.36	517.00	566.46	559.37	566.79
(1,3,3)	2217.91	2171.28	1973.48	2172.97	2153.17	2189.25
(2,1,1)	25.34	24.94	23.93	25.18	24.64	23.20
(2,1,2)	229.30	225.72	216.86	227.75	221.69	211.59
(2,1,3)	886.23	870.58	844.52	876.85	863.40	832.91
(2,2,1)	59.78	58.45	53.25	59.04	58.37	56.87
(2,2,2)	545.35	533.20	490.08	537.91	533.70	521.67
(2,2,3)	2057.96	2016.90	1843.66	2010.93	2025.00	1991.81
(2,3,1)	82.46	80.80	80.44	81.57	80.90	80.18
(2,3,2)	744.17	733.47	725.25	735.47	731.72	726.27
(2,3,3)	2755.53	2719.82	2687.69	2717.55	2718.83	2704.19
(3,1,1)	29.65	28.98	27.48	29.17	28.67	27.03
(3,1,2)	268.41	261.75	251.53	264.99	263.59	249.49
(3,1,3)	1038.75	1016.59	969.07	1026.95	1016.03	952.58
(3,2,1)	63.62	62.17	56.85	62.64	62.50	59.64
(3,2,2)	580.00	568.14	524.25	571.83	569.69	545.01
(3,2,3)	2160.77	2109.43	1950.05	2124.85	2121.32	2043.55
(3,3,1)	92.40	90.89	91.27	91.50	90.80	91.61
(3,3,2)	832.25	822.27	821.84	826.63	820.06	825.85
(3,3,3)	3051.82	3009.67	3012.35	3034.37	3014.31	3028.68
Mean	799.49	783.88	743.50	786.94	781.09	776.22

Table 8.11: Mean Rewards with Different Search Strategies - Type I Rewards

Classes	Methods				
(T,N,R)	BEST	BESTH	TABUMV	TABUEX	RAND10
(1,1,1)	17.44	17.36	17.30	17.41	17.32
(1,1,2)	159.89	158.81	158.40	159.85	158.16
(1,1,3)	628.62	626.06	624.55	628.11	623.32
(1,2,1)	48.11	47.71	46.64	47.99	47.62
(1,2,2)	438.09	435.18	425.21	436.30	434.57
(1,2,3)	1656.65	1646.77	1602.39	1649.97	1644.25
(1,3,1)	61.73	61.37	59.40	61.22	61.04
(1,3,2)	561.72	557.99	542.69	557.81	555.07
(1,3,3)	2153.22	2143.66	2087.94	2132.61	2131.99
(2,1,1)	20.01	19.85	19.75	19.89	19.77
(2,1,2)	182.44	180.53	178.97	181.40	180.00
(2,1,3)	709.66	703.90	699.79	707.07	702.28
(2,2,1)	54.99	54.29	53.50	54.51	54.25
(2,2,2)	498.25	493.57	481.95	493.95	490.89
(2,2,3)	1877.46	1853.94	1825.39	1858.71	1854.62
(2,3,1)	75.11	74.40	73.94	74.21	74.26
(2,3,2)	683.37	676.96	668.92	674.49	675.56
(2,3,3)	2543.83	2517.66	2507.25	2502.76	2521.41
(3,1,1)	23.26	22.99	22.90	23.14	22.84
(3,1,2)	212.23	210.39	209.81	210.79	210.44
(3,1,3)	827.52	820.14	811.88	821.60	820.18
(3,2,1)	56.85	56.12	55.23	56.18	56.33
(3,2,2)	514.20	505.64	503.41	509.46	508.55
(3,2,3)	1977.38	1950.09	1916.88	1944.17	1959.53
(3,3,1)	80.65	80.21	80.06	80.03	80.35
(3,3,2)	723.38	719.57	717.00	719.13	720.24
(3,3,3)	2695.72	2679.64	2677.34	2673.14	2684.70
Mean	721.55	715.36	706.24	714.66	715.17

Table 8.12: Mean Rewards with Different Search Strategies - Type II Rewards

with STEEPEST ASCENT, to obtain good solutions in reasonable time.

One important feature to note with the TABUEXCH method, is that the moves involved with the Tabu Search phase all involve inter-route improvement routines, with intra-route improvement routines being used for intensification purposes only. Therefore, if all the customers are included in the solution, as is often the case for the problem classes with N = 2 or 3, then the Tabu Search is unlikely to be effective, so intra-route improvement routines would be more appropriate to be included in the search routine.

8.1.4 The Combined MCP

After testing the individual components that comprise the MCP, we are now in position to test the overall problem. The MCP is a problem consisting of tasks and time-dependent rewards, which are based on time windows, and is classified as $subset, tw_j, T_{fix}, TDR_{inc}|1||sub, \max R$ according to the scheme from Chapter 6.

The Combined MCP can be defined by the contraints (12)–(24) in Section 8.1.2 but with the objective function

$$\max \sum_{i=1}^{n} y_i R_i(t_{i1}, t_{i2}, .., t_{in_i})$$

where for each customer i, the reward is a function $R_i(t_{i1}, t_{i2}, .., t_{in_i})$ of the time at which some or all of the subtasks of customer i are visited by the service vehicle. The time window constraints for the OPTW can also be included, in order to restrict the time of servicing some of the sub-tasks.

For provisional testing, we have selected five parameters of the test problems, and created classes of problems according to these parameter settings. The parameters we select are: the numbers of customers that can be serviced (N), the time window strictness (T), the load that is carried (L), the variability in the maximum reward value (R) and the type of reward function that we consider.

We create 54 classes of test problems, by using three values of N and T, and two of L, R and type ofreward function. For the numbers serviced, we use the same settings as for the MCPTDR. With low numbers included (N = 1), we create small sections of route, to base our constraints upon, and we ensure that these sub-routes are feasible, candidate solutions. With medium numbers serviced (N = 2), we create a feasible route containing all the customers, and set the time limit so this route is a feasible solution to the problem. With customers being

easily serviced (N = 3), we create a route containing all the customers, use this to generate our task constraints, and we set the time limit to be twice the time taken to visit all the customers, with the time windows randomly applied the within this time limit. For time window strictness, we set the time windows so that the widths are, respectively, between 10 and 30% (T = 1), 40 and 60% (T = 2) and 70 and 90% (T = 3) of the time limit.

For load sizes, we selected two levels, so that either only one customer load may be carried at a time (L = 1), or up to six customers' loads may be carried at a time (L = 2). With variability in the maximum reward, we use the low variability (R = 1) and high variability (R = 2) options, as used previously, i.e., reward is from the distribution $U(10, 20)$ or $U(10, 100)$. For both the reward variability and the load size, instead of using three levels (as used when the relevant constraints were applied in isolation) we used two levels, in order to reduce the number of classes we consider.

The reward functions we use are the piecewise linear function, as previously shown in Figure 6.2, and the TASKTDR reward function of Figure 6.4. With the piecewise linear function, we generate the width for which the reward is at its maximum level, to be a proportion, p, of the overall time window width, with $p \in (0, 0.25)$, and we randomly position this interval within the overall time window. This creates problems for which the time at which the maximum reward is available is quite low, so the slopes of the reward functions are reasonably low. With the TASKTDR function, we penalize the rewards for deviations from the desired service times, and for excessive time that the customer is in the service vehicle. When we have high loads, each customer must be carried out consecutively, so there is no flexibility allowed in the routing and, thus, the riding time will always be equal to the minimum riding time. In this case, the TASKTDR function is equivalent to our piecewise linear function, so we exclude this combination from our test problems. For the TASKTDR function, we placed the time window at which the reward is at its maximum in the middle of the time window of availability and set the slopes so that the reward reached zero at the extremes of the time window. Our penalty for excessive time in the vehicle was set so that the maximum riding time allowed was between two and three times the minimum riding time. This reward function, as we have set it, is theoretically able to become negative, but we consider the minimum reward possible to be equal to zero. We are able to apply our adaptation

of the routine of Dumas et al [43] to these problems, in order to find waiting times within a fixed vehicle route (as was discussed in Section 6.3.3). Were the reward of a customer be allowed to become negative, our technique will not guarantee to obtain the optimal solution. Our calculations assume that the penalty for excessive riding time penalty is unbounded, and so when a reward becomes zero, the effect of the riding time penalties are exaggerated. We believe that this isn't a serious shortcoming, as cases with zero reward are unlikely to feature in good solutions.

As was the case with our previous preliminary analyses, we randomized out a number of features of the problem, which were: the form of the layout, the method of generating the routes on which to base the constraints and the number of points in the tasks (which again we take to be either two or three). We set the number of points which comprise the tasks to be equal to 60, as this was a number that allowed manageable computational times for the previous problems. Again we create 60 test problems for each problem class, which gives us a total of 3240 test problems.

The solution methods that we employed are again variants of INSERTION, and these are:

- GREEDY — we insert customers in decreasing order of their maximum reward value. We insert the customer into the first available vehicle, where the customer is inserted in a position so that the reward received is equal to the maximum available reward and none of the rewards of customers in the current solution are affected. This routine involves the insertion of non-compulsory waiting time, in order to allow the maximum reward to be obtained and, when the reward function involves penalties for the excess riding time, the points of a task must also be serviced consecutively in order to obtain the maximum reward.

- B4B — this is bang-for-buck, where the customer that has the highest ratio of additional reward to additional distance is inserted to the current vehicle. If no customer can be inserted, a new vehicle is started.

- B4BAPP — this is also bang-for-buck, but the customer is only considered for adding to the end of the current route. The time at which the customer is considered to be serviced is either immediately, or when it achieves its maximum reward, depending on which gives the maximum bang-for-buck ratio. If no customer can be inserted, a new route is started.

- CHEAP — the customer that adds the least distance to the current route is added, and if none can be added, a new route is created. This method does not consider the reward values at all, and is therefore suited for problems for which routing considerations are important.

- CLUSREW — we create clusters of customers, using the CLUSTERING method of Algorithm 8.1, in which clusters are combined if the bang-for-buck ratio of the combined cluster exceeds the maximum of the ratios for the clusters separately. When creating the clusters, we impose artificial time windows on the final point of the task, to ensure that the customer is serviced with reasonable reward. The best setting we found was to ensure that each customer was serviced so that it received at least 90% of its maximum reward. With reward functions that depend upon the riding time, artificial time windows are also placed on the first point of a task, to restrict the riding time penalty. This is an adaptation of the artificial time windows that were found to be effective for the MCPTDR, and it is used to ensure that the reward functions of the customers within the cluster are compatible. These clusters are inserted into routes according to the bang-for-buck ratio of additional reward to additional time, and a new route is created if no cluster can be added to the current route.

- CLUSTIME — we, again, use the CLUSTERING method of Algorithm 8.1, and we combine clusters if the total time for the combined cluster is less than some proportion of the time to service the clusters separately. The most effective value of the proportion was found to be 0.7. We insert the clusters in the same manner as for CLUSREW.

We tested our heuristics on the 3240 test problem instances, and the results are shown in Tables 8.13 and 8.14. The computational times required for these methods are given in Table 8.15, in the form hh:mm:ss. From these Tables we see that the CHEAP method is clearly the least effective method considered and, as for the MCPTDR, the lack of concern for the reward values leads to this method being inappropriate with time-dependent rewards. The B4B method is the most effective at constructing initial solutions, particularly when non-compulsory waiting times are not allowed. These two methods have the least total computational time, although they have by far the greatest time requirements for the construction

phase. We attribute the low computational time requirement for the final solution for these methods to be due to the inability to obtain good solutions for CHEAP and the few improvements possible from the solution found by B4B. The length of time required in the construction phase shows just how computationally expensive insertion routines can be for the combined MCP. These two methods are the only ones that consider the insertion of each customer in each position of the current route, and thus the time requirement is large. The B4B method also evaluates the reward for each customer for each position, and the additional time for this method over the CHEAP method is in part due to the computational requirement of this evaluation.

The preliminary analysis of the results shows that the other four methods each have particular types of problems for which they are best-suited. The method B4BAPP takes very little time to construct, and since it considers the potential reward of each insertion, we see that it is able to be significantly improved by the addition of waiting times. This method has the highest mean for problems for which $N = 1$, since the consideration for the reward and the additional time, as well as its consideration for the potential reward, make it the most effective when few customers are served.

The GREEDY method is another that obtains initial solutions very quickly and this method obtains the highest mean for the problem classes with $N = 3$. This strength is consistent with the manner in which the method attempts to service as many customers as possible so that their maximum reward value is obtained. The CLUSREW method has the highest overall mean reward value, and appears to be quite robust. The CLUSTIME method obtains the highest overall means for type II reward functions, but since it is not directly concerned with the rewards of the customers when creating the clusters, it will not be very robust.

Our preliminary conclusions are, thus, that the most effective method is CLUSREW, and this would be the method we would, in general, use. If we knew that few of the customers would be included in the solution, we would use the B4BAPP method and if the customers were able to be easily inserted, we would use the GREEDY method. As anticipated, the Combined MCP appears to create a wider range of instances than the previous types of problem we considered, and so it appears to favour different methods in different situations, rather than favouring particular methods for all problem instances.

Classes	Methods					
(T,N,L,R)	GREEDY	B4B	B4BAPP	CHEAP	CLUSREW	CLUSTIME
(1 1 1 1)	75.77	77.00	77.64	74.84	75.33	73.89
(1 1 1 2)	295.28	315.35	321.00	308.48	308.32	302.13
(1 2 1 1)	261.78	266.95	263.51	260.50	265.21	266.31
(1 2 1 2)	990.75	1000.90	982.31	978.68	1005.53	1004.40
(1 3 1 1)	294.83	290.63	307.15	289.97	290.16	288.42
(1 3 1 2)	1112.71	1101.24	1111.73	1075.86	1109.05	1090.02
(2 1 1 1)	90.28	93.97	95.09	92.39	92.58	93.48
(2 1 1 2)	359.89	367.12	367.46	360.38	363.00	358.54
(2 2 1 1)	273.60	279.40	274.00	266.92	279.82	276.33
(2 2 1 2)	1086.82	1077.99	1038.05	1028.42	1099.82	1071.11
(2 3 1 1)	345.06	335.07	340.39	332.29	339.60	333.36
(2 3 1 2)	1319.14	1286.67	1284.43	1270.58	1305.69	1285.96
(3 1 1 1)	103.83	105.84	106.95	103.73	106.08	106.33
(3 1 1 2)	401.08	409.50	410.03	395.57	399.83	411.21
(3 2 1 1)	288.09	290.91	285.40	284.35	297.57	290.70
(3 2 1 2)	1124.85	1119.86	1078.98	1065.95	1117.14	1124.42
(3 3 1 1)	368.30	358.23	362.53	359.91	356.53	357.93
(3 3 1 2)	1375.60	1349.16	1335.37	1355.14	1332.93	1342.95
(1 1 2 1)	72.69	74.62	75.19	73.29	72.40	72.14
(1 1 2 2)	292.25	292.95	293.29	285.63	287.78	291.29
(1 2 2 1)	253.43	258.55	252.01	246.07	255.78	258.51
(1 2 2 2)	967.55	1003.48	990.10	923.64	988.73	994.09
(1 3 2 1)	311.93	311.13	312.03	301.78	314.66	310.07
(1 3 2 2)	1187.20	1188.99	1178.09	1124.71	1201.13	1175.37
(2 1 2 1)	91.82	92.10	93.42	90.21	92.02	90.28
(2 1 2 2)	356.27	361.05	361.08	352.59	349.56	350.47
(2 2 2 1)	260.04	269.80	260.83	249.99	269.15	273.56
(2 2 2 2)	994.81	1005.80	1004.31	930.40	1014.83	1024.65
(2 3 2 1)	356.08	349.62	347.86	329.66	351.80	352.93
(2 3 2 2)	1322.32	1304.26	1303.86	1234.37	1328.00	1325.45
(3 1 2 1)	98.70	99.07	100.05	98.73	100.34	98.61
(3 1 2 2)	392.24	391.97	399.86	384.89	384.18	395.07
(3 2 2 1)	266.31	271.81	270.57	259.13	276.53	277.39
(3 2 2 2)	1039.19	1048.08	1040.40	982.73	1065.96	1054.65
(3 3 2 1)	379.82	372.01	369.48	368.69	377.95	374.85
(3 3 2 2)	1382.84	1367.49	1362.27	1345.89	1387.34	1375.02
Unimp	350.25	469.87	297.91	341.53	350.63	423.13
Wait	350.25	490.41	465.14	389.34	393.39	449.12
Total	560.92	560.79	557.13	541.29	562.84	560.33

Table 8.13: Testing Methods Reward Type I for Combined MCP

Classes	Methods					
(T,N,L,R)	GREEDY	B4B	B4BAPP	CHEAP	CLUSREW	CLUSTIME
(1 1 2 1)	60.03	63.26	65.28	62.676	62.02	63.27
(1 1 2 2)	239.14	244.88	252.57	239.88	242.70	238.63
(1 2 2 1)	190.86	191.24	191.80	178.80	191.66	194.95
(1 2 2 2)	733.91	741.24	735.45	678.42	732.27	744.96
(1 3 2 1)	238.11	219.08	234.05	209.21	235.78	241.27
(1 3 2 2)	904.51	828.73	874.15	788.19	889.25	910.26
(2 1 2 1)	77.79	78.79	79.09	78.08	78.29	79.39
(2 1 2 2)	302.58	308.55	310.82	297.87	300.55	306.00
(2 2 2 1)	203.94	204.62	205.73	184.81	206.91	206.24
(2 2 2 2)	800.17	803.65	790.02	716.87	797.91	794.65
(2 3 2 1)	292.84	279.53	286.39	271.89	291.88	296.42
(2 3 2 2)	1102.84	1095.13	1112.57	1065.45	1120.19	1146.60
(3 1 2 1)	85.52	86.79	88.94	85.25	85.43	86.83
(3 1 2 2)	342.67	350.45	351.94	342.40	344.22	345.90
(3 2 2 1)	209.40	208.22	208.38	194.11	208.37	208.63
(3 2 2 2)	797.04	798.57	793.48	726.33	798.58	803.84
(3 3 2 1)	304.01	301.02	289.56	298.02	300.09	302.94
(3 3 2 2)	1159.40	1143.50	1096.52	1132.52	1150.32	1149.23
Unimp	265.62	365.37	207.50	148.66	217.98	334.32
Wait	265.62	368.12	383.70	156.28	233.73	343.53
Total	446.93	441.51	442.60	419.49	446.47	451.11

Table 8.14: Testing Methods Reward Type II for Combined MCP

	Construction Techniques					
Phase	GREEDY	B4B	B4BAPP	CHEAP	CLUSREW	CLUSTIME
Initial	00:00:24	01:24:55	00:00:19	00:29:54	00:05:38	00:03:58
Wait	00:00:24	01:25:29	00:00:25	00:30:00	00:05:42	00:04:00
Final	14:02:59	05:29:08	12:34:36	05:50:30	13:16:32	08:16:25

Table 8.15: Overall Computational Requirements of Combined MCP Heuristics

We also consider the effect of the different improvement routines, on the effectiveness and computational time requirements of the heuristic. We compare the routines using the construction method CLUSREW, and the results are given in Tables A.4 and A.5. The routines considered are ASCENT (which is our full implementation of STEEPEST ASCENT), MOVE and CROSS (which are the intra-route improvements with just Move and Cross respectively), NOINTRA (which is ASCENT without the intra-route improvement routines) NOMOVE (which is ASCENT without the Move routine) and EXCH (which is ASCENT using the Exchange routine instead of Generalized Exchange). The computational times of these routines are respectively: 13 hours and 16 minutes, 1 hour and 14 minutes, 48 minutes, 11 hours and 48 minutes, 30 hours and 29 minutes and 6 hours and 16 minutes.

We see that the routines that only use Move and Cross terminate quickly, but are also very ineffective. The Cross routine is more effective than Move when L = 1 (means are 456.527 vs 429.123), while Move is more effective with L = 2 (means are 531.030 vs 551.648). This is due to the inability of Cross to be effective when the tasks are inter-linked within the routes. The ineffectiveness of these versions of STEEPEST ASCENT, compared with the full routine, illustrates that the Generalized Exchange routine is very effective.

The removal of the intra-route improvement routines reduces the effectiveness of STEEPEST ASCENT routine significantly. The computational time is reduced by approximately 11%, which indicates that the intra-route improvement routines are not a large contributor to the computational time requirement. The reduction in the solution quality is most noticeable when N = 3 as, in these problems, the mean objective value with STEEPEST ASCENT is 760.130 and without intra-improvements the mean is 680.235. When N = 3, all the customers can be included in a single route, in which case the intra-route improvements become more important routines, as the inter-route improvement routines will not be applied within STEEPEST ASCENT.

By using the Exchange routine, instead of Generalized Exchange, less than half of the computational time is required. The solution quality is significantly affected also, but this would form a compromise between the computational time and solution quality. By removing the Move routine, we find that improvements in the solutions obtained may be made. This improved solution quality, with an increase in computational time, is due to the additional employment of the

Generalized Exchange routine, which again shows the effectiveness of this routine.

When creating efficient improvement routines for the Combined MCP, we need to consider the implications of the routines. The most effective results we found were with the NoMove method, which involves the repeated application of the Generalized Exchange routine followed by Cross exchange (although this also proved to be by far the most time-consuming). Significant gains were made by applying this method, thus, if time allows, this would be the implementation of Steepest Ascent that we would use to obtain effective solutions to these problems.

Our next consideration is to attempt to find effective methods using more advanced search techniques. As we found previously, the computational time required in order to solve the 3240 test problems is quite significant, and so the efficiency of the routines is an important consideration. We carry out four versions of local search for our test problems, and the results are shown in Tables 8.16 and 8.17. The column Best shows the mean objective value for each problem class, which is found by taking the highest reward received by one of the methods from the Table for each problem instance. The column BestH gives the mean of the highest rewards found by the six heuristics featured in Tables 8.13 and 8.14. The method Rand involves taking the best solution of six random solutions (using RandomConstruct), each of which are improved by Steepest Ascent. R10NoEx, involves 10 solutions found by RandomConstruct, with each being improved by a version of Steepest Ascent that does not include the Generalized Exchange routine. The final two routines, TabuMv and TabuCr, are versions of TabuImp but with the only inter-route routine allowed being respectively Move and Cross.

Tables 8.16 and 8.17 show that the two Tabu Search methods are very ineffective for these problems. This shows that the routines considered (Move and Cross) are not able to effectively be used within our current implementation of Tabu Search for the problem classes of the Combined MCP that we consider. The R10NoEx method is mostly quite ineffective, compared to the other routines considered, although it does appear to be best suited for problems where customers are easily serviced ($N = 3$) and time windows are not too strict ($T \geq 2$).

The difference between the Rand6 and BestH methods, is that Rand6 involves finding six random solutions, whereas BestH uses six varied construction methods. The results show that the generation of varied, well-constructed solutions,

Classes	Methods					
(T,N,L,R)	BEST	BESTH	RAND	R10NOEX	TABUMV	TABUCR
(1 1 1 1)	82.79	81.80	78.21	78.98	76.67	81.38
(1 1 1 2)	337.77	332.59	316.46	320.77	314.63	331.00
(1 2 1 1)	279.03	276.20	273.65	269.82	248.30	274.117
(1 2 1 2)	1054.15	1044.43	1037.32	1021.81	927.50	1032.39
(1 3 1 1)	318.28	315.66	304.07	305.87	292.78	300.55
(1 3 1 2)	1185.53	1169.34	1145.92	1154.78	1103.07	1146.26
(2 1 1 1)	98.42	97.75	95.98	95.90	93.21	97.41
(2 1 1 2)	384.49	382.79	375.71	376.65	368.93	378.49
(2 2 1 1)	291.67	288.83	287.76	283.45	254.45	287.58
(2 2 1 2)	1135.68	1125.89	1125.34	1116.79	1027.28	1115.48
(2 3 1 1)	359.01	353.61	353.99	353.70	346.31	350.67
(2 3 1 2)	1373.19	1359.94	1353.14	1361.29	1329.29	1344.82
(3 1 1 1)	111.29	110.56	107.67	108.48	106.29	110.29
(3 1 1 2)	432.26	429.64	421.04	422.81	414.52	427.58
(3 2 1 1)	307.32	304.68	303.62	298.43	274.83	302.97
(3 2 1 2)	1172.99	1160.93	1161.20	1147.34	1057.97	1152.51
(3 3 1 1)	380.39	376.08	376.27	377.57	373.34	373.90
(3 3 1 2)	1418.13	1403.55	1402.68	1409.69	1394.61	1402.40
(1 1 2 1)	79.76	78.37	77.35	76.52	75.37	75.52
(1 1 2 2)	311.03	309.09	299.75	299.50	296.57	297.89
(1 2 2 1)	273.79	271.66	267.02	233.70	216.78	180.07
(1 2 2 2)	1052.43	1044.24	1035.19	915.91	857.96	729.91
(1 3 2 1)	330.63	326.85	323.60	311.50	300.84	279.24
(1 3 2 2)	1260.21	1239.90	1241.00	1194.54	1146.11	1067.40
(2 1 2 1)	98.57	97.55	96.02	95.97	94.30	95.00
(2 1 2 2)	381.43	377.36	372.88	369.34	364.99	367.60
(2 2 2 1)	287.53	284.40	282.85	253.55	234.18	202.77
(2 2 2 2)	1077.60	1066.65	1065.79	960.28	900.02	743.68
(2 3 2 1)	368.50	364.69	363.57	359.52	348.21	325.46
(2 3 2 2)	1375.33	1363.28	1360.61	1344.69	1309.57	1214.59
(3 1 2 1)	106.58	105.40	105.53	103.83	100.61	102.29
(3 1 2 2)	418.07	415.18	413.26	408.15	396.63	400.92
(3 2 2 1)	292.91	289.97	287.78	261.59	245.05	206.16
(3 2 2 2)	1123.70	1111.91	1109.34	1003.00	925.85	797.36
(3 3 2 1)	388.97	385.84	386.11	385.01	380.19	364.49
(3 3 2 2)	1426.54	1417.44	1417.91	1413.77	1401.71	1329.51
Total	593.78	587.89	584.04	569.29	544.41	535.82

Table 8.16: Testing Search Routines for Reward Type I for Combined MCP

Classes	Methods					
(T,N,L,R)	BEST	BESTH	RAND	R10NOEX	TABUMV	TABUCR
(1 1 2 1)	67.71	66.76	64.56	64.82	63.96	66.46
(1 1 2 2)	265.94	262.54	251.75	247.93	248.42	260.35
(1 2 2 1)	204.55	202.98	201.20	189.10	174.15	184.61
(1 2 2 2)	781.07	773.92	771.52	716.76	670.64	696.41
(1 3 2 1)	257.15	252.01	247.61	243.75	240.09	237.08
(1 3 2 2)	982.85	956.91	942.91	930.07	902.33	900.45
(2 1 2 1)	82.55	82.13	81.35	81.19	79.36	81.94
(2 1 2 2)	321.01	319.48	316.53	315.60	312.24	318.25
(2 2 2 1)	218.83	216.78	216.33	207.16	188.17	205.89
(2 2 2 2)	845.95	840.14	837.46	801.30	737.89	787.45
(2 3 2 1)	311.34	306.06	305.13	306.44	302.64	299.68
(2 3 2 2)	1198.00	1178.14	1177.09	1181.52	1163.38	1156.75
(3 1 2 1)	91.22	90.91	90.11	90.72	88.44	90.03
(3 1 2 2)	362.24	360.41	360.04	358.40	348.25	359.35
(3 2 2 1)	219.92	217.85	217.43	211.69	194.44	208.34
(3 2 2 2)	842.50	837.67	835.07	808.90	758.73	789.57
(3 3 2 1)	316.67	313.15	313.44	314.62	312.72	311.22
(3 3 2 2)	1204.34	1192.46	1186.02	1196.75	1191.49	1183.82
Total	476.33	470.58	467.53	459.26	443.19	452.09

Table 8.17: Testing Search Routines for Reward Type II for Combined MCP

improves over the use of random solutions, since the BESTH method outperforms the RAND method.

Of the methods we considered, the BESTH method appears to be the most effective, at this stage. However, testing a full implementation of Tabu Search using Generalized Exchange, on a smaller set of problems is required in order to get a better idea as to the absolute effectiveness of these methods, and we will perform some representative testing of this, later.

General Tasks

We extend the version of the Combined MCP further, to allow for the case where the precedences are not fixed. This problem can be formulated as for the Combined MCP, but we allow a relaxation of constraint set (19). Therefore we use contraints (12)–(18) and (20)–(24) in Section 8.1.2, and the objective

$$\max \sum_{i=1}^{n} y_i R_i(t_{i1}, t_{i2}, .., t_{in_i})$$

and with the constraint set (19) replaced as is appropriate.

This problem is classified as

$$subset, tw_j, T_{rtg}, TDR_{inc}|1||sub, \max R$$

according to the classification scheme of Chapter 6. In this case we have all customers consisting of tasks of three sub-tasks. Each task consists of two pick-ups and a delivery, and we allow the pick-ups to be carried out in either order. Thus our version of constraint set (19) for this problem is that:

$$t_{i1}, t_{i2} \leq t_{i3} \leq T_{max}, \forall\ i = 1, \ldots, N$$

In order for our programs to be able to handle this more general form of task, we had to adapt the way we stored the tasks and the precedences, as was discussed in Section 6.2.1.

The other major alteration that is required with general tasks and with time-dependent reward functions, is the manner in which we store the times of servicing. For our TASKTDR function, we require the time at which the task is started and the time at which it is finished. We identify the times of servicing by scanning the route, and using the variable $got\_task(j)$ to indicate whether we have found the times of serving customer j. We initially set $got\_task(j) = 0\ \forall j$, and as we scan

through the route, if in position i we find that $got\_task(point\_to\_task(i)/N) = 0$, then we set the start time for the task to be the time of servicing i, and set $got\_task(point\_to\_task(i)/N) = 1$. If we have found the last point of a task, we store the service time as our final time, and we can then calculate the reward for the task.

We create 54 classes of problems, using the same problem classes as for the Combined MCP with fixed tasks. We generate 30 problem instances for each problem class, which gives a total of 1620 instances. We are just carrying out a preliminary investigation of this problem, and so we are not specifically interested in the statistical significance of the results; hence we consider fewer instances than for previous problem types.

The construction methods that we carry out for these problem instances are:

- CHEAP — we add the customer that adds the least additional distance to the current solution. This method ignores the rewards that are present and focuses purely on the distances. We consider two different implementations, where we either include the task in the order it is given, or we consider both possible orderings and select the customer and ordering that adds the least additional distance.

- B4B — we add the customer with the highest ratio of additional reward to additional distance. Again we consider either the order in which the task is given, or we consider both possible orders and select the best of these.

- CLUSREW — this method is implemented in the same manner as for the Combined MCP with fixed tasks. We create the clusters using the tasks in the order that they are given, and we again ensure that the tasks are placed in the cluster where at least 90% of the available reward is gained by each task.

For each initial solution we apply our full STEEPEST ASCENT method. The adaptations that are required in order to handle the general tasks, are to allow the customer to be placed in either order for the Move and Generalized Exchange routines, and to alter the method for checking that a sub-path may be moved within the Cross routine, by ensuring that the first customer of the sub-path is the first point visited from a task that is included in the route. With our previous storage of a vehicle's route, a sub-task could be identified as being the first that

is serviced from a task if it had no predecessor. With the general tasks we ensure that each task that is commenced within the candidate sub-path is also finished within this sub-path.

The results from the testing we carried out are given in Tables 8.18 and 8.19. The methods considered are CHEAP1 and CHEAP2 (which are CHEAP, where the tasks are respectively considered for insertion only in the order given or in either order), B4B1 and B4B2 (which are B4B with the tasks again considered for insertion only in the order given or in either order), and CLUSREW (as described above). The Tables give the mean reward obtained for each class of problem (given by (time window, number served, load type, reward variability)), with the unimproved and overall total means and with the computational time required (in the form hh:mm:ss) also given in the second Table.

The overall means for these methods are respectively 371.927, 368.974, 386.495, 385.266 and 388.646. The highest overall mean for the methods is for CLUSREW, and this appears to be relatively effective for all problems. The two methods based on CHEAP are the least effective overall, and they don't appear to be well-suited to these problems. It is interesting to note that the methods that only consider inserting the customers in the given order, i.e., CHEAP1 and B4B1, both obtain higher overall rewards than the comparable method where insertion may be carried out in either order. The computational time is lower when insertion in any order is allowed, and this appears to be due to the greater compatibility between the initial and final solutions. The higher overall mean for single order insertion shows that there is little to be gained by constructing a solution by inserting in the best order, and in fact it might be advantageous to just insert in a given order and allow the improvement routines to obtain effective solutions.

We now consider the differences between the two methods which allow the customers to be inserted in the given order or in different orders, and for this purpose we look at the methods B4B1 and B4B2. We see from the computational times that the construction using bang-for-buck in just the given order takes a total of 16 minutes and 8 seconds, while with either order allowed, the total time is 33 minutes and 30 seconds. The total time to the nearest minute for the improved solutions is 6 hours and 46 minutes when insertion is only in the given order, while it is 5 hours and 5 minutes when either order is allowed. This shows that a large reduction in computational time is gained by only inserting in the given

Classes	Methods				
(T,N,L,R)	CHEAP1	CHEAP2	B4B1	B4B2	CLUSREW
(1 1 1 1)	49.46	49.38	49.63	49.89	48.09
(1 1 1 2)	210.38	212.91	214.92	214.16	213.87
(1 2 1 1)	198.33	198.00	201.63	200.26	201.46
(1 2 1 2)	740.78	729.08	766.74	753.56	772.31
(1 3 1 1)	215.05	214.70	212.58	211.46	217.22
(1 3 1 2)	815.71	800.62	801.53	790.97	808.93
(2 1 1 1)	62.67	62.55	63.56	63.55	62.92
(2 1 1 2)	259.62	256.53	261.70	262.81	261.96
(2 2 1 1)	210.77	208.07	215.02	216.71	218.30
(2 2 1 2)	755.91	740.89	770.00	761.75	776.95
(2 3 1 1)	245.08	244.09	246.46	245.59	251.77
(2 3 1 2)	894.23	895.57	908.39	904.59	908.16
(3 1 1 1)	71.51	70.91	72.65	72.98	72.29
(3 1 1 2)	289.30	288.03	293.41	293.69	291.30
(3 2 1 1)	208.38	206.62	215.03	215.28	218.65
(3 2 1 2)	802.86	837.86	852.69	854.03	862.86
(3 3 1 1)	259.95	260.33	259.56	260.54	261.16
(3 3 1 2)	1038.37	1037.81	1047.89	1042.52	1031.69
(1 1 2 1)	54.75	55.21	55.51	56.10	54.54
(1 1 2 2)	235.69	234.92	238.19	237.20	230.36
(1 2 2 1)	197.86	198.42	206.04	207.80	209.07
(1 2 2 2)	743.34	745.35	788.84	784.03	788.87
(1 3 2 1)	213.66	210.35	225.27	220.24	224.34
(1 3 2 2)	770.52	750.68	805.01	811.39	800.76
(2 1 2 1)	66.64	67.92	69.09	68.99	66.41
(2 1 2 2)	262.09	264.83	271.02	270.58	268.56
(2 2 2 1)	194.89	193.37	209.96	209.87	214.85
(2 2 2 2)	719.74	716.18	765.84	771.61	792.75
(2 3 2 1)	240.20	238.98	256.47	256.97	257.59
(2 3 2 2)	902.63	880.73	974.06	966.41	962.58
(3 1 2 1)	73.71	73.34	73.98	74.17	74.41
(3 1 2 2)	289.37	291.36	308.38	305.12	291.64
(3 2 2 1)	203.52	194.46	211.82	211.47	215.59
(3 2 2 2)	783.49	767.41	842.75	833.10	839.72
(3 3 2 1)	257.94	256.03	263.69	261.87	269.90
(3 3 2 2)	953.77	945.42	984.02	972.04	1004.63
Unimp	250.30	249.76	350.23	354.82	295.76
Total	402.56	399.97	416.76	414.81	417.96

Table 8.18: Testing Methods for General Tasks with Reward Type I

Classes	Methods				
(T,N,L,R)	CHEAP1	CHEAP2	B4B1	B4B2	CLUSREW
(1 1 2 1)	42.34	42.04	44.30	44.42	42.92
(1 1 2 2)	169.81	169.04	176.68	175.96	179.23
(1 2 2 1)	150.39	145.32	153.22	156.06	155.63
(1 2 2 2)	549.29	527.38	565.51	567.93	571.51
(1 3 2 1)	142.18	142.62	153.45	155.85	162.86
(1 3 2 2)	541.34	514.60	595.63	601.38	603.00
(2 1 2 1)	52.35	53.15	54.21	54.48	53.62
(2 1 2 2)	211.76	211.21	217.54	216.88	217.84
(2 2 2 1)	148.83	147.28	157.80	156.63	158.76
(2 2 2 2)	562.38	556.21	600.58	602.90	608.19
(2 3 2 1)	192.02	197.99	201.49	199.28	206.71
(2 3 2 2)	711.77	720.11	747.19	751.05	768.53
(3 1 2 1)	61.77	61.42	62.74	62.86	61.36
(3 1 2 2)	245.61	242.29	249.26	251.13	249.16
(3 2 2 1)	159.89	158.70	168.29	167.22	169.63
(3 2 2 2)	622.42	602.02	649.34	643.83	653.25
(3 3 2 1)	203.96	208.10	211.30	208.18	211.56
(3 3 2 2)	823.77	826.23	858.82	855.04	866.68
Unimp	110.30	113.12	257.99	276.00	173.01
Total	310.66	306.98	325.96	326.17	330.02
Time	06:48:40	05:25:13	06:45:36	05:04:30	11:01:05

Table 8.19: Testing Methods for General Tasks with Reward Type II

order, although the total time is greater to obtain the improved solution. This is, however, somewhat arbitrary, as we are basically just selecting an order for insertion and improving upon this solution. We also carried out the insertion by just allowing the customer to be included in the initial solution in the reverse order, and the total time for the improved solution in this case is 5 hours and 14 minutes. The difference in this case, though, is that there is a total of 922 customers that are unable to be included, since they are infeasible in the reverse order; but, even with the smaller effective problem size, the computational time for this fixed order is greater than when either order is allowed.

Of interest is the comparison of the effectiveness of the methods for different problem definitions. We consider the problem where all customers must be included in the order in which they are given, thereby defining fixed precedences. The overall mean reward for CHEAP is 355.256 and the total computational time required is 1

hour and 43 minutes; the overall mean for B4B is 370.371 with total computational time of 1 hour and 29 minutes; and the overall mean for CLUSREW is 377.477 with total computational time of 3 hours and 38 minutes. This form of the problem is identical to that we considered in the previous section, and the CLUSREW method is the most effective for these problems also. The solution quality increases as the precedence constraints are relaxed, and the computational time required is increased dramatically. This is due to us considering each order when evaluating the insertion involved for the Move and Generalized Exchange routines.

We have carried out a preliminary investigation into the possible effect of allowing the order in which the sub-tasks are serviced to be varied. This requires some changes to the manner that the data is handled, as the precedence constraints and times of servicing are different. With these changes, the methods that were used previously for the Combined MCP may still be valid, with the previously applied CLUSREW method finding superior solution quality to that obtained by the other methods tested. We have seen that the relaxation of the precedence constraints significantly increases the computational time required for our improvement techniques, and the relaxation of these constraints in the construction phase creates solutions that take much less time to improve. The method of handling the tasks in the construction phase, however, doesn't appear to greatly affect the final solution obtained, and this again shows the power of the improvement routines. More investigation into this area is required, particularly in order to allow the other variations of the tasks that were considered in Chapter 5.

8.2 Further Testing

8.2.1 Comparing New Heuristics with Previous MCPTDR Heuristics

We found previously, that the methods of Erkut and Zhang [45] were not effective for the decreasing reward problems that we considered, with time windows present, and therefore our new methods were superior. We now consider the performance of the best of our heuristics, on the problems that have previously been used for the MCPTDR. These problems have no limit on the time that the service vehicle is in use, with all customer rewards linearly decreasing over time. The paper by Erkut and Zhang [45] describes the parameter settings that they used to create random

test problems, and we randomly generate our own test problems, according to these parameters. Customers are randomly positioned within the square $[0, 10]^2$, with the distances between the customers being given by the Euclidean distance. The initial values of the reward are uniformly distributed in the range $(40n, 80n)$, where n is the number of customers, and two distributions for the rate at which the rewards are decreasing are given, with these uniformly distributed in the range $(1, 11)$ and $(1, 16)$. We create 100 test problems, with each distribution of the slopes, where we set $n = 60$.

For the test problems, we implement the 2-step greedy method of Brideau and Cavalier [20], which we call BRIDEAU, and the 2-step loss minimization method of Erkut and Zhang [45], which we call ERKUT. We also implement the REWAPP method, which seemed to be the most effective of the heuristic methods that we discussed previously in Section 8.1.3, for the problems where all rewards are decreasing. For REWAPP we set the parameter $\mu = 1$, since this is one tenth of the parameter that we were using when our locations were located in a square of length 1. The results from our testing are shown in Table 8.20, where the column "Class" indicates the parameter setting for the slopes that we are using. The column "Phase" indicates whether we are comparing the results from the unimproved construction heuristic, i.e., UNIMP, the construction heuristic improved by 2-opt, i.e., 2-OPT or with the initial solution improved by STEEPEST ASCENT, i.e., ASCENT. We also include, in Table A.6, the p-values from testing the results using the Wilcoxon Signed-Rank Test. In this Table, a negative p-value indicates that the mean for the method stated in the column, is less than that for the method stated in the top row.

For both distributions of the slopes, with the unimproved solutions and those improved by the application of 2-opt, the BRIDEAU method is significantly less effective than each of the other methods, at a 5% significance level, according to the Wilcoxon Signed-Rank Test. There is no significant difference between the effectiveness of ERKUT and REWAPP for the unimproved solutions and with those improved by 2-opt. With STEEPEST ASCENT, there is no significant difference between any of the three methods.

From these results we conclude, as Erkut and Zhang did, that the ERKUT heuristic is superior to the BRIDEAU method, when the unimproved rewards and

		Methods		
Phase	Class	ERKUT	BRIDEAU	REWAPP
UNIMP	$U(1,11)$	204733.675	204489.096	204891.234
	$U(1,16)$	200011.476	198871.921	199757.140
	Overall	202372.576	201680.508	202324.187
2-OPT	$U(1,11)$	205938.743	205701.768	205968.075
	$U(1,16)$	201552.360	200985.032	201591.446
	Overall	203745.551	203343.400	203779.760
ASCENT	$U(1,11)$	206636.628	206595.967	206642.365
	$U(1,16)$	202459.376	202346.575	202435.017
	Overall	204548.002	204471.271	204538.691

Table 8.20: Testing Methods on Erkut's Data

those improved by 2-opt for each method are compared. When we carry out STEEPEST ASCENT, there is no significant difference between these methods. We also conclude that there is no significant difference between the effectiveness of our REWAPP method and the ERKUT method for these problems.

We extend the problems generated by Erkut and Zhang, to create test problems with higher variability of the slopes of the reward functions. We create five problem sets, with the slopes coming from the distributions: $U(1,26)$, $U(1,51)$, $U(1,101)$, $U(1,201)$ and $U(1,401)$. These problem sets have increasing mean slopes, and therefore the number of customers that can be served while having positive reward, is decreased. The three methods that we applied to the previous problem sets, i.e., ERKUT, BRIDEAU and REWAPP, were all implemented, in the manner we believe they were originally implemented. Thus all customers are added to a single solution route, even though a customer may be placed in a position where its reward received is zero. The results from these new problems are given in Table 8.21 and the p-values are given in Table A.7.

When comparing the unimproved solutions for the methods, and those improved by 2-opt, it is apparent that for these problems, the BRIDEAU method is the least effective. By the Wilcoxon Signed-Rank Test at a 5% level, both ERKUT and REWAPP are superior to BRIDEAU for all problem classes, for both the unimproved solutions and those improved by 2-opt. With the unimproved solutions we find that ERKUT is superior to REWAPP for the two lowest ranges of the slopes, whereas REWAPP is superior with slopes in the range (1,201). With solutions improved by

		Methods		
Phase	Class	ERKUT	BRIDEAU	REWAPP
UNIMP	$U(1,26)$	**189953.429**	187955.432	189076.002
	$U(1,51)$	**169048.454**	163635.864	166304.296
	$U(1,101)$	129391.381	125243.171	129839.030
	$U(1,201)$	92122.533	90129.462	**94027.399**
	$U(1,401)$	64197.260	62212.455	65023.742
	Overall	128942.611	125835.277	128854.094
2-OPT	$U(1,26)$	192728.297	191868.398	192572.337
	$U(1,51)$	**174162.046**	171465.231	172295.875
	$U(1,101)$	134810.350	131549.167	134791.005
	$U(1,201)$	95347.463	93937.125	97193.983
	$U(1,401)$	66000.002	64847.213	**67075.972**
	Overall	132609.632	130733.427	132785.834
ASCENT	$U(1,26)$	194234.419	194191.157	194217.865
	$U(1,51)$	176617.530	176550.679	176629.752
	$U(1,101)$	141063.149	141101.415	141173.093
	$U(1,201)$	101260.421	101707.548	102222.599
	$U(1,401)$	69768.562	69930.106	**70556.342**
	Overall	136588.816	136696.181	136959.930

Table 8.21: Testing Methods on Erkut's Data - Extended

2-opt ERKUT is superior to REWAPP for slopes in the range (1,51), while REWAPP is superior with slopes in the range (1,201).

We see that there are vast improvements in the solutions to these problems that may be made by carrying out STEEPEST ASCENT, rather than just 2-opt. When there are many customers for whom the reward is zero, it is not possible to easily make improvements by making a customer's reward become positive, as the effects of a 2-opt move can be far-reaching. For these problems, routines that move customers and strings of customers, as is the case with Or-opt, are far more effective than 2-opt. The trade-off that is made is in the computational time requirements. For example, with REWAPP, the time required to obtain the initial solutions to the 500 problems, is 1 minute and 18 seconds, the time to obtain the solution that is improved by 2-opt is 65 minutes and 54 seconds, while the time for STEEPEST ASCENT is 241 minutes and 14 seconds. Thus in selecting a solution method, we need to weigh up whether the improved effectiveness by using the more advanced techniques, is more important than the concerns with the additional computational

time. With this in mind, we tested the effectiveness of just using the 1-Or exchange on the initial solutions. For REWAPP, we found the mean reward values for the five ranges of slope values, were respectively 192837.237, 174199.252, 138631.941, 100494.871 and 69349.883. The overall mean was 135102.637 and the total time taken was 87 minutes and 37 seconds. With the Wilcoxon Signed-Rank Test at a 5% level, we find that this method is superior to 2-opt for the four highest ranges of slope parameters, but it is inferior to the full improvement routine for each of the problem classes. This therefore, seems to be an effective compromise between the solution quality and the computational time requirements.

For the solutions with STEEPEST ASCENT, there is no significant difference between the effectiveness of BRIDEAU and ERKUT. REWAPP is significantly more effective than both BRIDEAU and ERKUT with the highest range of the slope values and also significantly more effective than ERKUT for the the second highest slope parameter, as tested using the Wilcoxon Signed-Rank Test Test at a 5% level. This suggests that the existing methods are ineffective when fewer customers have positive reward.

We conclude that the construction heuristic of Erkut and Zhang is effective for the range of problems over which they tested their heuristic. This however is a very restricted form of the problem, and as fewer customers have positive rewards in the solution, i.e., as the rate of decline in the rewards increases, this method becomes far less effective, and our new REWAPP method becomes superior. The improvement routine 2-opt, is less appropriate as the numbers of customers with zero reward increase. In this case we have shown that a string move routine, such as Or-opt, may be more effective. We showed that the use of the 1-Or routine is significantly more effective than the 2-opt routine, although with increased computational time requirement. Future work could involve finding more advanced local search operators, with a path exchange routine likely to enable further improvements.

8.2.2 Testing the Absolute Quality of OPTW and MCPTDR Solutions

The MCPTDR

We previously tested the average case performance of a number of heuristic and metaheuristic methods for obtaining solutions to versions of the OPTW and MCPTDR. This enables us to obtain some measure of the relative performance of the heuristics, but we are now interested in finding the absolute effectiveness of these methods, by testing on problems for which we have a known optimal solution. We use the fact that if all customers are included in the solution for the OPTW, and if all customers are serviced while their reward is at its maximum for the MCPTDR, then we have the optimal solution.

For the MCPTDR we are considering reward functions to be of the piecewise linear form previously shown in Figure 6.2, and so this requires time windows over which the reward values are positive. We create a number of test problems by varying three factors: the variability in the maximum reward values (R), the form of the time windows (W) and the type of tour on which the time windows are based (T). For the parameter, R, we have three values:

1. the maximum rewards being the same (equal to 20.0)
2. low variability (uniformly distributed in the range $[10, 20]$)
3. high variability (uniformly distributed in the range $[10, 100]$)

For the time windows, we define the 'inner time window' to be the time span over which the reward is at its maximum value, and the 'outer time window' is the time span when the reward is positive. We also have three values for the parameter, W:

1. tight time windows for both the inner and outer time windows
2. tight time window for the inner time window and a loose time window for the outer time window
3. loose time windows for both the inner and outer time windows

We uniformly distribute the width of the inner time window of customer i to be a given proportion k_i of the time limit of the problem instance, $LIMIT$, while

the width of the outer time window is given by $K_i * LIMIT$. For the inner time window, with tight time windows k_i is uniformly distributed in the range $[0, 0.05]$ while for loose time windows k_i is uniformly distributed in the range $[0.1, 0.2]$. For the outer time window, with tight time windows K_i is uniformly distributed in the range $[0.1, 0.2]$, while with loose time windows we have K_i uniformly distributed in the range $[0.3, 0.6]$.

The methods we use for generating the tour types, T, are:

1. the best insertion method
2. the randomized nearest neighbour method
3. the random append method

We set the time limit of the problem to be 5% longer than the length of the tour that we base the time windows upon. We calculate the time at which each customer is serviced in this tour and randomly place the inner time window around this service time, so that the customer can be serviced without any waiting time required. We place the outer time windows symmetrically around the inner time window, although if this leads to the beginning of the outer time window being less than zero or the end of the window exceeding the time limit, we preserve the time window width but move the time at which the outer time window starts, so the whole time window fits within the scheduling period.

By varying each of these three variables to each of their three levels, we obtain 27 problem types with various characteristics. The different reward variabilities will affect the attractiveness of the customers, which mostly affects the selection decisions. The time window widths create problems with varying difficulties of inclusion and of serving with maximum reward, and this mostly affects the scheduling decisions. Varying the type of tour, affects the form of the solution route and this mostly affects the routing decisions.

We are interested in the relationship between the problem size, and the effectiveness of the heuristics and the computational time required. Therefore we create our test problems with respectively 20, 40, 60 and 100 customers in the problem. We create 100 test problems for each problem class, by randomly generating 100 problem layouts and assigning the customers to these layouts. For each size of problem, we have a total of 2700 test problems, and we record the total available reward for each instance.

Algorithm 8.2 function ATMAX

```
Order the customers in decreasing order of the maximum reward
for Each customer do
    for Each route do
        while (Customer has not yet been placed) do
            if(Customer can be placed with its maximum reward received
              and existing customers' rewards are unaffected)
                Place customer in position with least change in distance
            end
        end
    end
    if (Customer is still unplaced) then
        Insert customer into a new route
    end
end

end
```

We are using the REWAPP and CHEAP methods, that we previously applied to the MCPTDR, as the CHEAP method involves the consideration of just the insertion times and the REWAPP method considers the reward of all eligible customers when selecting which customer to insert. These methods approach the customer selection decision in completely different ways, so these two methods should obtain solutions that are diverse, which should enable a wider range of the solution space to be considered. We also apply the ARTIF method, which is the implementation of our artificial time window method ARTIFINSERT, with the time windows initially set to encompass the times when at least 50% of each customer's maximum reward is available. The other construction method we apply is the ATMAX method, which we describe in Algorithm 8.2. This method places customers in the solution in decreasing order of their maximum reward, such that the maximum reward for each customer is obtained. In preliminary testing on the different problem classes considered in Section 8.1.3, we found that this method was not very robust, although it was effective for problems in which most customers can be serviced at times where their maximum reward is obtained, so this method appears to be suited to the problem types we are solving here. For each of these methods we construct an initial solution and use STEEPEST ASCENT to obtain an improved solution, with the ARTIF method also including a secondary phase in which the

time windows are relaxed and STEEPEST ASCENT is again applied.

The final method we consider is our full TABUIMP metaheuristic. During the search, we stop the metaheuristic once we have found an optimal solution, and the use of this stopping criterion is able to dramatically reduce the computational time required. The knowledge of a tight upper bound on the solution quality may be very beneficial in all metaheuristic search, as the search may be terminated once the standard stopping criteria on number of iterations or computational time are met, or if the heuristic solution is within a given tolerance of the solution upper bound.

In Table 8.22 we give the number of optimal solutions that were found by each method for each problem class, with 20 customers available. We also give the total computational time required (in the form mm:ss) to implement each of the methods. Throughout the analysis, we consider a solution to be optimal if the total reward received is within 0.01% of the total available reward, as this allows for errors that have come through the rounding of the reward values. The total number of the 2700 instances where the optimal solution was found is respectively 2195, 2118, 2315, 2555 and 2698 for the five methods in the Table. Therefore TABUIMP finds the optimal solution for all but 2 of the problem instances considered. The least effective of the methods is CHEAP, although it obtains the optimal solution for each of the problem instances with tour type 1, in which the time windows are based upon a tour obtained by a cheapest insertion method. The ATMAX method obtains the optimal solution in almost 95% of the problem instances, and so this method appears to be the most effective of the construction heuristics to apply to these problems. The ARTIF method takes the least time to implement, and the TABUIMP method also obtains solutions very quickly, due to the method stopping as soon as an optimal solution is obtained.

Table 8.23 shows the proportion of optimal objective value that is found by each method, and from this Table we see that the TABUIMP method obtains on average, 99.99% of the available reward. For comparing the robustness of the methods, we calculate the minimum proportion of optimal solution value that is obtained for each of the methods, and these are respectively 0.82174, 0.69692, 0.87284, 0.85408 and 0.97609. TABUIMP also appears to be very robust, as its maximum deviation from the optimal solution is only 2.39% of this optimal solution. The CHEAP method is the least robust, as its minimum reward value is only 69.69% of the

Class	Construction Techniques				
(W,T,R)	REWAPP	CHEAP	ARTIF	ATMAX	TABUIMP
(1,1,1)	98	100	100	99	100
(1,1,2)	98	100	100	100	100
(1,1,3)	95	100	98	99	100
(1,2,1)	91	72	94	96	100
(1,2,2)	87	72	92	95	100
(1,2,3)	77	65	86	97	100
(1,3,1)	99	83	98	98	100
(1,3,2)	91	85	97	99	100
(1,3,3)	85	77	99	97	100
(2,1,1)	90	100	95	100	100
(2,1,2)	90	100	93	98	100
(2,1,3)	81	100	85	99	100
(2,2,1)	63	61	65	94	100
(2,2,2)	71	57	71	93	100
(2,2,3)	61	49	72	94	100
(2,3,1)	76	68	80	96	100
(2,3,2)	77	62	78	98	99
(2,3,3)	65	48	69	100	100
(3,1,1)	85	100	92	97	100
(3,1,2)	86	100	87	98	100
(3,1,3)	85	100	84	90	100
(3,2,1)	75	68	84	83	100
(3,2,2)	77	68	86	87	100
(3,2,3)	73	70	76	85	100
(3,3,1)	81	70	84	92	100
(3,3,2)	72	72	76	86	99
(3,3,3)	66	71	74	85	100
Total	2195	2118	2315	2555	2698
Time	02:26	02:06	01:00	01:21	02:24

Table 8.22: Number of Optimal Solutions Found — n = 20

optimal solution.

Since the problems tested here involve time-dependent rewards, with all customers able to be included in the solution, we carry out a new version of Tabu Search, TINTRA, which we believe is more appropriate with this form of problem as it involves intra-route improvement routines in the Tabu Search phase. Within TINTRA, the improvement move we consider is INTRAREPOS which is a form of 1-Or exchange, in which a customer is removed from a route and repositioned within the same route. For this process we store the tabu status for the customers, so that a customer that is repositioned within the search phase can not be repositioned again for a further $n/2$ iterations, where n is the number of customers, and we store the tabu status of the routes themselves, so that no solution can be repeated. Within TINTRA, we call upon the INTRAREPOS routine when all the customers are included within the solution route, and we use the standard inter-route improvement routines if not all customers are in the solution route or if no feasible intra-route moves are available.

For these test problems we found that TINTRA obtains an optimal solution for each of the 2700 test problems, with a computational time (2 minutes and 30 seconds) that is similar to that for TABUIMP. Due to the promising results obtained, we will consider the TINTRA routine in our further testing in this section.

We now consider the performance of the best of the heuristics, found by taking the highest reward value obtained by the four heuristics (REWAPP, CHEAP, ARTIF and ATMAX) compared with the Tabu Search methods. The total number of optimal solutions found by the best of the heuristics is 2682 and the total proportion of optimal objective value is 0.99997 for the best of the heuristics, whereas we found that TINTRA identified the optimal solution to each problem instance. The total computational time required is 6 minutes and 53 seconds for the heuristics, while it is 2 minutes and 30 seconds for TINTRA. Finding solutions for the five heuristics takes considerably longer than TINTRA, which is because we are calculating the solution for each heuristic regardless of the performance of the heuristics, whereas with TINTRA we stop our searching as soon as an optimal solution is found.

We now consider one test problem for which the Tabu Search methods outperform the other heuristics, and this is with layout 78 and with the variables for T = W = N = 2. The solution found by TABUIMP and TINTRA for this problem is optimal, and it is shown in Figure 8.1. For this figure, the reward that is received

Class	Construction Techniques				
(W,T,R)	RewApp	Cheap	Artif	AtMax	TabuImp
(1,1,1)	0.99896	1.00000	1.00000	0.99950	1.00000
(1,1,2)	0.99856	1.00000	1.00000	1.00000	1.00000
(1,1,3)	0.99633	1.00000	0.99922	0.99986	1.00000
(1,2,1)	0.99715	0.97842	0.99957	0.99927	1.00000
(1,2,2)	0.99578	0.97991	0.99913	0.99846	1.00000
(1,2,3)	0.99045	0.97637	0.99598	0.99935	1.00000
(1,3,1)	0.99949	0.98308	0.99948	0.99900	1.00000
(1,3,2)	0.99485	0.98624	0.99907	0.99964	1.00000
(1,3,3)	0.98913	0.97493	0.99961	0.99940	1.00000
(2,1,1)	0.99374	1.00000	0.99798	1.00000	1.00000
(2,1,2)	0.99446	1.00000	0.99765	0.99840	1.00000
(2,1,3)	0.99229	1.00000	0.99325	0.99964	1.00000
(2,2,1)	0.98919	0.96909	0.99199	0.99888	1.00000
(2,2,2)	0.99035	0.96168	0.99071	0.99849	1.00000
(2,2,3)	0.98553	0.95530	0.99376	0.99961	1.00000
(2,3,1)	0.98966	0.97700	0.99237	0.99861	1.00000
(2,3,2)	0.99133	0.97214	0.99332	0.99965	0.99976
(2,3,3)	0.98604	0.97095	0.99022	1.00000	1.00000
(3,1,1)	0.99327	1.00000	0.99727	0.99819	1.00000
(3,1,2)	0.99620	1.00000	0.99600	0.99951	1.00000
(3,1,3)	0.99656	1.00000	0.99599	0.99820	1.00000
(3,2,1)	0.99654	0.99185	0.99775	0.99684	1.00000
(3,2,2)	0.99791	0.99212	0.99821	0.99888	1.00000
(3,2,3)	0.99718	0.99314	0.99781	0.99878	1.00000
(3,3,1)	0.99850	0.99774	0.99902	0.99941	1.00000
(3,3,2)	0.99805	0.99593	0.99888	0.99883	0.99999
(3,3,3)	0.99735	0.99707	0.99862	0.99934	1.00000
Total	0.99345	0.98643	0.99648	0.99920	1.00000

Table 8.23: Average Proportion of Optimal Objective Value Found — n = 20

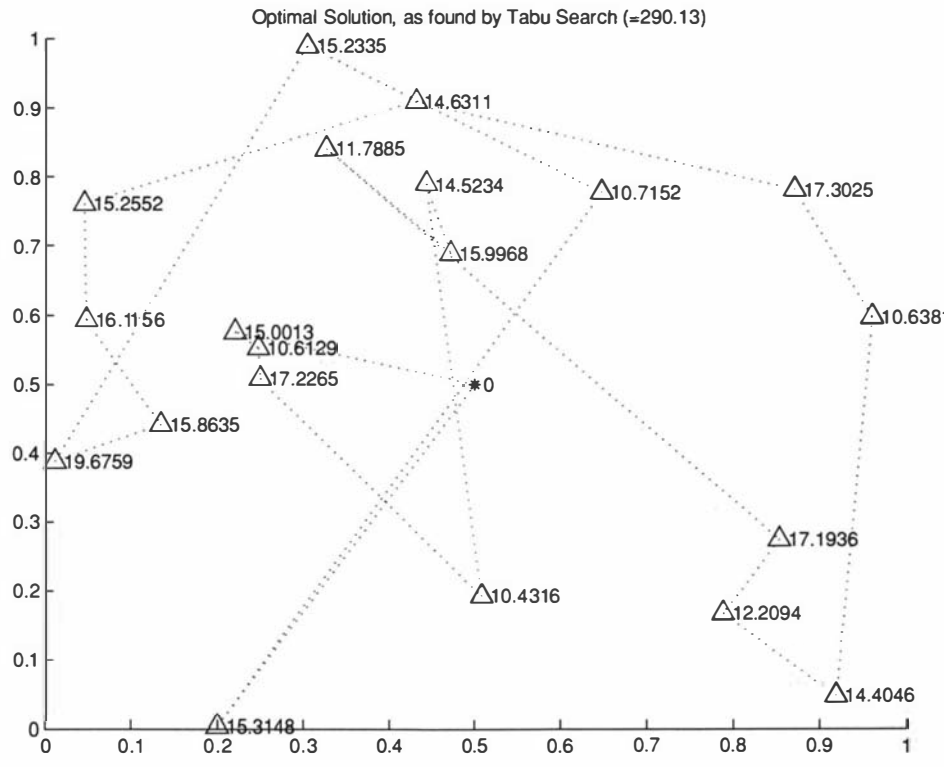

Figure 8.1: Tabu Search Solution for Problem 2093

for servicing the customer, is given as the label at the customer's location. A triangle indicates that the reward that is received is equal to the maximum available. The solution begins by servicing the cluster of three customers to the left of the depot, and proceeds around the path until finishing with the isolated customer at the bottom left of the service area.

The final solution with methods REWAPP, CHEAP and ARTIF are all the same, and this is shown in Figure 8.2. The solution begins by servicing the cluster of customers to the left of the depot, although the vehicle leaves this cluster to service the customer located above this cluster, before continuing. It is the premature servicing of this customer that prevents an optimal solution from being able to be found, by using strictly improving moves. Of note here, is that although the final solution obtained is identical, the initial solutions obtained by the different construction methods are very varied. The initial solution obtained by CHEAP is particularly poor, as it involves servicing two customers at the very beginning of their time window, when their reward is zero. This solution obtains a total reward of 132.75, whereas the initial rewards received by REWAPP and ARTIF are respectively 253.05 and 271.27.

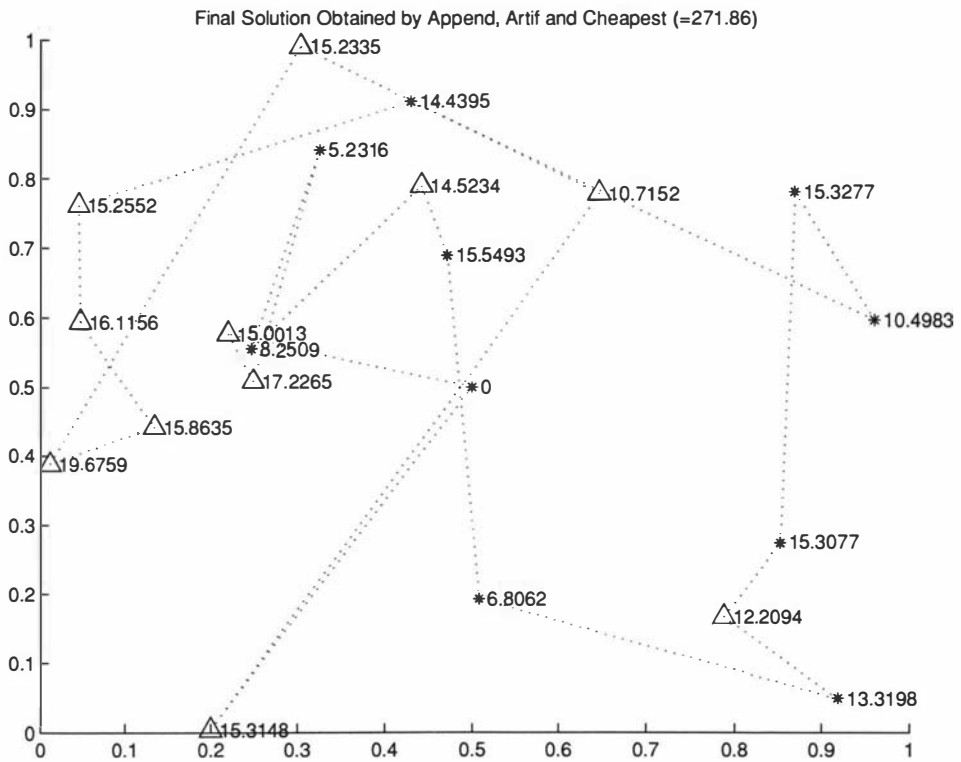

Figure 8.2: Solution found by REWAPP, CHEAP and ARTIF for Problem 2093

The solution found by ATMAX is shown in Figure 8.3. This also is the initial solution found by the construction technique, and involves servicing all bar one of the customers, with all serviced customers being serviced while their reward is at its highest level. The solution route, as it stands, could be improved using the Generalized Exchange routine to allow the excluded customer to be serviced. This was unable to occur, since the evaluation of the current solution requires the use of non-compulsory waiting time, in order to evaluate the solution correctly. The solution route begins by servicing the cluster of three customers to the left of the depot. The ordering of servicing this cluster does not allow for the excluded customer to be serviced, while allowing the remaining customers to be serviced at their maximum reward. The next customer that is included has a maximum reward of 14.5234, and its reward reaches its maximum value at time 1.2597. The vehicle arrives at this location at time 0.7529 and due to the specifications of the heuristic, waits until the maximum reward is available before servicing the customer.

When we carry out our improvement routines, we remove any non-compulsory waiting time. In the solution route given, the elimination of the waiting time leads to a solution that services a number of customers before their maximum reward

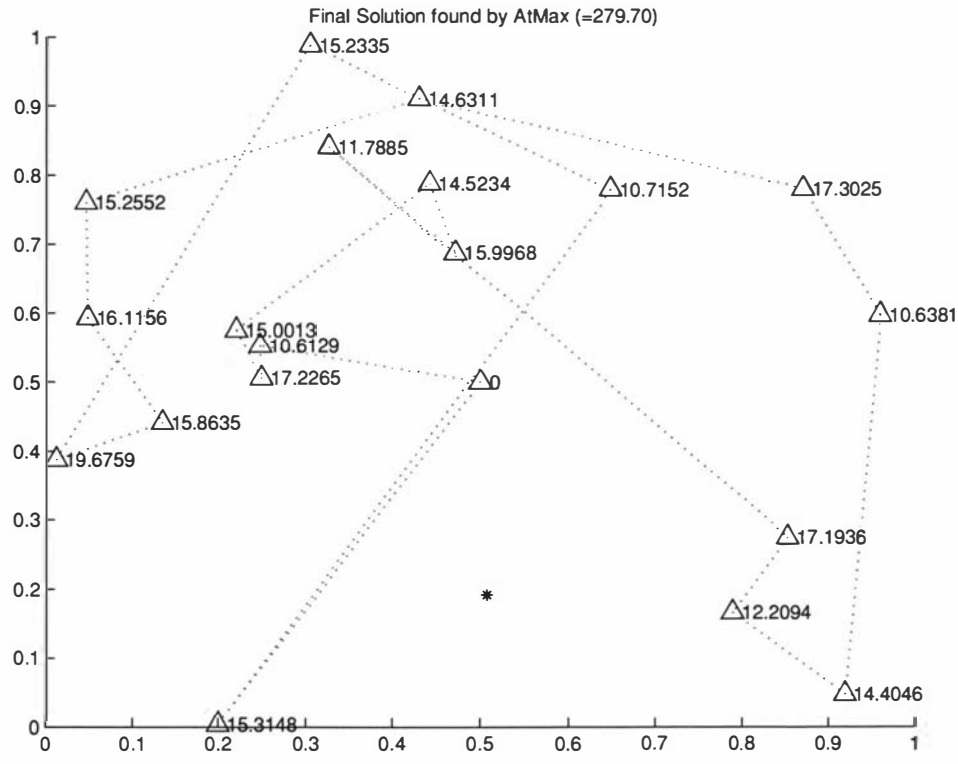

Figure 8.3: AtMax Solution for Problem 2093

is available, and so the resulting tour is quite poor. Therefore the improvement routines create a solution that is very different from the one shown, and so the optimal solution can not be obtained. In this case, if we imposed hard constraints to ensure the customer is serviced at its maximum, then the optimal solution would be obtained by applying the Generalized Exchange routine.

The ATMAX method creates solutions that can be effective if most of the customers are serviced at their optimal value. The additional structure that is imposed during the construction phase, i.e., the addition of hard time windows, results in non-compulsory waiting time that our improvement routines are unable to deal with, and so these routines may prove to be very ineffective. The frequent calculation of waiting time for our Tabu Search method, i.e., after every 10th iteration, increases the likelihood that we will obtain a good solution where non-compulsory waiting time is required. For other problems where waiting time is required in order to obtain reasonable solutions, the evaluation of each move according to the reward that would be gained if this waiting time is included, may be a reasonable approach. We previously employed this approach, using the method GWIMP, and found that this is potentially effective when there are significant gains that may be

made by including optional waiting time, although this method was computationally expensive.

In response to this perceived shortcoming of our ATMAX method, we adapt the method in a manner similar to that used for our artificial time windows, to create a heuristic that we call NEWMAX. We create the initial solution in the same manner as for ATMAX, but we then artificially set the time windows to be equal to the time at which the maximum reward is available. We carry out STEEPEST ASCENT using this time window, and then continue by resetting the original time windows, adapting the solution and carrying out STEEPEST ASCENT again.

We test the effectiveness of NEWMAX for the set of problems with 20 customers available. The method finds the optimal solution for 2687 of the 2700 test problems, which is a large increase from the 2555 optimal solutions found by ATMAX, and the average proportion of the optimal solution value found by NEWMAX is 0.99992. The computational time required for NEWMAX is 38 seconds, so this obtains solutions more quickly than the other heuristics considered. The minimum proportion of maximum reward found by NEWMAX is 0.95000, for a problem where all rewards are the same and one of the customers is not placed in the solution route.

If we only consider the initial phase of NEWMAX, i.e., the construction phase and implementation of STEEPEST ASCENT with the artificial time windows imposed, then the computational time required is 14 seconds, the number of optimal solutions found is 2631 and the average proportion of the optimal solution value found is 0.99894. The major gain by relaxing the time windows and searching again, is in the robustness of the search, as the minimum proportion of the optimal reward obtained in the first phase is 0.85000, whereas after relaxing the time constraints, we always obtain at least 0.95000 of the optimal solution. Therefore the initial phase for this method is very effective in obtaining high quality solutions quickly, although there is still room for improvement by fine-tuning the solution by relaxing the time constraints.

This new method NEWMAX, involves adding the customers in non-increasing order of their maximum reward, in a position such that the maximum reward is obtained. The solution is improved by applying STEEPEST ASCENT, with all rewards at their maximum value, and then removing any non-compulsory waiting time, updating the solution and re-applying STEEPEST ASCENT. This method is similar to ARTIF, where the artificial time windows are set so that the maximum

reward must be received, with the difference being that the method of constructing the initial solution involves adding the customer with highest available reward, rather than adding the one that has the highest ratio of aggregate score to insertion time. We therefore create a new method, ArtMax, which is our artificial time window method, but with all customers serviced at a time where their maximum reward is obtained.

For the set of problems with 20 customers, ArtMax obtains the optimal solution for 2689 of the 2700 problem instances, and obtains an average of 0.99994 of the optimal objective value. The minimum proportion of optimal obtained is 0.95271 and the total computational time required is 57 seconds. Therefore, this method appears to be of similar effectiveness to the NewMax method, it is also quick to implement, and so these methods give promising results that warrant further investigation.

We now consider the solutions obtained by our previous heuristics with 40 customers available, where we also consider the NewMax, ArtMax and TIntra methods. Since the Cheap method appears to be relatively ineffective for problems where the base tour is not a TSP tour, we do not consider this method further. The number of optimal solutions and the computational time required (in the form h:mm:ss) are included in Appendix A in Table A.8 with the average proportions of optimal solution reward shown in Table A.9. In these Tables the names of the methods have been abbreviated due to space restrictions.

From these Tables we see that the the most difficult problems for these heuristics to find an optimal solution for, are with type 2 time windows, and it is for these problems that the NewMax and ArtMax methods prove to be superior to the other methods. For these problems, there is a long time period over which the customer may be serviced, although there is only a short time period for which the maximum available reward may be collected. The forcing of customers to be serviced at times where their reward is at its maximum value, enables the optimal solution to be found, where the other methods are more likely to accept solutions where the reward is high, although without ensuring the maximum reward is obtained.

If we just consider the two implementations of Tabu Search, we see that TabuImp obtains 2450 optimal solutions, while TIntra obtains 2515 optimal solutions. The increased number of optimal solutions suggests that the TIntra method is the

more effective of the Tabu Search routines, so it is this method we compare with the other heuristic methods.

The most effective heuristic methods are NEWMAX and ARTMAX, and we compare the effectiveness of these methods with TINTRA. The number of optimal solutions found by TINTRA, NEWMAX and ARTMAX are respectively, 2515, 2545 and 2566 of the 2700 instances, so via this measure of performance, the TINTRA method is outperformed by the other methods. When we test the pairwise difference in the effectiveness of these methods, according to the Wilcoxon Signed-Rank Test to a 5% level, we see that NEWMAX is more effective than TINTRA for the three problem classes with W and T both equal 2, while TINTRA is superior for the four classes with W = 3 and with T = 1 or 2. ARTMAX is more effective than TINTRA for the same three problem classes for which NEWMAX was superior, while TINTRA is superior for two problem classes with wide time windows and T = 1. ARTMAX is more effective than NEWMAX for one of the problem classes, while NEWMAX is not superior for any problem classes, and we therefore consider that ARTMAX is slightly superior to NEWMAX.

The classes for which TINTRA is the more effective method, are all with W = 3, i.e., with wide time windows for which the maximum reward is available. This creates problems for which there is a lot of flexibility in the routing allowed, and the additional searching that is part of the TINTRA method, appears to be of most benefit for these cases. The classes for which NEWMAX and ARTMAX are more effective than TINTRA, are all with type 2 time windows, and with the medium length tour. For these problems, the requirement of servicing while the reward is at its maximum, as is the case with the heuristics, leads to consistently superior solutions to those obtained by Tabu Search, where a number of customers are serviced while their reward is 'near' its maximum value rather than at its maximum.

The difference in the computational time requirements between the two methods ARTMAX and TINTRA, i.e., approximately 12 minutes vs 2 hours and 41 minutes, would suggest that the ARTMAX method would be our solution method of choice for these problems. The advantages of the TINTRA method are the superiority with wide time windows and the robustness of the method, i.e., the worst performance found for TINTRA was to obtain 0.97634 of the optimal reward, whereas the worst performance of the NEWMAX method obtained 0.95098 of the reward

that is available.

In Tables A.10 and A.11 we give the performance of the heuristics and computational time requirements (in the form hh:mm) for 60 customers. From these Tables we can again see that the most effective of the heuristic methods are NEWMAX and ARTMAX. The methods ARTIF and REWAPP, which were the most effective methods for the general problems of Section 8.1.3 prove to be the least effective, which demonstrates the benefit from tailoring specific methods for these problems. The two implementations of Tabu Search are of similar effectiveness, although TINTRA obtains a greater number of optimal solutions, so we consider this method in further testing. The fact that TABUIMP obtains more effective solutions for certain types of problems, e.g., with W = 2 and T = 2, indicates that the use of the intra-route improvement routine is not always more effective than the inter-route improvement routine, and perhaps a more effective solution method may allow for inter-route improvements at certain times of the search.

If we test the effectiveness of NEWMAX and ARTMAX by the Wilcoxon Signed-Rank Test at a 5% level, we find that ARTMAX is more effective for one of the problem classes, while NEWMAX is not more effective for any. Therefore we consider ARTMAX to be the more effective method, and continue by comparing the effectiveness of this method with TINTRA. By the Wilcoxon Signed-Rank Test, we find that ARTMAX is more effective than TINTRA for 12 of the problem classes, while TINTRA is more effective for five. All five classes for which TINTRA is superior have W = 3, i.e, wide time windows, whereas all of the problem classes for which ARTMAX is superior, have W = 1 or 2. The minimum proportion of optimal reward found by ARTMAX is 0.96323, while the minimum proportion is 0.9764 for TINTRA, which again shows that TINTRA seems to be more robust, and therefore less likely to obtain poor solutions. The computational time required for ARTMAX is 1 hour and 2 minutes, so this method is the second quickest of the solution routines, and is significantly quicker than TINTRA which takes over 27 hours.

For problems with 100 customers, the computational time requirements prevented us from solving all 2700 of the problem instances with TABUIMP and TINTRA, and we only solved 25 problem instances from each problem class with these method. The Tables, A.12 and A.13, give the average performance and computational times (in the form hh:mm:ss) for the other five heuristics for each of the 2700 instances, while Tables A.14 and A.15 give the results for all of the heuristics over

the 675 instances (25 of each problem class) for which we obtained solutions using the Tabu Search methods. The computational time taken for TABUIMP to obtain the 675 solutions was approximately 60 hours, with TINTRA taking approximately 70 hours.

Over the 2700 problem instances, the methods REWAPP and ARTIF, obtain an optimal solution for very few of the problem instances, apart from with W = 3. The NEWMAX and ARTMAX methods obtain a number of optimal solutions, and appear to be the only methods that are able to consistently obtain optimal solutions with large numbers of customers. When we compare the effectiveness of NEWMAX and ARTMAX, by the Wilcoxon Signed-Rank Test at a 5% level, we find that ARTMAX is more effective for four of the problem classes, while NEWMAX is not more effective for any, so the use of the subgravity score in the construction phase appears to again lead to improved results.

Over the subset of problems for which the Tabu Search methods were applied, the TABUIMP method proved to be slightly more effective than the TINTRA method. Therefore the inter-route improvements prove to be more effective than the intra-route improvement routine that we consider. This again suggests that an improved implementation of Tabu Search may incorporate both the intra-route and inter-route improvement routines, or a more effective form of intra-route improvement routine that affects more than just the one customer.

We now compare ARTMAX, which was the most effective of the heuristics combined with steepest ascent, with TABUIMP on the subset of problems for which TABUIMP was applied. For these problems, TABUIMP obtains very few optimal solutions, apart from with time window type 3, but it still appears to obtain very good solutions, as indicated by the high proportion of the optimal solution value obtained. For these problems the minimum proportion of the optimal solution value obtained by TABUIMP is 0.98721, while the minimum for ARTMAX is 0.97725, so the superiority in worst encountered performance appears to be the major advantage of TABUIMP. Carrying out the Wilcoxon Signed-Rank Test at a 5% level, we find that ARTMAX is more effective for eight of the problem classes, while TABUIMP is superior for one. TABUIMP still appears to be relatively effective with wide time windows, but due to the excessive computational time of this method and the general superiority of the ARTMAX method over the problem instances solved, the ARTMAX method would be the one we would recommend for use on

		Artificial Window Threshold			
Number		1.0	0.9	0.7	0.5
20	Prop	0.99593	0.94481	0.88222	0.85741
	Obj	0.99994	0.99945	0.99809	0.99648
	Time	00:00:57	00:00:55	00:01:03	00:01:00
40	Prop	0.95037	0.66407	0.51333	0.47778
	Obj	0.99962	0.99798	0.99515	0.99305
	Time	00:12:18	00:10:13	00:11:52	00:12:37
60	Prop	0.88667	0.41519	0.32370	0.30296
	Obj	0.99935	0.99758	0.99442	0.99214
	Time	01:02:23	00:44:12	00:53:17	01:00:13
100	Prop	0.72667	0.28407	0.25963	0.25333
	Obj	0.99862	0.99721	0.99471	0.99318
	Time	10:18:50	05:25:36	06:56:29	08:14:25

Table 8.24: Results from Varying the Artificial Time Windows

problems of this type, i.e., large problems where we know that most customers can be serviced at their optimal value.

This testing has shown that for problems where all customers can be serviced at their optimal time, we can adapt methods based upon artificial time windows, in order to improve upon the effectiveness of the methods. For the problems tested, we found that setting artificial time windows about the time that the reward is at its maximum value, achieved the best results. In Table 8.24, we give the results over these 2700 problem instances, with the four different numbers of customers, by setting the artificial time windows at four different levels. These artificial windows are set so that respectively, 1.0, 0.9, 0.7 and 0.5 of each customer's maximum reward value is received at the time the customer is serviced in the initial phase of the heuristic. We give the proportion of optimal solutions, the average proportion of the optimal objective function and the total computational time required (in the form hh:mm:ss), for each time window width. These results show that there is a significant drop-off in the solution quality in moving from having the time windows at the maximum value of the reward, to having the windows at 90% of the maximum reward. The solution quality reduces monotonically as the artificial time windows get wider for these problems.

We further test the effect of these artificial time windows, by creating a set of test problems where the time windows are based on receiving proportions of reward

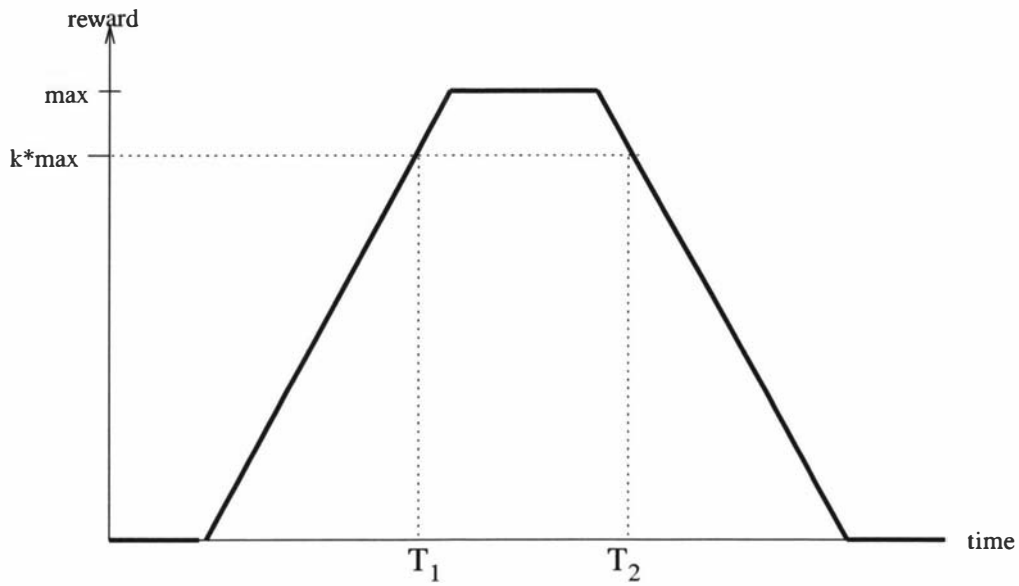

Figure 8.4: Reward Function for Testing

other than the maximum. We create test problems by creating tours such that all the available customers are serviced, with the time a customer is serviced in the base tour, being the time at which the customer's reward is a given proportion, k, of its maximum reward. Thus, we set the reward function to be as in Figure 8.4, if the customer is serviced at time T_1 or T_2 in the base tour. We randomly select whether the time of service will be when the customer's reward is increasing or decreasing, although we ensure that the first customer's reward is increasing and the last customer's reward is decreasing, and set the time limit of the problem to be equal to the time taken for the base tour. We create three sets of problem instances by setting k to be respectively 0.95, 0.9 and 0.8, and we generate 100 problem instances for each set, where we have 60 customers and we have the settings W=2, T=2 and R=3, that were previously described. This type of problem was chosen because the time windows enable the greatest flexibility in the solutions, which gives greater scope for differentiating between the solutions obtained.

The results from this testing are shown in Figure 8.5, where we use our artificial time window method, with the artificial time windows set at eight different proportions of the maximum reward value and the symbols noting which parameter values were tested. The y-axis gives the average proportion of the total available reward that is received by each method, where we note that if each customer can be serviced at a time when a proportion k of its maximum reward is available, then k is a lower bound on the proportion of reward that can be collected.

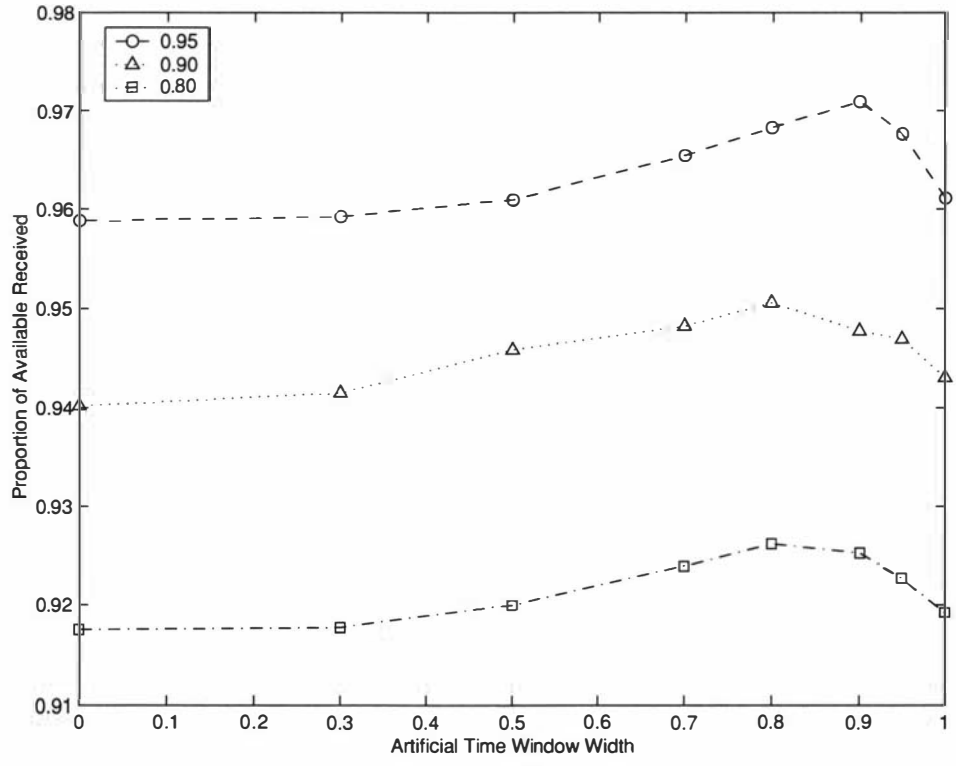

Figure 8.5: Solution Quality vs Artificial Window Setting

The computational time, to the nearest minute, that is required to solve these 300 problem instances is 37 minutes (window = 1.0), 27 minutes (window = 0.95), 19 minutes (window = 0.9), 12 minutes (window = 0.8), 11 minutes (window = 0.7), 9 minutes (window = 0.5), 10 minutes (window = 0.3) and 13 minutes (window = 0.0). Therefore the problems where the improvement phase took the longest time, were the ones with the narrowest time windows, with the time in the improvement phase being reduced until only half of the reward is within the time window. This is an indication of the compatibility of the initial solutions obtained and the final ones, as shorter computational time is required when fewer improvements are required.

The plot of the results show that for these problems, setting the time windows to encompass the time for which the reward is at its maximum value, does not obtain the best results. When we base the time windows on 95% of the maximum reward, the highest mean is obtained by setting the time windows about the time when 90% of the reward is available, and for rewards based on 80% and 90% of the maximum available, the highest means are with the time windows set around the time that 80% of the maximum reward is available.

The set of problems considered, involved all customers being able to be included in the solution route, with the minimum proportion of reward that is available being varied. When we set the time of servicing in the base route to be when the reward is slightly below its maximum, i.e., at 95% of the maximum, the policy of setting time windows at the maximum, which was the best policy when all customers could be serviced at their maximum, was no longer the most effective policy. Therefore this choice of time window width does not appear to be very robust, as small deviations from the required settings, lead to significant drop-offs in the performance of the heuristic. We found previously in Section 8.1.3 that the most effective method for the general forms of the problems considered, was to set the time windows around the time when at least half of the maximum reward is available. Therefore, we see that in order to obtain the most effective results from the artificial time windows, we need to select the level of the artificial time window effectively, and this requires some information about the type of problem being considered.

The OPTW

We carry out our testing on the OPTW, by using the data that we used in the previous section for the MCPTDR. For these problems, there is again the known optimal solution, with all customers being serviced. We create problem instances by setting the time windows to be equal to the time when the customer's reward is at its maximum. We note here that since problems with time window types 1 and 2 were both generated with the same specifications for the time the reward is at its maximum, there is no difference between the classes with these different time window types.

For these problems we carry out the four heuristics (GREEDY, KANTORB4B, CHEAP and APPEND) that we previously considered for the OPTW. Since finding an optimal solution to this problem consists of finding a feasible solution for a TSPTW problem, we also consider a TSPTW type of method, HARDEST, which is an insertion method whereby the customers are inserted in non-decreasing order of their time window width.

For these problems, with 20 customers, we give the number of optimal solutions obtained in Table 8.25. Solution methods 1 to 5 are respectively APPEND, CHEAP, B4B, HARDEST and GREEDY. The final four methods are different versions of Tabu Search. Methods 6 and 7 refer to our standard TABUIMP method, where

the first column refers to the solutions obtained via the initial stage, i.e., creating five random solutions and applying STEEPEST ASCENT to them, and the second column refers to that which has been improved via the Tabu Search. Methods 8 and 9 refer to a variation of TABUIMP, which we call TABUSTART, where we begin by applying the five heuristics of columns 1 to 5. Column 8 refers to the best solution obtained at the end of the initial phase, i.e., the best results from the five heuristics applied, with column 9 involving the solution after applying the Tabu Search.

From the Table we see that the heuristics are reasonably effective in obtaining the optimal solution. With the tight time windows, i.e., W = 1 and 2, each of the heuristic methods obtains the optimal solution for almost every problem instance. The more difficult problems are with the wider time windows, as there are greater numbers of routing options, and therefore a greater number of solutions that may be obtained that do not allow the improvements to obtain the optimal solution. From column 8, we see that between the heuristics, the optimal solution is obtained for all but one of the 2700 test problems considered. The optimal solution is obtained when Tabu Search is applied to this solution, as can be seen from the results for TABUSTART. The random initialization technique obtains the optimal solution for all of the problem instances, as is shown in column 6, so the Tabu Search method need not be applied to these problems. TABUIMP and TABUSTART took a total of 17 seconds and 18 seconds respectively, so the computational time for these problems is minimal.

The results with 40 customers are given in Table A.16 and there is a greater amount of distinguishability in the results obtained. We see that with the wider time windows and these time windows based on a good TSP tour, i.e., tour type equal to 2, the CHEAP method finds an optimal solution for each of the problem instances, whereas the other heuristics obtain relatively few optimal solutions. The problems with T = 3, particularly with the wider time windows, prove to be the easiest problems to solve, and this is due to there being many optimal solutions, of which many have significantly shorter distance travelled than for the base tour.

The best of the heuristics obtain an optimal solution for 2692 of the test problems, with the Tabu Search of TABUSTART obtaining an optimal solution for 7 of the remaining problems. The problem instance where this method did not obtain an optimal solution, was one where the TABUIMP method obtained the optimal

Class	Solution Methods								
(W,T,R)	1	2	3	4	5	6	7	8	9
(1,1,1)	100	100	100	100	100	100	100	100	100
(1,1,2)	100	100	100	100	100	100	100	100	100
(1,1,3)	100	100	100	100	100	100	100	100	100
(1,2,1)	99	100	99	99	99	100	100	100	100
(1,2,2)	100	100	100	99	100	100	100	100	100
(1,2,3)	99	100	97	99	99	100	100	100	100
(1,3,1)	100	99	100	100	99	100	100	100	100
(1,3,2)	100	99	100	100	100	100	100	100	100
(1,3,3)	100	99	100	100	100	100	100	100	100
(2,1,1)	100	100	100	100	100	100	100	100	100
(2,1,2)	100	100	100	100	100	100	100	100	100
(2,1,3)	100	100	100	100	100	100	100	100	100
(2,2,1)	100	99	98	99	99	100	100	100	100
(2,2,2)	100	98	99	100	100	100	100	100	100
(2,2,3)	100	98	98	99	98	100	100	100	100
(2,3,1)	100	100	100	100	100	100	100	100	100
(2,3,2)	100	100	100	100	100	100	100	100	100
(2,3,3)	100	100	100	100	100	100	100	100	100
(3,1,1)	92	100	91	94	94	100	100	100	100
(3,1,2)	95	100	94	98	93	100	100	100	100
(3,1,3)	93	100	92	97	91	100	100	100	100
(3,2,1)	95	94	94	97	90	100	100	100	100
(3,2,2)	96	94	92	98	93	100	100	99	100
(3,2,3)	97	94	93	97	89	100	100	100	100
(3,3,1)	98	100	97	99	96	100	100	100	100
(3,3,2)	97	100	99	99	97	100	100	100	100
(3,3,3)	97	100	100	99	96	100	100	100	100
Total	2658	2674	2643	2673	2633	2700	2700	2699	2700

Table 8.25: Number of Optimal OPTW Solutions Obtained - n = 20

solution in the initialization phase. The issue here, is that the solutions obtained by the heuristics are all similar, with Tabu Search unable to obtain an optimal solution from these starting solutions. Also of note is that the best of the solutions found by the five heuristics (given in column 8) obtained a greater number of optimal solutions than our TABUIMP (given in column 7). Therefore it appears that with our current Tabu Search method, the generation of high quality initial solutions is an important factor in obtaining good final solutions. The computational time required for TABUIMP is 4 minutes and 35 seconds, while TABUSTART takes 1 minute and 51 seconds, so the generation of good solutions, of which many are optimal, leads to reduced computational time for our implementation where we stop the search when we have obtained an optimal solution.

For 60 customers, the results are shown in Table A.17 and again we see the superiority of the CHEAP heuristic for the problems with tour type 1 and the wider time windows. The inferiority of TABUIMP on these problems, leads to the best of the five heuristics obtaining a greater number of optimal solutions than TABUIMP (2674 vs 2669). On the other types of problems considered, the TABUIMP method appears to be quite effective, with the greater initial solutions of TABUSTART leading to better final solutions. The main advantage of TABUSTART is that it begins with good solutions for the problems with tour type 1 and the wider time windows, thereby surmounting the shortcoming of the random initialization technique. There is again a major difference in computational time, with TABUIMP taking 36 minutes and 14 seconds, while TABUSTART takes 6 minutes and 49 seconds.

The findings from above are further shown and somewhat amplified in Table A.18 for 100 customers. Here the TABUIMP method is ineffective again with tour type 1 and the wider time windows, and it obtains far fewer optimal solutions than the best of the five heuristics for these problems. The computational time requirement for TABUIMP is 11 hours and 3 minutes, while TABUSTART takes 54 minutes to solve these problems. The most effective of the heuristics is HARDEST, although CHEAP is very effective with tour type 1. The TABUSTART method obtains an optimal solution for most of the problems considered, although there are still 40 problems for which it is unable to service all the customers. Of these, TABUIMP finds an optimal solution for 26, which leaves 14 problem instances for which neither Tabu Search method could obtain a feasible solution to the TSPTW.

We note that for each combination of time window and tour type, we created three problem instances by creating rewards with three different variabilities. For each of the templates for which the rewards are applied, all customers are serviced for at least one of the reward variabilities. This shows that there is nothing inherently in the time and locations that prevents the search methods from being able to include all customers, and it is therefore the overall effect of the time, location and reward dimensions that create difficulties for the methods.

Overall, we see that the TABUSTART method is effective in obtaining solutions to the given problems. The TABUIMP method, was unable to find the optimal solution for a number of the problems for which the time windows were based upon TSP tours, and so the major gain made by the TABUSTART method is in the unfair practice of applying the same method for solving a problem as was used for generating the problem instance. The time windows cause difficulties for the TABUIMP method, as it was unable to always recover from poor initial solutions, particularly for the TSP based problems with wide time windows, for which there are many feasible solutions available, but few of the solutions obtain the optimal reward. The construction of good initial solutions is an important factor in the effectiveness of the TABUSTART method, particularly as the construction of initial solutions by TSP based construction methods obtains very good solutions for these 'difficult' problems. The use of good initial solutions also significantly reduced the computational time required to solve these problems and this was due to the tight upper bound on solution quality that we had, whereby we were able to stop the search when the upper bound was attained.

We note that the problem instances considered in this section are the same as those that were considered for the MCPTDR in the previous section, except the time windows that were applied to the OPTW were those that were used to incorporate the time for which the customer's maximum reward was available in the MCPTDR. These problem instances are therefore more restricted versions of the MCPTDR instances, and thus solutions to the OPTW will also be feasible for the MCPTDR. We find that the imposition of the hard time windows about the time at which the reward is at its maximum, results in improved solutions being obtained by the solution methods, particularly with Tabu Search. We compare the results obtained, in Table 8.26, where TABUIMP1 is our implementation of TABUIMP for the MCPTDR, TABUIMP2 is our implementation of TABUIMP for the OPTW and

		Solution Methods			
Num Custs		TABUIMP1	TINTRA	TABUIMP2	TSTART
20	Num	2698	2700	2700	2700
	Prop	1.00000	1.00000	1.00000	1.00000
	Time	00:02	00:02	00:00	00:00
40	Num	2450	2515	2688	2699
	Prop	0.99957	0.99965	0.99993	1.00000
	Time	02:22	02:41	00:05	00:02
60	Num	1617	1655	2669	2697
	Prop	0.99856	0.99848	0.99988	0.99999
	Time	21:49	27:22	00:36	00:05
100	Num	266	254	639	665
	Prop	0.99790	0.99741	0.99952	0.99991
	Time	$\approx$ 60 hours	$\approx$ 70 hours	02:46	00:14

Table 8.26: Comparison of MCPTDR and OPTW Tabu Search Implementations

TINTRA and TSTART are the methods we used for the MCPTDR and OPTW respectively. We include the number of optimal solutions, the proportion of the overall optimal solution obtained and the computational time required (in the form hh:mm) for each method for each of the numbers of customers considered. With 100 customers, we recall that the MCPTDR methods only solved 675 of the problem instances, so it is over this subset of instances that we compare the methods.

The Table shows that the methods for the OPTW are far more effective than the MCPTDR methods in terms of the number of optimal solutions obtained and the total solution quality. Also, the computational time is far lower for the OPTW methods, and factors that cause the reduced computational time include the fact that the methods terminate when an optimal solution is found, and the reduced problem space in the presence of the hard time windows. Another important factor in the reduced computational time is the OPTW methods considering the customers to have constant rewards, so the actual reward received from a customer is not a decision variable; the time of servicing is only calculated in order to ensure feasibility, rather than for evaluating the solution quality.

These results show that modelling the MCPTDR as a hard time window problem may result in improved solution quality, and also increase the tractability of the problem by allowing techniques to be applied more readily. In the above case, we applied artificial time windows that were used throughout the search process,

and these led to improved solutions being obtained. These time windows were appropriate due to our knowledge that all the customers could be serviced while their reward was at its maximum value, thus the time windows incorporated the time interval for which each customer would be serviced within an optimal solution. In more general problems, we are not likely to know when the customer should be serviced, so artificial time windows of this exact form are unlikely to be applicable. The gains in computational time requirement by considering the problem to involve constant rewards, cannot be taken advantage of in this case, but gains may be made by restricting the time window over which the customer is available. The important factor for this method, is the identification of appropriate time windows, with an adaptive approach, i.e., time windows varying throughout the course of the search process, being one that might be particularly beneficial.

8.2.3 Testing Tabu Search Methods for the MCPTDR

We now consider a couple of small variations to the MCPTDR and consider the impact on our solution methods, particularly the ones involving Tabu Search. Previously, the reward received for each customer was zero at the start and the end of the time window, and for these problems there is some confounding present, as the tightness of the time windows also affects the rate of decline in the rewards, and so the impact of the time window width can not be assessed independently of the slopes that are present. We consider a set of problems for which the minimum reward need not be zero, and look at the effectiveness of our methods for these problems.

To generate these problems we use the 60 layouts that we used previously in Section 8.1.3, and for each layout we create a tour through each location, according to our randomized nearest neighbour method. We set the time limit of the problem to be equal to the length of the tour, and assign the time windows for each customer so that they may be serviced on this tour, although not necessarily at a time for which the maximum reward may be received.

We vary the types of problems generated by using three different settings of the parameter T, which relates to the widths of the time windows of the customers. The time window width of customer i is given as a proportion, k_i, of the time limit, with k_i uniformly distributed between 0.1 and 0.3 (T = 1), 0.4 and 0.6 (T = 2) and 0.7 and 0.9 (T = 3). For customer i, we randomly position the time window

of availability, $[t_{i1}, t_{i4}]$, around the time the customer is serviced in the base tour. The time, t_{i2}, that the customer's reward first reaches its maximum is uniformly distributed in the range $(t_{i1}, t_{i1} + 0.25 * (t_{i4} - t_{i1}))$, and the time, t_{i3}, when the reward begins to decrease is uniformly distributed in the range $(t_{i1} + 0.5 * (t_{i4} - t_{i1}), t_{i1} + 0.75 * (t_{i4} - t_{i1}))$.

Our reward functions are again of the form given in Figure 6.2, and we create four problem classes for each class of time window width, by varying the parameter R, which affects the range of values that the initial and final reward may take, i.e., the rewards at time t_{i1} and t_{i4}. The maximum reward, max_i, for each customer i, is uniformly distributed in the range (10,100), and we set the initial reward to be a proportion a_i of max_i, and the final reward to be a proportion b_i of max_i. For R = 1 we set a_i and b_i to be in the range (0.0,0.5), for R = 2 we use the range (0.5,1.0) and for R = 3 we use the range (0.0,1.0). For R = 4, we set the reward at the start and end of the time window to be equal zero, so this is the form of the reward function that we have previously considered.

Having the minimum value of the reward function to be greater than zero, can be considered to involve modelling constant penalties for non-service. Here, the minimum reward value that may be claimed, can be considered to be a penalty that is incurred for not servicing the customer. As the proportion of the maximum available reward that this penalty takes increases, the relative priority of ensuring the customer is serviced will increase, at the expense of ensuring the customer is serviced at the 'best' time.

By obtaining four forms of the reward function and three distributions of the time window width, we obtain 12 problem classes, with each containing 60 layouts of 60 customers. Our first consideration is the effectiveness of our heuristics on these data sets, and the results from this testing are shown in Table 8.27, with the methods consider being the same as were previously tested on the MCPTDR. For the ARTIF method we found that there was little difference between setting the time windows around a given proportion of the reward or around a given proportion of the length of the time windows, so the results given in the Table are with the time window being a given proportion of the time between when the maximum and minimum available rewards can be collected. For these problems, we again found that the best value of the proportion was about 0.5, so this value was used to obtain the results in the Table.

Classes	Construction Techniques					
(T,R)	B4B	BRIDPOT	REWAPP	B4BAPP	CHEAP	ARTIF
(1,1)	2797.47	2799.70	2824.95	2793.97	2759.70	2843.12
(1,2)	3082.00	3089.24	3092.92	3084.47	3069.03	3096.29
(1,3)	2882.98	2915.88	2917.68	2921.42	2880.89	2932.68
(1,4)	2689.38	2704.27	2714.25	2696.68	2656.98	2761.47
(2,1)	2928.91	2911.72	2914.08	2928.16	2850.74	2972.86
(2,2)	3082.39	3099.21	3097.48	3103.54	3052.72	3094.75
(2,3)	2979.06	3021.47	3002.08	3005.05	2931.01	3024.55
(2,4)	2817.88	2829.89	2856.57	2857.50	2779.50	2901.18
(3,1)	3017.96	3017.77	2977.58	2998.34	3000.78	3049.36
(3,2)	3194.81	3209.19	3206.75	3199.37	3191.62	3187.24
(3,3)	3141.87	3130.20	3139.72	3137.34	3130.05	3151.95
(3,4)	2933.55	2946.05	2931.00	2932.61	2899.42	2994.35
Total	2962.35	2972.88	2972.92	2971.54	2933.54	3000.82

Table 8.27: Heuristic Results with Non-zero Initial Rewards

The Table shows that the ARTIF method obtains a much higher overall mean reward than the other methods. If we look at the performance of this method on the individual problem classes, we see that the relative effectiveness of this method is not as high for R = 2 and to a lesser extent, for R = 3. When we carry out pairwise tests on the difference between the results obtained by ARTIF against those for each of the other methods (tested via the Wilcoxon Signed-Rank Test at a 5% level, with a total of five tests carried out), we find that for type 2 rewards, the ARTIF method is significantly more effective than CHEAP for one problem class, but it is not more effective than any of the other methods for any of the other problem classes. We find, however, that we can obtain higher rewards with the ARTIF method for R = 2 when we use a lower value of the time window width parameter, so that the minimum proportion of the maximum reward that needs to be collected from each customer is lower. Thus, the parameters used in the method need to be changed in order to increase the effectiveness for these problems.

With R = 3, the ARTIF method is not significantly more effective than either BRIDPOT or B4BAPP for any of the time window widths, but it is significantly more effective than each of the other methods for all classes with type 1 and type 4 rewards. Therefore, the ARTIF method with time window width parameter 0.5, is the most effective of the methods for the problems with rewards that may be very

variable. For these problems, there is a greater advantage in focusing on servicing customers at times where their reward is highest, and by inducing the customer to be serviced while its reward is high, improved solutions can be found. More effective results may be obtained with less variability in the available reward, i.e, with R = 2 and 3, by loosening the required reward parameter, and this demonstrates again that the selection of the width of the artificial time windows is an important decision in determining the effectiveness of any method that exploits their use.

We also use some implementations of Tabu Search on these problem instances, in order to see if improved solutions may be obtained. We recall that in the previous investigation into the OPTW, we found that improved solutions could be found for a set of problems where all customers could be serviced, by using Tabu Search starting from the solutions obtained by a number of effective heuristic routines. We wish to consider this approach further for these problems for the MCPTDR.

We create four different implementations of Tabu Search for these test problems. TABU1 is our standard implementation of TABUIMP, where the construction phase consists of randomly selecting which customer to insert next, and putting this customer in its best position in the current route. TABU2 involves an adaptation of this construction method, where we randomly select which customer to include next, and append it to the end of the current route, and this enables a greater range of initial solutions to be obtained. TABU3 involves the five initial solutions being obtained by different heuristic routines. The heuristics we consider are B4B, BRIDPOT, REWAPP and CHEAP and ATMAX, which is a method that inserts customers in non-increasing order of maximum reward, in a position so that their maximum available reward will be collected. Therefore the methods we consider are quite diverse as, for example, CHEAP does not consider reward at all when generating solution routes while ATMAX does not explicitly consider the distances when selecting which customers to include. Our fourth implementation of Tabu Search, TABU4, involves generating initial solutions by using a wide range of the parameter within the ARTIF method, described previously. Therefore, in order to create five initial solutions, we set the artificial width parameter for ARTIF to be successively 0.0, 0.25, 0.5, 0.75 and 1.0. These different parameters direct the search in different areas, which should also create diverse solutions.

The results obtained from this testing are shown in Table 8.28, where the columns TABU3A and TABU4A give the results obtained by the initial phase of

Classes	Search Methods						
(TW,RT)	TABU1	TABU2	TABU3A	TABU3	TABU4A	TABU4	RAND10
(1,1)	2894.68	2885.38	2872.38	2893.42	2877.36	2879.68	2883.64
(1,2)	3151.99	3148.03	3134.01	3148.63	3130.71	3139.62	3150.29
(1,3)	2985.70	2985.09	2967.47	2982.87	2964.60	2970.02	2980.29
(1,4)	2794.90	2791.97	2765.65	2792.39	2768.66	2777.17	2783.57
(2,1)	3037.82	3036.11	3014.30	3030.64	3009.38	3011.42	3036.85
(2,2)	3163.68	3159.15	3143.07	3153.85	3137.66	3146.25	3162.63
(2,3)	3088.51	3087.51	3067.17	3083.59	3056.51	3068.73	3089.87
(2,4)	2966.63	2964.97	2924.42	2958.11	2939.07	2942.09	2961.90
(3,1)	3120.35	3119.73	3076.45	3112.67	3083.32	3099.84	3104.35
(3,2)	3246.57	3250.09	3235.56	3243.80	3228.20	3236.68	3242.75
(3,3)	3203.71	3202.29	3183.84	3204.75	3184.38	3198.19	3203.93
(3,4)	3068.17	3061.89	3013.49	3054.97	3033.12	3054.75	3050.12
Total	3060.23	3057.68	3033.15	3054.97	3034.41	3043.70	3054.18

Table 8.28: Results from testing Tabu Search Methods with Non-zero Initial Rewards

TABU3 and TABU4 respectively, i.e., the best of the results obtained by the five heuristics considered. We also include a column RAND10, which gives the results obtained by taking 10 applications of RANDOMCONSTRUCT, applying STEEPEST ASCENT to each of these, and storing the best result obtained.

From the Table we see that the highest overall mean is obtained by the standard implementation of Tabu Search, and this appears to be marginally more effective than the other methods. We therefore test the differences between the results obtained by this method, and each of the other methods given in the Table (a total of four tests). When we test the differences between the standard TABU1 method and the other implementations of Tabu Search, using the Wilcoxon Signed-Rank Test to a 5% level, we find that there is no significant difference between TABU1 and TABU2 for any of the problem classes, so the method of creating random initial solutions didn't significantly affect the effectiveness of the methods. TABU1 is significantly more effective than TABU3 for two of the problem classes, and significantly more effective than TABU4 for nine of the problem classes; this shows that for these problems, a more effective method is to start with random solutions, and letting the search mechanisms obtain good, final solutions, rather than trying to find good initial solutions using some form of biased techniques. Randomization

enables certain features of the problems to be ignored, and thus more diverse solutions are obtained, which is an important factor for allowing the search to move away from local optima.

If we compare TABU1 with RAND10 we find, using the Wilcoxon Signed-Rank Test at a 5% level, that TABU1 is significantly more effective for five of the problem classes, while RAND10 is not significantly more effective for any of the classes, which shows the merits of the Tabu Search procedure. What is of most interest here is the computational times involved as TABU1 takes a total of approximately 24 hours to run while RAND10 takes approximately 32 hours. Since TABU1 begins by creating five random solutions and improving each of these by using STEEPEST ASCENT, we would estimate that this procedure takes approximately 16 hours. The search procedure for our Tabu Search implementation therefore takes approximately 8 hours, which is half as long as generating five random solutions and improving these. This shows that the full implementation of our STEEPEST ASCENT search procedure is quite computationally expensive, and our Tabu Search procedure, when well-constructed, can be a more efficient method.

All the implementations of Tabu Search are able to significantly improve over the results that were obtained via the non-random construction heuristics that were improved by using STEEPEST ASCENT. This is as we would expect, and we suggest that the previous case where the heuristics obtained results that were competitive with our implementation of Tabu Search, were just due to the few improvements that were able to be made.

The comparison of Tabu Search with the steepest ascent heuristics is obviously not a fair one, as the computational time required for each of these heuristic methods, was approximately 30 minutes, whereas the Tabu Search methods took between 17 and 26 hours to complete. So if very fast solutions were required, the heuristics by themselves seem to find reasonable solutions, although these can be greatly improved if there is enough time to run a more sophisticated search procedure like Tabu Search. Also, since STEEPEST ASCENT applied to well-constructed solutions took approximately 30 minutes to obtain solutions, while obtaining solutions to 10 randomly generated solutions with STEEPEST ASCENT took over 30 hours, we see that the large computational burden arises by using this improvement routine from poor initial solutions. Therefore the construction of good initial solutions from which to apply STEEPEST ASCENT is an important step in reducing

Classes	Search Methods		
(T,R)	TABUIMP	TABU0.5	TABU0.75
(1,1)	2894.68	2854.29	2762.56
(1,2)	3151.99	3035.58	2851.07
(1,3)	2985.70	2902.81	2770.02
(1,4)	2794.90	2792.19	2738.68
(2,1)	3037.82	3009.68	2947.02
(2,2)	3163.68	3078.24	2958.32
(2,3)	3088.51	3033.33	2927.98
(2,4)	2966.63	2958.73	2902.15
(3,1)	3120.35	3101.20	3031.73
(3,2)	3246.57	3191.07	3078.64
(3,3)	3203.71	3168.10	3057.69
(3,4)	3068.17	3061.26	3010.61
Total	3060.23	3015.54	2919.71

Table 8.29: Comparison of Standard Tabu Search with Artificial Time Windows

the computational requirement.

Our final concern with these test problems, is to consider the efficacy of the artificial time windows themselves, within a more advanced search routine. The time windows in this case can be used to give some form of service guarantee, so that a customer's desired service time (as represented by the time at which the reward is at its maximum value) is not violated by too great an extent. Therefore the customer's level of service is explicitly considered, and the time windows are able to ensure that the selected customers will all be serviced to a reasonable standard.

We carry out an implementation of Tabu Search, with hard artificial time windows created and used throughout the search. We create two different methods by setting the time windows at respectively 0.5 and 0.75 of the time between when the minimum and maximum rewards are available. The methods obtained are TABU0.5 and TABU0.75, and the results from these methods are given, along with those for TABUIMP, in Table 8.29.

From the Table we see that TABUIMP obtains a higher overall mean than TABU0.5, with TABU0.5 having a far higher mean than TABU0.75. If we carry out the Wilcoxon Signed-Rank Test on the two methods with artificial time windows, we find that at a 5% level, the TABU0.5 method is significantly more effective

than TABU0.75 for all 12 problem classes. This shows that the time windows used within the TABU0.75 method are too strict to allow effective solutions to be obtained, and so these time windows would only be applicable when high levels of servicing are required, with the cost of this approach being the much lower overall reward that is obtained.

When we compare TABUIMP with TABU0.5, again by the Wilcoxon Signed-Rank Test at a 5% level, we find that TABUIMP is significantly more effective for seven of the problem classes. These problem classes are with R = 2 and 3, with each value of T, and with R = 1 and T = 1. With R = 2 and 3, there is less variability in the reward that can be received from each customer at different times, so there is a higher priority to maximizing the number of customers serviced. The restrictions from applying the artificial time windows lead to fewer customers being able to be serviced, and thus less effective solutions being obtained.

Of interest with this comparison, is that there is no significant difference between the effectiveness of the TABUIMP and TABU0.5 methods with R = 4 (which is the form of reward function we previously considered as our primary reward function) and the only significant difference with R = 1 is with tight time windows. With tight time windows, the time of availability restricts the number of customers that are able to be serviced, so further reducing the time of availability limit the numbers serviced and reduce the effectiveness of the method. With R = 4, the TABU0.5 method ensures that at least half of the available reward is received from each serviced customer, and this allows a reasonable service guarantee, without unduly affecting the solution quality. One additional benefit of the application of the artificial time windows, is the reduction in computational time that comes about by restricting the time each customer is available, which restricts the size of the neighbourhoods that need to be evaluated by our local search operators.

Therefore we have seen that when a Tabu Search method incorporates hard, artificial time windows, that are set at an appropriate level, effective solutions are able to be obtained for certain types of problems. These solutions are obtained in a shorter amount of computational time, and involve fewer customers being serviced, but those that are serviced receive a higher level of service. This approach appears to be effective for reward functions where there is a large amount of variability in the reward received over time, particularly when the actual time windows are not too restrictive, and would also be appropriate for many practical situations for

which the level of customer service is of greater concern than the actual reward received.

8.2.4 Clustered OPTW and MCPTDR Problems

The previous problems we have considered have been based upon tours in which the optimal solution can be obtained without the inclusion of non-compulsory waiting times. We now investigate the effectiveness of our methods for problems for which waiting time is required in order to service the customers effectively. The problems of this type that we consider, involve small clusters of customers that need to be pieced together to create a solution route.

The method we use for generating the clusters, was previously described in Section 7.2.1. We create templates of clusters containing between four and six points, where these points are placed within the Euclidean $[0, 0.1]^2$. We create a total of 60 points, so the last cluster may contain fewer customers than the others. For each cluster, we use a self-correcting method for generating the relative time windows within each cluster so that there is only one time feasible path through the customers. We then randomly allocate the location of the clusters so the points are positioned within the Euclidean $[0, 1]^2$, and we set the actual values of the time windows. We create three different types of problems by varying the parameter O, which relates to the level of overlap between the clusters. For O = 1, we ensure that only one cluster at a time is feasible, for O = 2, two clusters at a time are feasible (and it is not possible to service the customers from more than one cluster) and for O = 3, we randomly allocate the time windows throughout the interval $[1, 5.5]$. The time windows were sufficiently separated in the cases with O$\leq$2, that it was always possible to service the first customer from the next cluster upon completing the service of the current cluster. The problem classes identified are ones where the ability to schedule all the customers from a cluster is the important concern, where the selection of the clusters to service is also required and where decisions relate to which clusters to service and which customers within the cluster to select.

The first problems we generate in this manner, are instances for the single-vehicle OPTW. We create nine classes of problems, by selecting three different distributions of the rewards of the customers (given by the parameter R), for each of the three values of O. The reward distributions are with all customers having a reward equal to 10.0 (R = 1), high variability, i.e., rewards from the distribution

Classes	Construction Techniques						
(O,R)	CLUSTER	APPEND	GREEDY	CHEAP	B4B	BEST	TABU
(1,1)	600.00	596.00	595.20	594.00	595.00	599.20	600.00
(1,2)	3297.61	3290.52	3288.81	3287.79	3288.34	3294.79	3297.61
(1,3)	3285.70	3267.25	3269.56	3276.81	3271.64	3282.80	3285.70
Mean(1)	2394.43	2384.59	2384.52	2386.20	2384.99	2392.26	2394.43
(2,1)	337.40	320.40	329.80	330.40	333.40	335.80	337.40
(2,2)	1944.58	1887.13	1827.38	1923.29	1930.35	1937.60	1943.95
(2,3)	2178.03	2137.06	2154.32	2163.41	2161.30	2173.40	2176.93
Mean(2)	1486.67	1448.20	1437.17	1472.37	1475.02	1482.27	1486.09
(3,1)	298.60	321.40	320.40	324.20	326.00	331.40	334.40
(3,2)	1682.11	1874.54	1818.29	1877.71	1884.30	1919.86	1926.28
(3,3)	1845.80	1984.30	1997.09	1983.57	2004.84	2030.46	2037.88
Mean(3)	1275.50	1393.41	1378.59	1395.16	1405.04	1427.24	1432.85
Total	1718.87	1742.07	1733.43	1751.24	1755.02	1767.26	1771.13
No Cross		1721.55	1705.79	1653.90	1714.07	1753.95	

Table 8.30: Testing Clustered OPTW Instances

$U(10, 100)$ for R = 2, and cluster-dependent distributions for R = 3. For these cluster-dependent rewards we partition the interval (10,100) into k equal length intervals, where k is the number of clusters, we obtain a random ordering of the clusters and set the reward so that the rewards for the customers from the jth cluster via this ordering are uniformly distributed in the range $(10+(j-1)*(100-10)/k, 10+j*(100-10)/k)$. This reward distribution favours particular clusters, whereas the other two distributions involve the 'attractiveness' of a cluster being determined according to the individual rewards of the customers in the cluster.

The results from this OPTW testing, are shown in Table 8.30, where 50 instances were solved for each problem class. The rows 'Mean(i)' refer to the mean reward obtained by each method over the problem instances for which O = i. The column CLUSTER refers to a method obtained from the problem generation phase, which involves inserting whole clusters into the solution route. This method therefore directly obtains the optimal solution for O = 1 and 2, and for O = 3 we use a simple branch-and-bound method to find the optimal solution that can be obtained by servicing all the customers from a cluster consecutively.

The four methods: APPEND, GREEDY, CHEAP and B4B were the methods that were previously applied to the OPTW in our preliminary testing and BEST

refers to the highest reward obtained by these four methods. The method TABU, is our full implementation of TABUIMP, and it took a total of 1 hour and 49 minutes to run, whereas the four heuristics each took approximately 3 minutes to obtain their solutions.

The last row, 'No Cross' refers to the implementation of the four heuristics without the use of the Cross routine. The solutions obtained to the problems with O = 1 were identical with and without the Cross routine, and the major difference is with O = 2, i.e., when there are two clusters available at a time and decisions as to which cluster to service need to be made. These decisions become trivial when all customers are placed in tours, and the Cross routine is able to exchange the clusters between routes. For the CHEAP routine, we find that the mean reward obtained for the problem classes with O = 2 decreases from 1472.37 to 1295.25 when the Cross routine is removed. In this heuristic, the selection decisions are made, without referring to the available rewards, and this method therefore requires the application of appropriate improvement routines for problems where the selection of customers is an important decision.

Overall, we see that for these problems, the TABUIMP method is quite effective, and it obtains the optimal solution to 297 of the 300 problems which have O = 1 or 2. These problems are relatively easy to solve, as there are few options as to where a customer may be placed, and one would therefore expect that the optimal solution would be obtained, particularly by a metaheuristic method. In each of the instances for which the TABUIMP method fails to attain optimality, there is one customer that is unable to be serviced within the feasible path through its cluster. In this case, a specific form of search that focuses on the suboptimal cluster, would lead to improved results, so some modification of the metaheuristic to perhaps allow for temporary infeasibility may be useful.

For the problem instances for which O = 3, we find that the TABUIMP method obtains the highest mean rewards, as would be expected. The CLUSTER method is ineffective for these problems, which shows that critical decisions to depart from a partially serviced cluster, are required in order to obtain effective solutions to these problems. We find that, with O = 3, the TABUIMP method services customers from on average 7.52 clusters for R = 1, 7.70 clusters for R = 2, and 7.40 clusters for R = 3, and services all the customers within each cluster, on average, a proportion 0.669, 0.586 and 0.669 of the time for R = 1, 2 and 3 respectively. Therefore, the

lowest completion proportion is for R = 2, in which the variability between the rewards are highest and there is no systematic relationship between the clusters and rewards. The fewest clusters are serviced when the distributions of the rewards are based upon the clusters, in which case greater gains are obtained by completing the service of a cluster which contains high rewards.

We now create problem instances for the MCPTDR using the same approach. We generate the time windows, using the same methods as were used for the OPTW in this Section, but these time windows now represent the time at which the customer's reward is at its maximum for the piecewise linear reward function. If (t_{1i}, t_{2i}) is this inner time window for customer i, we set the overall time window to be $(t_{1i} - 2(t_{2i} - t_{1i}), t_{2i} + 2(t_{2i} - t_{1i}))$, with the decline in the reward function set so the reward at the start and the end of the overall time window is equal to zero. This creates problems for which there is a much wider time of availability for servicing the customer, compared with the time at which the maximum reward is gained, i.e., the customer's reward is at its maximum value for one fifth of the time over which the customer may be serviced. These wider time windows were chosen since we previously found that reward functions of this type created problems for which our methods had most difficulty in obtaining optimal solutions.

We used the same levels of the overlap as for the OPTW, with again there being large separation between the times that successive clusters were available for O$\leq$2. This form of problem creates the additional feature that each cluster can be arrived at prior to the start of its time window. Since the reward at the time the customer becomes available is zero, we need to incorporate techniques which allow for the inclusion of non-compulsory waiting times, in order to service the customers effectively.

Our first testing, shown in Table 8.31, tests the effectiveness of the construction techniques that we found to be the most effective when applied to the MCPTDR, in the preliminary testing of Section 8.1.3. These methods are therefore REWAPP, BRIDPOT (given as BRID in the Table) and B4BAPP, and all three of these methods obtain their initial solution by evaluating the potential reward that may be received from the customer, rather than the reward that will be received at the time at which the service vehicle arrives at the customer's location. We also apply three versions of the ARTIF method, which was the method that proved to be the

Classes	Construction Techniques						
(O,R)	CLUST	REWAPP	BRID	B4BAPP	ART1.0	ART0.9	ART0.5
(1,1)	600.00	507.00	568.83	572.07	595.00	593.08	546.74
(1,2)	3290.75	2820.87	2926.69	2817.26	3275.95	3256.92	3045.65
(1,3)	3322.87	2675.38	2783.22	2754.62	3301.23	3286.41	3027.60
Mean(1)	2404.54	2001.08	2092.92	2047.99	2390.73	2378.81	2206.66
(2,1)	335.20	282.81	299.09	301.73	320.80	325.38	304.43
(2,2)	1939.69	1694.21	1710.69	1643.43	1880.36	1892.00	1794.76
(2,3)	2199.15	1837.60	1963.98	1937.38	2162.33	2151.17	2004.94
Mean(2)	1491.35	1271.54	1324.59	1294.18	1454.50	1456.19	1368.04
(3,1)	292.80	313.16	316.19	317.81	320.78	326.09	328.05
(3,2)	1661.03	1793.63	1767.41	1773.70	1857.79	1879.32	1872.15
(3,3)	1861.44	1897.83	1907.18	1904.85	1999.93	2061.87	2022.80
Mean(3)	1271.76	1334.87	1330.26	1332.12	1392.83	1422.43	1407.67
Total	1722.55	1535.83	1582.59	1558.10	1746.02	1752.47	1660.79
Time		08:47	07:02	13:02	09:46	09:28	08:32

Table 8.31: Testing Construction Techniques for Clustered MCPTDR Instances

most effective of the methods from Tables 8.6 and 8.7, particularly for the problems that contained piecewise linear reward functions. The versions of ARTIF we consider are ART1.0, ART0.9 and ART0.5, in which the time windows encompass the time at which, respectively, 100%, 90% and 50% of the maximum reward is available. These levels were chosen, since we know that the maximum reward may be claimed from each customer for problems with O = 1 and 2, so we include the high levels of reward for this case, and for O = 3 we have a more general problem, so we use the threshold of 50% of the maximum available reward, which was the most effective value we found for the general problems of Tables 8.6 and 8.7. The other method given in the Table, CLUST, gives the maximum reward that can be received from servicing whole clusters, where each customer in the cluster is serviced at a time when its maximum reward is available. This method will again be optimal for the problem classes for which O = 1 or 2. Table 8.31 gives the results in the same format as in Table 8.30, except an additional row 'Time' is included, and this gives the total computational time (in the format mm:ss) that was required by each of the methods.

Table 8.31 shows that the methods based on the ARTIF method are the most effective on these problem instances. The methods ART1.0 and ART0.9 are the

most effective for the problems with O = 1 and 2, and these are the problems for which we know that in the optimal solution, all the customers will be serviced at a time for which their maximum reward will be received. This again shows the advantage of the appropriate selection of the width of these artificial time windows, according to some knowledge of the problem characteristics. These methods do not have the shortest computational times, unlike in the preliminary testing of Section 8.1.3, as in these problems, particularly for the problems with O = 1 or 2, the feasible neighbourhoods are small and the relative gains from further restricting the neighbourhoods, by applying the artificial time windows, are diminished. Therefore the additional time spent in applying STEEPEST ASCENT twice, is not less than the gains by starting the full search from good solutions, which were created in the presence of the artificial time windows.

Of the other methods considered, the BRIDPOT method proves to be the most effective. For the current problem classes, the approach of the BRIDPOT method, which considers the next two customers when assessing the appending of a new customer to the route, thus appears to be more effective than the myopic B4BAPP method and the subgravity approach of the REWAPP method, which assesses the effect of all other available customers, when most of these customers are not relevant to the routing decisions, particularly for the problem classes for which O = 1 or 2.

We now consider the effectiveness of different implementations of our Tabu Search methods on these test problems. We applied our full implementation of Tabu Search, TABUIMP, to the set of problems and found that the solution quality obtained was quite poor, since our method evaluates each potential alteration of the routes according to the reward that will be received when the customer is serviced at the time the vehicle arrives at the customer. Since large periods of waiting time are required in the solution, the reward that is used for evaluation is considerably lower than the actual reward that could be received, and thus, this method is inadequate for these problems. Therefore the methods we created were intended to allow more effective evaluation of the reward received.

We created the three methods: TS1.0, TS0.75 and TS0.5, which involved the permanent application of artificial time windows on each customer. These time windows were set so that a minimum proportion of the maximum available reward would be received for each customer, where the proportion of reward received is

given as the number within the name of the method. Another method we implemented was T WAIT, which is a Tabu Search method, in which each proposed alteration to the current solution is evaluated exactly, by calculating the reward that would be received if the move was made (including the optimal waiting time for the new solution). Due to the additional computational requirements of evaluating the solution exactly, we employ a restricted neighbourhood within T WAIT, where the only inter-route improvement routine we include is the inter-route Move routine.

The final method we consider is TADAPT, which is an implementation of Tabu Search in which we vary the width of the time windows, so as to attract the search toward certain areas of the search space at various times during the search. We begin by creating initial solutions, using the original time windows, and commence the Tabu Search phase by randomly selecting routes from the current memory, until all customers are included within a route. The details of the Tabu Search iterations are given in Algorithm 8.3.

We define the current time window of customer $i, \forall i = 1, \ldots, n$ to be given by $(c_{i,0}, c_{i,1}, c_{i,2}, c_{i,3})$ and the original window is given by $(o_{i,0}, o_{i,1}, o_{i,2}, o_{i,3})$. After each iteration of the Tabu Search, we randomly select *num_changed* customers, with probability proportional to the minimum of t_i, the number of iterations since customer i's time window has been adapted, and non_i and sol_i, which are the number of iterations that the customer has been excluded from or included in the solution. For the customer in the solution we set non_i to be a large number, and we set sol_i to be large when the customer is not in the solution. If the customer is currently serviced in the solution, we attempt to increase the width of the time window for the customer, which allows greater flexibility in the current route. If the customer is not currently serviced in the solution, we decrease the width of the time window, so as to force the customer to be evaluated for service at a time at which its reward is high, which ensures the attractiveness of the reward is high and therefore promotes the insertion of the customer into one of the current solution routes. The maximum width of time window allowed is equal to the original time window width, while the minimum allowed width is equal to the length of time over which the customer's maximum reward may be obtained.

We adapt the time windows by selecting a proportion p, in the range (0.05,0.20), and changing $c_{i,0}$ by a value of $p * (o_{i,1} - o_{i,0})$ and changing $c_{i,3}$ by a value of

Algorithm 8.3 function TADAPT

Randomly select an improvement routine,
with probability proportional to previous success
Carry out best non-tabu move for the routine
Update the Current Solution
for $i = 1$ **to** n **do**
    $t_i \leftarrow t_i + 1$, $MIN_i \leftarrow min(t_i, sol_i, non_i)$, $p_i \leftarrow m * MIN_i$
end
for $i = 1$ **to** *num_changed* **do**
    Randomly select a customer, a, according to the probabilities, p_k
    $\delta \leftarrow 0.05 + U(0,1) * 0.15$
    $t_a \leftarrow 0$, $p_a \leftarrow 0$
    if (customer a is in the solution) **then**
        if $(c_{a0} - \delta * (o_{a1} - o_{a0}) \geq o_{a0})$ **then**
            $c_{a0} \leftarrow c_{a0} - \delta * (o_{a1} - o_{a0})$
            $c_{a3} \leftarrow c_{a3} + \delta * (o_{a3} - o_{a2})$
        else
            $c_{a0} \leftarrow o_{a1}$
            $c_{a3} \leftarrow o_{a2}$
        end
    else
        if $(c_{a0} + \delta * (o_{a1} - o_{a0}) \leq o_{a1})$ **then**
            $c_{a0} \leftarrow c_{a0} + \delta * (o_{a1} - o_{a0})$
            $c_{a3} \leftarrow c_{a3} - \delta * (o_{a3} - o_{a2})$
        else
            $c_{a0} \leftarrow o_{a0}$
            $c_{a3} \leftarrow o_{a3}$
        end
    end
end
Update the Current Solution to Ensure Feasibility

end

$p * (o_{i,3} - o_{i,2})$, where the direction of the change, i.e., whether we increase or decrease the time window width, is determined by whether i is serviced in the current solution or not. If the new width exceeds the allowable time window width, i.e., $c_{i,0} < o_{i,0}$ we reset the time window width at its minimum level, i.e., $c_{i,0} = o_{i,1}$ and $c_{i,3} = o_{i,2}$, and continue. Similarly, if the new width is less than its allowed minimum, i.e., $c_{i,0} > c_{i,1}$, we set the time window width at its maximum level, i.e., $c_{i,0} = o_{i,0}$ and $c_{i,3} = o_{i,3}$, and continue. Since the time window for each customer is equal to its original time window at the start of the Tabu Search phase, we see that the first time that each customer which is serviced within the solution routes has its time window altered, its time window will be set to just encompass the time at which the maximum reward is available.

One feature which may occur when the time windows are updated, is that if the time window is made narrower, the current time of servicing the affected customer may need to be delayed, which may make the current route become infeasible with regard to the time window constraints. We overcome this problem, by removing the first customer whose time window is violated, and inserting it into the first available route, and continuing to exclude customers until the current solution becomes feasible again. There are many other features which could be included within the TADAPT method, but we create a quite simple method, in which, after every iteration, we alter the time windows of *num_changed* customers. We found that reasonable results were obtained in preliminary testing with three time windows changed in each iteration, i.e., setting *num_changed* = 3, and it enables many time windows to be altered within the course of the search, without evolving the time windows so quickly as to dramatically alter the current state of the problem from one iteration to the next.

We illustrate the solutions obtained by this process in Figures 8.6 and 8.7, for problem number 322, which has O = 3 and R = 3. This is a single vehicle problem, so we refer to the route with the highest current reward as the solution route, with all other customers placed within other routes. Figure 8.6 shows the best solution that has been obtained during the initialization phase, with the x-axis giving the time dimension for the problem and the y-axis gives the reward for the customer, with the height indicating the available reward for this customer and the vertical dotted line extending down to the actual value of the reward that is received when the customer is serviced. The x-value for this line indicates the time at which the

customer is serviced in this solution. We only show the customers that are in the current solution, with the solid horizontal bar showing the time window for which the customer's reward is at its maximum value, and the dotted bars at either end indicating the time at which the customer may be serviced. We note that the distribution of the rewards favours certain clusters over others, so the customers from the same cluster will have similar time windows and similar maximum reward values. The initial solution obtains a total reward of 1855.736, and we see by the length of a number of vertical dotted sections that many of the customers are serviced at a time for which their reward received is much lower than its potential.

Figure 8.7 shows the solution obtained at the end of the TADAPT routine, with the first plot showing the current time windows that are being used within the search and the vertical bar again showing the difference between the potential and actual reward received, whereas the second plot shows the original time windows for each of the customers in the solution, with the y-value here corresponding to the customer number. The total reward obtained by this method is 2613.993, so the effective inclusion of some new clusters, together with the intensification into servicing existing clusters at a higher level, leads to a much higher reward than in the initial phase. The customers that are serviced are mostly serviced at a high level, as indicated by the few vertical dotted lines, although there are a few customers serviced between the times of 4.0 and 4.5 for which lower rewards are obtained. We see from the second of the plots from this Figure, that the customers serviced at this time come from different clusters, so there is no expectation of these customers being time window compatible, which means that this does not show an obvious flaw in the route obtained (which would be the case if the customers within a single cluster were poorly serviced, when we know that they may all be serviced while their reward is at its maximum). This solution was obtained quite early on in the search procedure and in the overall problem the mean time window width is 0.2591 at the time this solution is obtained, whereas the mean initial time window width was 0.5062 and the mean minimum time window width allowed is 0.1012. For the customers within the solution itself, the mean time window width is 0.1942, so the time windows of the customers in the solution are, on average, narrower than those for the customers outside of the solution route, so this phase of the search has led to improved solutions by intensifying the search around times for which the maximum reward may be received for many of the customers within the solution

route.

The overall results from our testing are shown in Table 8.32, which gives the mean reward obtained by each method for each set of problems, with summaries of the mean reward and total computational time required to obtain the solutions (in the form h:mm:ss), given for each level of the overlap parameter, O. The CLUST method obtains the optimal solution for problems which have O = 1 or 2, and since this method involves taking the optimal paths from the generation process, this method is not a method per se, but more of a diagnostic tool, so we do not include the computational times required. As noted earlier, the results show that the Tabu Search method TIMP, which evaluates each customer's reward directly, according to the reward received if the customer is serviced when the service vehicle arrives at the customer's location, obtains very poor results and the direct evaluation appears to be unsuitable for problems for which waiting time needs to be included in order to obtain solutions that are of reasonable quality. Our TIMP method does periodically call on the waiting time insertion algorithm, but since the times of servicing do not consider the potential reward for the customers, the gain from the waiting time insertion is insufficient to enable effective solutions to be obtained.

The three Tabu Search methods based on fixed, artificial time windows, i.e., TS1.0, TS0.75 and TS0.5, all obtain solutions to these problems in a relatively short amount of computational time. The TS1.0 method obtains the optimal solution for each of the problem instances for which O = 1 or 2, as all customers are serviced at a time for which their maximum reward is available in these problems, and the method only considers servicing customers while their maximum reward is available. Neither TS0.75 nor TS0.5 are particularly effective for the problem classes for which O = 1 or 2, but with O = 3 the TS0.75 method is more effective than both the TS1.0 and TS0.5 methods, particularly with the larger variabilities in the maximum reward values.

The final two methods, TWAIT and TADAPT, appear to be the most effective for the problems tested, as these two methods obtain a much higher overall mean reward than the other methods. We see that the TWAIT method obtains the optimal solution for almost all of the problem instances for which O = 1, although it is slightly less effective for the problems with O = 2, particularly for the higher levels of reward variability. For these problems, since we do not include the Cross exchange routine within the TWAIT method, the selection of which of the two

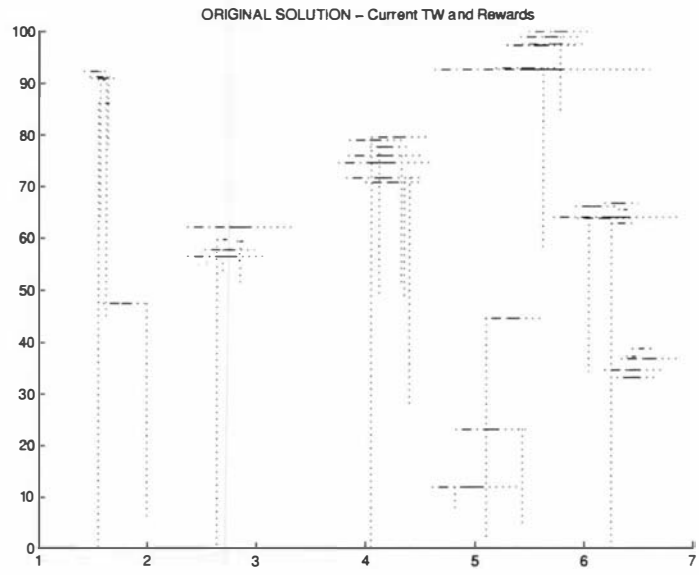

Figure 8.6: Initial Solution for TADAPT - Reward = 1855.736

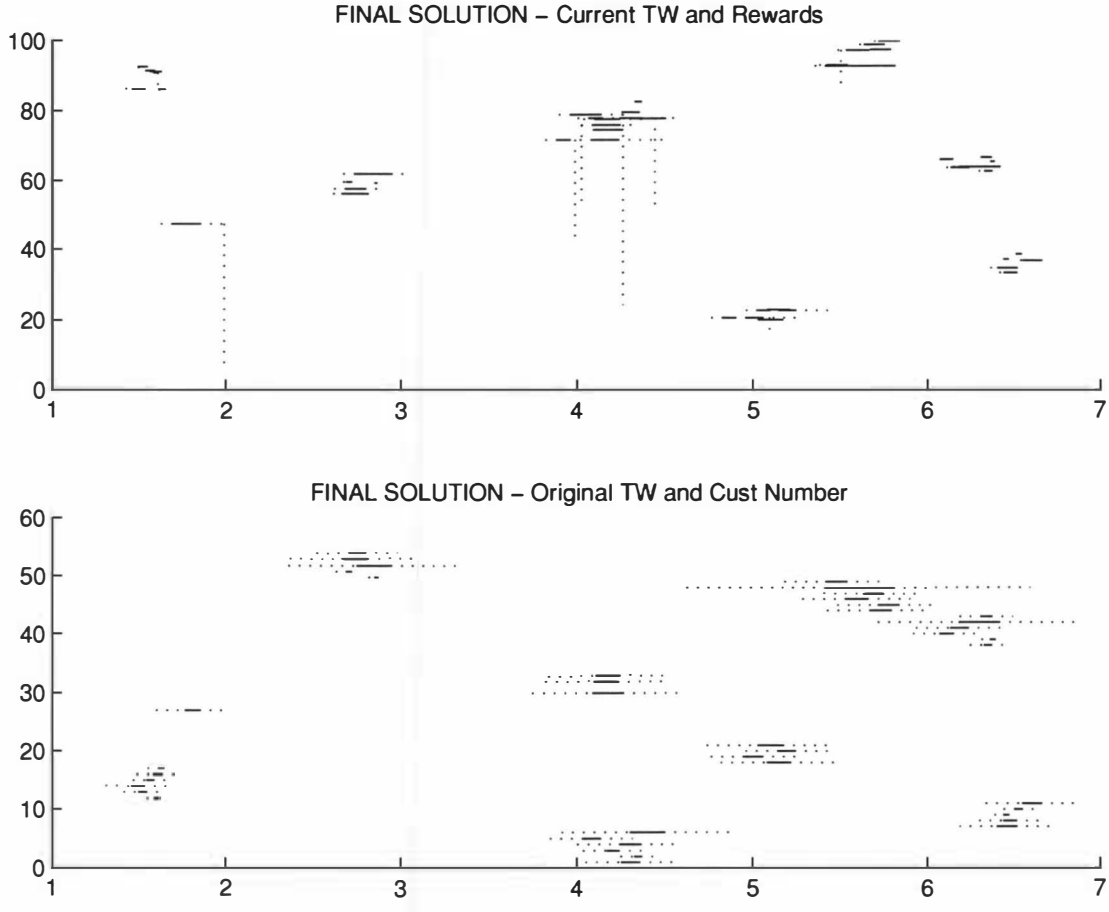

Figure 8.7: Final Solution for TADAPT - Reward = 2613.993

Classes	Construction Techniques						
(O,R)	CLUST	TIMP	TS1.0	TS0.75	TS0.5	TWAIT	TADAPT
(1,1)	600.00	495.80	600.00	580.86	554.51	600.00	598.44
(1,2)	3290.75	2839.83	3290.75	3205.58	3084.89	3290.66	3284.50
(1,3)	3322.87	2753.85	3322.87	3217.93	3071.75	3322.81	3316.27
Mean(1)	2404.54	2029.83	2404.54	2334.79	2237.05	2404.49	2399.74
Time(1)	—	0:24:57	0:01:02	0:10:47	0:13:07	0:16:57	0:27:08
(2,1)	335.20	286.89	335.20	326.32	313.29	332.95	333.13
(2,2)	1939.69	1699.30	1939.69	1892.94	1828.17	1902.57	1930.65
(2,3)	2199.15	1851.36	2199.15	2130.61	2044.19	2177.31	2184.42
Mean(2)	1491.35	1279.18	1491.35	1449.96	1395.22	1470.94	1482.73
Time(2)	—	3:12:39	1:34:20	1:29:21	1:38:46	2:24:39	4:06:48
(3,1)	292.80	323.77	326.20	342.06	341.12	353.35	349.43
(3,2)	1661.03	1848.69	1864.76	1945.62	1925.84	1972.31	1985.81
(3,3)	1861.44	1961.81	2039.38	2091.95	2059.84	2153.04	2140.69
Mean(3)	1271.76	1378.09	1410.11	1459.88	1442.27	1492.90	1491.97
Time(3)	—	3:52:00	2:17:00	2:39:04	2:49:26	5:58:34	5:00:21
Total	1722.55	1562.37	1768.67	1748.21	1691.51	1789.45	1791.48

Table 8.32: Results from Different Tabu Search Implementations for Clustered MCPTDR Instances

available clusters to service in the current route becomes an important decision, as there is little chance to be able to interchange the clusters within the search phase. The decision of which cluster to include is pretty arbitrary, because we only consider one customer at a time in the insertion phase, and the choice of poor clusters to service will likely lead to poor results being obtained. For the problem classes for which O = 3, which are the most interesting of these problems, the TWAIT method obtains the highest overall mean reward, although the computational time required is much higher for these problems. For the problem classes with O = 1, the TWAIT method obtains an optimal solution in which the reward obtained is equal to the maximum available reward, so the search process is terminated quickly for these problems. With O = 2, the size of the neighbourhood available to the Move routine is very small, so this method again obtains solutions quickly, and since global improvements are so difficult to obtain, this method operates reasonably quickly. With O = 3, there are larger neighbourhoods available, as the clusters are able to overlap a lot more, and improvements to the solution are more easily found, so the search process takes much longer for these problems. The solution quality appears to be highest for this method although the large increase in the computational time as the neighbourhood size increases, is a concern for its general applicability.

The TADAPT method seems to improve over the other Tabu Search methods for which the customer rewards are evaluated for the earliest time at which the customer may be serviced, although it can not obtain as effective solutions as the TS1.0 method for the classes for which O = 1 or 2. In these problems, the time of servicing within the artificial time windows is directly linked to the optimal time for servicing the customers, so the TS1.0 method has an unfair advantage for these problems. We see that the computational time required for TADAPT is the highest of the methods considered with O = 1 or 2. The additional time required to update the current routes upon the alteration of the time windows, particularly when the alteration leads to the route becoming infeasible, is the major contributor to the additional computational time. This additional time is necessary as it enables the search to go into other regions of the solution space, that may have been deemed to be unattractive when evaluation only considered the original time windows. For the problem classes for which O = 3, the overall mean for this method is similar to that obtained by the TWAIT method, and much higher than for the other

methods, so the adaptation of the time windows appears to enable better solutions to be obtained.

Since we are mostly interested in the effectiveness of our TADAPT method, our testing involves finding the significance of differences between the results obtained by this method and each of the other Tabu Search methods (a total of five tests). Our testing again involves the application of the Wilcoxon Signed-Rank Test to these pairs of methods.

When we test the TS1.0 method against TADAPT, we find that TS1.0 is significantly more effective for all six problem classes for which O = 1 and 2, while TADAPT is significantly more effective for the three problem classes with O = 3. TADAPT is significantly more effective than TS0.75, TS0.5 and TIMP for all nine problem classes, which demonstrates the improvements that may be made by allowing the time windows to be adapted. When testing TADAPT against TWAIT we find that TWAIT is significantly more effective for the three problem classes for which O = 1, as well as two problem classes which have O = 3, whereas TADAPT is not significantly more effective for any of the problem classes. The TWAIT method appears to be better suited to the problem classes for which O = 1, as the neighbourhood size is small and the effective evaluation of a proposed move is important in determining the effectiveness of the move. For the problem classes for which O = 3, the TWAIT method was significantly more effective than TADAPT for two of the problem classes, and for the third problem class, the median of the differences between the solutions (TWAIT - TADAPT) was also positive. The difference, though, was the TADAPT method obtained much larger rewards for certain of the problem instances, and the ability to use the other improvement routines, i.e., Generalized Exchange and Cross, in certain situations, enabled more effective solutions to be obtained. The much larger neighbourhood sizes for these routines prevented their inclusion within the TWAIT routine due to the computational times becoming excessive.

We conclude that the TADAPT method of adapting the time windows throughout the course of the Tabu Search, is a promising method for obtaining effective solutions to various versions of the MCPTDR. The advantage of this method is that it is able to search much greater regions of the solution space, and the generality of the method should enable it to be used for wide-ranging types of problems. We tested the method on a set of problems for which waiting time is required in order

to be able to obtain reasonable solutions, and the method appeared to work reasonably effectively on these problems, and was able to outperform some other Tabu Search implementations for many of the problem classes considered. We found previously that setting the artificial time windows at an appropriate level may lead to improved results being obtained, although much needs to be known about the problem in order to be able to set these time windows. TADAPT may be used instead, to allow the time windows to set themselves, particularly if we implement this method so that the time windows enable intensification and diversification of the search, with smaller alterations made to the time windows if the solution is in a promising region, and larger changes made at other times. Many other adaptations to the TADAPT method may be explored in order to improve upon its effectiveness, with alterations such as different initial time window widths and different rates of change for different customers, so that certain customers may have their time window fixed at times that have been found to be effective throughout the search, appearing to be quite appropriate. One of the disadvantages of this approach is the additional computational time required, as the update of the solution, in the presence of the adapted time windows, may require a lot of additional effort.

For the test problems considered, the most effective of the methods appeared to be TWAIT, and this method involves a more restricted neighbourhood definition being combined with a more computationally expensive evaluative tool. This approach is similar to that used by Verweij and Aardal [183] within their Tabu Search routine for a pick-up and delivery problem, as they applied a simple node move to the current solution in order to create the new solution, with the effect of the proposed change being determined by a routine that found the optimal size of loads to be carried for the proposed solution. The TWAIT method requires the optimal time to service a set of customers to be able to be obtained, in order for the optimal waiting time method to be applied. This drawback, of requiring the reward functions to be tractable, may be partially overcome by the use of effective heuristic methods for estimating the best reward that may be obtained for a proposed solution route, with the effectiveness of the TWAIT method being directly linked to the effectiveness of this estimation method.

8.2.5 Testing Tabu Search Methods for the TRCP

We now test two different implementations of Tabu Search on some new problem instances for the TRCP, for which we are interested in the computational time and relative effectiveness of the different versions of the methods. We study the effect of the problem size on the computational time, and create problem instances for which there are 40, 60, 80 and 100 points, from which the tasks are comprised. We generate five layouts containing the required number of points, by creating Euclidean, "rattled grids". For a rattled grid, if there are n points to be placed, we create a k by k grid (where $k = \lceil \sqrt{n} \rceil$) over the available problem area (in our case, $[0,1]^2$). For each point, we randomly select an empty cell of the grid to assign the point's location and randomly generate the exact location for the point, within the cell. This generation method ensures the customers are evenly spread throughout the problem area, and prevents clusters of any significant size from being formed. The travel time between each pair of points was assumed to be equal to the Euclidean distance.

For each layout we generate 16 test problems by creating two levels of the parameters for four features of the problem: the load size, number of customers served, distribution of the rewards and method for generating the tasks. Each task consists of either two or three points, which must be visited in a given order and there is a load of a given size which must be transported between successive points within the task. For the parameter for the load sizes (L), we create the tasks so that either only one load at a time may be included in the service vehicle, for L = 1, or a few loads may be in the vehicle at at time, for L = 2. The capacity of the vehicle is set to be equal to 6 units, so if L = 1, each task consists of a pick-up of 6 units followed by either one delivery of the 6 units, or one delivery of 1 unit followed by a second delivery of 5 units. For L = 2, each task consists of a pick-up of 2 units, followed by either a delivery of the 2 units or two deliveries each of 1 unit.

We again create appropriately sized sections of route for which to assign the precedence constraints. For low numbers of customers serviced, (N = 1), we create sections of route containing 10 points, and set the time limit so that each of the sets of customers may be serviced completely, i.e., each of the sections is a feasible candidate solution. The method for generating the tasks, G, determines the method that is used to generate the sections of route, where for G = 1, a matching is

heuristically obtained, while for G = 2, a randomized nearest neighbour method is used to create a route through the customers, and this route is partitioned into the required sections. The tasks are obtained by taking consecutive pieces of the sections, containing the number of points that comprise the task. This approach enables greater control over the form of the task created, rather than using the recursive approach we previously considered, as the tasks are determined directly from the routes, rather than having the interaction between the allowed load change and location affecting the task allocation.

For high numbers of customer serviced, N = 2, we create a tour through all of the points, using a cheapest insertion approach (for G = 1) and randomized nearest neighbour (for G = 2), with the tasks allocated to consecutive sections of the tour. The route obtained was improved by applying Or-opt and 2-opt, and the time limit was set to be equal to 80% of the tour length, so as to ensure that all customers were unable to be serviced within the solution route (thereby preventing the case in which our Tabu Search methods stop prematurely, after finding a solution in which all customers are serviced). For the reward parameter, R, the rewards are randomly generated in the range $[10, 100]$ when R = 1, and the reward is a constant multiple of the total time required to service the task.

We generate a total of 320 test problems, by generating five layouts for each of the four numbers of points, and generating 16 problem classes for each layout. In order to compare the effect of the different values of G, we give a summary of the lengths of the tasks generated over the 320 problem instances, in Table 8.33. In the Table, Sect1 and Sect2 are the lengths of the sections within the tasks, for G = 1 and G = 2, respectively and Task1 and Task2 similarly define the overall length of the tasks created. The summary statistics given are the mean, standard deviation, minimum, lower quartile, upper quartile and maximum.

Table 8.33 shows that there is a large difference between the lengths of the tasks generated by the two methods, with far longer tasks and sections generated when G = 2. Both methods include the same shortest section, which has a length of 0.0070, but the other statistics are much higher for G = 2. We will subsequently examine the effect that the different task generation methods has on the solutions produced and on the effectiveness of the solution methods employed.

The methods we implement on the test problems are TABUMOVE, TABUIMP and BESTH. The TABUMOVE method is our implementation of Tabu Search which

Method	Mean	StDev	Min	Q1	Q3	Max
Sect1	0.1110	0.0476	0.0070	0.0799	0.1355	0.4551
Sect2	0.2768	0.2224	0.0070	0.1138	0.3758	1.2971
Task1	0.1630	0.0817	0.0070	0.0979	0.2166	0.5631
Task2	0.4028	0.3065	0.0070	0.1636	0.5684	1.8891

Table 8.33: Summary of Section and Task Lengths Generated

only considers the inter-route Move routine, whereas TABUIMP is our full implementation of Tabu Search, which allows inter-route improvements to be made by using the Move, Cross and Generalized Exchange routines. The other method, BESTH, involves taking the best of the solutions obtained by the five most effective heuristics of Table 8.4, i.e., B4BTASK, AGGAPP, CLUSDIST, CLUSREW and CHEAP, with each solution being improved by the application of the STEEPEST ASCENT routine.

We illustrate the solutions in the two test problems shown in Figures 8.8 and 8.9. Figure 8.8 shows the tasks that are included in test problem 34; this test problem contains 40 points, L = 1, N = 1, R = 2 and G = 1. Figure 8.9 shows the tasks that are included in test problem 36, and this problem has the same parameters as problem 34, except G = 2. In each Figure, the label associated with each location is the order in which the locations within the task must be serviced. We see from these Figures that there is a large difference in the forms of the tasks that are created by the two methods, as with G = 1 there is no overlap between the tasks, while with G = 2 there are a large number of pairs of tasks which cross-over each other.

The solutions obtained by applying TABUIMP and TABUMOVE to problem 34 are shown in Figures 8.10 and 8.11, respectively, with the corresponding solutions for problem 36 shown in Figures 8.12 and 8.13. In each Figure the label given next to each location is the load that is in the service vehicle after completing the service at the location. The capacity of the vehicle is equal to 6 units, so only one customer may be in the vehicle at any one time. We see that for the test problems, the solution routes obtained by the different methods are very different, although the total reward obtained is quite similar. The TABUIMP method is slightly more effective for each instance, as might be expected, given the more advanced search routines considered. The computational time required by the TABUMOVE method

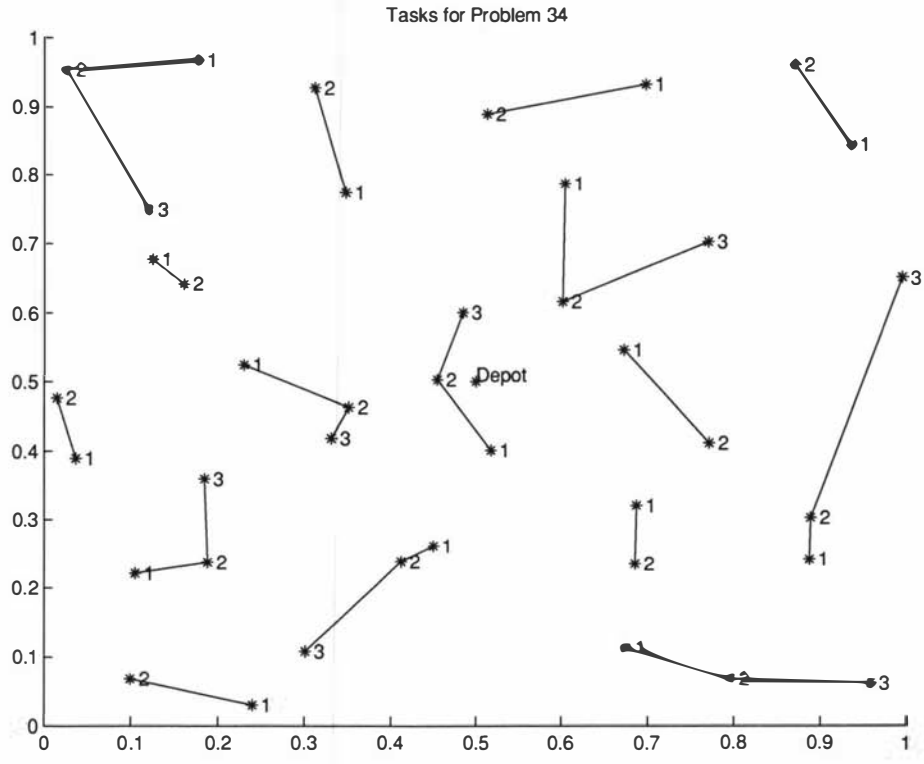

Figure 8.8: Test Problem 34 for the TRCP

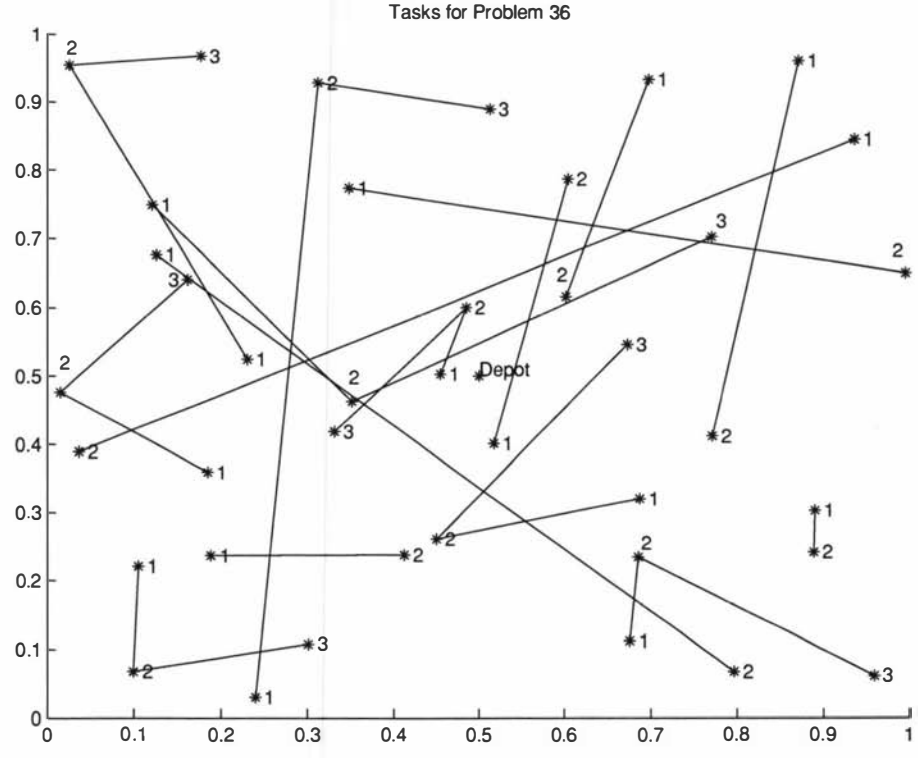

Figure 8.9: Test Problem 36 for the TRCP

in order to obtain each of the solutions is approximately 1 second, whereas the TABUIMP method takes approximately 5 and 7 seconds, respectively, to obtain the given solutions.

The results from the testing on the 320 problem instances containing 40 points are shown in Table 8.34. For the computational times, the different runs of the same method were run successively, and the elapsed time (in seconds) to obtain each solution was recorded. These times are not completely accurate, especially when the computational times are short, but since we are more interested in the times for their comparative values, rather than for their actual values, we don't consider this to be a significant problem. Also of note is that the methods have been implemented in a much more efficient manner than was used in Section 8.1.2. We identified the provisional insertion of the tasks into the solution routes, as is used within the Move and Generalized Exchange routines, as a large contributor to the overall computational time required, so we attempted to carry out this step as few times as possible. Thus we assessed the possible effect of each proposed move, and only checked the feasibility of the move if improvements were possible, and we tested the feasibility of the time and capacity constraints at the earliest possible phase, in order to prevent unnecessary searching within unviable regions.

From Table 8.34 we see that the most effective of the methods appears to be TABUIMP, as it obtains the highest mean reward for 12 of the 16 problem classes and it is tied for the highest mean reward for the remaining four classes. The BESTH method appears to be more effective than the TABUMOVE method, as it obtains a higher mean reward for 13 of the problem classes.

When we consider the computational times, we find that the times required for BESTH and TABUMOVE appear to be quite similar for the problems for which $L = 1$, but the BESTH method appears to obtain solutions in slightly less time when $L = 2$. The price to be paid for the additional solution quality obtained by the TABUIMP method, appears to be the quite significant increase in the computational time, compared with the other two methods. There appears to be a greater computational time requirement for the problem instances for which $L = 1$ (where all selected tasks must be serviced consecutively) than for $L = 2$ (where the tasks may be interspersed within the route). When $L = 1$, there is little flexibility allowed in the routing of customers, as, when inserting a customer into a current route, the subtasks of each customer may only be placed directly after the

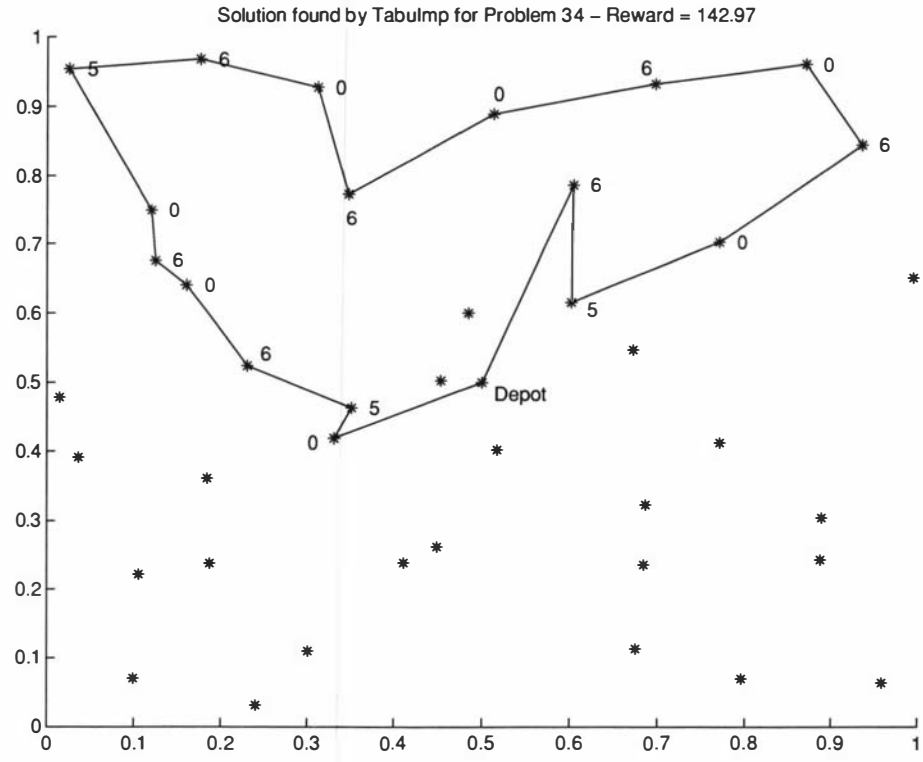

Figure 8.10: Solution found by TABUIMP for Test Problem 34

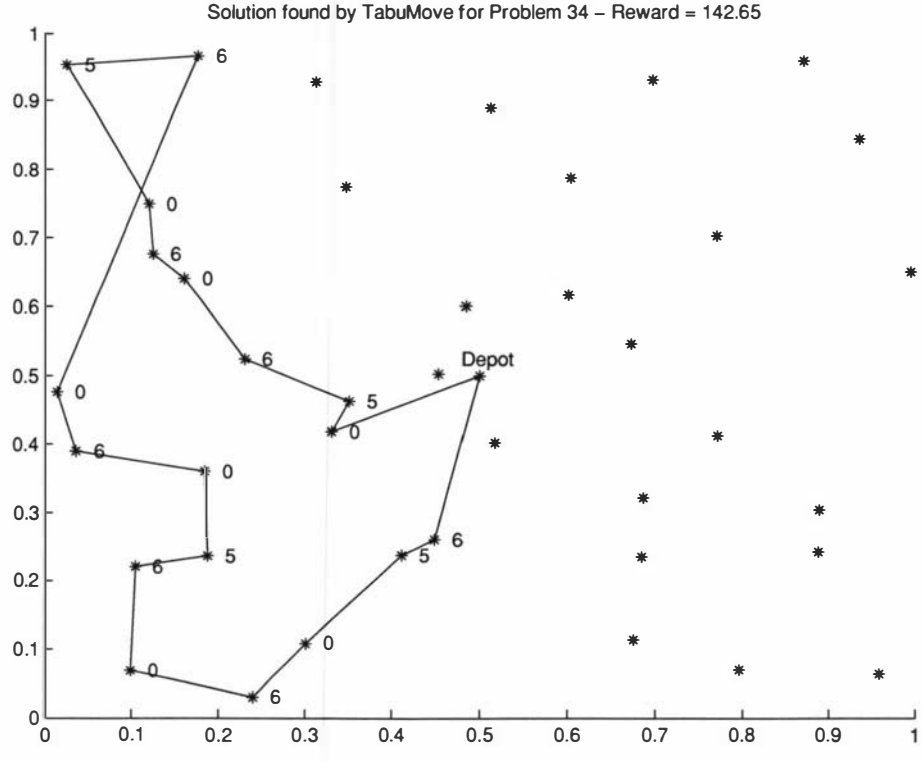

Figure 8.11: Solution found by the TABUMOVE for Test Problem 34

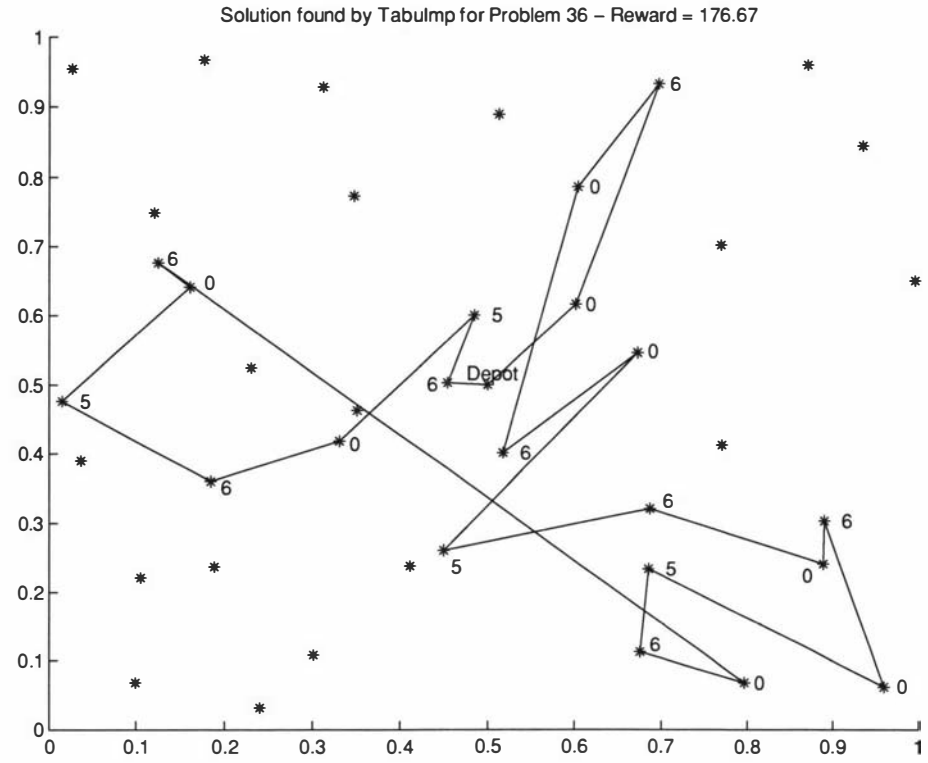

Figure 8.12: Solution found by the TABUIMP for Test Problem 36

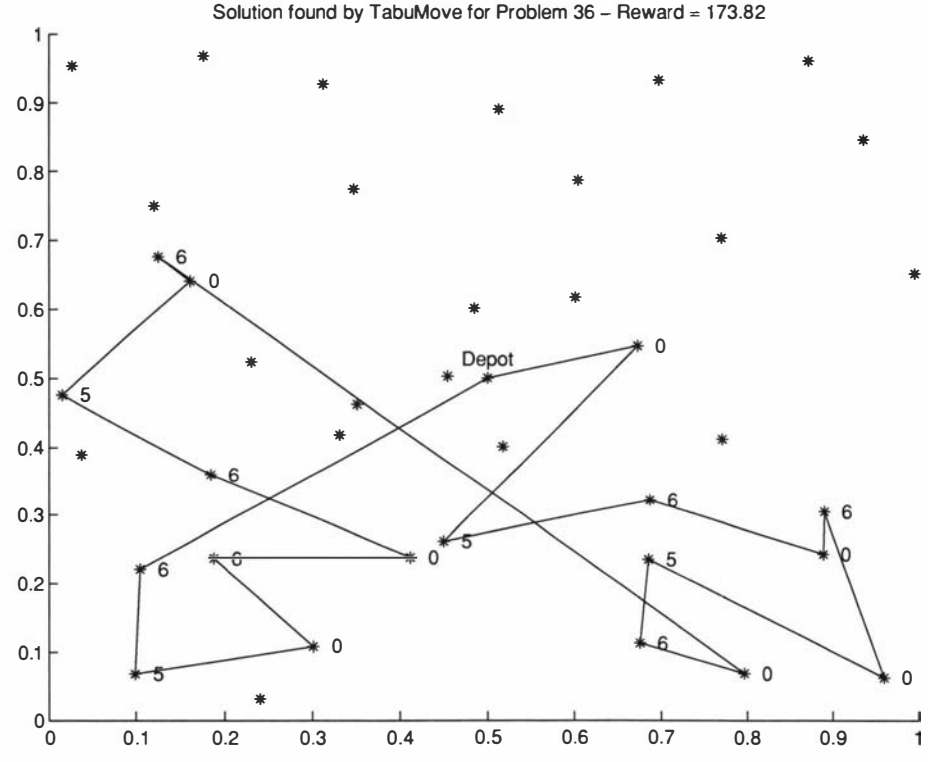

Figure 8.13: Solution found by the TABUMOVE for Test Problem 36

Class	TABUMOVE		TABUIMP		BESTH	
(L,N,R,G)	Reward	Time	Reward	Time	Reward	Time
(1,1,1,1)	504.012	0.6	509.691	4.0	509.691	0.4
(1,1,1,2)	624.371	0.6	634.524	4.2	627.735	0.0
(1,1,2,1)	142.887	0.4	144.071	4.4	144.071	0.4
(1,1,2,2)	193.724	0.2	196.612	5.6	193.329	0.8
(1,2,1,1)	822.729	0.2	840.584	3.0	840.584	0.4
(1,2,1,2)	823.481	0.6	831.852	3.4	831.852	0.4
(1,2,2,1)	281.117	0.2	288.299	4.6	286.862	0.0
(1,2,2,2)	333.113	0.4	344.514	4.2	340.001	0.4
(2,1,1,1)	487.100	0.2	491.453	6.2	481.265	0.2
(2,1,1,2)	643.205	1.0	659.825	5.4	655.254	0.2
(2,1,2,1)	151.512	1.0	154.000	7.8	152.944	1.0
(2,1,2,2)	248.042	1.2	268.127	7.4	259.969	1.0
(2,2,1,1)	792.266	2.6	797.345	7.0	791.613	0.8
(2,2,1,2)	860.275	1.6	871.915	5.4	866.000	0.8
(2,2,2,1)	276.032	3.2	282.429	9.6	278.817	1.4
(2,2,2,2)	349.918	2.0	366.070	4.8	355.878	0.2

Table 8.34: Mean Reward and Computational Time for TRCP with 40 points

last point of one of the customers in the route. This leads to there being smaller neighbourhoods, compared to instances for which there is allowed to be significant separation between the positions of successive sub-tasks of a customer, and the resulting lower time required to evaluate the neighbourhood, leads to great reductions in the computational time required for the methods.

The results from testing problem instances containing, respectively, 60, 80 and 100 points are given in Tables 8.35, 8.36 and 8.37. The TABUIMP method seems to be the most effective of the methods, and it obtained the highest mean reward for 40 of the 48 problem classes of Tables 8.35, 8.36 and 8.37. The method BESTH obtains the highest mean reward for the eight remaining problem classes, while the TABUMOVE method obtains the lowest mean reward for all but four of the problem classes.

The computational times in Tables 8.35, 8.36 and 8.37 show that the time required for the TABUMOVE routine increases more slowly as the problem size increases, compared with the other methods, which results in this method taking less time than the BESTH method for larger problems. We believe that this is due to the improvement routines that are used, as the Generalized Exchange and Cross

Class	TABUMOVE		TABUIMP		BESTH	
(L,N,R,G)	Reward	Time	Reward	Time	Reward	Time
(1,1,1,1)	634.477	1.0	640.595	12.6	636.102	1.0
(1,1,1,2)	810.717	0.4	836.408	13.0	826.999	1.4
(1,1,2,1)	166.822	1.0	168.079	13.4	166.733	0.6
(1,1,2,2)	225.397	1.2	241.375	16.8	240.123	1.0
(1,2,1,1)	1194.437	0.8	1199.449	11.2	1196.789	1.0
(1,2,1,2)	1223.704	0.4	1239.261	11.2	1232.788	1.0
(1,2,2,1)	391.141	0.8	397.833	14.0	397.739	1.2
(1,2,2,2)	471.997	0.8	477.502	13.4	476.323	1.0
(2,1,1,1)	593.320	1.0	593.304	19.4	595.753	2.0
(2,1,1,2)	866.705	1.4	902.395	19.6	906.302	2.4
(2,1,2,1)	167.982	1.4	169.189	25.2	165.634	2.0
(2,1,2,2)	303.180	1.8	307.770	26.4	304.270	2.6
(2,2,1,1)	1166.676	6.4	1181.563	18.8	1180.876	5.4
(2,2,1,2)	1193.142	3.4	1197.910	10.6	1199.700	2.0
(2,2,2,1)	412.700	4.2	414.618	23.4	415.029	4.6
(2,2,2,2)	508.673	2.2	536.475	11.6	528.100	1.6

Table 8.35: Mean Reward and Computational Time for TRCP with 60 points

Class	TABUMOVE		TABUIMP		BESTH	
(L,N,R,G)	Reward	Time	Reward	Time	Reward	Time
(1,1,1,1)	653.910	1.8	669.888	25.0	659.098	2.0
(1,1,1,2)	852.221	1.4	883.764	27.8	877.344	2.4
(1,1,2,1)	180.168	1.6	179.839	26.8	180.267	2.0
(1,1,2,2)	226.752	1.4	240.257	35.6	238.249	2.4
(1,2,1,1)	1551.065	1.2	1582.506	20.2	1578.679	2.6
(1,2,1,2)	1682.655	1.4	1717.566	25.8	1709.328	2.6
(1,2,2,1)	510.160	1.0	529.692	29.6	530.037	3.6
(1,2,2,2)	658.683	1.0	672.316	37.4	668.614	3.2
(2,1,1,1)	709.975	1.6	725.704	30.8	724.478	4.0
(2,1,1,2)	1019.571	2.0	1093.460	41.8	1070.948	5.4
(2,1,2,1)	177.453	2.6	178.965	44.6	176.503	3.6
(2,1,2,2)	344.210	2.6	371.646	70.6	345.831	6.0
(2,2,1,1)	1512.656	16.8	1550.781	38.2	1539.865	15.2
(2,2,1,2)	1705.914	4.2	1747.155	15.4	1727.301	4.4
(2,2,2,1)	526.421	11.4	542.509	48.8	540.504	9.4
(2,2,2,2)	643.605	3.8	665.245	23.6	659.695	3.8

Table 8.36: Mean Reward and Computational Time for TRCP with 80 points

Class	TABUMOVE		TABUIMP		BESTH	
(L,N,R,G)	Reward	Time	Reward	Time	Reward	Time
(1,1,1,1)	698.162	2.4	725.384	45.4	725.486	3.4
(1,1,1,2)	1012.195	2.0	1046.188	55.0	1047.567	5.6
(1,1,2,1)	177.038	2.8	183.024	52.8	182.718	3.0
(1,1,2,2)	303.167	2.2	325.200	83.8	317.923	5.2
(1,2,1,1)	1927.033	2.0	1957.082	64.8	1944.567	6.8
(1,2,1,2)	2070.231	2.2	2091.677	50.0	2085.062	6.4
(1,2,2,1)	633.157	1.6	650.297	63.4	646.882	6.4
(1,2,2,2)	773.977	2.0	791.046	69.6	791.232	6.2
(2,1,1,1)	717.676	2.6	730.141	57.8	712.355	6.4
(2,1,1,2)	1081.085	2.8	1154.733	80.8	1149.985	13.4
(2,1,2,1)	178.114	3.6	186.215	63.4	186.665	7.0
(2,1,2,2)	348.131	3.2	371.491	122.8	355.671	14.4
(2,2,1,1)	2059.009	21.6	2096.070	60.6	2088.572	54.8
(2,2,1,2)	2188.577	2.8	2232.462	18.8	2216.944	5.6
(2,2,2,1)	668.284	17.2	681.972	98.2	677.052	30.4
(2,2,2,2)	883.351	4.8	900.833	27.0	895.975	5.2

Table 8.37: Mean Reward and Computational Time for TRCP with 100 points

routines, which are employed by both the TABUIMP and BESTH methods, take much longer to implement as the problem size increases. One noticeable feature, that is exaggerated as the problem size increases, is, in the problems for which L = 2 and N = 2, there is a large difference between the computational time required for the problem classes for which G = 1 and those for which G = 2. In each of these problem classes, for each of the methods, the generation type G = 1, i.e., having all tasks generated from sub-paths that are created according to some distance minimization criterion, leads to much greater computational times than the generation method G = 2, in which tasks are generated from distributions that follow less geometric properties. The TABUMOVE and BESTH routines appear to be very affected by this discrepancy in the times, as, particularly for the problems containing 80 and 100 points, the computational times required for the problems for which N = 2, L = 2 and G = 1, are much higher than for any of the other problem classes. What is also interesting is the fact that for the problem classes for which N = 1 and L = 2, the computational times required by the TABUIMP and BESTH methods are much higher when G = 2 than for G = 1. As we showed in Table 8.33, the tasks generated by method G = 2, are much longer than those

generated by method G = 1, so we now seek to find out why the longer tasks result in shorter computational times with high numbers served and small loads carried, and why this behaviour is somewhat different with fewer customers serviced.

We examine the solution obtained by the TABUIMP method on test problem 174, the tasks of which are shown in Figure 8.14, where the label for each location shows the position that this location is serviced within the task. This problem instance contains 80 points, and the parameters used in problem generation are: N = 2, L = 2, R = 2 and G = 1. The solution obtained by the TABUIMP method for this problem instance is shown in Figure 8.15, with the label given to each location indicating the load in the service vehicle after completing the desired service at the location. The total computational time required by the TABUIMP method for this problem was approximately 31 seconds.

We compare the test problem with problem instance 176, which is generated in the same manner as problem 174, except, G, the task generation method is equal to 2. The tasks for this problem instance are shown in Figure 8.16 and the solution obtained by the TABUIMP method is shown in Figure 8.17. The total time required to obtain this solution was approximately 15 seconds, so the computational time is about half of that which was required for test problem 174. The main reason for the difference in the computational times is the form of the solution obtained. We see that the solution for problem 174 many of the tasks are serviced consecutively, in which case the load in the vehicle does not exceed 2 units. We find that the average load in the vehicle upon the completion of service at each of the locations in the route, is 1.9859. The solution for problem 176 consists of much greater intertwining of customers, and, in fact, the load in the vehicle is never equal to zero from the time the first pick-up is made until the last delivery is completed. This would prevent the Cross exchange routine from being able to be used, as no stand-alone paths will exist (the load in the vehicle being equal to zero is a necessary requirement of the end-point of a stand-alone path). In this case the average load in the vehicle after servicing each location is 4.16667.

The implications of the form of the solution are felt when attempting to insert customers into the current solution. When there are low loads in the vehicle, there is little restriction on the position to place the initial subtask of the customer and there are many feasible positions to place the remaining subtasks, particularly involving large separation between the consecutive subtasks. We demonstrate the

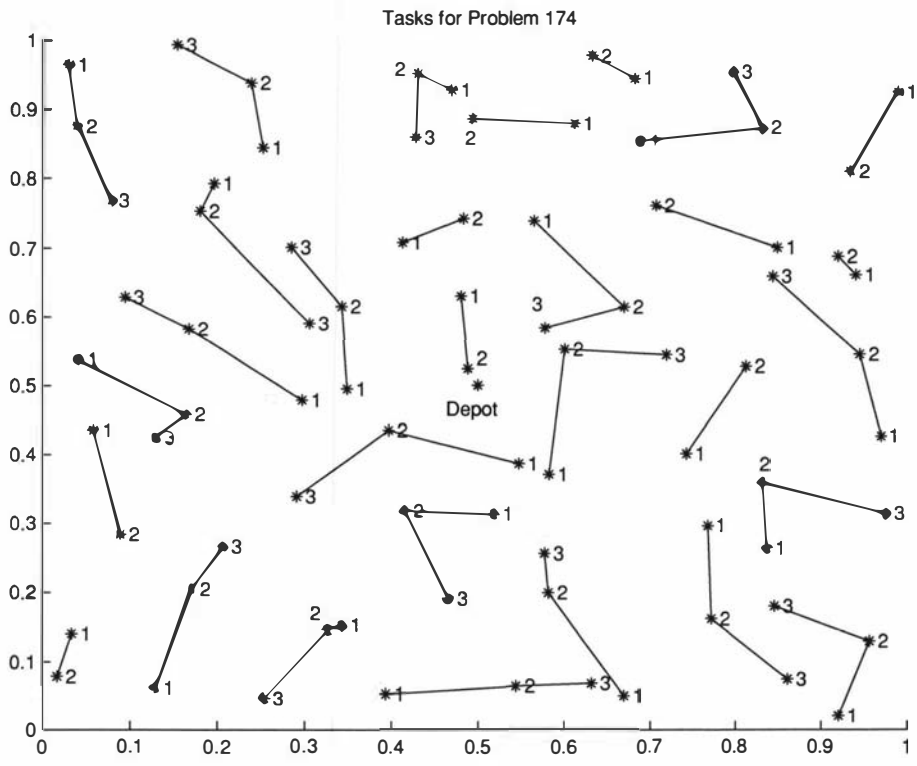

Figure 8.14: Test Problem 174

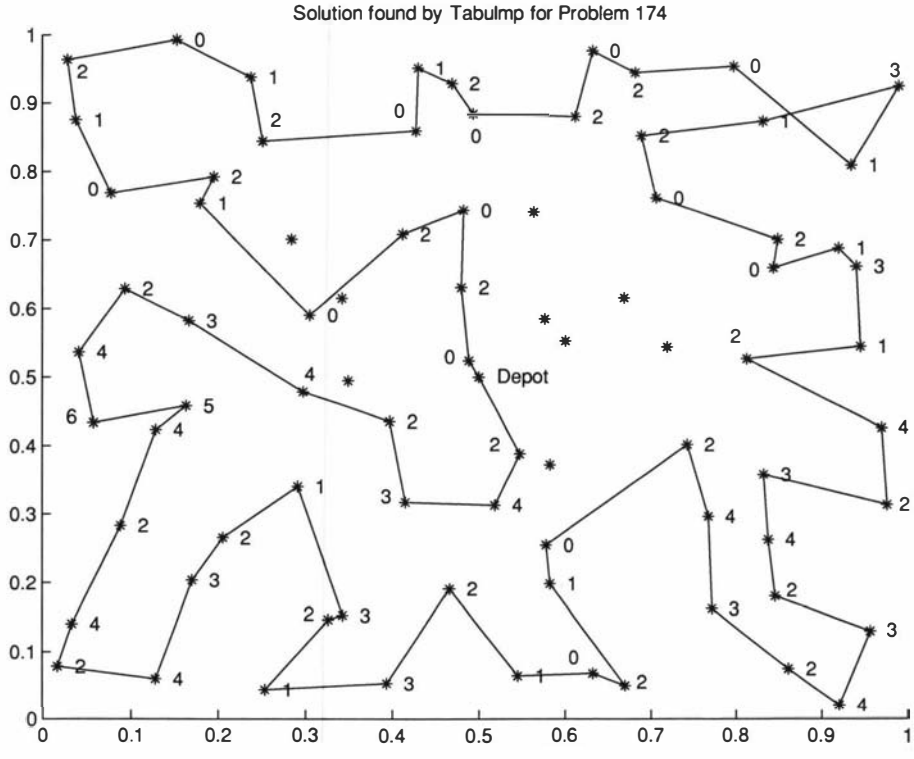

Figure 8.15: Solution Obtained by TABUIMP for Test Problem 174

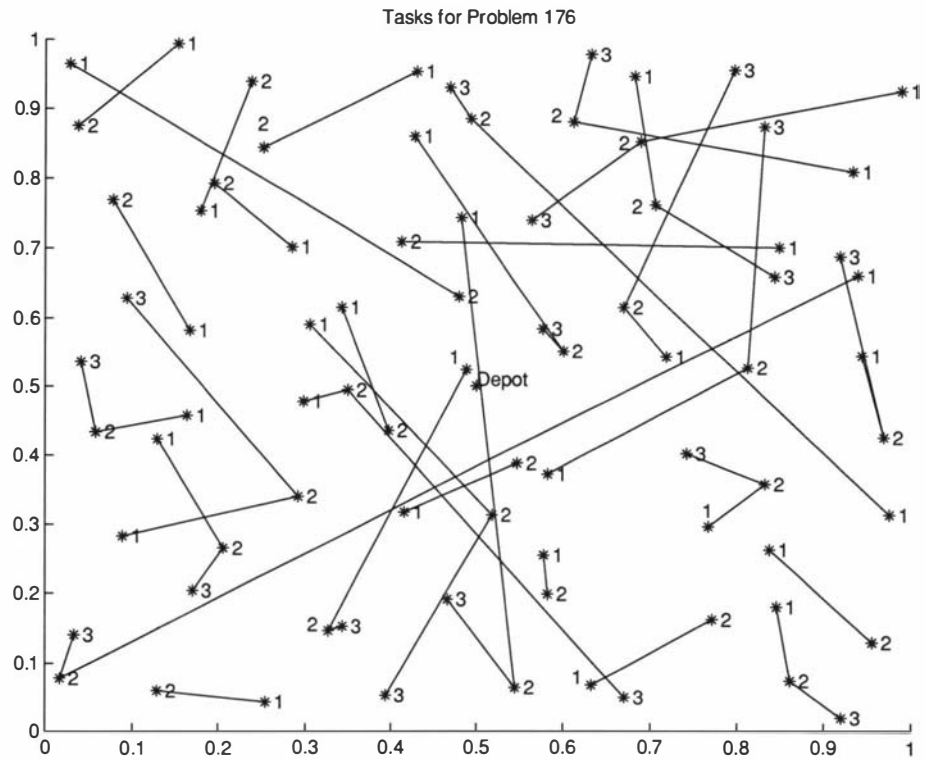

Figure 8.16: Test Problem 176

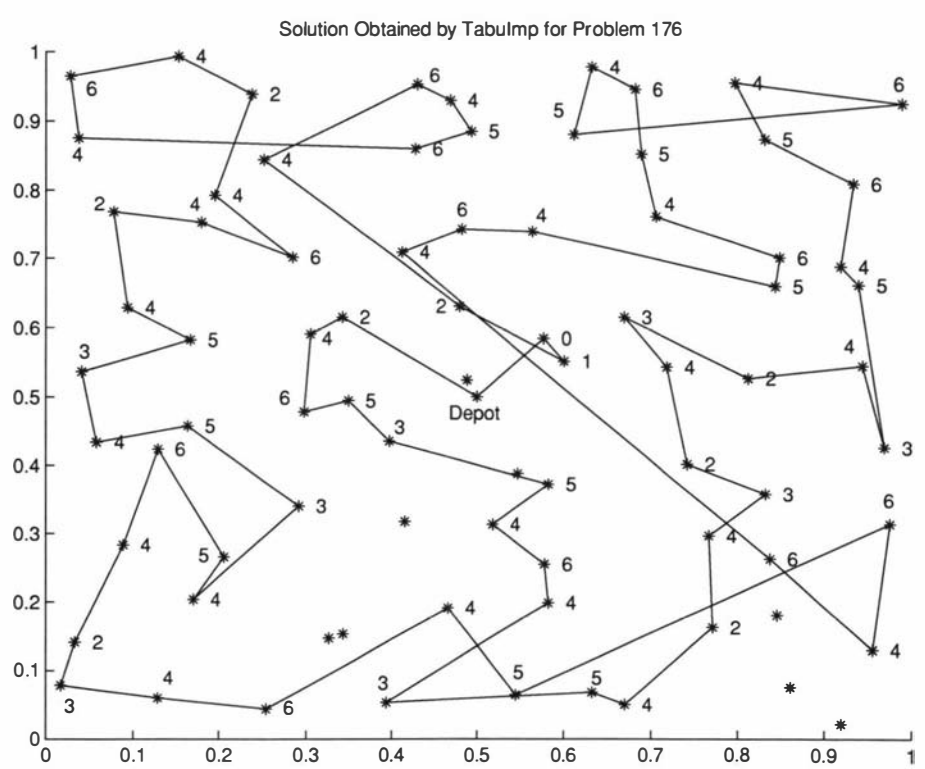

Figure 8.17: Solution Obtained by TABUIMP for Test Problem 176

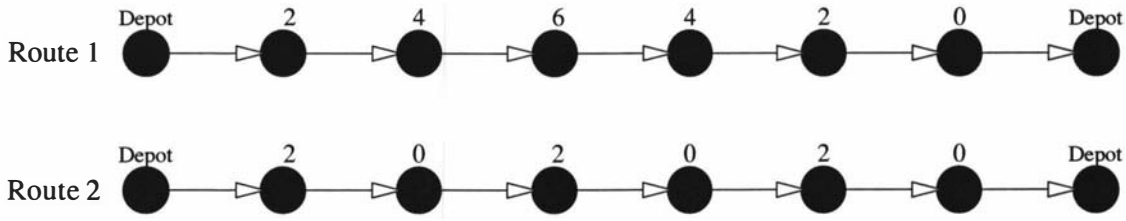

Figure 8.18: Possible Solution Routes

insertion process on the solution routes shown in Figure 8.18, in which the labels for each node indicate that this node corresponds to the depot or gives the load that is in the vehicle upon the completion of the service at this node. The two solutions each consist of three tasks, with each task consisting of a pick-up of 2 units and a delivery of 2 units, and we refer to the depot occupying positions 0 and 7 of the routes, with the serviced customers occupying positions 1 to 6. Route 1 corresponds to an extreme case of the type of solutions found for problem instances for which G = 1, while route 2 is more akin to the solutions found for G = 2.

We consider the insertion of a customer, again consisting of a pick-up of 2 units and a delivery of 2 units, into each of the routes. When inserting into route 1, the vehicle capacity being full at position 3 dramatically reduces the number of feasible insertion positions that are available. Therefore, when inserting the pick-up of the customer in position 1, the delivery can be placed only in positions 1, 2 and 3. The feasible insertion positions available, given as the pair (*pick-up location, delivery location*), are: (1,1), (1,2), (1,3), (2,2), (2,3), (3,3), (5,5), (5,6), (5,7), (6,6), (6,7) and (7,7), so there are a total of 12 positions in which it is feasible to insert the customer without violating the capacity constraint. In route 2, the capacity constraint does not restrict the insertion positions at all, so when inserting the pick-up in position i, there are $8 - i$ positions available for inserting the delivery. Therefore there are 28 positions in which it is feasible to insert the customer without violating the capacity constraint, so there is a much larger neighbourhood associated with inserting the customer into route 2 than there is for route 1. Since the insertion neighbourhood is used within the Move routine, and is used twice within the Exchange and Generalized Exchange routines, the larger neighbourhood size with low loads results in longer computational times as greater numbers of potentially feasible solutions need to be evaluated.

The main increase in computational time for task problems relates to the insertion of a number of points for each task, with the difference between the position of

the successive points of a task contributing significantly to the time required. For example, if there are n positions in a tour in which it is possible to insert subtasks, there is no binding capacity constraint and the task to be inserted consists of two subtasks, then, if we require the inserted task to be serviced consecutively, there is a total of n possible insertion positions, while if we allow a separation of at most one node between the subtasks of the inserted task, the total number of insertion positions is equal to $2n-1$. If we allow any degree of separation between the points, then the total number of insertion positions is equal to $\sum_{i=1}^{n} i$, which is equal to $n(n+1)/2$. If we consider the insertion of three subtasks per task, the number of feasible positions is much higher again, as there are a total of $\sum_{i=1}^{n} \frac{i(i+1)}{n}$ positions to insert the task.

The above calculations show that, with no constraints on the loads carried along the route, the number of feasible insertion positions increases dramatically as the separation between consecutive subtasks is allowed to increase and as the size of the tasks also increase. The introduction of capacity constraints may decrease the number of insertion positions, with the effect of these constraints being influenced by how binding they are in the current vehicle route. Loads that are high, i.e., near the capacity of the service vehicle, will restrict the routing options significantly, whereas low loads will not have a significant influence on the routing.

For the solution routes obtained within Tables 8.34, 8.35, 8.36 and 8.37 we summarize the loads carried in the solution routes obtained by the TABUIMP routine on the problem classes for which N = 2 and L = 2. These results are shown in Table 8.38, and are to be used to compare the loads carried between the classes for which G = 1 and G = 2. The column 'Mean Load' gives the average load that is in the service vehicle upon the completion of the service of each subtask and the column 'Num High' gives the average number of subtasks for which the load in the vehicle is greater than or equal to 5 units upon the completion of the subtask's service. Since each load consists of a pick-up of 2 units and the service vehicle has a capacity of 6 units, the figures given in the column Num High refer to the number of positions in the solution route where it is not possible to insert the first point of a new customer.

From the Table we see that there is a huge difference between the loads in the solutions obtained for the problem classes for which G = 1 and G = 2, with there being much higher average loads and much more frequent situations in which

Num	Mean Load		Num High	
	G = 1	G = 2	G = 1	G = 2
40	2.69	3.28	2.2	7.8
60	2.43	3.65	5.3	15.5
80	2.47	3.63	6.3	20.4
100	2.55	3.86	9.5	31.7
Total	2.53	3.60	5.8	18.8

Table 8.38: Load Data Obtained from Classes with N = 2 and L = 2

the load in the vehicle prevents customers from being able to be inserted, for the problem classes for which G = 2. It is this higher load which plays a large role in restricting the size of the available neighbourhoods for inserting customers and this restricts the computational time required to evaluate the neighbourhood for insertion, which therefore reduces the computational time required for the method. Since the tasks are intermingled to a greater extent for the problem classes for which G = 2, the loads carried are higher and there are large savings in the computational time compared with the classes for which G = 1.

In order to test the influence of the loads and other features on the computational times, we set up regression models for the data from Tables 8.34, 8.35, 8.36 and 8.37. We create an overall model for all the problems, and we also create four models, one for each combination of N and L. Within the models, our response variable is the running time of the TABUIMP method, and we test the influence of N (NUMSERVED), L (LOADTYPE), R (REWTYPE) and G (GENTYPE), which are our generation parameters, as well as the number of points (NUMPTS, which we partition into the effects of the different numbers by using the indicator variables NUM40, NUM60 and NUM80), the number of customers (NUMCUSTS), the mean load in the vehicle (MEANLOAD), the number of high loads in the vehicle (NUMHIGH), as was used in Table 8.38, and the proportion of available points that are included in the solution route (PROPSERVED). The overall model, together with the statistical tests on the significance of the parameters, is :

TIMES = 119 - 118 NUM40 - 86.5 NUM60 - 49.5 NUM80 - 1.65 NUMCUSTS
+ 1.54 LOAD - 9.78 NUMSERVED + 8.37 REWTYPE + 2.70 GENTYPE
+ 3.94 MEANLOAD - 0.944 NUMHIGH + 40.2 PROPSERVED

Predictor	Coef	StDev	T	P
Constant	119.15	38.37	3.11	0.002
NUM40	-117.70	22.35	-5.27	0.000
NUM60	-86.52	15.03	-5.76	0.000
NUM80	-49.545	7.907	-6.27	0.000
NUMCUSTS	-1.6510	0.9044	-1.83	0.069
LOAD	1.535	1.960	0.78	0.434
NUMSERVE	-9.775	5.443	-1.80	0.073
REWTYPE	8.367	1.576	5.31	0.000
GENTYPE	2.701	1.954	1.38	0.168
MEANLOAD	3.938	1.777	2.22	0.027
NUMHIGH	-0.9441	0.1544	-6.12	0.000
PROPSERV	40.22	13.75	2.92	0.004
	S = 14.06		R-Sq = 74.2%	

Therefore, in this model, by far the greatest influence is for the number of points in the problem (NUM40, NUM60 and NUM80), and the sequential sum of squares that these factor explain are 63034, 55793 and 34840 with the next highest value being 6012 for REWTYPE. We note that the effect of having 100 points in the problem is included within the constant, and the parameters of NUM40, NUM60 and NUM80 are the difference between the times with this number of points and with 100 points, i.e., on average the difference between the times in moving from 80 to 100 points is 49.545. The overall R^2 value of 74.2% is quite satisfactory, given the number of different problem types considered.

The next model we fit is for LOADTYPE = 1 and NUMSERVED = 1, and the model is:

TIMES = - 12.1 + 73.2 NUM40 + 31.9 NUM60 + 8.9 NUM80
+ 3.06 NUMCUSTS + 6.11 REWTYPE + 1.93 GENTYPE
- 32.8 MEANLOAD + 4.82 NUMHIGH - 132 PROPSERVED

Predictor	Coef	StDev	T	P
Constant	-12.13	81.72	-0.15	0.882
NUM40	73.18	35.16	2.08	0.041
NUM60	31.88	22.70	1.40	0.165
NUM80	8.92	11.81	0.76	0.453
NUMCUSTS	3.058	1.318	2.32	0.023
REWTYPE	6.106	1.619	3.77	0.000
GENTYPE	1.930	2.368	0.82	0.418
MEANLOAD	-32.75	13.19	-2.48	0.015
NUMHIGH	4.818	1.214	3.97	0.000
PROPSERV	-131.95	42.51	-3.10	0.003
	S = 7.217		R-Sq = 91.2%	

This model shows a very good fit to the data as the R^2 value is very high. The significant factors to a 5% level are NUM40, NUMCUSTS, REWTYPE, MEANLOAD, NUMHIGH and PROPSERV. The positive coefficient for REWTYPE shows that the computational time required is higher for the problem class in which the reward offered is a function of the minimum distance that must be travelled in order to service the task, compared with the problems with randomly distributed rewards.

For the problems with LOADTYPE = 1 and NUMSERVED = 2, the regression model we obtain is:

TIMES = - 103 - 73 NUM40 - 59.9 NUM60 - 38.1 NUM80
+ 2.12 NUMCUSTS + 6.29 REWTYPE + 1.46 GENTYPE
+ 50.2 MEANLOAD - 2.10 NUMHIGH + 8.2 PROPSERVED

Predictor	Coef	StDev	T	P
Constant	-103.5	172.2	-0.60	0.550
NUM40	-73.3	103.8	-0.71	0.483
NUM60	-59.93	70.23	-0.85	0.396
NUM80	-38.06	35.10	-1.08	0.282
NUMCUSTS	2.120	2.247	0.94	0.349
REWTYPE	6.294	1.763	3.57	0.001
GENTYPE	1.460	1.910	0.76	0.447
MEANLOAD	50.23	36.03	1.39	0.168
NUMHIGH	-2.096	2.452	-0.85	0.396
PROPSERV	8.22	90.08	0.09	0.928
	S = 7.547		R-Sq = 91.0%	

For this set of problems, the only significant variable at a 5% level is REWTYPE, and the positive coefficient again shows that the running times are higher for the problems in which the rewards are a function of the distance required to service the customer. With this reward distribution, long tasks will have higher rewards, so there are fewer customers that will not feature in the problem, due to them being 'unattractive'. There is no significant effect for the numbers of available customers for these problems so the running time is not significantly affected by the size of the problem in these cases.

We now consider the problem set for which LOADTYPE = 2 and NUMSERVED = 1, and these problems seemed to showed a trend that the running times were longer for GENTYPE 2. The regression model for these problem classes is:

TIMES = 59.0 - 61.6 NUM40 - 52.9 NUM60 - 31.8 NUM80
- 0.09 NUMCUSTS + 12.3 REWTYPE + 19.9 GENTYPE
- 9.61 MEANLOAD + 1.38 NUMHIGH - 44.9 PROPSERVED

Predictor	Coef	StDev	T	P
Constant	58.99	83.96	0.70	0.485
NUM40	-61.58	51.09	-1.21	0.232
NUM60	-52.85	33.78	-1.56	0.122
NUM80	-31.83	17.22	-1.85	0.069
NUMCUSTS	-0.091	1.999	-0.05	0.964
REWTYPE	12.332	3.541	3.48	0.001
GENTYPE	19.905	8.492	2.34	0.022
MEANLOAD	-9.607	6.517	-1.47	0.145
NUMHIGH	1.3804	0.9241	1.49	0.140
PROPSERV	-44.86	36.99	-1.21	0.229
	S = 15.43		R-Sq = 81.1%	

For this model, the only significant parameters to a 5% level are REWTYPE and GENTYPE. The coefficient for GENTYPE is positive which shows that the time required to run TABUIMP is higher for G = 2, which is with the longer tasks. When we fit a model which also includes the number of customers serviced in the solutions, we find that this parameter has a significant effect, whereas GENTYPE is no longer significant, and we suggest that the effect of GENTYPE is due to there being greater numbers of customers serviced when G = 2, as the time limit is much longer. This is therefore a result of the way we set the time limit for the problems, rather than being a direct result of the type of tasks considered.

The last regression model we fit is for the problem class for which LOADTYPE = 2 and NUMSERVED = 2. We have already considered this problem class in some detail, and have noted the dramatic difference between the loads in the vehicle for the different methods of generating tasks, but now we verify whether this actually does affect the running time of the TABUIMP method. The regression model fitted is:

TIMES = 216 - 133 NUM40 - 96.3 NUM60 - 54.1 NUM80
- 1.76 NUMCUSTS + 8.07 REWTYPE - 11.1 GENTYPE
+ 16.9 MEANLOAD - 1.51 NUMHIGH - 85.5 PROPSERVED

Predictor	Coef	StDev	T	P
Constant	216.32	64.67	3.34	0.001
NUM40	-133.43	35.83	-3.72	0.000
NUM60	-96.30	24.13	-3.99	0.000
NUM80	-54.08	13.11	-4.13	0.000
NUMCUSTS	-1.760	1.473	-1.19	0.236
REWTYPE	8.073	2.601	3.10	0.003
GENTYPE	-11.082	3.751	-2.95	0.004
MEANLOAD	16.852	5.017	3.36	0.001
NUMHIGH	-1.5142	0.3240	-4.67	0.000
PROPSERV	-85.48	36.24	-2.36	0.021
	S = 11.520		R-Sq = 82.0%	

This model shows that each of the parameters NUM40, NUM60, NUM80, REWTYPE, GENTYPE, MEANLOAD and NUMHIGH are all significant to a 1% level, with the parameter PROPSERV also appearing to influence the computational time required. The coefficients of the parameters NUM40, NUM60 and NUM80 show that there is a non-linear relationship between the running time and the number of points, as the time required increases at a higher rate when higher numbers of points are included. Again, the running time is higher for the distribution of rewards relating to the distance required for servicing, and the negative coefficient for GENTYPE shows that the time required is significantly higher when there is little spatial separation between the tasks, which we had hypothesized previously. The load parameters, MEANLOAD and NUMHIGH also have a combined effect on the running times, as the number of high loads, which shows the number of positions in which no insertion may be made, has a negative correlation with the running times and also a significantly negative coefficient. MEANLOAD also has a negative correlation with the running times, but the positive coefficient for MEANLOAD is used to penalize routes with medium loads throughout the route, but that don't have high numbers of positions for which no insertions may be made. The PROPSERV parameter again has some influence on reducing the computational time, as there are fewer routing options when the proportion of customers served are high.

When we create a new regression model for this problem set, just considering the data for which there are 80 or 100 points in the layouts, neither of the load

parameters MEANLOAD and NUMHIGH, have significant parameters. In these cases, these parameters are so highly correlated to GENTYPE, that the GENTYPE parameter accounts for the influence of the loads in the vehicle.

Therefore, for the TRCP we have implemented our Tabu Search methods on a new set of test problems and found that for certain classes of problems, i.e., with low loads and high numbers of customers served, there are large differences in the behaviour of the problems according to the methods for generating the tasks. We found that, for these problems, the computational time required for our methods was much lower when we generated the tasks from consecutive points from a randomized nearest neighbour tour (G = 2), rather than generating tasks that were from consecutive points from a good TSP solution tour (G = 1). We found that longer tasks were created by the method G = 2, and the tasks were intermixed more within the solution routes, which led to higher loads in the service vehicle and a greatly reduced neighbourhood for the insertion of customers (that is used by our inter-route improvement routines).

The implication of this finding is that in order for our methods to be able to be efficiently applied to these task problems for which the capacity constraints are non-binding, we may need to restrict the neighbourhood that we consider for our improvement routines. One way to do this, would be to restrict the insertion of the subtasks of a customer, so they may only be inserted adjacent to one of their p nearest neighbours within the solution route, where p is a user-defined parameter that determines the size of the neighbourhoods considered. This is the approach that has been used when applying the GENI method [60], in order to reduce the computational time requirement by disregarding insertions that are not likely to result in worthwhile solutions. Care must be taken with this approach, as the diversification of solutions, that may be obtained by not placing the customer in its 'best' position, is an important feature of Tabu Search methods so the elimination of these moves may restrict the effectiveness of this search routine.

The other major feature of the computational times, was that the method of generating the rewards according to the distance required to service the customer, led to consistently longer computational times required by the TABUIMP method for each of the problem classes. With randomly generated rewards, customers with long distances but low rewards can be excluded from consideration, which effectively reduces the size of the problem considered. With rewards relating to

the distances, all tasks need to be considered, as the customers which have long distances required receive higher rewards, so there is less variability in the attractiveness of the customers. These problems therefore appear to be 'harder' ones to solve, which makes them of greater interest for further investigation.

8.2.6 Testing Tabu Search Methods for the Combined MCP

We now test some further instances of the form of the Combined MCP that we tested in Section 8.1.4, and we use the problem instances for testing the full implementation of our Tabu Search method. We generate problem instances in a similar manner to that which we used for the TRCP in Section 8.2.5, but there are additional requirements, associated with the time-dependent rewards, which need to be included within these instances.

We set the number of points that comprise the tasks to be equal to 40 and 60, and for each number of points we create two rattled grid layouts. The parameters that we vary to create the problem classes are:

- L, the size of the loads to be carried. For L = 1, the size of each load is equal to half of the vehicle capacity, so only two loads may be in the vehicle at any one time. For L = 2, there are no loads to be carried, so there is no capacity restriction on the vehicle.

- N, the number of customers that may be serviced within the solution route. For N = 1, which is the case with few customers serviced, we create sections of route containing 5 points and use these to assign the relevant constraints to the customers. For N = 2, larger numbers of customers may be serviced, so we create a TSP tour around the points, and assign the constraints to the tour.

- T, the width of the time window over which the service of the task may be completed. We set the time window width to be a proportion, k, of the time limit for the problem instance, where $k \in (0.1, 0.2)$ for T = 1 and $k \in (0.2, 0.6)$ for T = 2.

- G, the method of generating the tasks. We apply the task constraints to consecutive points from the sections of route, where, for G = 1, we base the constraints upon matchings or TSP tours found by a best insertion method,

whereas for G = 2, we generate our sections of route from our randomized nearest neighbour method.

- R, the type of reward function for each of the tasks. For these problems we partition the time window for servicing the last point of a task into three equal length intervals. The middle interval is the time period over which the customer's reward is at its maximum, whereas the first and third intervals are the times over which the rewards are increasing and decreasing, respectively. For R = 1, there are no penalties for excess riding time, whereas for R = 2 and R = 3 we set the penalty for excess riding time so that the reward is equal to zero if the riding time exceeds four times its minimum duration and double its minimum duration, respectively.

Therefore we obtain a total of 48 problem classes for each number of points and with two repetitions of each problem class and three numbers of points considered, we obtain a total of 192 problem instances. One adaptation to our method that we use in this case, is to set an earliest time for which a customer's service may be commenced, with this being equal to the beginning of the time interval for the completion of the task less the maximum riding time allowed, while the reward may remain positive. This additional feature restricts the problem space by limiting the time for which the customer may be serviced. In order to handle this additional time window, we needed to update our method for calculating the waiting time for a fixed route, as we need to explicitly consider the time at which each time-constrained subtask is serviced. Previously the only time-constrained subtasks were the final points of a task, so the evaluation of the customer's reward explicitly considered its time window of availability, but with waiting time allowed at other points, which would previously have not been considered, we need to restrict the time window over which the linked sections are able to be serviced within the waiting time algorithm.

The results from this testing are shown in Tables 8.39 and 8.40. The methods we employ are TIMP (which is the implementation of our full Tabu Search routine, TABUIMP), TMOVE (which is our implementation of TABUIMP, but with the Move routine being the only inter-route improvement routine included, and HEUR (which involves implementing each of the six heuristics that were used in Tables 8.13 and 8.14, with each initial solution improved by the use of the STEEPEST ASCENT algorithm and the application of our waiting time insertion algorithm.

Class	N = 1						N = 2					
	TIMP		TMOVE		HEUR		TIMP		TMOVE		HEUR	
L,G,T,R	r	t	r	t	r	t	r	t	r	t	r	t
1,1,1,1	78.3	23	78.3	4	75.7	1	311.0	30	288.8	2	303.9	0
1,1,1,2	75.6	16	74.9	4	71.6	2	299.0	21	298.3	2	303.9	0
1,1,1,3	75.6	16	75.6	3	69.6	1	287.8	13	279.3	2	303.9	1
1,1,2,1	94.9	150	92.8	22	91.5	12	285.5	79	280.8	4	291.5	1
1,1,2,2	92.8	114	88.3	18	91.5	11	268.9	60	268.9	4	291.1	1
1,1,2,3	94.9	77	88.3	13	91.5	6	268.6	43	268.6	4	291.1	1
1,2,1,1	135.7	32	135.7	6	131.9	4	425.8	20	411.4	3	429.9	1
1,2,1,2	125.7	17	112.9	6	131.9	2	376.7	12	353.3	3	429.9	1
1,2,1,3	115.8	10	111.5	2	131.9	1	372.1	15	370.6	3	429.9	1
1,2,2,1	193.7	111	177.7	23	190.1	12	452.7	149	455.6	8	452.0	4
1,2,2,2	179.2	101	172.5	24	179.0	12	420.5	110	419.4	8	429.3	4
1,2,2,3	166.3	57	166.2	18	165.1	6	382.9	96	380.9	9	419.0	4
2,1,1,1	63.5	14	63.5	4	58.5	1	287.5	12	234.9	2	294.5	0
2,1,1,2	62.3	13	62.3	4	58.5	1	262.7	12	243.5	2	288.0	1
2,1,1,3	62.3	15	62.3	3	58.5	0	252.7	13	246.7	2	285.5	1
2,1,2,1	90.2	61	90.2	9	87.0	3	291.6	34	246.2	6	278.4	3
2,1,2,2	90.2	60	83.4	8	87.0	2	272.0	40	257.9	7	268.5	2
2,1,2,3	90.2	49	83.4	8	87.0	2	271.7	46	238.4	6	267.6	2
2,2,1,1	136.2	39	134.5	6	135.9	2	424.5	40	332.9	8	433.4	2
2,2,1,2	128.8	40	123.7	8	135.9	2	375.4	37	336.2	7	427.3	2
2,2,1,3	118.9	25	126.5	7	135.9	2	364.7	35	337.5	7	427.3	2
2,2,2,1	152.7	70	144.7	12	151.4	5	456.2	144	436.7	21	446.8	16
2,2,2,2	144.3	37	142.0	13	148.5	6	382.8	169	364.3	22	373.4	12
2,2,2,3	132.0	86	128.7	13	148.5	6	354.3	152	341.3	20	372.0	15

Table 8.39: Mean Reward and Computational Time for the MCP with 40 points

Table 8.39 shows that the TMOVE method appears to be the least effective of the methods for many of the problem instances. This method is also somewhat erratic, as, for example, with L = 2, N = 2, G = 1 and T = 1, the reward obtained is highest with R = 3, i.e., with the highest penalties for excess riding time, and lowest for R = 1, for which there is no penalty for excess riding time. Of particular interest is the effectiveness of the HEUR method, especially for problems for which N = 2 and L = 1, and for these problems the HEUR method quickly obtains solutions, of which many outperform the solutions obtained by the TIMP method. For two of these problem types, the solutions obtained achieve the same reward value for each type of reward function, which shows that the solution routes involve all customers being serviced consecutively, regardless of whether the reward function penalizes for excessive riding time or not. Looking at the solutions obtained in more detail, we find that most of the solutions that are counted for the HEUR method involve the GREEDY method, in which customers are added to the routes in decreasing order of their maximum reward value and the customers are positioned so that the maximum available reward is received from each customer.

For the set of problems given in Table 8.39, we find that the mean reward obtained by the TIMP method over the problems for which N = 1 is 112.517, the mean for HEUR for these problems is 113.065, while the mean obtained by the GREEDY method is 112.121. The respective means for these methods with N = 2 are 339.480, 355.760 and 352.874, so the GREEDY method itself obtains a much higher mean reward than the TIMP method for problems with high numbers serviced.

The results of the testing, with 60 points comprising the tasks, are shown in Table 8.40. This again shows that the TMOVE method is the least effective method for most of the problem instances. If we again look to summarize the performance of the methods over the different values of N, we see that for N = 1 the mean reward obtained by TIMP, HEUR and GREEDY, respectively, are 124.508, 125.441 and 121.309, while for N = 2 these means are 489.834, 504.457 and 496.555, so again the simple GREEDY method obtains higher rewards than TIMP with high numbers serviced. The reason for the advantage for the GREEDY method is the manner in which it assesses the rewards for inserting a customer in the construction phase. This method adds the customer at a time so that their maximum reward is obtained, and this involves the implicit introduction of non-compulsory waiting

Class	N = 1						N = 2					
	TIMP		TMOVE		HEUR		TIMP		TMOVE		HEUR	
L,G,T,R	r	t	r	t	r	t	r	t	r	t	r	t
1,1,1,1	92	14	89	2	94	1	423	9	401	4	399	2
1,1,1,2	92	15	89	4	94	1	390	7	374	3	385	2
1,1,1,3	89	17	89	3	94	2	371	8	357	3	383	1
1,1,2,1	88	68	86	9	105	5	386	13	380	14	399	5
1,1,2,2	88	59	88	8	103	4	374	11	369	11	391	6
1,1,2,3	84	61	82	8	103	4	372	13	364	10	395	4
1,2,1,1	166	41	158	5	143	2	590	27	573	6	567	3
1,2,1,2	151	42	155	6	132	3	525	31	514	2	563	2
1,2,1,3	140	35	148	5	130	2	504	36	483	3	563	2
1,2,2,1	171	155	170	21	156	10	586	38	576	21	581	13
1,2,2,2	167	157	165	24	165	10	553	31	543	21	550	17
1,2,2,3	159	165	158	18	158	8	530	32	530	18	528	13
2,1,1,1	87	145	85	19	90	11	454	168	347	19	487	8
2,1,1,2	87	93	84	12	90	9	429	64	334	14	487	8
2,1,1,3	85	50	84	8	90	8	416	52	338	8	487	6
2,1,2,1	97	680	98	60	119	73	454	703	391	89	450	89
2,1,2,2	99	274	96	37	119	65	439	305	376	72	439	101
2,1,2,3	99	244	97	41	119	39	430	554	387	60	423	54
2,2,1,1	149	588	144	43	136	59	579	580	451	59	580	49
2,2,1,2	137	168	130	14	120	28	511	137	445	16	555	14
2,2,1,3	122	127	113	10	114	16	502	40	489	8	551	8
2,2,2,1	196	1102	184	154	191	207	742	1225	689	152	747	145
2,2,2,2	178	726	173	151	178	214	613	779	584	81	613	109
2,2,2,3	168	902	166	72	168	123	584	460	572	34	584	56

Table 8.40: Mean Reward and Computational Time for the MCP with 60 points

time. The TIMP method includes the technique for including waiting time to the problem after every tenth iteration, but it does not explicitly consider the impact of the waiting time at other phases during the search. This method therefore obtains relatively poor results due to not being able to effectively deal with situations for which waiting time is required in order to obtain obtain reasonable quality solutions. The development of techniques that are able to handle these waiting times therefore appears to be an important step in improving the applicability of the TIMP method to more general cases of the MCP, with adaptive time windows or efficient application of the waiting time insertion routine being potentially beneficial for this purpose.

CHAPTER 9

Dynamic Subset Selection Routing

We now consider a further practical extension to subset selection routing problems, involving customer requests for service arriving in real-time, with the vehicle routes also needing to be devised in real-time. This situation is known as a dynamic problem, according to the definition of Psaraftis [146, page 151], who stated that

> "a vehicle routing problem is 'dynamic' (also known as 'real-time', or 'on-line') if information (input) on the problem is made known to the decision maker or is updated concurrently with the determination of the set routes."

Therefore, by this definition, any problem where information arrives as the routes are being created constitutes a dynamic vehicle routing problem. The distinction is made between problems with dynamic decision making and ones where the decisions are made based on a fixed, complete set of data, e.g., for the time-dependent travelling salesman problem the time to travel between two points may depend upon the time at which the travel takes place, but routing decisions are carried out before the route is started and assuming that the travel times are known.

We set out to model a situation where there are customers arriving for service, and we can choose whether or not to service each customer according to the profitability of the service. The extension we address here involves the decision-making process being carried out with partial information. We consider the possibilities for

dynamic subset selection vehicle routing models, and explore practical strategies for solving some problem classes for these problems.

9.1 Previous Work on Dynamic Vehicle Routing Problems

In order to identify the context for dynamic subset selection problems, we now consider previous work on dynamic routing problems, where we are particularly interested in how subset selection has been or may be applied within the given models. The development of the area of dynamic vehicle routing has lagged behind the work in static problems, and the definition of the problem area is a relatively recent innovation. Psaraftis [145] noted the lack of development of this area, and described 12 factors that are able to distinguish between static and dynamic problems. As we discuss later, most of these factors are also relevant with dynamic subset selection problems, although there are some extra considerations that should be dealt with.

Psaraftis [145] went on to define the Dynamic Travelling Salesman Problem (DTSP), which involves customer requests for service arriving from a set of known locations, according to a Poisson process, where the optimal route to service all the customers, according to a given objective, is sought. The objective may be to minimize the total time taken or to minimize the waiting time for each customer.

In a follow-up survey, Psaraftis [146] described the advances in technology that have led to the increased relevance of dynamic vehicle routing problems. These include the improvements in communications and vehicle tracking, e.g., global positioning systems, which allow the routing coordinator to keep track of the current location of the service vehicle. Further advances in Internet-based systems that have been developed since this paper was written, together with the immediacy requirements that current society dictates, have led to even further requirements for dynamic routing models.

The definition of the DTSP led to a number of theoretical articles that consider similar problems, and exploit results from the well-developed statistical field of queueing theory. Bertsimas and van Ryzin [14] defined the Dynamic Travelling Repairman Problem (DTRP), which is equivalent to the DTSP, except customer locations are defined in the Euclidean plane and the objective is to minimize the waiting time for each customer. A number of different solution techniques were

considered, where these methods were chosen so that achievable theoretical results could be utilized, rather than for finding effective solutions. For example, a travelling salesman policy is employed, where the salesman will always serve a fixed number of customers before returning to the depot. This policy has provable results, based on the known asymptotic expected length of a TSP tour, but it involves a lot of dead time (in unnecessarily returning to the depot), so is unlikely to be of practical use.

Bertsimas and van Ryzin developed similar theoretical results for further versions of the DTRP with multiple vehicles (see [15]) and general distributions of the interarrival times and customer locations (see [16]). Further theoretical results, including a proof of the asymptotic optimality of methods under certain conditions, have been obtained by Papastavrou [131, 132], where the solution methods considered again enable queueing results to be utilized. An interesting extension is the work of Swihart and Papastavrou [170], which extends the theoretical results for the DTRP to a dynamic version of the DARP. Again theoretical results were obtained for tractable solution methods, although through simulation it was discovered that a simple solution method (for which theoretical behaviour could not be ascertained) based on selecting the customer nearest to the current location, was able to outperform the methods for which theoretical results were obtained.

The theoretical analysis of dynamic routing problems enables the effectiveness of methods to be compared with asymptotically optimal behaviour of the problems (although this behaviour is unlikely to ever exist). The most important results are in the structure of the problem, through finding what factors affect the theoretical behaviour of a problem. For example, Swihart and Papastavrou [170] identified that the asymptotic lower bound on solution quality for the dynamic version of the single vehicle DARP does not depend on the capacity of the vehicle, so other factors are of greater importance.

Recent research into dynamic vehicle routing problems has involved many practical considerations being explored. Lund, Madsen and Rygaard [117] investigated the behaviour of a practical solution policy for the VRPTW under varying degrees of dynamism, where each arriving customer request is inserted into the current solution routes. They found that the amount of warning given, i.e., the length of time between when the request for service is lodged and the start of the required time window, was an important factor in determining the effectiveness of their method.

Savelsbergh and Sol [159] described a routing situation where a proportion of the customers become known during the course of the day. Their solution method creates permanent routes for customers whose request needs to be serviced in the near future and provisionally places the remaining customers. One novel feature of this method is the consideration of the effect of customers that haven't yet arrived, as the number of vehicles scheduled for use in the following day is determined according to the expected demand present.

Gendreau, Guertin, Potvin and Taillard [59] described an implementation of Tabu Search for a dynamic routing problem with soft time windows. The Tabu Search process is run while the vehicles are actually operating, and this search process is stopped whenever a new request is lodged or one of the vehicles reaches its destination. A new customer is trialled for insertion into one of the current feasible tours in the adaptive memory, and the customer is discarded if it can not be included within any of these existing routes. This method assumes that the only time a vehicle's route may be changed is when it is situated at a current customer location.

Ichoua, Gendreau and Potvin [86] developed a method that enables a solution vehicle to deviate from its route at any time during the routing. When a new request arrives at time t, a given time δt is allowed for the computing new solution routes, where the evaluation takes place according to the location at which each vehicle will be situated at time $t + \delta t$. Customers arriving while this evaluation is taking place are assessed for their feasibility according to the current set of feasible solution routes in the adaptive memory; those customers for whom no feasible insertion is found are rejected, while those which can be inserted are added to a list of provisional customers to be included later. One problem is that a customer which can be added to the adaptive memory, may not be able to be added to the solutions that are obtained after the Tabu Search has been implemented, which causes delayed notification of non-service to the customers involved.

These methods described above were each tested on problems where all customers were able to be serviced within the solution routes, e.g., Lund et al. [117], Gendreau et al. [65] and Ichoua et al. [86] all tested their methods on the test problems of Solomon [166], where unserviced customers are a result of the solutions for the known customers not being in a form that enables the complete feasible solutions to be obtained. Therefore the objective is to minimize the routing costs,

where an attempt is made to service all customers. It is theoretically possible to service all the customers in these problem instances, although there is no way to guarantee that the dynamic method will be able to find such a solution. In some cases, in order to be able to service all customers, routes that are poor with respect to the current set of available customers are required, and these routes are unlikely to be adopted unless information about future customers is known.

These dynamic models need to include a capacity to reject customers if they can not be included within the current routes, but no consideration has been given to actually modelling the non-service. These models are therefore unable to effectively deal with problem cases where there are a significant number of customers that are unable to be included, so are inappropriate within subset selection objectives.

A dynamic situation where subset selection is addressed is in the paper by Kleywegt and Papastavrou [97]. They considered the case of distribution companies deliveries of loads, where they describe the form of the process as being LTL, where loads carried are less than truckloads, so more than one request can be carried in the vehicle at one time. Kleywegt and Papastavrou described the situation we are looking to model, and stated that [97, page 127]:

> "Although many LTL carriers have long followed the policy of accepting as many loads as they can handle, they do have the option of rejecting requests and several are investigating this option in more selective ways."

The paper describes the long-haul distribution process between depots, where the loads to be carried are located at these depots. Each customer request consists of a batch of loads, where given volumes of loads need to be carried between given pairs of depots. When a new request for service arrives, it is assumed that the price, or reward for the service, is negotiated and the distributor is able to accept the customer (thereby receiving the agreed reward) or reject the customer, which incurs a cost. The objective is to maximize the net profit of the system, where the profit is the reward for servicing less the operating and rejection costs.

The process is modelled in an impractical manner so as to obtain theoretical results. This involves the travel time between any two depots being given by a Poisson Process with a given rate, which allows the state transition probabilities to be found for the Markov Decision Process Model that is developed. The value of a given policy is given according to these transition probabilities, and an optimal

policy is to accept a customer's request if the customer can be feasibly serviced and the expected net gain is positive, i.e., if the reward offered exceeds the change in costs. The paper mentions that approximation methods have been developed for this form of problem, although the main contribution is in the development of theoretical results for a newly modelled situation.

The introduction of subset selection, creates parallels with the field of selection policies. One well-known problem in this field is the secretary problem, in which candidates for a number of positions arrive sequentially and they can be accepted for a position or rejected. The objective of this problem is to maximize the expected value of the selected candidates.

Another related problem is the Dynamic and Stochastic Knapsack Problem (DSKP), which has been studied by Papastavrou, Rajogopalan and Kleywegt [133] and Kleywegt and Papastavrou [98]. In the DSKP, loads of given sizes arrive and they can be added to the current knapsack (which is a container with a given capacity), thereby earning a known reward, or rejected, which incurs a penalty. There may be a holding cost on the time the loads are held in the knapsack, and there may be a given time horizon over which customers arrive. This case can allow for the problem to be stopped at any stage, which involves the knapsack being dispatched. The objective of the DSKP is to maximize the expected net reward received from policies which are able to accept loads and to dispatch the knapsack. The papers found optimal solution policies for certain forms of the problem with equal loads, where the optimal policy is to accept the customer according to a simple threshold ratio of reward to size of load.

9.2 Modelling Dynamic Subset Selection Routing

We now consider the practicalities involved with the modelling of dynamic subset selection routing problems, and develop some specific models. The problems we consider involve customer requests for service arriving at a central dispatcher and, for each request, a decision as to whether or not to service the request is made. We consider two different situations: where the requests arrive prior to the period over which the routing is to take place, and where the requests arrive as the service vehicles are carrying out the planned service.

We believe that these situations are relevant for the types of problems that may occur in practice. A recent article on current vehicle routing trends by Partyka and Hall [134], describes how the Internet has affected the vehicle routing problems that require solution today. They observed [134, page 29] that:

> "The Internet has created three classes of home delivery: substitute, next day and same day."

Same day and substitute delivery services involve customer requests arriving while the service vehicles are operating. Substitute delivery involves Internet ordering replacing traditional ordering methods, and service in this case may or may not be constrained to given time periods, i.e., pizza delivery is an immediate request while home shopping orders may take much longer to deliver. Same day service involves the requests being taken while service vehicles are operating, which requires the vehicle routes to be updated while the servicing is taking place. If there is a spare service vehicle available, it may be possible to create a new solution route, otherwise the updating just involves the current vehicle routes. Next day servicing involves the decisions being made prior to carrying out the resulting routes. The case that Partyka and Hall [134] gave for this form of problem is for home-delivery grocery shopping, where requests for service are taken the day before the deliveries will be made. Subset selection problems may arise within any of the forms of delivery described above when the demand arriving exceeds the capacity of the service company, and they may also occur in certain cases when the service of a customer is deemed to be too costly to warrant servicing. Next day and same day delivery may lead to dynamic subset selection problems, and we describe here a general model for these problems and develop more specific models in the ensuing sections.

We define a *dynamic subset selection routing problem* (DSSRP) to be a problem where requests for service arrive while vehicle routes are being determined, and some customers may be denied serviced. The information that evolves in real-time is the set of eligible customers, and decisions on the composition of the vehicle routes may be made in light of the changing information. The objective of such a problem is to maximize some measure of profit, where this objective may be evaluated over all the customers, or over the subset of customers who are serviced.

We now consider DSSRPs in the context of the 12 important features of dynamic vehicle routing problems, as defined by Psaraftis [145]. These points, together with

our opinions on their relevance to DSSRPs, are:

1. *Time dimension is essential* — the time dimension is essential for a DSSRP, as customers are arriving during the course of a time period and so when decisions are made, only a subset of the available customers, i.e., those whose requests have arrived up until the current time, are known of with certainty. The time dimensions may also relate to the urgency for serving customers, in the case where customers have desired times or intervals for servicing and also if there are penalties for the deviation between the service time and the desired time.

2. *Problem may be open-ended* — one of the major advantages in modelling subset selection problems is the elimination of feasibility concerns, as any customer that can not be included in the solution can be discarded. The allowance of an open-ended problem, is mostly a concession towards feasibility, as it ensures that all customers will be able to be serviced at some stage. For DSSRPs, the problem may still be open-ended in cases where the rewards received are non-constant, but for problems where the rewards are constant and there are no costs for servicing, i.e., the OP in the static situation, a given time limit is required in order to create meaningful problems.

3. *Future information may be imprecise or unknown* — if all future information is known with certainty, then the problem can be treated as a static problem, with all decisions taking account of the known future customer arrivals. This problem would then be a static problem, with release times defining when the customers become available. In order for the problem to be truly dynamic, some unknown or uncertain information needs to be become known with certainty as the problem proceeds.

4. *Near-term events are more important* — for DSSRPs, events that are known are of greater importance than those which are only predicted or expected. However, given a limit on the time available for servicing, there may be inefficiencies if servicing capacity is overly allocated to currently known customers, at the expense of future customers (which may be of greater value). When all customers must be serviced, devising good routes for the currently known customers is an important feature of effective solution routines, whereas for DSSRPs, the allocation of resources to customers that ultimately are not

beneficial for inclusion may result in poor solutions and solution methods. Therefore the value of future information is greater in the case of a DSSRP. Difficult situations arise with how to deal with customers who place advanced requests for service, as future routes may not be known with any certainty and the commitment to servicing such a customer may lead to poor solutions being obtained. Therefore customers that are isolated with regard to their time of servicing, would appear to be of lower attractiveness than ones with similar rewards, due to the greater uncertainty as to whether they will be beneficial when the actual routes are implemented.

5. *Information update mechanisms are essential* — the ability to process customers as they arrive and to update the list of available customers, is a necessary element of dealing with any instance of a DSSRP. Also essential is the ability to keep track of the locations of the service vehicles at any stage, as these are needed in order to be able to alter the routes when new customer information arrives.

6. *Resequencing and reassignment decisions may be warranted* — the arrival of a customer request that is beneficial may warrant this customer's inclusion within the solution routes, and the current routes may need to be dramatically altered in order to accommodate the new customer. One important new feature of a DSSRP is that the decision making may involve the removal of a customer that has previously been scheduled. Thus, if the problem definition allows for such a case, it may be advantageous to choose to not service a customer after previously placing the customer within the solution routes. This introduces an additional improvement move, that of excluding a customer from service, and needs special consideration to be taken to ensure that the service requirements of the problem are not violated. For instance, a cost may be associated with the exclusion of a customer, with an infinite cost being used to prohibit the exclusion of a customer for whom service may be critical.

7. *Faster computation times are necessary* — the arrival of customers in real-time leads to routing decisions needing to be made quickly, so that the decisions relate to the current state of the problem. The Tabu Search method of Ichoua et al [86] considers this explicitly, through allowing a given amount of

computational time to elapse, and evaluating the routes according to where the vehicles will be at the end of this time period. Techniques such as this may also be appropriate for a DSSRP, but the time element will no longer just affect the routes involved, as the level of customer service needs to be explicitly considered. For a DSSRP, the important consideration with regard to the route updates, is to enable the server to quickly relay their intentions to the customers. Therefore, if a customer is to be excluded from service, immediate notification may result in lower customer dissatisfaction, so we see the quick updating of routes as an even more important feature of solution methods for a DSSRP. The techniques that are appropriate, then, are ones that allow very quick updates to the solutions, with a trade-off being that the quality of solution is not as high as might be obtained with more advanced, time-consuming methods.

8. *Indefinite deferment mechanisms are essential* — indefinite deferment mechanisms are required in dynamic routing problems to ensure that all customers are serviced. Cases where such a mechanism may be required in a DSSRP are where a customer is accepted for servicing and there is an infinite cost of excluding a customer at a later stage. If the customer's service is delayed too significantly, the customer may balk (thereby refusing service) and so mechanisms, such as the incorporation of time windows, may be required in order to ensure that all selected customers will be serviced to an acceptable standard. Balking can cause significant problems to the routing as a service vehicle may be allocated to a customer that will not accept service, which means the vehicle resources will have been assigned inefficiently. Otherwise, if there is no restriction on which customers are to be serviced and there is freedom to accept and reject customers at any stage, there is no need to prevent deferment as non-service is an important feature of a DSSRP.

9. *Objective function may be different* — for a static subset selection routing problem, the objective that is commonly used is to maximize the net benefits of servicing. One additional feature that may be necessary (as described above) is the inclusion of a cost of exclusion that will penalize for excluding a scheduled customer at a later stage. Another feature that may be useful is in the evaluation of a change to a solution, where an opportunity cost in terms of possible future improvements may be considered when allocating time to

the servicing of a currently known customer. Therefore, rather than just considering the current benefits, the effect on future benefits may need to be allowed for, and so the objective function being evaluated may be different.

10. *Time constraints may be different* — this feature is used to allow greater flexibility in decision-making, so that infeasible solutions may be prevented through servicing a customer outside of its imposed time window, thereby dealing with the time window as if it is soft. For a DSSRP, this may be of benefit in allowing a customer to be serviced outside of its time window, rather than incurring as large a cost for excluding the customer from service.

11. *Flexibility to vary vehicle fleet size is lower* — obtaining access to additional service vehicles may not be possible in real-time situations, in which case the best solution with the current fleet size is sought. If it were possible to obtain additional vehicles, a possible occurrence may be the allocation of previously scheduled customers to additional vehicles, so that improvements may be made through being able to service a newly arriving customer request.

12. *Queueing considerations may become important* — queueing considerations have previously been considered due to the theoretical results that can be exploited (see [14, 15, 16]). These are of benefit due to the requirement that all customers be serviced, but with the ability to exclude non-beneficial customers within a DSSRP, these queueing results effectively become irrelevant.

These features demonstrate how DSSRPs may be considered within the framework for previous dynamic vehicle routing problems. The major alteration that is required is in the consideration of customer service when a customer is excluded from service, and this involves additional features that are not present within static subset selection problems. The fact that it may be beneficial to delay the routing decisions in order to take more information into account, will require some mechanism for ensuring the decision making is not deferred indefinitely. The consideration of customer service also leads to faster solution methods being required, which would prevent the use of exact methods, and so simple, effective heuristics will be essential.

We now look at some specific models of DSSRPs, and consider the practicalities that may be involved. We consider two different cases: where customer requests

arrive the day before service is required, and with customer requests arriving while service is taking place.

9.3 Next Day Dynamic Subset Selection

We now consider in more detail the types of problems that may arise with next day subset selection problems. We consider that there are two cases that are relevant with next day delivery:

1. The static case, where all customer requests are known when the routing decisions are made, and these decisions are made at the start of the next day's routing period. All customer requests are recorded when they arrive, and all routing decisions are made with reference to the complete, recorded data.

2. The dynamic case, where customer requests need to be processed as they arrive. This involves decisions being made according to the present knowledge, and may involve allowances for anticipated future information.

We have previously considered the static representation of the problem, where we have complete information on the customers before selecting the solution routes. This is the case with the MCP that we described in Chapter 6, where we create solution routes according to the complete set of available customers, which are fixed and known with certainty. We now consider an on-line version of the problem, where each customer's request for service needs to be processed as soon as it arrives.

We assume that there are three possible outcomes that may occur when the customer request arrives. Firstly, the service provider may reject the customer immediately, thus refusing to service them (this might occur because the customer is not profitable enough or may be too difficult to service; there may be a cost that is incurred for rejecting the customer, with this reflecting the customer dissatisfaction and perhaps a future unwillingness to use the company's services). Secondly, the server may provisionally accept the customer's request (this option allows for greater flexibility, as it is possible to exclude the customer at some later stage, should a more profitable customer arrive). The third possibility is for the service provider to accept the customer unconditionally, thereby guaranteeing that the service will take place.

The essential element of this problem, which distinguishes it from a static problem, is the inclusion of some factor that penalizes for deferment of the decision process. If there were no such mechanism, there would be no advantage to creating routes with incomplete information, so customers would be scheduled just prior to the routes being carried out, thereby creating a static problem.

There are a number of possible mechanisms that we consider may be appropriate for hastening the decision process. One such mechanism may be differential rewards for provisional and permanent inclusion of the customer. Thus a lower reward may be offered for provisional inclusion, which may relate to the potential lack of service that a customer is afforded, as they are unable to make definite plans when unsure if their request will be accepted.

Another possibility may be the inclusion of a penalty that is a function of the time between the customer being provisionally included and the time that it is excluded from the solution. This will be a non-decreasing function of the delay of notification, with one possibility being to allow the server a given grace period over which no penalty is incurred. This case would promote waiting until the end of the given grace period before carrying out the routing decisions, as this will allow a greater amount of information (at no additional cost) to be referred to when the routing decisions are made. There may be a limit on the time allowed to decide whether to route the customer permanently or not. In this case, once a customer's request is processed as provisionally included, we are not allowed to exclude it from service after this time limit has elapsed. If the time limit is zero, then we have to immediately decide whether to service the customer or not. If a customer has an infinite penalty for delayed notification, this will ensure that this customer will not be excluded once its request has been accepted.

Other penalty functions may be incorporated within this model, for example, where there are differential rates for which the server is penalized, according to the level of service supplied. Thus, if we have a problem where there are time windows on servicing the customers, there may be a higher penalty for excluding a customer from service than for servicing the customer outside its time window. In this case, one of the routing options may be to service the customer poorly (i.e., outside its time window, thereby avoiding the incurrence of the penalty), rather than excluding them from service.

The model of penalties for non-service may also be extended, to become time-dependent. Without penalties, any customer whose reward is not sufficient may be simply excluded, but with penalties, all customers may need to be accounted for when the routes are created. Thus with penalties increasing indefinitely, all customers need to be included at some stage, with this situation being particularly relevant for multi-period routing. The penalties may also be dependent on other penalties incurred, *viz*, if the cumulative effect of customer dissatisfaction exceeds the sum of the individual penalties.

This dynamic routing model also lends itself to a more general dynamic situation, where there are some customers that request same day servicing, and thus the vehicle routes need to be adapted on-line. We look in more detail at the same-day service case later on in this chapter. The addition of immediate request customers, along with those requesting next-day service, may require extra flexibility in the routing, to enable the inclusion of these more urgent customers. The relative importance of the different customers may be affected if there are different distributions of rewards, for example, if the customers with more urgent requests are willing to pay a higher amount for service. A further generalization would be to allow the service to take place on any of the following days, and with multi-period time-dependent rewards which affect the viability of the servicing.

An example of this problem is a courier company which only takes requests for service during the following day. There is a given fleet of vehicles available for servicing the customers, and as such, there is a limit on the number of customers that may be serviced. Therefore, whenever a request for service arrives, we assume that a decision must be made as to whether to service this customer, reject the customer, or provisionally include the customer. This decision relates to the information that is already known about the current routes, and should also allow for customers whose requests are not yet known.

Other extensions of the model may be to incorporate customers consisting of tasks that require servicing, time windows on when the customer may be serviced and time-dependent rewards on the time at which the service is initiated and/or completed. These are the extensions that we considered previously, in creating the MCP.

9.3.1 The Dynamic OP

We now develop a specific dynamic subset selection model, based on the static Orienteering Problem (OP). This model incorporates customer requests (involving the service of a single location) arriving during the course of a day, with service being requested for the following day. The server may choose to reject the customer immediately, which incurs no cost, or provisionally service the customer. With provisional service, if the customer is excluded at a later time, a penalty for delayed notification is incurred, where this penalty is linear in the length of the delay. The objective is to maximize the net reward received for a feasible route that ends at the depot prior to the end of the day, where the net reward is the reward for the customers that are serviced, less the penalties for delayed notification. With this model, the server must assess the impact a customer will have on the current route, and the effect on the viability of future insertions.

Since the customers consist of single locations and the rewards are constant for each customer, this problem is a dynamic version of the OP, and we thus term this problem the Dynamic OP (DOP). The dynamism comes from the incomplete information available when making decisions. This model allows the server to reject as many customers as they desire at no cost, and is as such, perhaps most appropriate for instances where the number of customer requests greatly exceeds the capacity of the server.

This problem is quite similar to the DSKP, as it was described by Kleywegt and Papastavrou [98]. For the DOP the resource usage of a customer, which is the time required for insertion in the solution route, is dependent on the current solution, whereas for the DSKP the resource usage (which is the size of the load) is independent of the solution. Because the loads come from a known distribution, exact results can be obtained for the DSKP, whereas the distribution of the insertion times in the DOP are unknown, and so the results obtained for the DSKP are not applicable. Optimal solution policies for the DSKP involved accepting customers according to given thresholds, and this idea does seem to be one that can be applied to the DOP. Also, since customers can be excluded at a later stage (with the incurrence of an appropriate penalty), there is an additional level of flexibility that can be taken advantage of in the DOP.

We create a simulation environment to test the effectiveness of some heuristics for this dynamic problem. The simulation is over an eight hour day, and the

customers are randomly positioned in the Euclidean $[0, 2]^2$. We create a number of layouts, with 200 potential customer locations stored for each. There is a single service vehicle, which is initially located at the depot, positioned at the centre of this square. The travel time, in hours, between any two points is set equal to the Euclidean distance between these points. We generate the time at which each customer's request becomes available, from a Poisson process, with the arrival rate for customers being an input parameter. We create different problem classes through varying the distribution from which the rewards are generated, the rate at which customers arrive for service and the rates of the delay penalties.

We create 18 problem classes, through using three values of the Poisson arrival rate, 2.5, 5 and 10 per hour, two distributions of the reward value, $U(10, 20)$ and $U(10, 100)$ and three values of the delay penalty, 0.5, 0.2 and 0.05 of the customer's reward per hour of delay. The two distributions of rewards have dramatically different variabilities, which creates problems with marked differences in the 'attractiveness' of the customers to the service provider. The different arrival rates of the customers, create instances of low, medium and high demand, and these affect the extent to which future information needs to be considered. The different delay penalties affect the urgency of making permanent decisions. These delay penalties are given as a function of the customer's reward, so a customer that has high benefits will also incur high penalties if excluded at a later stage.

Solution Methods

For the DOP, we developed a number of solution methods for preliminary investigation of the problem. We seek to quickly generate solutions, and provide for almost immediate updates of the solution when a new customer arrives. For this case we prefer to insert customers into the current solution, rather than completely recreating the solution when a new customer arrives, due to the reduced computational time that is required. We believe that exact techniques (or even time-consuming heuristic techniques) may be inappropriate for these problems, as we are attempting to make almost instantaneous decisions and the computational time requirement for these methods may lead to costly delays of notification.

The first method we consider is termed MAXGAIN, which we describe in Algorithm 9.1. Upon the arrival of a new service request, the current solution route is updated, through attempting to find the best updated solution from the current set

Algorithm 9.1 function MAXGAIN

```
// Initialize the solution route, T
sizetour ← 0, T ← {0}
for each arriving customer i do
    mindist ← LIMIT
    for (j = 1 to sizeroute + 1) do
        c_j ← the extra distance for feasibly inserting i in position j of T
        if (c_j < mindist) then
            best_j ← j, mindist ← c_j
        end
    end
    if (mindist < LIMIT) then
        Insert customer i in position best_j
    else
        Place customer i in a route S = 0–i–0
        Apply STEEPEST ASCENT to routes S and T
        Reject any customer that is in route S
    end
end

end
```

of customers plus the newly available customer. Any customer that is not included in the solution route, is assumed to be permanently excluded, thus incurring the penalty for the delay between when this customer's request arrived and when it was excluded.

This method begins with an empty solution route and the first customer is placed in this route if it is feasible to do so. Initial processing occurs when each new customer arrives, where, if it is possible to insert the customer into the solution route, it is inserted in the position that adds the least time. If not, the customer is put into a new 'non-solution' route, and a version of STEEPEST ASCENT is used to create improvements. If a customer is moved from a non-solution route, into the solution route, the overall effect is to increase the solution value by the value of the moved reward plus the delay penalties that are saved through servicing this customer. If a customer is moved from the solution route, into a non-solution route, the solution value is decreased by the value of the moved reward plus the delay penalties incurred through excluding this customer. The steepest ascent we carry out involves reducing the time of the tour, and then the inter-route routines of Move and General Exchange, to create improvements. For each routine, the improving move that maximizes the change in solution value is carried out.

A second method we consider is BATCHMAXGAIN, which is equivalent to MAXGAIN except that instead of processing each customer immediately, we break the time limit into non-overlapping intervals $[0, T], (T, 2T], \ldots ((k-1)T, kT]$, where $T = \frac{LIMIT}{k}$. Each customer whose request arrives in the interval $((m-1)T, mT]$ will be processed at time mT.

Worst-Case Performance

Our problem consists of maximizing the net reward and we now consider the worst-case performance of our online heuristics, compared with those derived from an offline algorithm. This is similar to competitive analysis (see [19]), where the effectiveness of an online heuristic is defined by finding the maximum ratio, under any input stream, of the objective value obtained by the offline algorithm compared with that for the online algorithm. In our case, the optimal offline algorithm involves solving the static OP version of the problem. This is transformed into a solution for the dynamic problem by assuming that each customer that is not serviced in the OP solution is notified of its non-service at the time the request is

made. Thus, no delay-of-notification penalties are incurred, and the optimal OP solution is identical to the optimal DOP solution.

Suppose we have a scenario where multiple customer requests are generated from one of two locations, each a travel time k from the depot, with the travel time between these locations being $2k$. With a time limit set at $3k$, it is not possible to visit both locations in a solution tour. Suppose that successive customer requests alternate between these locations, and the m'th customer request arrives at time $m\Delta$, with the reward for servicing this customer being equal to $R + m\delta$, with δ diminutively small compared with R. The penalty for delay of notification is assumed equal to a constant η for each customer. The first customer request will be included immediately, and it has a total reward of $R+\delta$, and no delay penalty. The second customer request arrives at time 2Δ, and since it can not be included in the current solution route, it is placed in a new route. The impact of servicing the second customer instead of the first customer, is the new reward less the new penalties, minus the old reward less the old penalties. This is $(R+2\delta-\eta\Delta)-(R+\delta-0)=\delta-\eta\Delta$. For any problem instance where η is less than δ/Δ, the second customer will be serviced while the first customer is excluded.

For each successive customer, the change through excluding the customer that is currently in the vehicle is equal to $\delta-\eta\Delta$, and so the newly arrived customer will be serviced. If there are N customers who request service, the total reward received by this method will be equal to $R+\delta+(N-1)*(\delta-\eta\Delta)$. The optimal solution would be to accept all the customers from one of the locations, and the total reward for this solution would be $\frac{N+1}{2}R+N^2\delta$ if N is odd, and $\frac{N}{2}R+N(N+1)\delta$ if N is even. This is the optimal offline reward value, and as $N\rightarrow\infty$, we find that the worst-case ratio of MAXGAIN to the offline solution, is zero.

We now consider the worst-case performance of the BATCHMAXGAIN heuristic. We consider the situation where there is an isolated customer that arrives at time $(k-1)T+\epsilon$, and whose penalty for delay of notification approaches ∞. If there is no bound on the delay penalties incurred, then we may need to include this customer in the solution, whereas the optimal policy would be to exclude this customer as soon as its request arrives. If this customer is such that no other customers can be included in the service route while enabling this customer to be serviced, and the reward for this customer is zero, then the worst case competitive performance follows. This particular instance could involve the objective function

value becoming negative; if further, there were two customers with large delay penalties, which could not be included together in the same solution, then the worst case would approach $-\infty$.

The advantage of batch processing of customers is that it enables a greater number of customers to be considered when creating the solutions. The disadvantage comes through the delay of notification penalties that have will have accrued already for customers, if they are rejected when they are initially processed. Therefore, this method will not be effective if there are too many customers being evaluated that are unable to included in the solution. This method would be expected to be superior to the single consideration heuristic when the delay penalties are small compared to the reward values, as the effect of the penalties is less than the gains through using more information in the routing.

Computational Results

The preliminary results of our testing are shown in Table 9.1. We generate the arrival streams where we randomly assign each customer to one of the unique locations within the service area. The columns under the label Types give the problem classes we are considering for the DOP. The column DP shows the three different delay rates considered, with these being respectively, 0.5, 0.2 and 0.05 of the reward for each customer. The column headed (A,R), gives respectively the parameter for the arrival rate and the reward variability. The arrival rates are given in decreasing order, i.e., 10.0, 5.0 and 2.5 respectively, with the reward variability also given in decreasing order, i.e, from the distributions $U(10, 100)$ and $U(10, 20)$ respectively. The objective function value we are attempting to maximize is given in the column DOP Obj, while the total reward of the serviced customers is given in column OP Obj, i.e., the difference between the values in columns OP Obj and DOP Obj is the average penalty that is incurred for delayed notification.

The third method, BatchB4B, is the same as BatchMaxGain, except that each customer is considered for inclusion in the current solution according to the bang-for-buck ratio of change in objective function value to change in time. The customer with the maximum ratio is added at each step. The total computational time for each of the methods is approximately 12 minutes (programmed in C, and run under Linux 6.2 on a Pentium II, 400MHz with 128 MB of RAM) to solve the 1800 problem instances given in Table 9.1. Thus the average computational

Types		Methods					
DP	(A,R)	MaxGain		BatchMaxGain		BatchB4B	
		DOP Obj	OP Obj	DOP Obj	OP Obj	DOP Obj	OP Obj
0.5	(1,1)	2082.890	2202.234	1753.493	2402.505	1738.105	2388.630
	(2,1)	1582.413	1646.360	1484.240	1695.621	1485.838	1696.284
	(3,1)	1042.613	1050.752	1035.618	1056.987	1036.018	1056.943
	(1,2)	442.827	448.230	344.016	521.510	350.342	526.152
	(2,2)	380.319	383.144	354.354	404.475	355.243	405.159
	(3,2)	284.156	284.634	279.135	285.405	279.578	285.768
Total		969.203	1002.559	875.143	1061.084	874.187	1059.823
0.2	(1,1)	2342.604	2463.213	2298.841	2594.301	2283.793	2585.116
	(2,1)	1684.028	1743.033	1663.716	1768.738	1664.614	1769.710
	(3,1)	1048.173	1054.860	1047.469	1058.011	1047.712	1058.264
	(1,2)	492.675	504.538	505.827	583.999	502.895	580.682
	(2,2)	400.790	405.859	402.901	425.164	401.813	424.824
	(3,2)	286.107	286.847	286.164	289.049	285.831	288.579
Total		1042.396	1076.392	1034.153	1119.877	1031.110	1117.863
0.05	(1,1)	2598.018	2659.117	2638.938	2733.748	2644.936	2739.430
	(2,1)	1785.102	1810.291	1798.561	1830.926	1795.052	1827.369
	(3,1)	1060.046	1062.960	1064.069	1067.162	1064.405	1067.441
	(1,2)	591.704	606.164	621.328	647.605	621.349	647.868
	(2,2)	430.537	437.146	441.482	450.788	441.926	451.358
	(3,2)	288.793	289.685	289.483	290.818	289.562	290.823
Total		1125.700	1144.227	1142.310	1170.175	1142.872	1170.715

Table 9.1: Results from Preliminary Testing for the DOP

time for a problem instance is well under a second, which meets the quick updates criteria we considered important.

We see from Table 9.1 that, as the delay penalties decrease, the batch processing methods become relatively more effective. We also see that, comparing the OP Obj column values, the batch processing methods result in greater total reward in the vehicle, than is the case for MaxGain. This shows that the delay in order to obtain more information before making routing decisions enables additional total reward to be collected for each of the problem classes. The additional information comes at a cost of the penalties that are incurred if a customer can not be included at the time the evaluation is carried out. We see that for high delay penalties, the gains made through delayed decision making are far less than the cost of the delays, but the relative performance of the two batch heuristics improves as the

delay penalties decrease.

If we compare the effectiveness of the batch methods with the MAXGAIN method by the Wilcoxon Signed Rank Test at a 5% level, we see that the MAXGAIN method is significantly more effective for seven of the first nine problem classes from the table compared to BATCHMAXGAIN and eight of the first nine compared to BATCHB4B. These problem classes have either high delay penalties or high reward variability and medium arrival rates. The two batch methods are both significantly more effective than MAXGAIN for the problem classes with low delay penalties and reward variability and with the two highest arrival rates, with the BATCHB4B method also significantly more effective for the problem class with high reward variability and arrival rate and with the low delay penalties. This shows that there are significant improvements that may be made through delaying the routing decisions for the problems with low delay penalties. With the low reward variability, since the penalties incurred are a function of the available reward for a customer, there are lower delay penalties incurred than with higher reward variability. Therefore the cost of not being able to service a customer, after waiting before attempting to insert it, will not be as great and this explains why the performance of the batch heuristics is in general greater for the lower reward variability.

We now consider in more detail the effectiveness of the techniques, through considering the penalties that are incurred. The difference between the columns DOP Obj and OP Obj is the average penalty incurred for delay of notification. We see that with rates of delay penalties of 0.5 and 0.2, the performance of the MAXGAIN method is adversely affected due to the incurrence of significant penalties. These occur through a customer being provisionally included in the solution when it becomes available, but then being excluded at a later stage. If we restrict the solution method from excluding customers that are already in the solution, then the mean objective value obtained for the six problem classes by MAXGAIN are respectively, 1550.526, 1322.815, 994.059, 393.835, 357.331 and 275.342, with an overall mean of 815.651. The ineffectiveness of this version of MAXGAIN shows that the ability for recourse is an important feature of the method for these problems.

Another detrimental feature of the MAXGAIN method is its insertion of the first customers that can be feasibly included in the solution route, without taking into account the attractiveness of these customers. This has the potential to prevent

future insertions from being made, if an isolated customer is inserted into the route and can not be excluded at a later stage. We only allow the insertion of a new customer if the cost of any customers that are identified for exclusion in order to accommodate the new customer, is less than the additional reward for the insertion. Therefore a distant customer that has high reward is unlikely to be able to be removed, and it has the potential to have a detrimental effect on the solution obtained.

Testing Further Solution Methods

The significance of the delay penalties for the MAXGAIN method shows that potential improvements may be made through implementing a method that reduces these penalties. The negative impact of a 'poor' customer that cannot be excluded at a later stage is also an important occurrence that we are attempting to diminish. We therefore propose a method that selects whether or not to accept the customer, according to some criterion other than just whether the customer can be profitably inserted into the current vehicle route. We consider both the reward value and the insertion time, and create two new solution methods which attempt to reduce the penalties incurred. We found that, through rejecting few of the customers, we obtain similar results to using MAXGAIN, but, by effectively rejecting the poor customers, we are able to obtain improved solutions, particularly for problems where the demand is high.

The method REJECTB4B involves the calculation of the bang-for-buck ratio of reward to minimum insertion time for adding the customer to the current route. If this ratio is less than a given threshold, we reject the customer from consideration. If the ratio is acceptable, we insert the customer into the current solution, if this is feasible, or we place the customer in its own vehicle, and carry out inter-route improvements. For the calculation of the bang-for-buck ratio, we scale the rewards so they are in the range $(0, 1)$, as with uniformly distributed data, and this gives the expected percentiles which we can use to determine the probability that this reward value will be exceeded. This method is thus a version of MAXGAIN, except we pre-process the customers to immediately reject those which are unlikely to be beneficial.

Of note is that, as the search procedure continues through the insertion of additional customers, the expected insertion time will reduce and therefore the

threshold ratio becomes effectively lower as more customers are included. In this manner, the probability of rejecting a customer will be reduced for later customers, which appears to be a sensible approach since the effect of including a poor customer is greater at the start of the time period. We found that the method was improved through initializing the solution route in a different manner, namely including the first customer if its insertion time is less than 1.5 (which is an estimate of the average insertion time) or its scaled reward value exceeds 0.9. In this manner we reject the worst customers that will be unlikely to feature in the optimal solution.

The method REJECTREW involves the customer being rejected from consideration if its reward value is too low and the insertion time is too high. If the customer is acceptable we insert the customer in its best position in the solution route, if this is feasible, or we place it in its own vehicle and carry out the inter-route improvements. We again used the scaled reward values, and found that our methods are more successful if the threshold reward value is decreased as time increases, as this again creates stricter criteria early on when the inserted customer has greater influence. For the acceptable insertion time, we estimate the average insertion time, and accept any customer whose insertion time is less than this average insertion time. We use the result of Beardwood, Halton and Hammersley [11], which states that the asymptotic expected distance (and thus time, since we assume that the service vehicle travels at unit speed) of a TSP tour through n points is $\beta_{TSP}\sqrt{na}$, where a is the area of the region where the points are placed and β_{TSP} is a constant, which has been estimated to be approximately equal to 0.72 (although we use a slightly weaker estimate of 0.75, which allows greater numbers of customers to be considered). We thereby estimate the expected insertion time for adding a point, when there are k points already in the solution, to be $1.5 * (\sqrt{k+1} - \sqrt{k})$, since the area we are considering is 4.

For the REJECTB4B and REJECTREW methods we test the effectiveness of the heuristics for a number of different values of the threshold bang-for-buck ratio and initial required reward respectively. The results of this testing are shown in Figure 9.1, where we test the bang-for-buck ratio for the values of 0.2 up to 1.0, and we test the values of the initial required reward of 0.4 up to 0.95. The solution for the parameter value of 0.0, is that obtained by the MAXGAIN method, as this is the version of these two heuristics when all customers are considered for insertion. These plots, and ones that we will consider in the following sections, show the

mean solution value for each problem class for the given value of the parameter for the heuristic. The lines on the plot are found through interpolation between the values obtained from the tested parameters. For each graph the lines 1, 2 and 3 refer to the high reward variability with the penalties being respectively 0.5, 0.2 and 0.05. Lines 4, 5 and 6 refer to the low reward variability with these penalties. The top graph is with the highest delay penalties, the second is with medium delay penalties and the bottom graph is with the low delay penalties.

From the graphs we see that there appear to be substantial gains that may be made through incorporating the rejection criteria given, particularly with high arrival rates. With low arrival rates, the mean objective value appears to decrease consistently as the parameter values increase and therefore greater numbers of customers are rejected from consideration. This result is expected, as with low arrival rates almost all customers can be included in the solution and so the rejection of customers is unnecessary and therefore leads to poorer solutions. We also see that as the rates of delay penalties decrease, the parameter value at which the maximum mean objective value is obtained also decreases. Therefore, for high delay penalties, it is beneficial to reject more customers, as the delay penalties make it more difficult to reject these customers at a later time.

Since, using this approach, no gains seem possible with low arrival rates, we will concentrate on the problems with the two higher arrival rates. We identify the parameter setting for each of the RejectB4B and RejectRew methods that give the highest overall mean objective value with the two different arrival rates. Since better results are obtained through rejecting greater numbers of customers when the delay penalties are higher, what we are in fact doing here is averaging out the effect of the penalties. This obtains the most robust of the methods, but is unlikely to be as effective as a method that is tailored for the specific penalties.

Of the values we tested, the highest mean for RejectB4B with high arrival rates is with a threshold ratio of 0.5 (method B4B.5) and for medium arrival rates the maximum is for the ratio of 0.3 (method B4B.3). For the RejectRew method the highest mean for high arrival rates is with initial scaled reward required of 0.8 (method Rew.8), while with medium arrival rates the maximum is with reward of 0.6 (method Rew.6). Not surprisingly, for high arrival rates the most successful parameter settings involve rejecting greater numbers of customers than for the most effective values for the medium arrival rates.

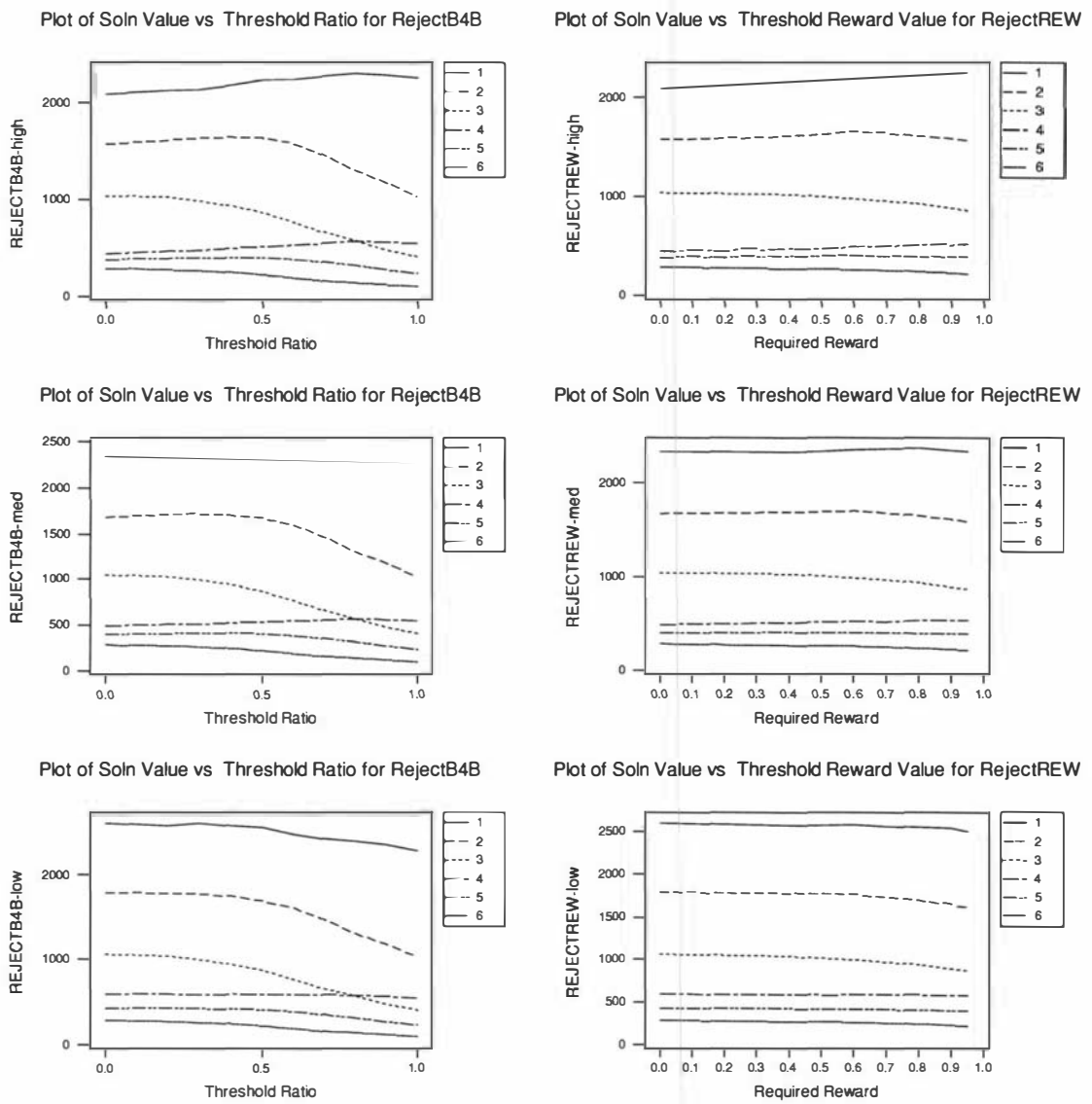

Figure 9.1: Testing Parameter Values for Methods for the DOP

We also consider an additional method, which relies solely on the times considered, and we call this method TSP. We note that the form of a solution to the DOP will consist of the TSP tour over the selected customers, and our method thus operates through attempting to create a rejection criterion so that the number of customers serviced is maximized. We select a square of a given size, that is centred on the centre of the area over which the customer requests may appear. Since we have customers uniformly distributed over the space, we note that the expected number of customers within a given area a of total area A, will be equal to $\frac{\lambda aT}{A}$ where λ is the customer arrival rate and T is the length of time over which customer requests may arrive (and also the limit on the duration of the service vehicle's route).

We again rely on the asymptotic results obtained by Beardwood et al. [11] to motivate our method. Via their result, we see that the expected length of a tour to service n customers will be $\beta_{TSP}\sqrt{na}$. For the customers arriving in area a, and with the depot included, we will expect to have $\frac{\lambda aT}{A}+1$ points in a tour. Therefore we substitute the expected number of points in the TSP tour into the equation for the length of TSP tour, in order to obtain the appropriate area so that the average number of customers within this area will equal the number of customers that can be serviced. Rearranging the asymptotic tour length formula, we find:

$$a = \frac{1}{n}(\frac{T}{\beta_{TSP}})^2$$

Using the result

$$n = \frac{\lambda aT}{A} + 1$$

we substitute and find:

$$a = \frac{-A\beta_{TSP} + \sqrt{(A\beta_{TSP})^2 + 4\lambda T^3 A}}{2\lambda T \beta_{TSP}}$$

Therefore, our policy is to accept all customers within a square with sides of length $\sqrt{a}$, centred on the depot. For the test problems we are considering, T is equal to 8.0 and A is 4.0. For $\lambda = 10.0$, we find $a = 2.46$; for $\lambda = 5.0$, we find $a = 3.46$; and for $\lambda = 2.5$ we find $a = 4.87$. Since the original region has area 4.0, our policy with $\lambda = 2.5$ is to accept all customers, and so this method is equivalent to MAXGAIN for these problems.

The results from this testing are shown in Table 9.2, with the results for MAXGAIN and BATCHB4B (given as MAXG and BB4B respectively) duplicated from

Types		Methods							
DP	(A,R)	REW.8	REW.6	B4B.5	B4B.3	MAXG	BB4B	TSP	OPS
0.5	(1,1)	**2220.9**	**2217.9**	**2224.0**	2134.6	2082.9	1738.1	**2396.3**	-342.7
	(2,1)	1622.8	**1663.1**	**1642.4**	**1645.2**	1582.4	1485.8	**1644.9**	1054.7
	(3,1)	930.3	982.3	864.5	988.2	1042.6	1036.0	1042.6	966.8
	(1,2)	**513.7**	**488.9**	**509.3**	**473.7**	442.83	350.3	**593.3**	-147.8
	(2,2)	393.7	397.8	398.4	397.4	380.3	355.2	**410.7**	254.5
	(3,2)	238.4	260.0	220.9	261.6	284.2	279.6	284.2	260.9
0.2	(1,1)	2377.5	2363.1	2397.2	2345.6	2342.6	2283.8	**2469.1**	1565.6
	(2,1)	1662.0	1711.8	1669.4	1719.6	1684.0	1664.6	1700.8	1540.3
	(3,1)	932.7	986.2	864.5	994.2	1048.2	1047.7	1048.2	1021.2
	(1,2)	**532.9**	**523.9**	**533.2**	**512.7**	492.7	502.9	**610.0**	375.5
	(2,2)	394.3	402.0	402.5	408.4	400.8	401.8	**418.5**	384.6
	(3,2)	238.5	260.6	220.9	262.0	286.1	285.8	286.1	278.1
0.05	(1,1)	2551.1	2576.1	2543.0	2588.6	2598.0	**2644.9**	2525.8	2533.5
	(2,1)	1692.1	1759.1	1686.0	1774.2	1785.1	1795.1	1745.5	1783.7
	(3,1)	932.9	991.4	864.5	998.5	1060.1	1064.4	1060.1	1052.6
	(1,2)	580.9	581.1	590.9	585.5	591.7	**621.3**	**635.9**	**643.1**
	(2,2)	404.4	418.5	410.0	423.3	430.5	**441.9**	435.0	**454.1**
	(3,2)	238.7	261.5	220.9	262.5	288.8	289.6	288.8	287.4
Total		1025.4	1047.0	1014.6	1043.1	1045.8	1016.1	1088.7	775.9

Table 9.2: Testing New Methods for the DOP

Table 9.1. The method OPS involves finding a solution to the static OP at the end of the time limit and incurring the appropriate penalty for each customer that isn't included in the OP solution. For this problem, we see that there is a constant penalty that is incurred for excluding a customer from the solution route, which is equal to the penalty that would be incurred for excluding the customer at the time when the routing decisions are made. Therefore, in our static solution method, we add these penalties to the reward values, and continue by using a method that appends customers to the current route according to their aggregate score found by the subgravity measure, with the solution obtained being improved via the application STEEPEST ASCENT. We test the significance of differences between the objective function values obtained by different methods, through using the non-parametric Wilcoxon Signed Rank Test, with numbers in bold indicating that at a 5% level (with seven comparisons made) the method is superior to MAXGAIN for this problem class.

From Table 9.2 we see that the TSP method obtains the highest overall mean objective value, and is the most effective method for many of the problem classes considered. This method proves superior to MAXGAIN for eight of the 12 problem classes for which the implementations are different, i.e., for A < 3, whereas MAXGAIN is only superior for two problem classes, with these both featuring low delay penalties. This method obtains an overall mean that is far higher than the means for the other methods, and as such, it appears to be the best method for the problem classes over which were tested in this experiment.

Of the other methods, we see that the implementations of REJECTB4B and REJECTREW are consistently superior to MAXGAIN for problems with high penalties, although not with low arrival rates. These methods are also relatively effective with medium delay penalties, apart from with low arrival rates, although the methods are only significantly more effective for the problem class with A = 1 and R = 2. With low delay penalties these methods are not effective, as it appears that fewer customers should be rejected as the low penalties enable effective substitution of customers to take place, which reduces the need to exclude the undesirable customers off-hand. For these low delay penalties, the methods that delay the scheduling of the customers, i.e., BATCHB4B and OPS become more effective. The OPS method is particularly effective with low variability in the rewards as, in this case, the objective of this method can focus on the reward collected. With the higher reward variability, the selection of customers becomes more important, and fewer customers are serviced than with lower reward variability, where the routing of the customers is the more important consideration. Thus with fewer customers served, there are higher penalties incurred, and this reduces the effectiveness of the static method.

We now consider the way the actual solution is obtained, by studying an, intentionally extreme, case where the use of the rejection criteria obtains a very different result from that obtained via the MAXGAIN method. The problem instance is our test problem number 3, which has A = 1, R = 1 and DP = 0.5, and there are 89 customers who request service. We will show the solutions obtained for this test problem by three of our methods, with the symbols for the plots of the solution

being given by:

- $*$ if the customer is included in the final solution
- $\diamond$ if the customer is not able to improve the current solution, so is rejected
- Δ if the customer is included in the solution, but is rejected at a later stage, thereby incurring a penalty
- $\circ$ if the customer is not considered for insertion

We give the solution obtained by the MAXGAIN method in Figure 9.2, where the reference number given for each customer is the order in which the customer arrives. The objective value obtained by this method is 1814.38. We see that this method inserts the first 16 customers that become available, with three of these (customers 11, 12 and 14) being removed from the solution route at a later stage. We also see that very few of the later customers are able to be included in the solution due to the route that has already been created, with none of the last 16 customers being able to be included. The solution route is quite unnatural, as it involves a number of long links between customers and therefore few of the available customers are serviced (a total of 31 of the 89 customers).

We give the solution obtained by the B4B.5 method in Figure 9.3, and the objective function value for this solution is 2706.58. This method immediately rejects the first three customers, with the distant fourth customer accepted due to its reward value of 93.723. The inclusion of this customer results in a much lower total objective value than might otherwise be obtained. Fortunately the following customer (customer 5) is able to be inserted and it is located near the depot, which enables subsequent insertions to be made. With this method, there is a real possibility of this isolated customer leading to many of the following customers being rejected, due to having too high an insertion time to allow the bang-for-buck ratio to be sufficient to allow the insertion to be considered. This method appears that it might be too 'harsh', in that some customers with small insertion time are excluded from consideration. For example customer 72, situated in location (0.5219,1.0356), is not considered (due to its reward only being 16.9219 (scaled reward of 0.0769)), even though the insertion time is low. Since this customer is one of the later ones to request service, including it would not unduly affect the routing as there are few customers that subsequently request service. This suggests that the use of a rejection criterion that accepts a greater number of customers as the time evolves would be more effective for this particular problem. We therefore

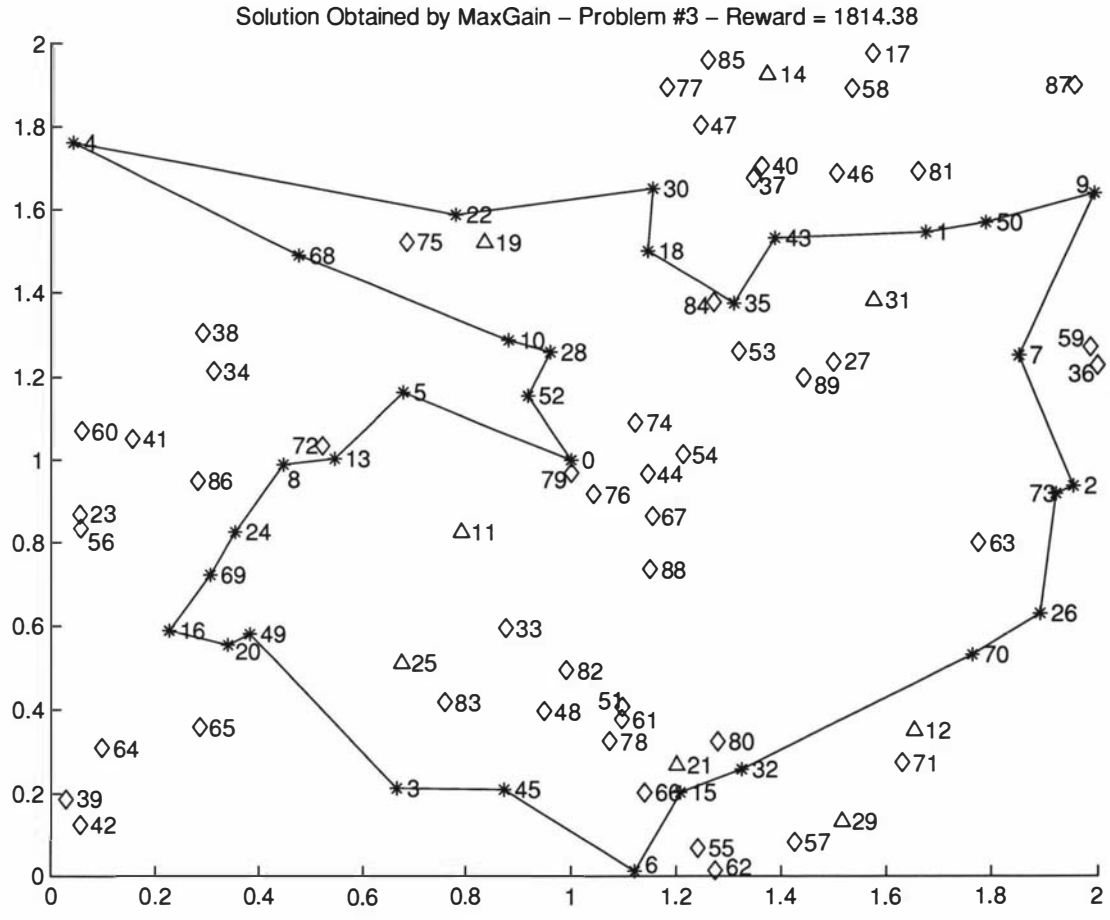

Figure 9.2: DOP Solution found by MAXGAIN for Problem 3

developed a version of the REJECTB4B method, where the required bang-for-buck ratio decreases as the time goes on. However, for the full set of problems we found that there was no significant difference between the overall effectiveness of the best parameter settings with reducing threshold ratio and constant ratio, and we therefore do not consider this implementation further.

We give the solution obtained by the TSP method in Figure 9.4, and this obtains a total objective value of 2890.85. We see from this solution that the outlying customer 4, which caused trouble for the other two methods considered, is excluded from consideration by the TSP method. The method inserts most of the customers within the central section, with four inserted customers being removed later, and seven eligible customers being unable to be included in the solution route. The problem appears to be well-solved by this method, with superior results due to the form of the data suiting it.

Although the TSP method is the most effective overall method for most of the

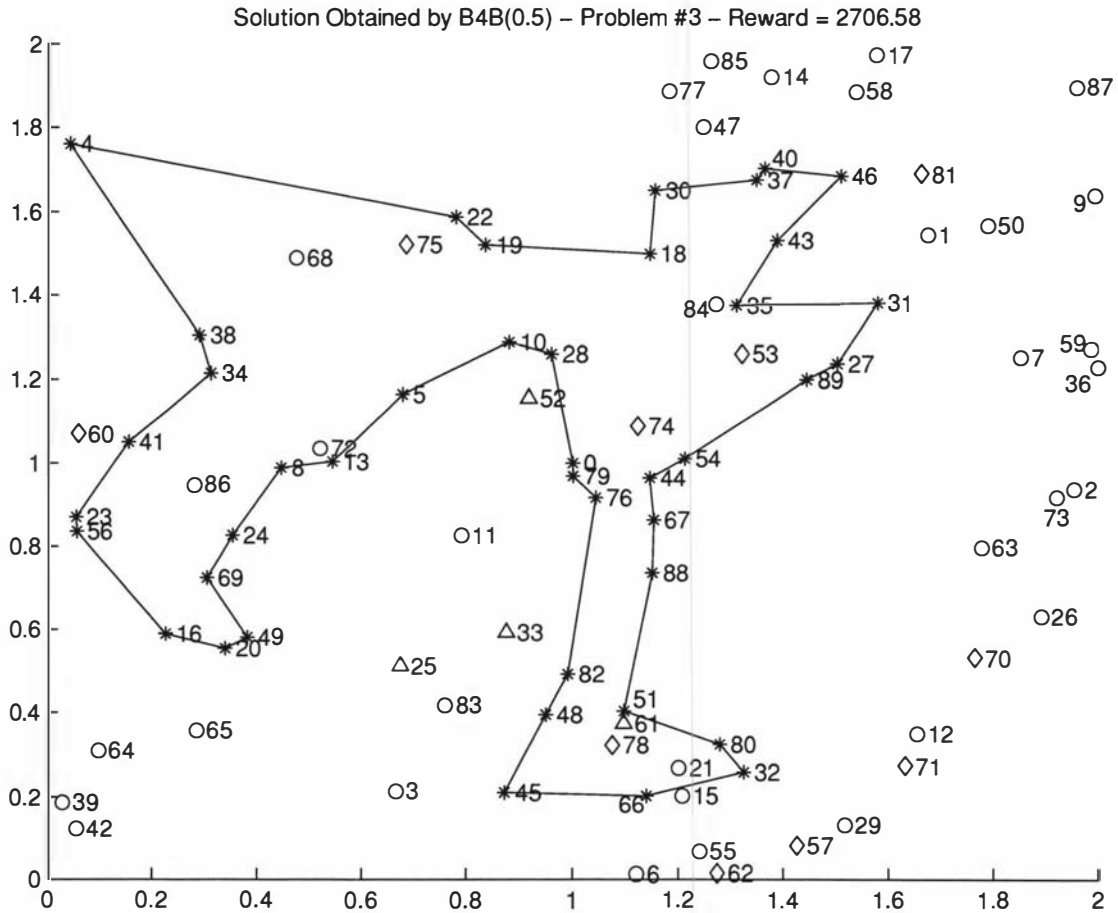

Figure 9.3: DOP Solution found by B4B.5 for Problem 3

problem classes that we tested, the lack of concern with rewards when evaluating a customer's eligibility, will lead to this method obtaining poor solutions for a number of problem instances. We show one such instance in Figures 9.5 and 9.6, which is for problem 60, which has A = 1, R = 1 and DP = 0.5. In Figure 9.5 we depict the solution obtained by the B4B.5 method, with the label of each customer giving the order in which the customer's request for service arrives. Figure 9.6 depicts the solution obtained by the TSP method, with the label of each customer giving the customer's reward. The objective value obtained by B4B.5 is 2872.04 while the TSP method obtains an objective value of 2311.70. The difference between the methods is that the B4B.5 method is able to service the sets of customers in the bottom right hand corner and at the top, toward the right hand edge, of the solution area, while these customers are not considered for inclusion by the TSP method. Since there are high rewards for customers in these sets, the B4B.5 method obtains a much greater objective value for this problem instance.

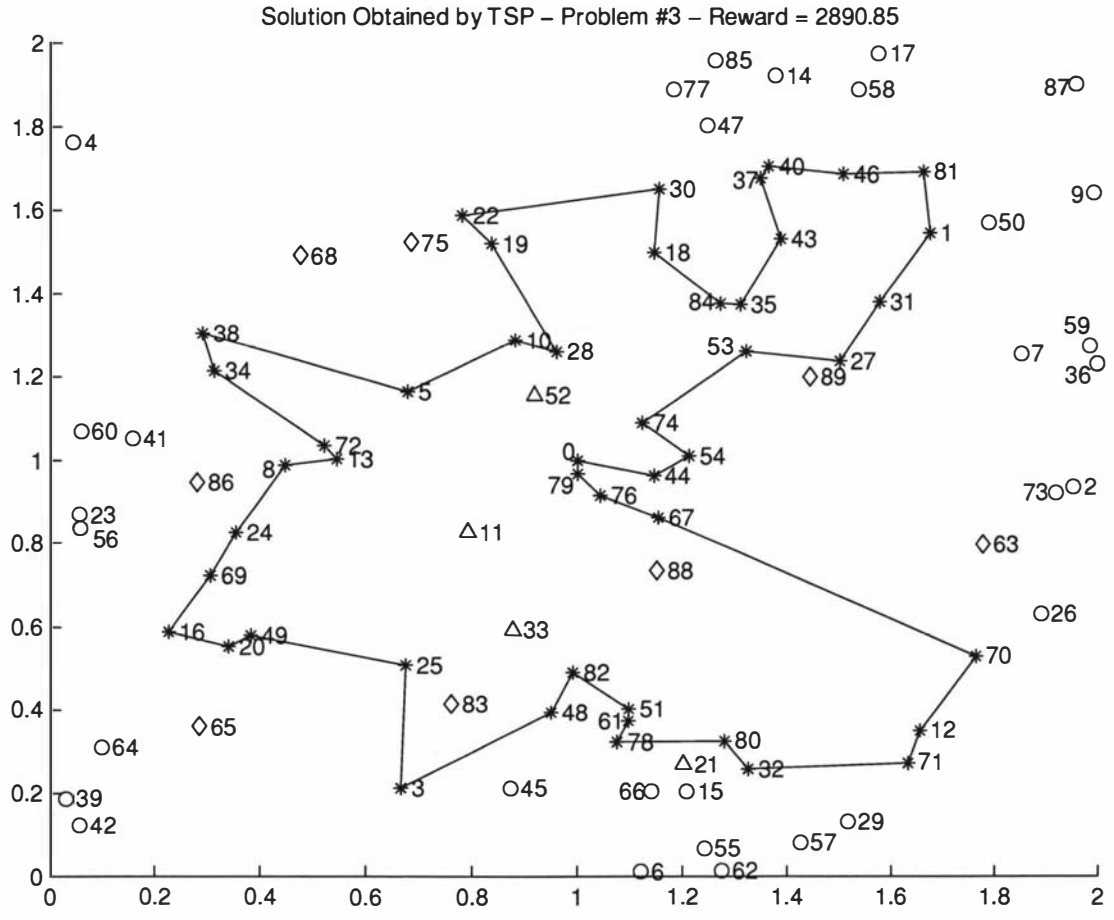

Figure 9.4: DOP Solution found by TSP for Problem 3

We now consider the first set of problems, i.e, those with high reward variability, high arrival rates and high delay penalties. For these problems there are 80.1 customers on average available for servicing, and the methods MAXGAIN and BATCHB4B both allow each of these customers to be considered for insertion into the solution route. The average performance of our other heuristics for these problems are given in Table 9.3. We see that the TSP method services by far the greatest number of customers, and obtains the highest overall mean while incurring the fewest penalties. Therefore it appears that the acceptance criteria of this method is the most appropriate for this set of problems, as it considers all of the customers that are closest to the origin, which enables greater numbers of customers to be serviced and it allows few customers whose inclusion will adversely affect the allowable remaining routing.

Considering the remaining methods, which take into account both the reward and time considerations, the method with the highest mean objective value is

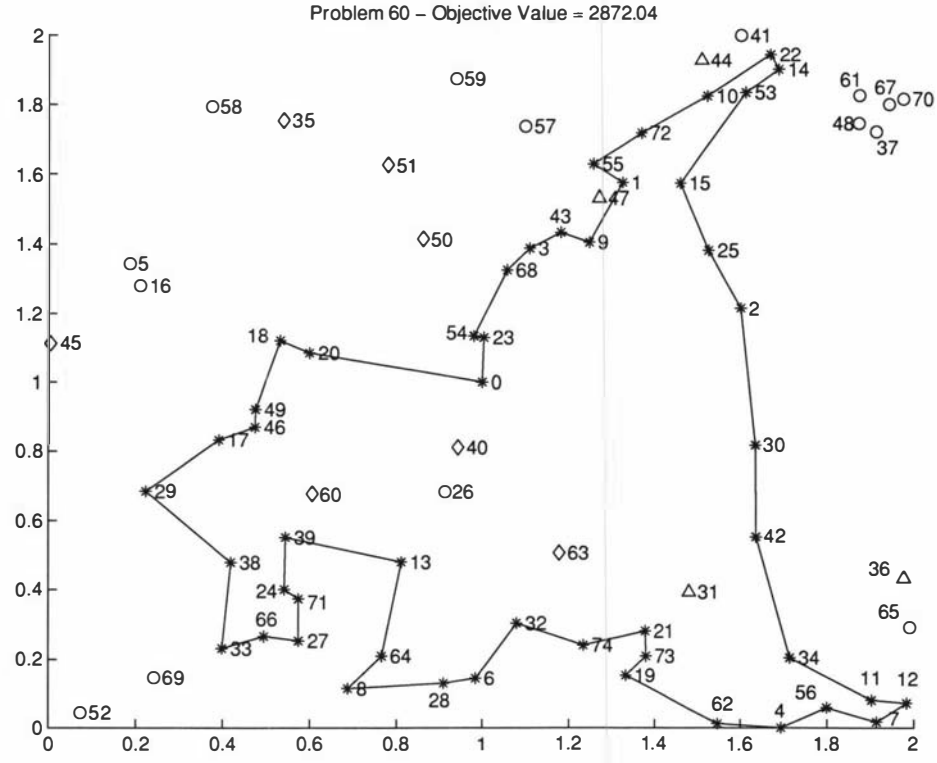

Figure 9.5: DOP Solution found by B4B.5 for Problem 60

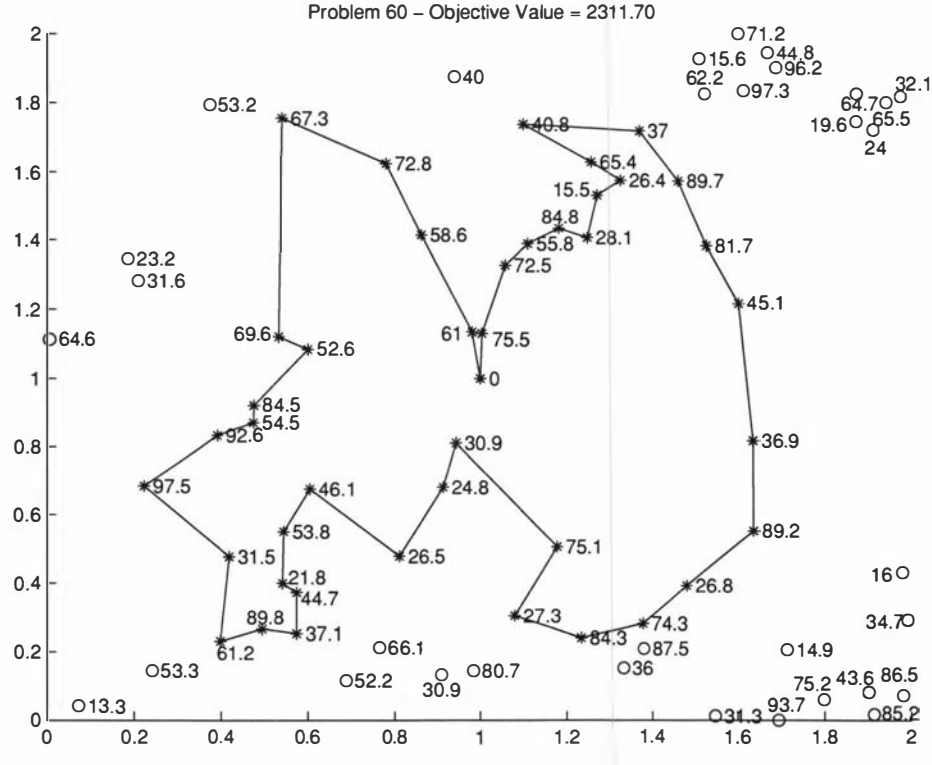

Figure 9.6: DOP Solution found by TSP for Problem 60

Method	Custs Served	Custs Rejected	Mean Objective	Mean Penalties
REW.8	33.98	18.86	2220.891	75.431
REW.6	34.22	13.77	2217.916	88.859
B4B.5	34.87	32.68	2224.016	71.217
B4B.3	33.65	20.66	2134.634	92.102
TSP	41.40	30.66	2396.278	62.200

Table 9.3: Average Performance for first Problem Class

B4B.5, and this method rejects the greatest number of customers, without considering them for inclusion in the current solution. This results in the solution incurring fewer delay penalties than these other methods, which leads to the method obtaining the highest mean, although the two implementations of REJECTREW both collect a higher total reward from the customers. These two REJECTREW implementations allow greater numbers of customers to be tested for insertion than the REJECTB4B methods, so they result in a greater number of provisionally included customers being excluded during the routing phase.

Testing the Robustness of the Methods

The previous results have shown that a solution method which only considers the customers nearest the depot, has been the most effective for a number of problem classes where the rewards and locations are uniformly distributed. We have also seen that methods which reject customers according to both reward and time criteria, may improve upon methods that consider all customers. These methods work through enabling later customers (which may be more profitable) to be included. We now consider a set of special problems, upon which to test the effectiveness of the heuristics under some different, non-uniform conditions.

The conditions that we consider are:

- the customers come from a distribution where the successive customers are far from each other. To generate these problems, for each set of available locations we randomly select a customer to be the first selected, and then randomly select one of the five farthest locations to be the location of the succeeding customer

- the rewards are an increasing function of the time at which a customer's

request for service is made known. The function we use in this case is $R(i) = 100 * (t_i/LIMIT)$, where t_i is the time at which the i'th customer requests service

- the rewards are a decreasing function of the time of request, in which case we use $R(i) = 100 * (LIMIT - t_i)/LIMIT$

We create four sets of problems by using two methods for generating the locations (the 'farthest' method from above, and random generation) and, for each of these, using the two methods for generating the rewards. For each of these methods we create 100 problem instances, and for each of these we create three problem instances by varying the delay penalties at three different levels, i.e., 0.5, 0.2 and 0.05 of the customer's reward. We use the same methods that were used in Table 9.2, and the results are given in Table 9.4. The problem types are given in the form (M, A), where M corresponds to the method of generating the customers, i.e., for $M = 1$ and 2, the streams of customers are randomly distributed, and they are placed according to the farthest criterion for $M = 3$ and 4. For $M = 1$ and 3, the rewards are increasing in the arrival time and with $M = 3$ and 4 the rewards are decreasing.

Of note is that the methods which only consider single insertion, i.e., all the dynamic heuristics apart from BB4B, will not reject any customer for problems where the rewards are strictly decreasing. These methods will reject a customer that has been placed in the solution route through swapping in a new customer, and can do so only if the new customer's reward exceeds the reward of the excluded customer. Therefore there are no delay penalties incurred by these methods for any of the problem classes with $M = 1$ or 3, and so the solution obtained is identical, regardless of the value of the delay penalty. Accordingly, we only include in Table 9.4 the results with decreasing rewards for the lowest delay penalties. The results for BB4B and OPS are lower than the given values when there are higher penalties, and since the values are lower than those for the static heuristics, we exclude these results from the table.

The most striking feature of the table, is the poor performance of the TSP method for problems where the arrival stream is from farthest locations. In these problems, the locations are biased towards the outside of the area available for servicing, and this method will reject many of the customers in this case. These results show a major weakness of this method; that it is unable to cope with

Types		Methods							
DP	(M, A)	REW.8	REW.6	B4B.5	B4B.3	MAXG	BB4B	TSP	OPS
0.5	(2,1)	1721.8	1621.7	1861.9	1574.9	1551.1	1490.8	2019.2	460.7
	(2,2)	1450.6	1395.4	1449.4	1407.2	1304.4	1294.6	1366.1	1072.6
	(2,3)	821.2	860.9	721.7	860.2	909.2	913.0	909.2	876.2
	(4,1)	2076.5	1937.3	2207.7	2205.4	1996.9	1761.7	1096.6	945.1
	(4,2)	1442.5	1419.0	942.3	1411.2	1384.4	1269.5	1182.4	1001.9
	(4,3)	781.0	827.4	397.3	701.6	923.5	911.0	923.5	860.3
0.2	(2,1)	2102.6	2081.6	2135.7	2129.9	2107.4	2122.4	2157.9	1735.8
	(2,2)	1506.2	1515.3	1469.9	1544.5	1523.8	1512.8	1505.8	1435.1
	(2,3)	819.9	860.3	721.7	861.0	932.9	934.4	932.9	920.0
	(4,1)	2482.5	2500.5	2264.5	2464.7	2397.9	2326.3	1096.6	2095.3
	(4,2)	1519.5	1564.1	941.7	1422.5	1468.0	1418.5	1219.9	1404.9
	(4,3)	791.4	843.3	397.3	701.6	932.9	926.5	932.9	913.2
0.05	(1,1)	2179.6	2179.6	2195.5	2179.2	2179.6	2416.5	2354.6	2388.0
	(1,2)	1632.0	1632.0	1637.5	1636.4	1632.0	1665.8	1623.6	1631.1
	(1,3)	977.4	977.4	897.4	968.9	977.4	977.9	977.4	963.3
	(2,1)	2362.2	2368.3	2302.6	2406.1	2400.6	2444.7	2232.5	2363.3
	(2,2)	1543.3	1587.2	1489.0	1610.9	1621.3	1624.3	1569.9	1623.1
	(2,3)	821.2	862.5	721.7	862.9	944.8	945.5	944.8	937.9
	(3,1)	2610.4	2610.4	2147.3	2613.2	2610.4	2640.4	584.2	2437.6
	(3,2)	1776.9	1776.9	872.8	1772.1	1777.0	1768.7	1188.9	1700.7
	(3,3)	1022.3	1022.3	476.5	1012.0	1022.3	1021.0	1022.3	1019.2
	(4,1)	2700.2	2730.7	2333.3	2654.9	2603.4	2626.1	1096.6	2700.4
	(4,2)	1551.4	1611.7	942.8	1442.2	1512.7	1515.0	1242.7	1624.2
	(4,3)	795.9	851.2	397.3	703.0	933.3	935.4	933.3	937.4

Table 9.4: Performance on Special DOP Data

problems where the customer locations are not uniformly distributed. This method works through attempting to include as many of the customers that are close to the depot as possible, and, if few customers are placed within the acceptance region, then this method will fail.

Table 9.4 shows that again the MAXGAIN method is, in general, the most effective method with low arrival rates. The BB4B and OPS methods are competitive for problems with low delay penalties, with these two methods being far more effective than the single insertion methods for the problems with high arrival rate, decreasing reward and the stream of customer locations being randomly generated. For these problems, these two methods are able to include far greater numbers of customers, i.e., there are on average 33.69 and 47.71 customers in the solution

routes for BB4B and OPS respectively, whereas REW.8, REW.6 and MAXG include on average 28.46 customers and B4B.5 and B4B.3 include on average 28.74 and 28.44 customers respectively. The major shortcoming for these methods is therefore their inclusion of all customers immediately, and their inability to insert later customers since the additional reward received will never exceed the reward that is lost through removing a customer.

The other main difference between the behaviour of the heuristics on the problem classes is that, with the successive customers arriving from distant locations, the B4B.5 method is ineffective, particularly for problem classes where the arrival rates and/or the delay penalties are low. In these cases, this method rejects too many customers from consideration according to their insertion ratio, and so the numbers of customers that are serviced is much lower than for the other methods.

We now consider how these methods operate, and we look at one measure of the performance of the heuristics, namely the quality of service that is offered. The measure that we consider is the number of customers excluded from service at such a time that the penalty incurred exceeds the available reward for the customer. The BB4B method services three customers in this manner for the problem class with high arrival rates and penalties, and with decreasing rewards and the customer order being randomly generated. None of the other methods obtain any such customers with reward decreasing, and it is not possible for the penalty to exceed the available reward with the low penalty rates, so we focus on the problem classes with increasing rewards and with the two higher penalty rates. In Table 9.5 we give the average number of customers with penalty incurred exceeding the available reward for each of the dynamic methods considered above.

The Table shows that there are far greater numbers of customers rejected from the solution route by the MAXG and BB4B methods, such that the penalty incurred exceeds the reward that would be available, than for the other methods considered. The TSP method also tends to reject a relatively large number of customers with high penalties, although with high arrival rates and the arrival stream having successive customers from distant locations, this method considers too few customers to require any customers to be rejected. This we consider to be a quite serious shortcoming of these methods, as they invoke such poor service for so many customers, and this is likely to start incurring loss of goodwill. If we were to restrict the time at which a customer may be excluded so that the penalties

Prob Types		Methods						
DP	(M, A)	REW.8	REW.6	B4B.5	B4B.3	MAXG	BB4B	TSP
0.5	(2,1)	3.33	3.74	1.63	3.72	12.57	13.48	4.87
	(2,2)	0.78	1.20	0.22	1.24	5.96	5.50	4.03
	(2,3)	0.02	0.02	0.00	0.03	0.54	0.46	0.54
0.5	(4,1)	0.45	0.46	0.16	0.45	11.90	10.81	0.00
	(4,2)	0.14	0.16	0.00	0.11	2.41	2.16	1.36
	(4,3)	0.02	0.04	0.00	0.00	0.25	0.19	0.25
0.2	(2,1)	2.30	1.82	1.78	1.70	3.29	4.82	4.00
	(2,2)	0.84	1.20	0.27	1.18	2.90	2.95	2.55
	(2,3)	0.02	0.02	0.00	0.02	0.40	0.36	0.40
0.2	(4,1)	0.43	0.33	0.16	0.46	5.42	5.92	0.00
	(4,2)	0.14	0.17	0.00	0.12	1.85	1.85	1.09
	(4,3)	0.01	0.03	0.00	0.00	0.21	0.13	0.21

Table 9.5: Average Number of Customers with Penalty Exceeding Reward

incurred are less than the available reward, which would seem to be a practical approach to guaranteeing reasonable service, then this would seriously affect the effectiveness of these methods. In this case, stricter initial rejection criteria would be required, in order to prevent the inclusion of these customers that would prove to be detrimental to the quality of solution obtained.

Table 9.5 shows the effect of the rejection criteria on the quality of the customer service that is afforded, where the rejection of customers according to both time and reward measures leads to fewer customers being excluded such that their penalty incurred exceeds their available reward. Therefore it appears that, for these problems, the methods which are able to reject customers before attempting to insert them into the solutions will provide better customer service. These methods will thus be the more appropriate methods if the customer service is an important factor.

We consider one further set of problems, where in this case the reward of the customers is equal to a constant multiple of the travel time from the customer location to the depot. This is a possible practical situation, where the reward received depends upon the difficulty of service or the inconvenience to the server.

We use the same data set with uniform locations, previously considered for the DOP, and we create two problem sets through setting the reward received at two different levels, i.e., equal to 10 and 50 times the travel time from the depot.

Types		Methods						
DP	(R, A)	REW.8	REW.6	B4B.5	B4B.3	MAXG	BB4B	TSP
0.5	(1,1)	225.47	230.97	227.51	226.76	226.76	-351.68	241.23
	(1,2)	185.34	197.24	179.86	192.74	192.74	18.72	198.98
	(1,3)	124.64	137.04	99.63	139.55	139.55	124.06	139.55
	(2,1)	1307.70	1347.77	1138.63	1343.85	1343.85	1295.81	1246.74
	(2,2)	973.18	1025.58	893.73	1033.88	1033.88	1015.97	1011.82
	(2,3)	656.53	711.57	522.00	733.44	733.44	729.75	733.44
0.2	(1,1)	229.81	233.07	228.15	228.77	228.77	18.38	240.95
	(1,2)	186.00	198.38	178.75	195.30	195.30	129.77	199.88
	(1,3)	124.90	137.11	99.59	139.60	139.60	134.46	139.60
	(2,1)	1445.12	1526.01	1181.72	1529.32	1529.32	1539.18	1298.12
	(2,2)	1006.19	1077.75	901.42	1096.23	1096.23	1103.16	1044.16
	(2,3)	657.32	718.06	522.11	740.29	740.29	738.00	740.29
0.05	(1,1)	252.28	260.05	234.55	260.05	260.05	222.38	248.74
	(1,2)	192.46	206.25	180.91	206.99	206.99	196.51	205.76
	(1,3)	125.00	137.66	99.58	140.75	140.75	140.50	140.75
	(2,1)	1544.72	1616.76	1218.52	1630.06	1630.06	1680.11	1327.78
	(2,2)	1034.04	1116.32	914.58	1135.01	1135.01	1152.97	1072.72
	(2,3)	662.81	723.91	522.30	747.66	747.66	747.29	747.66
Total		607.42	644.53	519.08	651.13	651.13	590.85	609.90

Table 9.6: Performance on DOP Data with Rewards Based on Travel Times

The results of our dynamic heuristics are shown in Table 9.6, where for the column Types, R refers to the reward type and A refers to the arrival rate. For this problem, we see that the B4B.3 method obtains identical solutions to those obtained by MAXGAIN, as this method rejects very few customers, and those that are rejected are unable to be included in the solution. The B4B.5 method is particularly ineffective with this method, which suggests that the bang-for-buck ratio, using the scaled rewards, is difficult to effectively apply to this type of problem. This is due to the non-uniformity of the rewards present, as the travel time from the depot does not follow a uniform distribution, which results in the rejection criteria not adequately reflecting the data present.

We see that the MAXGAIN method, and therefore also the B4B.3 method, are the most effective overall methods in this case. The BATCHB4B method is effective for problems with high reward variability, particularly with low delay penalties. The TSP method, conversely, is more effective with low reward variability, particularly with high delay penalties. With low reward variability, the location of the

customers is more important than the rewards, and this method will therefore be effective through including as many customers as possible from within the central region. With low delay penalties, the advantage is greater, as it is more important to select the customers that it is desirable to include in the final solution.

We see that the TSP method is ineffective with the high reward variability. In these cases, with higher rewards for the 'distant' customers that this method rejects from consideration, the method restricts itself to only considering the customers with low reward values. This method therefore requires uniformity of rewards or low reward variability, as the rewards are not considered within the rejection criterion, and so it is most effective for problems where the routing over-rides the selection of which customers to service.

The TSP method therefore is not robust, as it relies on certain forms of problem structure for it to be successfully applicable. The exclusion of customers with high rewards, on the basis of their location alone, is likely to result in poor solutions, and one could easily obtain problem instances for which this method will be very ineffective, e.g., if all customers are placed towards the outside of the service region, no customers will be serviced. Also, the fact that a customer with a given location, will not be serviced by this method regardless of its potential effect, could lead to some serious practical difficulties, which would likely render this method unworkable in practise.

We have considered in this section some cases of the DOP and have found solution methods that are suited to particular problem types. Since the problem is dynamic in nature, exact techniques can not be applied, but further work in identifying practical, robust methods that may be effective for instances where the problem characteristics, e.g., the dissatisfaction costs for particular customers, are not known with certainty, would help enhance the practicality of this area.

9.3.2 The Dynamic OPTW

We now consider an extension to the DOP in which all customers have time windows over which they are available to be serviced. We initially consider the problem with hard time windows, where reward is only received for servicing the customer within their time window. Since this problem is a dynamic version of the OPTW, we call it the Dynamic OPTW (DOPTW).

We create an environment for testing the DOPTW, similarly to that for the

DOP, where we again generate a number of layouts from which the customers may be located, and generate customer arrivals from a Poisson distribution. The arrival rates we consider for this problem are, respectively, 5, 2.5 and 1.25. We found that the inclusion of the time windows leads to fewer customers being able to be included than for the DOP. Therefore, in order to create similar problem classes to those for the DOP (where there were many rejected, some rejected and very few rejected) we reduced the arrival rates to these given levels.

In order to test the effect of the time windows on the problem, we set the time window widths at three different levels: all of width 4 hours, all of width 2 hours and all of width 1 hour. This creates problems where the time windows become successively more restrictive, which increases the importance of the time windows in the routing decisions. By having equal width time windows (instead of customer-dependent widths), we remove one factor that distinguishes between the customers, which enables consideration of the attractiveness of a customer to be focused on fewer features, i.e., the reward value and the combination between the temporal and spatial locations of the customer request.

For each time window width, we again create 18 classes of problems, through using the three levels of arrival rate given above, as well as the three levels of delay penalties and the two levels of reward variability given for the DOP. Again, we generate 100 problem instances for each problem class.

Computational Results

Wide Time Windows

Our first investigation is with the wide time windows, which we set to all be of width 4 hours. We initially consider the basic solution routines used for the DOP, with these appropriately altered to account for the time windows. The results from testing the methods MaxGain, BatchMaxGain and BatchB4B are given in Table 9.7. For this Table, the DOPTW columns gives the objective value that we are attempting to maximize, while the OPTW columns give the total reward of the customers serviced in the solution route.

These results show that MaxGain is far superior to the batch techniques when the delay penalties are high. By the Wilcoxon Signed Rank Test at a 5% level (with three comparisons being made), we see that MaxGain is superior to each of the batch methods for 11 of the 12 problem classes with the two highest rates of the

Types		Methods					
DP	(A,R)	MaxGain		BatchMaxGain		BatchB4B	
		DOPTW	OPTW	DOPTW	OPTW	DOPTW	OPTW
0.5	(1,1)	1070.576	1110.856	874.661	1171.500	866.559	1163.749
	(2,1)	766.062	784.076	695.792	786.045	702.712	791.348
	(3,1)	484.938	489.234	474.745	489.358	477.306	492.544
	(1,2)	253.245	255.732	184.437	270.772	185.956	273.336
	(2,2)	155.181	156.090	125.502	161.751	123.538	159.927
	(3,2)	136.537	136.803	131.567	135.980	131.259	135.500
0.2	(1,1)	1154.734	1196.886	1093.769	1229.901	1096.542	1231.220
	(2,1)	790.521	812.262	763.500	807.412	770.634	813.609
	(3,1)	490.402	493.592	489.862	497.074	487.489	494.856
	(1,2)	272.587	277.812	252.897	290.606	255.839	293.997
	(2,2)	157.978	159.928	148.987	165.267	147.861	163.971
	(3,2)	137.254	137.620	134.989	137.020	134.796	136.633
0.05	(1,1)	1222.598	1245.315	1239.156	1280.722	1249.719	1290.708
	(2,1)	815.474	826.604	811.265	825.261	819.472	833.974
	(3,1)	497.277	498.691	496.798	498.965	495.544	498.042
	(1,2)	299.886	305.291	307.316	320.424	307.285	320.137
	(2,2)	165.533	168.008	165.482	171.717	166.308	172.614
	(3,2)	138.590	139.141	137.280	138.208	137.694	138.658
Total		500.521	510.774	473.778	520.999	475.362	522.490

Table 9.7: Results from Preliminary Testing for the DOPTW - Wide TWs

delay penalties, with the exception being the problem with penalty factor of 0.2 and A=3 and R = 1. MaxGain is also superior for the problem class with low arrival rates, low reward variability and low penalties. The superiority of this method over these problem classes indicates that the additional gain in information through waiting before making the routing decisions does not exceed the delay penalties incurred through this waiting. The batch methods are not superior to MaxGain for any of the problem classes, so there are no significant net gains to be made through gathering more information before selecting the routes to carry out.

We found previously for the DOP that significant gains could be made through rejecting customers that were deemed to be undesirable and we now consider strategies for carrying out this rejection in the presence of time windows. The strategies that we previously considered were to reject a customer if its ratio of reward to insertion time was less than a given threshold, or if the reward value was too low and the insertion time was too high. We explore these rejection criteria with a number

of different parameter levels, and work towards developing a successful approach for these time window problems.

In Figure 9.7 we show the mean results obtained on these problems for the RejectB4B and RejectRew methods, as they were applied for the DOP. The graphs on the left show the mean rewards obtained by the RejectB4B method, for a number of different values of the acceptance threshold for the bang-for-buck ratio. These values were taken at intervals of 0.1 in the range 0.1 to 1.0 and also at 1.2, with the solution value for 0.0 being that obtained by the MaxGain method (which is a version of RejectB4B where all customers are accepted). The graphs show that there are gains in mean objective value that may be obtained through rejecting some of the customers, when there are high arrival rates and/or there are reasonably high delay penalties. For other problems, the MaxGain rationale of accepting all customers, provides more effective average objective values than rejecting some of the customers.

The graphs on the right of Figure 9.7 show the mean rewards obtained by the RejectRew method for a number of initial values of the required scaled reward value. We test with the values of 0.5, 0.6, 0.7, 0.8, 0.9 and 0.95, with the solution with value of 0.0 (corresponding to the MaxGain method) also given for comparative purposes. We only tested the initial required reward value down to 0.5, as it appeared that there was little difference between the behaviour of MaxGain and RejectRew with a threshold below this. We still accept any move where the insertion time is less than the estimated expected time that we obtained via the results of Beardwood et al. [11]. With time windows this measure will favour customers that will be added close to one of the customers in the current route, as customers with incompatible time windows will not be accepted via this criterion. This becomes more of a factor as the time windows become narrower, which may affect the suitability of this method.

The graphs show that with low arrival rates the mean rewards decrease consistently as the threshold increases and the highest means are obtained through employing the MaxGain method. There are gains through rejecting customers for the medium arrival rates with high delay penalties, and with high arrival rates.

We now update the methods considered in an attempt to find improved solutions for these problems. We create a new measure of the travel time from the candidate customer to the route, through setting this time to be the minimum time from

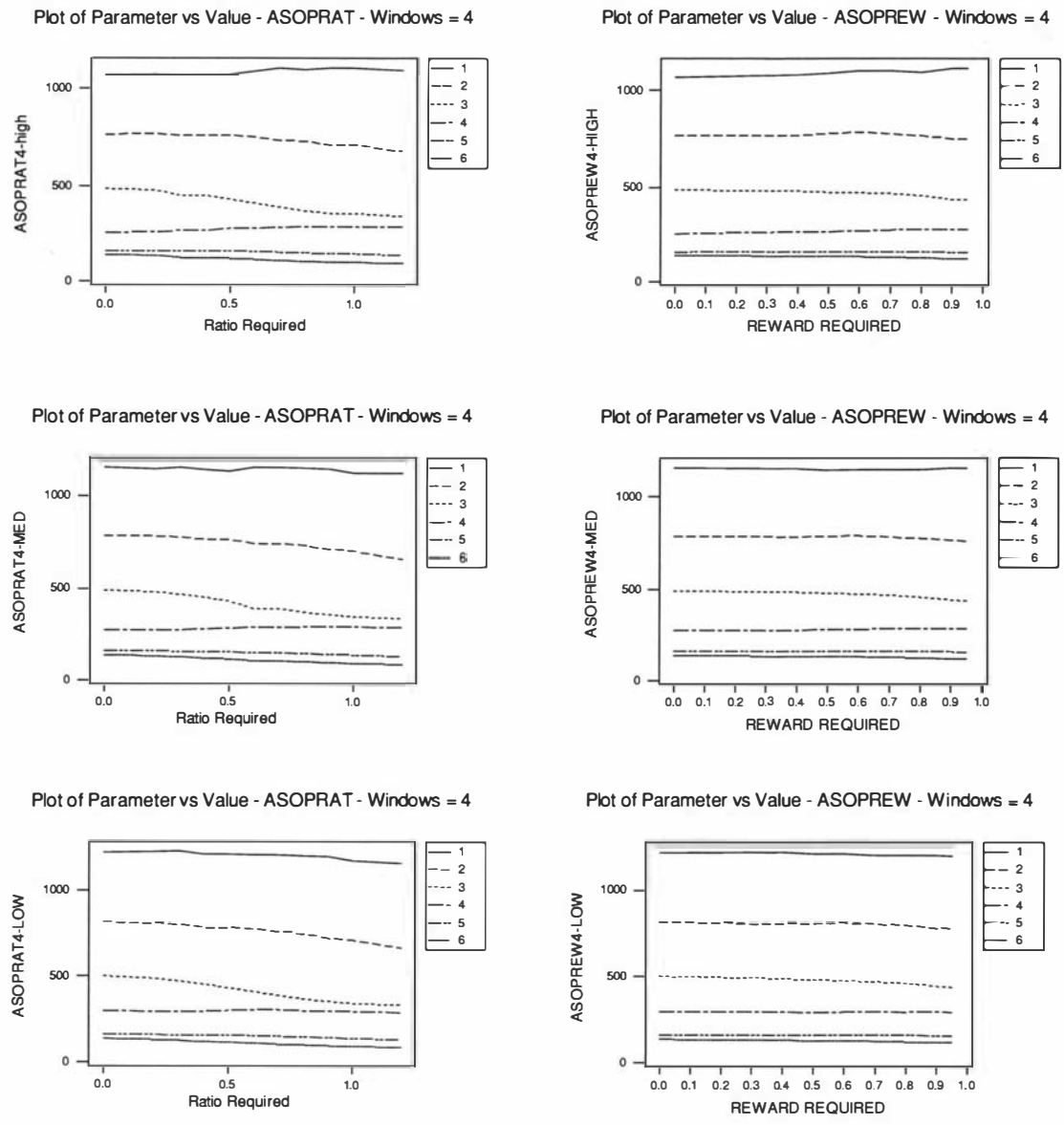

Figure 9.7: DOP Methods Applied to the DOPTW with Wide TWs

the customer to one of the customers on the route. This measure of closeness is independent of the actual route and depends only upon the customers in the route. The previous measure of time related to the insertion time, and would not consider a customer that couldn't be added between consecutive points in the route. This new measure of time allows the case where a customer is unable to be inserted between consecutive customers of the route, but may improve the solution route through being inserted into the solution route, whilst deleting one of the currently scheduled customers.

We use the new time measure for two versions of the REJECTB4B method, and the results obtained are shown in Figure 9.8. The plots on the left relate to using a bang-for-buck ratio that reduces as a function of the time since the start of the scheduling period. Therefore at time t_i, we accept a customer to be a candidate for insertion if the bang-for-buck ratio exceeds $Init*(1-t_i/LIMIT)$, where $Init$ is the initial required ratio and $LIMIT$ is the length of the time horizon during which customer requests may arrive. The plots on the right relate to the results from REJECTB4B, where we have a constant required ratio regardless of the time the customer's request becomes available. These plots are in similar format to those of Figure 9.7, where the plots are in order of the rates of penalties for delayed notification, lines 1 to 3 refer to high reward variability, and lines 4 to 6 refer to low reward variability, with these in decreasing order of the delay penalties.

For problems where the required ratio is reducing, we test initial ratios over the range 0.3 to 1.6, as these were sufficient to show the trends present. We see from the plots that the objective function value generally follows a curve, with increasing values up to a particular threshold value, and the values decreasing thenceforth. For the problems with constant threshold ratio we tested ratios of between 0.2 and 1.4, and the plots show there is a clear trend in the objective value, as it decreases dramatically after a given maximum value of the threshold ratio. The results of peak objective values are similar to, although consistently lower than, those obtained by the method with reducing threshold ratio.

There are six problem classes where the peak objective function value for these methods obtain higher means than those obtained by the MAXGAIN method, with five of these classes having high arrival rates and the remaining class having medium arrival rates. The peak mean objective value is always lower for the constant ratios, and so, for these problems, the decreasing ratios appear to obtain slightly better

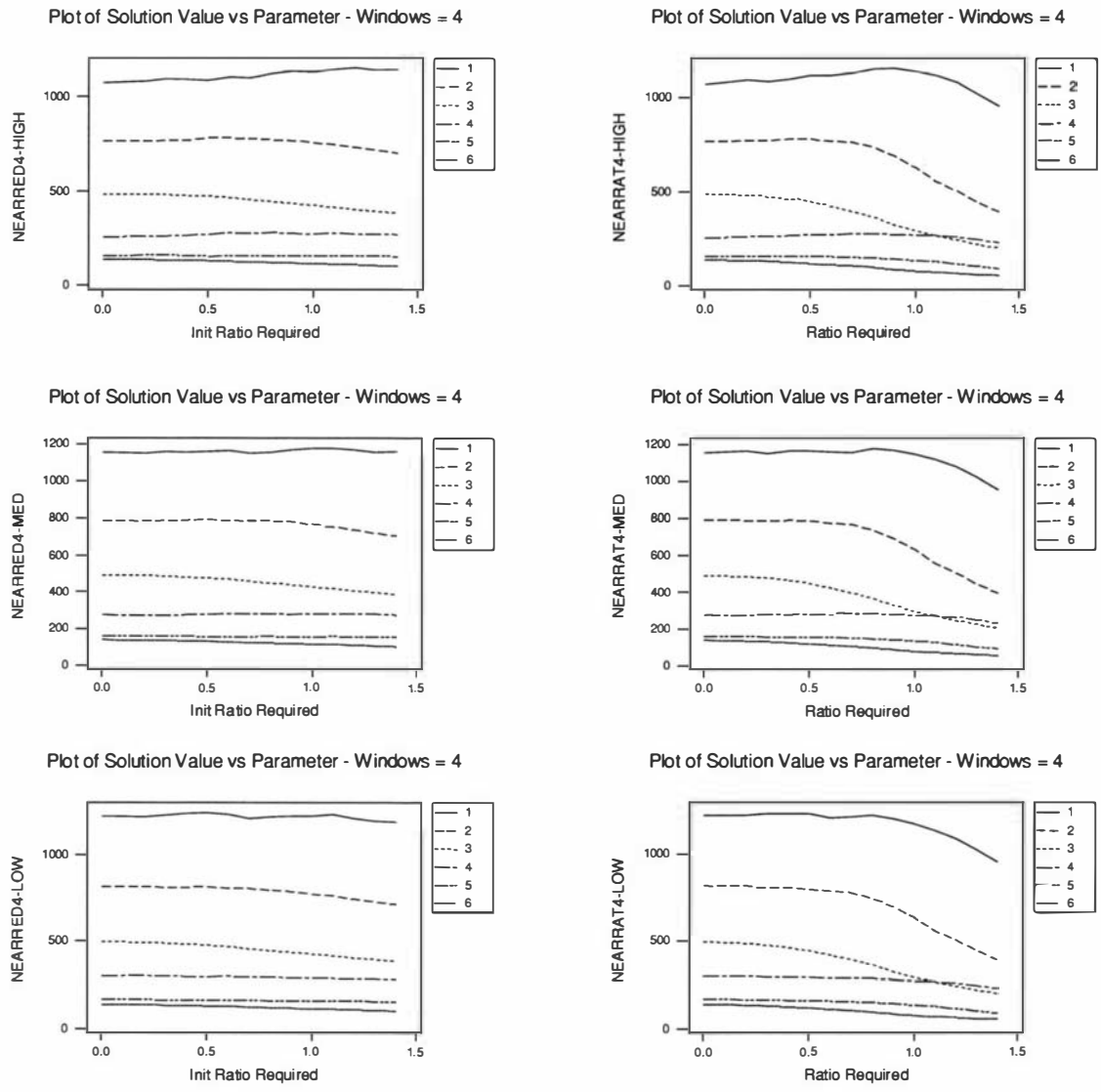

Figure 9.8: New Methods Applied to the DOPTW with Wide TWs

results.

We summarize the results by extracting the best performing of the parameter settings for different situations. We have seen our rejection routines are not effective for any problem classes where the arrival rates are low, so we concentrate on the problem classes with the two highest values of the arrival rate. We find the parameter setting that obtains the highest mean objective function value for each of these arrival rates for the different methods, and collate these. For the methods that were as for the DOP, the maximum overall mean for the high arrival rates come from the REJECTB4B method with threshold ratio of 0.7, while the maximum with medium arrival rates is REJECTREW method with initial threshold reward of 0.6.

For the methods with the new time measure, the highest mean for high arrival rates is with the threshold ratio decreasing and the initial value required being 1.1; the highest mean with medium arrival rates is again with decreasing threshold ratios, but with an initial value of 0.5.

We compare the results for these identified parameter settings with those obtained via the MAXGAIN method, and we also include the mean objective function value for the method of obtaining the best solution for the static OPTW version of the problem, with this method involving the processing of all customers at the end of the scheduling period. over which customer requests may be made.

The results are given in Table 9.8, where NR1.1 and NR0.5 are our methods using the new time measure, RAT0.7 and REW0.6 are our selected methods using the old time measure, MGAIN is our MAXGAIN method and OPTW is our static OPTW solution method. We also include the results from an implementation of the TSP method considered previously for the DOP. We note that with the arrival rates of 1.25 and 2.5 that are used for our data for the DOPTW, then the TSP method will accept all customers (acceptance area is equal to 6.83 and 4.87 respectively). Since this method is a duplication of MAXGAIN for these problems, and since there is a parallel between the proportion of customers available for these problems and the numbers that were available for the DOP, we implement the TSP method using the same acceptance areas as for the DOP with double the arrival rate, i.e., 2.48, 3.51 and 4.96 for arrival rates 5.0, 2.5 and 1.25 respectively. In Table 9.8, a number in bold indicates that this method is significantly more effective than MAXGAIN for this particular problem class, when tested by the Wilcoxon Signed Rank Test

Types		Methods						
DP	(A,R)	NR1.1	NR0.5	RAT0.7	REW0.6	TSP	MGAIN	OPTW
0.5	(1,1)	**1141.52**	1086.22	1107.12	1104.36	1085.71	1070.58	-166.62
	(2,1)	746.38	781.55	738.25	784.33	745.16	766.06	476.61
	(3,1)	416.42	474.09	388.68	471.90	484.94	484.94	448.48
	(1,2)	**277.11**	**269.66**	**283.90**	**267.83**	**286.13**	253.25	-116.52
	(2,2)	154.77	151.45	149.67	158.65	154.23	155.18	-32.94
	(3,2)	111.55	129.53	105.24	130.54	136.54	136.54	121.92
0.2	(1,1)	1176.54	1158.88	1155.64	1145.70	1117.13	1154.73	753.78
	(2,1)	753.96	795.13	747.80	792.26	760.62	790.52	704.55
	(3,1)	417.72	476.48	389.10	472.74	490.40	490.40	479.01
	(1,2)	278.90	274.58	286.99	278.83	**290.89**	272.59	163.36
	(2,2)	155.20	153.70	149.52	158.16	156.30	157.98	97.50
	(3,2)	111.61	129.45	105.24	130.58	137.25	137.25	132.74
0.05	(1,1)	1228.84	1239.51	1208.24	1215.62	1147.09	1222.60	1221.22
	(2,1)	763.89	815.93	761.66	812.35	779.99	815.47	823.71
	(3,1)	417.78	478.32	389.60	474.22	497.28	497.28	494.03
	(1,2)	286.77	295.41	307.97	297.34	299.50	299.89	309.01
	(2,2)	155.69	159.31	153.26	163.18	162.57	165.53	165.01
	(3,2)	111.87	129.67	105.61	131.05	138.59	138.59	138.98
Total		483.70	499.94	474.08	499.42	492.80	500.52	345.21

Table 9.8: Testing New Methods for the DOPTW - Wide TWs

at a 5% level (with six tests considered).

Table 9.8 shows that none of the methods are significantly more effective than MAXGAIN for problem classes with medium or low arrival rates. The NR1.1 method is more effective than MAXGAIN for both problem classes with high arrival rates and high penalties, although it's relative performance drops away dramatically as the arrival rates decrease. The performance of the RAT0.7 method drops away even more significantly as arrival rates decrease, and thus this method appears to only be suitable for problems with high arrival rates, particularly with low reward variability. We believe that this is due to the time measure that is applied, being more suitable with these rewards. This time measure relates to the insertion time to the current route and, since with low reward variability the routing aspect is more important than the selection, the time measure is compatible with the decisions being made. The new time measure relates to closeness of the customer to one of the customers in the current solution, and since this measure doesn't guarantee

route closeness, this method does not appear to be as effective when the routing is of primary importance.

The results show that, using techniques which reject customers according to their reward and time properties, that were previously found to be effective for the DOP, i.e., the RejectRat method, with parameters appropriately selected, leads to improved solutions over those obtained by MaxGain for certain problem classes with high arrival rates and reasonably high delay penalties. Using the new time measures also leads to improved results, with the methods we considered being more effective for problems with high reward variability.

The TSP method is more effective than MaxGain for two problem classes, namely those with low reward variability, high arrival rates and the two highest delay penalties. Thus, with these wide time windows and low reward variability, the routing aspects remain more important than the selection, which enables this method to be effective. The overall mean objective value obtained by this method is below that obtained by three of the other heuristics considered, which shows that the method is less appropriate for the time window problems than it was for the DOP, as the time windows are able to over-ride the closeness due to locational proximity.

The static OPTW method is unable to improve significantly upon the results obtained by the MaxGain method for any of the problem classes. We note that the OPTW method is able to service on average 22.6 of the 39.8 available customers with high arrival rates, 13.3 of the 19.9 available customers with medium arrival rates and 9.0 of the 9.9 available customers with the low arrival rates. These means are obviously higher than those obtained by the dynamic methods, but the penalty incurred for the excluded customers prevents the method from out-performing the dynamic methods.

Our final consideration is with the robustness of the methods. In order to test this, we calculate the highest objective value identified over all the different implementations that we tested, and we calculate the minimum proportion of this maximum value that is obtained by each of the methods for each of the problem classes. The results of this testing are shown in Table 9.9.

This Table shows that the method that obtains the poorest results is the RRat0.7 method. This method obtains a particularly poor solution for a test problem with low arrival rates, where only one of the customers is serviced. This

Types		Methods					
DP	(A,R)	NRED1.1	NRED0.5	RRAT0.7	RREW0.6	TSP	MGAIN
0.5	(1,1)	0.645432	0.509819	0.569709	0.569709	0.576089	0.561007
	(2,1)	0.624994	0.627907	0.512741	0.607224	0.606469	0.570192
	(3,1)	0.424918	0.648741	0.059317	0.590840	0.671140	0.671140
	(1,2)	0.489760	0.546235	0.577282	0.529610	0.549727	0.525934
	(2,2)	0.606831	0.548456	0.216369	0.571036	0.401617	0.558471
	(3,2)	0.271034	0.591056	0.233682	0.653829	0.677343	0.677343
0.2	(1,1)	0.671071	0.669611	0.642099	0.582902	0.650593	0.482848
	(2,1)	0.663201	0.634501	0.501473	0.651984	0.663249	0.739949
	(3,1)	0.424918	0.678369	0.059317	0.590840	0.671140	0.671140
	(1,2)	0.552348	0.551665	0.583372	0.536376	0.549337	0.582914
	(2,2)	0.606831	0.574385	0.216369	0.595131	0.539072	0.558471
	(3,2)	0.271034	0.591056	0.233682	0.653829	0.677343	0.677343
0.05	(1,1)	0.684231	0.718662	0.639643	0.716519	0.636580	0.651630
	(2,1)	0.645206	0.679910	0.496023	0.679910	0.663450	0.739286
	(3,1)	0.424918	0.661443	0.059317	0.590840	0.795557	0.795557
	(1,2)	0.539656	0.630890	0.705392	0.633700	0.569021	0.685449
	(2,2)	0.612502	0.595131	0.204020	0.595131	0.539922	0.621710
	(3,2)	0.271034	0.591056	0.233682	0.662385	0.677343	0.677343

Table 9.9: Worst-Case Performance for the DOPTW - Wide TWs

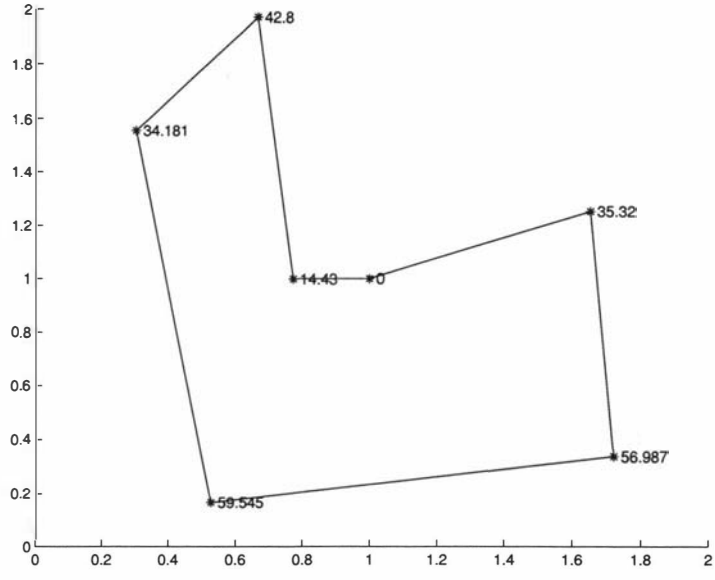

Figure 9.9: Worst Found Solution by RRAT0.7

problem is shown in Figure 9.9, where the solution obtained by MGAIN is shown, with the labels corresponding to the reward available for the customer. We see that for this problem instance there are only six customers available, and each can be serviced within a single vehicle route. The first customer that requests service is the one located at position (0.774, 0.998) with a reward of 14.429, and this is included in the service route by each of the methods. The RRAT0.7 method then rejects each of the customers in turn as their successive bang-for-buck ratios are: 0.229, 0.401, 0.310, 0.242 and 0.234. Thus the only customer serviced is the first one and the total reward received is 14.429, whereas it is possible to collect all the available reward (which is equal to 243.263). Table 9.9 again shows that the RRAT0.7 method method appears to be unsuitable for problems with low arrival rates, with the NRED1.1 also obtaining poor solutions with low arrival rates and reward variability. The most robust method, in terms of lowest proportion of maximum value obtained, is the RREW0.6 method, and, since the minimum proportion exceeds the minimum obtained by MGAIN, this suggests that it is possible to reduce the likelihood of obtaining poor results through rejecting some of the customers from consideration.

Medium Width Time Windows

Our next investigation is into medium width time windows, where we set all time windows to be of width 2 hours. We again begin our investigation through considering the basic solution routines used for the DOP, with these adjusted to account for the time windows. The results from testing the methods MAXGAIN, BATCHMAXGAIN and BATCHB4B are given in Table 9.10. This table shows that with these time windows there is little advantage from waiting in order to insert the customers at a later stage. Neither of the batch methods are significantly more effective than MAXGAIN for any of the problem classes, when tested by the Wilcoxon Signed Rank Test to a 5% level, whereas MAXGAIN is significantly more effective than each of the batch methods for the first 13 problem classes of the Table (each having a p-value of 0.0000 (4 d.p)). Therefore the gains in reward from waiting to schedule the customers (as indicated by the higher values in the OPTW columns for the batch methods compared with those for MAXGAIN), do not exceed the additional costs of the delays. This is due to the time windows in this case restricting the routing options, and so less additional reward can be claimed than for less-constrained problems.

We again consider the methods REJECTREW and REJECTB4B that were successful for the DOP, and vary the parameters for these methods. The plots of this testing are shown in Figure 9.10, where these are in the format previously considered with the wider time windows. For the results from the REJECTREW method, which are shown in the plots on the left, we vary the required scaled reward in the range 0.3 to 0.95. These results show little deviation across the range of parameter values, and therefore very little possible gain over the MAXGAIN method.

For the results from REJECTB4B, shown in the plots on the right, we vary the threshold ratio in the range 0.2 to 0.9. We see that there may be small gains in the objective value through increasing the threshold ratio up to a certain value, but as the threshold increases, the average objective value decreases. The gains are only present with high and medium arrival rates and with high and medium rates of the delay penalty, since, for low penalties and low arrival rates, the highest means are obtained with a threshold value of 0.0, i.e., the MAXGAIN method.

For later analysis, we identify the maximum mean objective value obtained among the two methods, for high and medium arrival rates. For high arrival rates, the maximum mean value is with the REJECTB4B method, with a ratio of 0.3,

Prob Types		Methods					
Pen	(Arr,RV)	MaxGain		BatchMaxGain		BatchB4B	
		DOPTW	OPTW	DOPTW	OPTW	DOPTW	OPTW
0.5	(1,1)	808.319	845.327	514.698	885.172	512.543	881.555
	(2,1)	577.713	600.721	471.349	614.777	473.623	616.914
	(3,1)	413.203	422.755	383.228	427.082	384.056	428.291
	(1,2)	192.392	194.282	113.161	212.067	112.764	211.281
	(2,2)	150.215	151.297	118.068	155.241	118.214	155.647
	(3,2)	101.910	102.029	93.974	103.202	94.158	103.301
0.2	(1,1)	862.652	898.937	753.368	918.049	762.699	927.103
	(2,1)	605.350	627.145	576.823	643.380	575.435	642.426
	(3,1)	422.413	430.797	413.810	434.645	414.759	435.488
	(1,2)	204.854	208.577	185.671	228.819	185.284	227.844
	(2,2)	152.848	155.348	145.530	162.405	146.113	162.793
	(3,2)	103.060	103.881	101.060	105.889	101.193	105.988
0.05	(1,1)	940.685	960.745	925.040	975.286	926.177	976.473
	(2,1)	647.425	659.723	647.954	670.210	646.276	668.681
	(3,1)	430.440	434.907	429.184	436.473	429.800	436.953
	(1,2)	225.285	229.654	232.226	245.877	230.591	244.433
	(2,2)	161.528	163.997	163.418	169.212	164.005	170.099
	(3,2)	106.255	107.361	107.111	109.113	106.949	108.959
Total		394.808	405.416	354.204	416.494	354.702	416.902

Table 9.10: Results from Preliminary Testing for DOPTW - Medium TWs

while, for the medium arrival rates, the maximum mean value is with the same method with threshold ratio of 0.2.

We also consider the methods using the new time measure, and the plots of the results obtained by these methods are given in Figure 9.11. The plots on the left show the mean objective value for the method where the threshold bang-for-buck ratio decreases, where the initial required ratios were tested at intervals of 0.1 in the range 0.5 to 1.2. These plots show that there are significant gains to be made through incorporating this rejection ratio, with initial ratios of approximately 1.0 seeming to give the best results for high arrival rates.

The plots on the right give the mean objective value for our method where the required bang-for-buck ratio is constant for the whole planning horizon. The values that we tested are at intervals of 0.1 in the range 0.4 to 1.2, and also at 1.4. These results show that there are again significant gains to be made through the use of this acceptance criteria, with large drop-offs in heuristic performance for ratios that are too large.

We find that the highest mean over these two methods with high arrival rates is with the threshold decreasing, and its initial value being equal to 1.1. The highest mean with medium arrivals is also with the reducing threshold, and this is with an initial value of 0.6. We give these results in Table 9.11 (with these methods given as NRED1.1 and NRED0.6), together with the best methods using the same criteria as for the DOP (REJECTB4B with ratios of 0.3 and 0.2), the MAXGAIN method and the static version of the OPTW. The numbers in bold again indicate that the method is significantly more effective than MAXGAIN for the problem class, as tested by the Wilcoxon Signed Rank Test at a 5% level. We found that the TSP method, which we previously considered, was relatively ineffective for these problem instances, so we chose not to test this method further. Since the temporal aspect of the 'closeness' of customers becomes more important as the time windows are tightened, the use of only spatial considerations in the TSP method, leads to ineffective solutions being obtained.

Table 9.11 shows that there are significant gains that may be made through selecting an appropriate policy for rejecting customers from consideration for certain classes of problems. The NRED1.1 method is significantly more effective than MAXGAIN for three of the problem classes, and these are with the high and medium arrival rates, in which we were interested. These problems also involve the highest

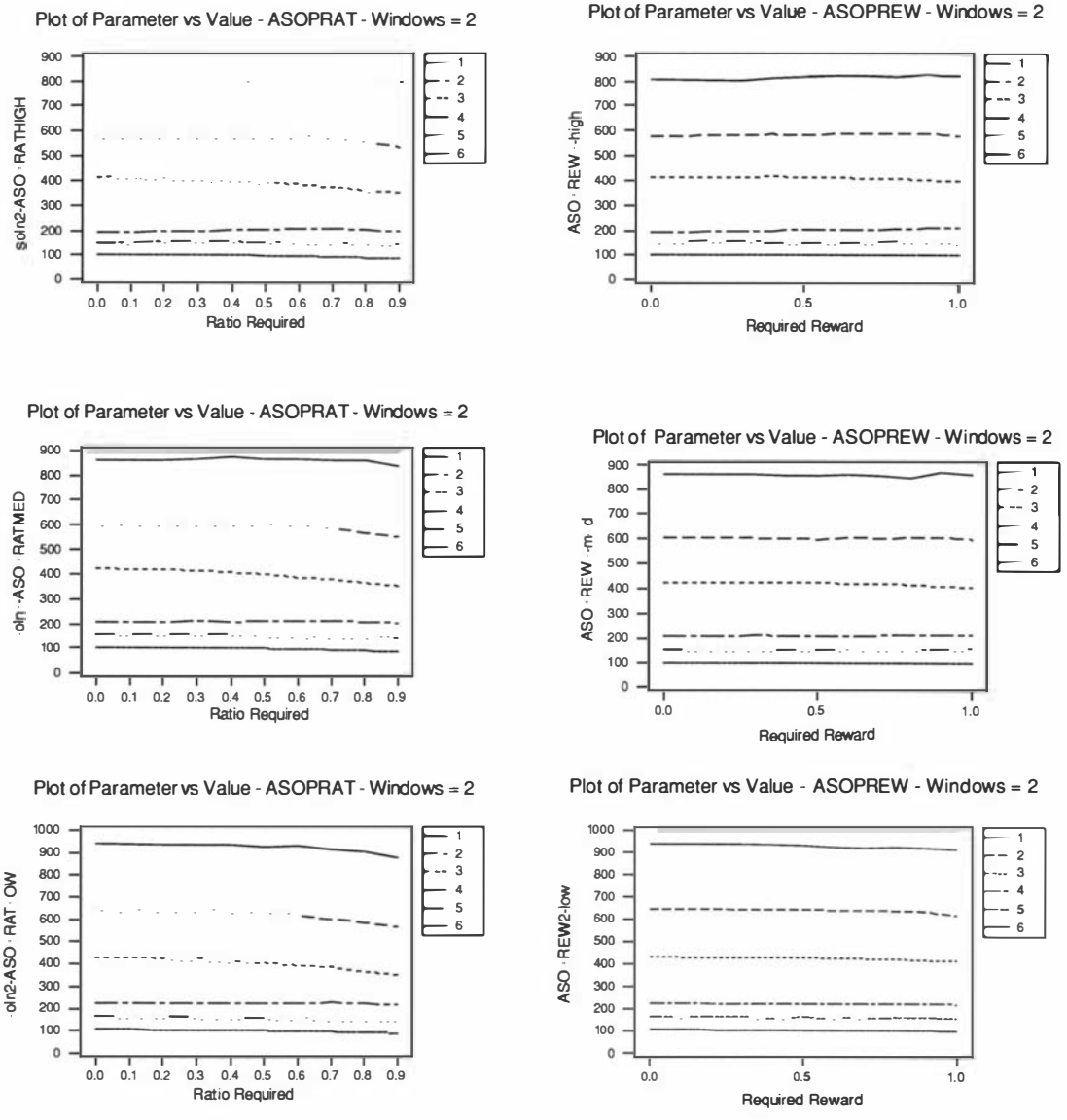

Figure 9.10: DOP Methods Applied to the DOPTW with Medium TWs

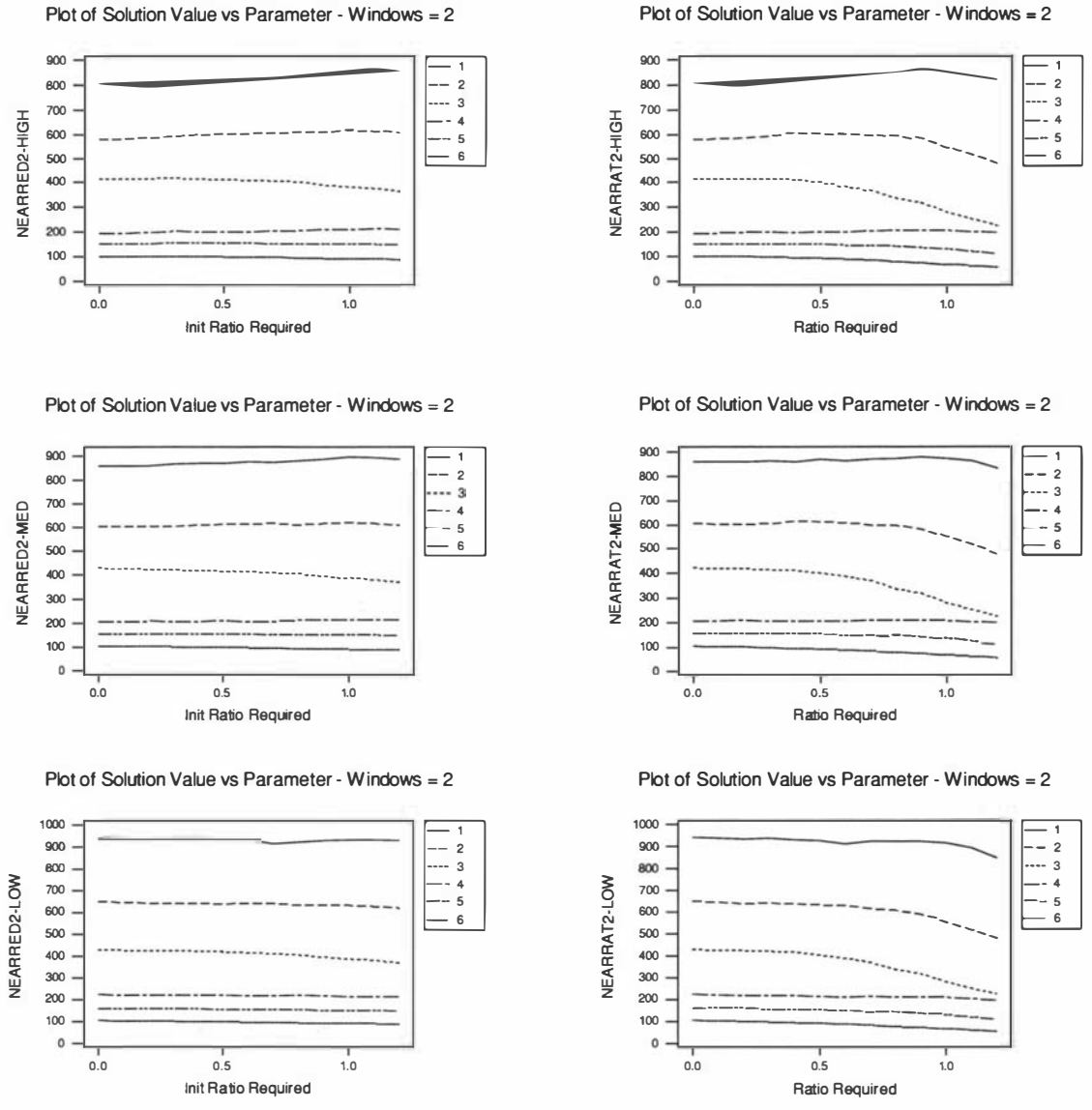

Figure 9.11: New Methods Applied to the DOPTW with Medium TWs

Prob Types		Methods					
Pen	(Arr,RV)	NRED1.1	NRED0.6	RRAT0.3	RRAT0.2	MGAIN	OPTW
0.5	(1,1)	**870.292**	834.184	826.446	811.314	808.319	-1049.729
	(2,1)	**613.661**	**605.645**	590.873	589.406	577.713	-75.012
	(3,1)	375.759	408.461	401.161	410.192	413.203	236.692
	(1,2)	**214.544**	201.566	199.652	198.904	192.392	-320.555
	(2,2)	148.959	153.550	151.682	151.882	150.215	-38.965
	(3,2)	90.271	97.611	99.233	100.906	101.910	60.539
0.2	(1,1)	896.477	877.915	873.512	869.983	862.652	238.574
	(2,1)	617.105	614.009	604.371	608.603	605.350	403.062
	(3,1)	380.637	411.788	405.984	415.966	422.413	363.057
	(1,2)	213.662	206.106	212.105	211.449	204.854	34.238
	(2,2)	149.653	153.733	151.683	152.846	152.848	91.958
	(3,2)	90.567	97.762	99.164	101.041	103.060	91.258
0.05	(1,1)	933.964	931.373	937.935	937.392	940.685	884.527
	(2,1)	626.013	639.039	639.366	644.537	647.425	635.953
	(3,1)	382.824	417.955	412.168	421.445	430.440	430.096
	(1,2)	217.061	219.527	226.018	224.849	225.285	218.010
	(2,2)	151.660	155.459	156.313	159.387	161.528	160.673
	(3,2)	91.292	98.052	100.323	102.452	106.255	107.172
Overall		392.467	395.763	393.777	395.142	394.808	137.308

Table 9.11: Testing New Methods for the DOPTW - Medium TWs

delay penalties, and it appears that for problems with these medium width time windows, we can only find improved solutions through rejecting certain customers, when these penalties are high. We see that the relative performance of this method decreases dramatically as the arrival rates decrease, so this method only appears suitable with high arrival rates. Neither the RRAT0.3 nor the RRAT0.2 method is significantly more effective than MAXGAIN for any of the problem classes.

We see that the static OPTW method is not more effective than MAXGAIN for any of the problem classes; this shows that any potential gain through waiting to service the customer does not exceed the additional delay penalties for customers that are rejected at this later stage. With wide time windows and with no time windows, we found there were possible gains through servicing all of the customers at the end of the time limit, but with these narrower time windows, there is less to be gained through waiting for more information. This is because, with tighter windows, there is a higher chance that the customer will not be able to be added, due to time window incompatibility. For the DOP we found that the static method became more effective as the arrival rate decreased, but with the time windows there is no guarantee that lowering the arrival rates will allow more customers to be served, since we are randomly generating the time windows. If we generated the customers from a known feasible tour, then the static method would be effective for these problems, but we do not make the assumption that such a tour exists here. In these problems we found that the OPTW method, on average, services 17.8 of the 39.9 available customers with high arrival rates, 11.7 of the 19.9 available customers with medium arrival rates and 7.5 of the 9.9 available customers with low arrival rates. Since the numbers served are lower than those with the wider time windows, this method performs more poorly than was previously the case.

We also give, in Table 9.12, the worst-case proportion of the best solution value found for each of these methods. The Table shows that the NRED1.1 method is particularly effective in avoiding obtaining poor solutions for problems with high arrival rates, especially with quite high delay penalties and high reward variability. In this case, the strict rejection criterion which eases as the time remaining for customers to arrive decreases, will prevent some inferior customers from being inserted early on in the search procedure, which would negatively impact the solution that is otherwise able to be obtained.

Prob Types		Methods				
Pen	(Arr,RV)	NRED1.1	NRED0.6	RRAT0.3	RRAT0.2	MGAIN
0.5	(1,1)	0.588313	0.427688	0.437892	0.437892	0.437892
	(2,1)	0.642660	0.628850	0.639519	0.544129	0.467938
	(3,1)	0.359667	0.597964	0.359667	0.516982	0.628011
	(1,2)	0.522180	0.504144	0.472152	0.472152	0.471135
	(2,2)	0.463041	0.470263	0.467672	0.467672	0.510335
	(3,2)	0.339604	0.476150	0.526039	0.584261	0.475761
0.2	(1,1)	0.587086	0.424074	0.434109	0.434109	0.434109
	(2,1)	0.580319	0.616996	0.602593	0.594934	0.594934
	(3,1)	0.359667	0.604496	0.359667	0.516982	0.635186
	(1,2)	0.522180	0.502409	0.555534	0.526340	0.526299
	(2,2)	0.463041	0.470263	0.469227	0.461498	0.461498
	(3,2)	0.339604	0.476150	0.526039	0.611631	0.586200
0.05	(1,1)	0.560397	0.560397	0.583529	0.583529	0.581880
	(2,1)	0.666817	0.699890	0.696799	0.693444	0.689141
	(3,1)	0.359667	0.630823	0.359667	0.516982	0.661513
	(1,2)	0.527724	0.513227	0.565748	0.537791	0.537791
	(2,2)	0.570724	0.470263	0.470004	0.470004	0.523491
	(3,2)	0.339604	0.476150	0.526039	0.633651	0.600548

Table 9.12: Worst-Case Performance for the DOPTW - Medium TWs

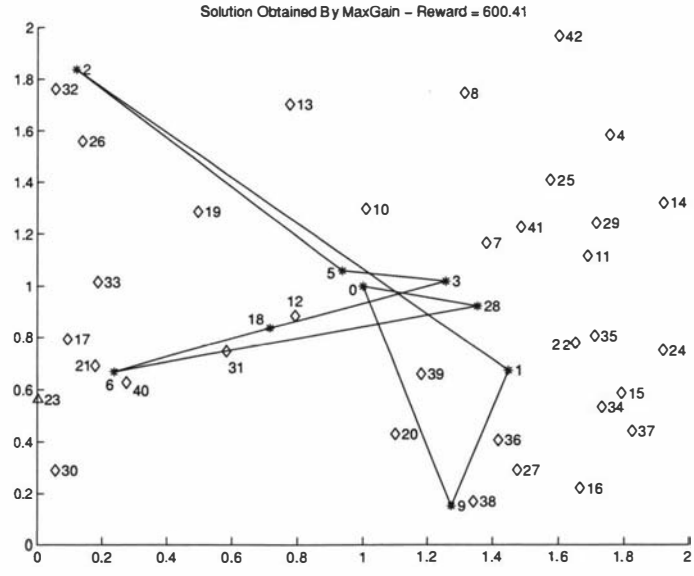

Figure 9.12: Solution Obtained by MAXGAIN for Problem 34

We illustrate this through analyzing a problem instance with high delay penalties and high arrival rates. The solution obtained by MAXGAIN is given in Figure 9.12, with the labels of the customers again representing the order in which their requests for service arrive. For clarity purposes, we have omitted the time windows and the rewards from the Figure, with the time windows in this case being incompatible with the locations, thereby causing the self-crossing tour. Again, the diamonds indicate that the customer could not have created an improved solution if it were added to the existing route. We see that, for this solution, five of the first six customers are able to be included, with only four of the remaining 36 customers able to be profitably included into the solution route (customer 23 is tentatively inserted, but is then excluded in favour of customer 28).

The solution obtained by NRED1.1 for this problem is given in Figure 9.13, where in this figure the circles indicate that the customer was rejected according to the bang-for-buck criterion. The main reason for the superior solution obtained by this method is the exclusion of the second customer, which is distant from the majority of the other customers. Since this customer has a large reward (85.97), there is a very short time span for which it is possible to exclude this customer through carrying out an improving exchange. In this problem, there is no such exchange that could occur, so this customer must remain in the solution once it has been placed (unless we allow for deteriorating moves or batch insertion). The NRED1.1 method, through excluding customer 2, is able to obtain far greater total

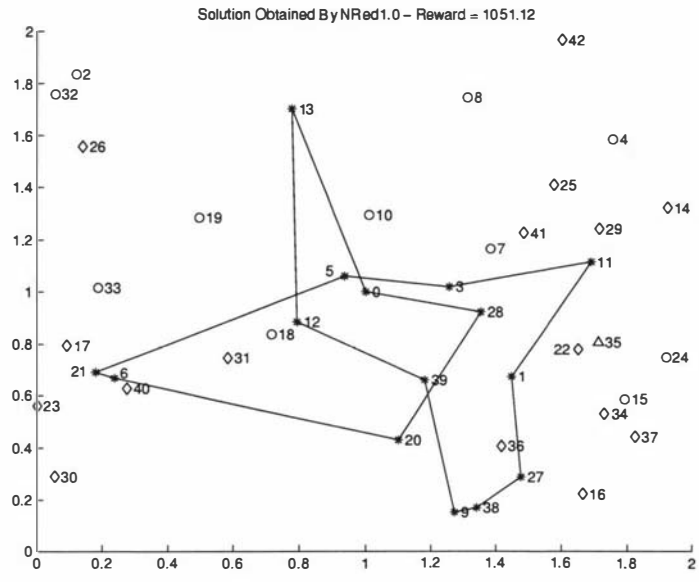

Figure 9.13: Solution Obtained by NRED1.1 for Problem 34

reward and is therefore more effective for this problem instance. We note that the worst-case performance of each of the heuristics, apart from NRED1.1, for delay penalties of 0.5 and 0.2, is for this particular problem instance, and due to the inclusion of the negatively influential customer. With delay penalties of 0.05, this customer is able to be excluded from the solution route at a later stage, and better solutions are consequently available.

We conclude from this section that, with the time windows considered, the NRED1.1 method is the most effective for problems where there are high arrival rates and high or medium delay penalties. This measure works through rejecting a number of customers, that could irretrievably impair the route created. This method is not effective for low arrival rates, as too many customers are rejected. With low delay penalties, the best method is to consider all customers for insertion (as is the case for the MAXGAIN method), as there is a greater chance of being able to salvage a solution through excluding a customer at a later stage.

Narrow Time Windows

Our next investigation is into problems with narrow time windows, where we set all time windows to be of width 1 hour. We again consider the effectiveness of the basic solution routines used for the DOP on problems with these time windows. The results from testing the methods MAXGAIN, BATCHMAXGAIN and BATCHB4B

Prob Types		Methods					
Pen	(Arr,RV)	MaxGain		BatchMaxGain		BatchB4B	
		DOPTW	OPTW	DOPTW	OPTW	DOPTW	OPTW
0.5	(1,1)	657.548	692.972	322.526	725.630	327.944	729.833
	(2,1)	508.455	529.044	375.310	546.481	370.852	542.939
	(3,1)	343.042	355.442	301.043	359.457	301.704	360.739
	(1,2)	169.882	171.682	75.044	183.978	74.467	183.414
	(2,2)	126.851	128.152	86.919	132.250	86.356	131.906
	(3,2)	90.424	90.954	78.274	92.004	78.406	92.136
0.2	(1,1)	715.069	750.158	592.203	769.783	599.800	776.686
	(2,1)	536.941	558.546	485.712	564.856	484.235	563.442
	(3,1)	352.716	363.352	338.462	367.601	338.462	367.601
	(1,2)	176.211	180.098	144.559	191.950	142.551	189.652
	(2,2)	130.007	132.247	118.282	138.496	117.760	138.080
	(3,2)	91.419	92.389	87.364	93.959	87.280	93.806
0.05	(1,1)	775.495	794.014	761.868	816.614	768.616	824.522
	(2,1)	559.079	569.282	556.835	580.865	557.499	582.272
	(3,1)	366.153	371.328	365.660	375.276	365.423	375.010
	(1,2)	190.558	195.072	190.974	205.103	191.181	205.474
	(2,2)	135.497	138.312	136.304	143.532	136.572	143.960
	(3,2)	93.234	94.472	94.511	96.838	94.320	96.665
Total		334.366	344.862	283.992	354.704	284.635	355.452

Table 9.13: Results from Preliminary Testing for DOPTW - Narrow TWs

are given in Table 9.13, and these show that there is no advantage to be gained from waiting before attempting to schedule the customers. The mean reward for MaxGain is greater than the mean for the batch techniques except for the problem classes with low arrival rates and low reward variability, where the means are very slightly higher. This shows that the cost of the delays exceeds any gains in the routing that are possible, which is the result we found previously for the problems with medium width time windows.

We again consider the techniques RejectRew and RejectB4B for these problems, and the plots of the results obtained are given in Figure 9.14. The plots on the left show the mean objective function for different ratios for the RejectB4B method, where we test with ratios at intervals of 0.1 in the range 0.2 to 0.9. These plots show that there may be gains through using the rejection criteria of this method with high arrival rates, since the mean reward obtained increases as the threshold ratio increases, until the threshold reaches a value of approximately 0.4.

The mean reward decreases steadily for higher threshold values and, with the two lower settings of the arrival rate or with the low rate of the delay penalties, the mean rewards decrease consistently as the threshold ratio is increased above zero.

The plots on the right are for the RejectRew method, where we test with the scaled reward that is initially required, being varied in intervals of 0.1 from 0.3 to 0.9 with the value 0.95 also tested. There appear to be some small gains that may be made through using the rejection criteria for high arrival rates also. Neither this method nor RejectB4B seem to be able to improve upon the results obtained by the MaxGain method for any of the parameter settings for problems with medium or low arrival rates and/or with low delay penalties. For further investigation we take the parameter setting for each method that obtains the highest mean reward for high arrival rates over the two higher rates of delay penalties. These parameter settings are 0.3 for RejectB4B and 0.6 for RejectRew.

We again test the methods with the new time measure on these problems, and the plots of these results are given in Figure 9.15. The plots on the left are with the fixed threshold ratio, and we tested the ratios of 0.1 to 1.1 in intervals of 0.1. With these problems we see that there are some gains that may be made with high arrival rates and the higher delay penalties, and with low delay penalties the mean reward decreases as the ratio increases. The plots on the right give the results with the decreasing threshold ratios, where we tested the initial values of 0.2 to 1.3 in intervals of 0.1. The results for this method seem to be very similar to those obtained with the constant ratio. The implementation that gives the highest overall mean with the highest arrival rates is with a decreasing threshold ratio, with initial value of 1.2, while the decreasing ratio with initial value of 0.6 gives the highest mean with the medium arrival rates.

The methods we consider further are therefore NRed1.2 and NRed0.6 (which are the methods with reducing bang-for-buck ratio threshold and the new time measure), RRat0.3 and RRew0.6 (which are the best parameter settings for RejectRew and RejectB4B), MGain (which is the MaxGain method) and the static OPTW solution. These results are given in Table 9.14, where the number in bold indicates that the method is significantly more effective than MaxGain for the particular problem class when tested by the Wilcoxon Signed Rank Test at a 5% level.

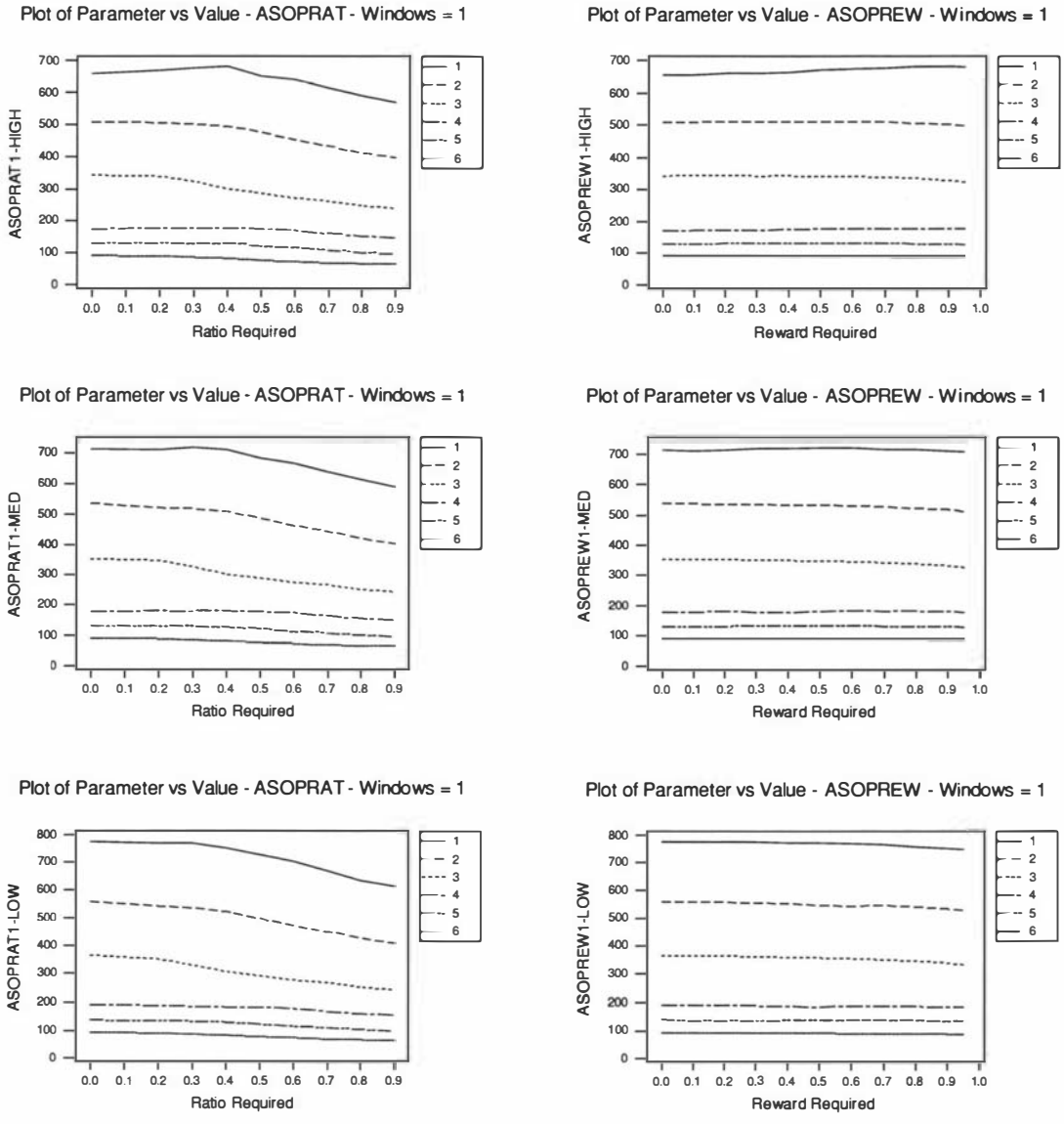

Figure 9.14: DOP Methods Applied to the DOPTW with Narrow TWs

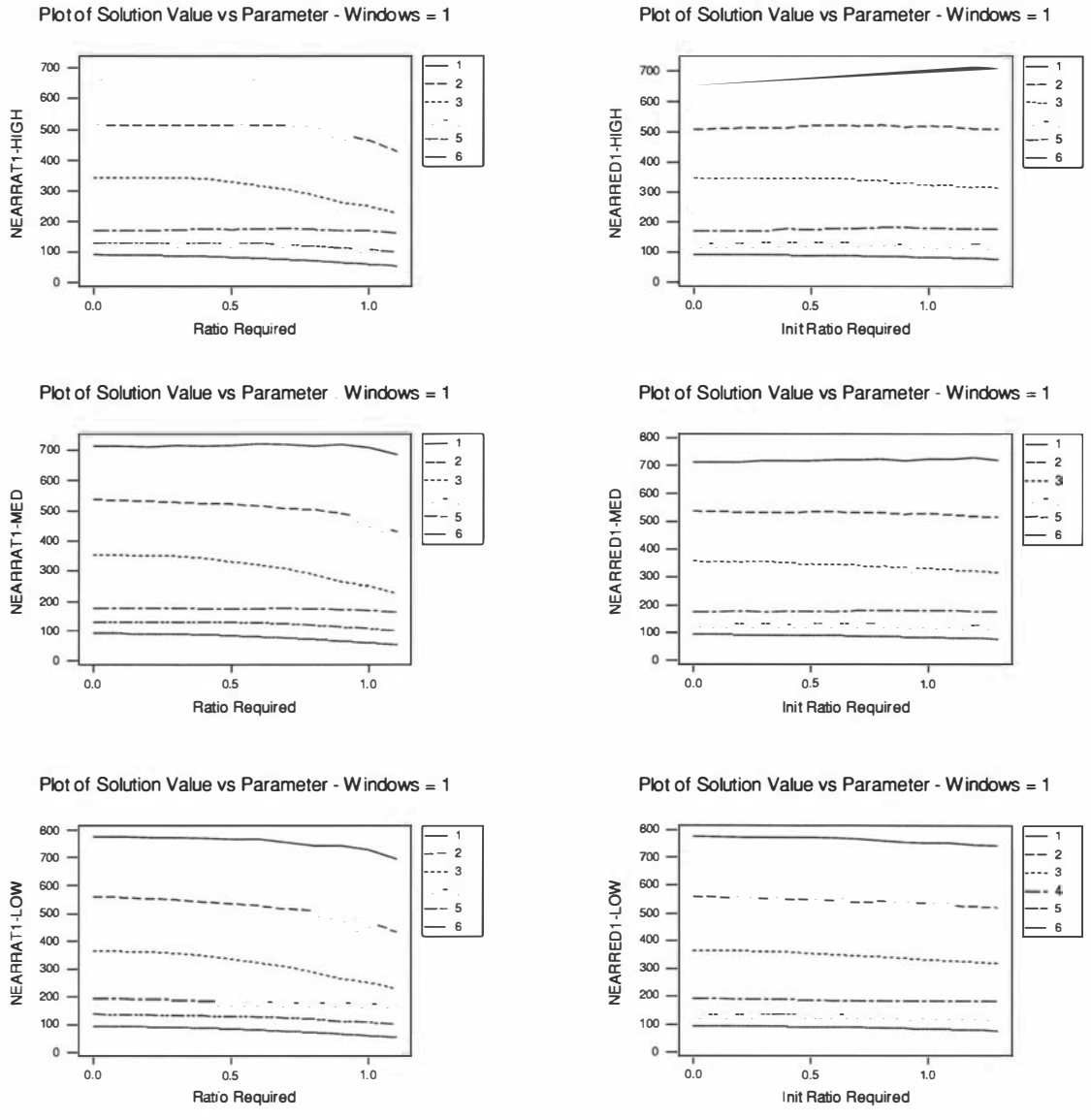

Figure 9.15: New Methods Applied to the DOPTW with Narrow TWs

Prob Types		Methods					
Pen	(Arr,RV)	NR1.2	NR0.6	RAT0.3	REW0.6	MGAIN	OPTW
0.5	(1,1)	**720.420**	**692.564**	**677.088**	**676.508**	657.548	-1574.971
	(2,1)	512.284	**523.634**	501.582	511.723	508.455	-327.948
	(3,1)	319.741	343.426	322.644	341.227	343.042	101.329
	(1,2)	177.707	177.958	175.432	**177.816**	169.882	-471.444
	(2,2)	125.373	130.254	127.421	131.227	126.851	-113.927
	(3,2)	79.305	87.682	84.569	88.795	90.424	14.849
0.2	(1,1)	732.901	721.782	720.346	723.314	715.069	-74.167
	(2,1)	519.762	536.673	518.636	529.557	536.941	245.345
	(3,1)	323.126	344.082	325.898	345.885	352.716	268.626
	(1,2)	179.383	176.702	178.252	181.558	176.211	-49.959
	(2,2)	124.723	131.278	128.736	132.929	130.007	49.158
	(3,2)	79.425	88.275	85.017	89.425	91.419	64.861
0.05	(1,1)	747.264	770.185	769.415	770.848	775.495	676.778
	(2,1)	526.145	547.125	535.633	544.885	559.079	531.776
	(3,1)	325.778	351.479	329.936	355.223	366.153	353.973
	(1,2)	180.471	182.683	185.366	186.452	190.558	164.523
	(2,2)	125.515	132.693	132.117	135.071	135.497	130.898
	(3,2)	79.666	88.867	85.211	90.401	93.234	90.799
Overall		326.610	334.852	326.850	334.047	334.366	4.472

Table 9.14: Testing New Methods for the DOPTW - Narrow TWs

From the Table we see that the four methods that reject customers from consideration are all able to significantly improve upon the results obtained by MAXGAIN for the problem class with high arrival rates, delay penalties and reward variability. For this problem class the NRED1.2 method is the most effective method, and it has a much higher reward than each of the other methods. The performance of this heuristic drops away dramatically as the arrival rate decreases, and it is therefore not likely to be robust enough to be of use for problems where we are not certain that the arrival rates will be high. The method also becomes less effective as the delay penalties decrease, as with lower delay penalties the rejection of customers is not as critical, since poor customers can more easily be rejected at a later stage.

The NRED0.6 method is significantly more effective than MAXGAIN for two of the problem classes, each having high arrival rates. This method is the only one that is significantly more effective than MAXGAIN for any of the problem classes with low or medium arrival rates, and this is with the medium arrival rate, high delay penalties and reward variability. The NRED0.6 method obtains the highest overall mean, but it is significantly less effective than the MAXGAIN method for six of the problem classes, with a total of five of these having low arrival rates and three having low delay penalties. The RREW0.6 method appears to be an effective, general method, and it is significantly more effective than MAXGAIN for two of the problem classes. This method doesn't obtain as high reward as the other rejection methods for the problems with high arrival rates, high reward variability and high penalties, but the advantage appears to be in its robustness. With low reward variability this method appears to be the most effective, which we again believe is due to the greater appropriateness of the insertion time criterion for these problems where the routing is of major concern. Also the effectiveness of this method doesn't deteriorate as quickly as for the other methods, as the arrival rate decreases.

The static OPTW method is very poor in this case and, for example, it is significantly less effective than MAXGAIN for all 18 problem classes. This method includes, on average, 14.7 of the 39.7 customers that are available with the high arrival rates, 10.0 of the 20.2 customers available with the medium arrival rates and 6.4 of the 9.8 available customers with the low arrival rates. Therefore, even with the low arrival rates, there are still a number of customers that are unable to be serviced and the penalties incurred prevent this method from being effective for these problems. The incompatibility of the time windows restricts the number of

Prob Types		Methods				
Pen	(Arr,RV)	NRED1.2	NRED0.6	RRAT0.3	RREW0.6	MGAIN
0.5	(1,1)	0.607456	0.571044	0.446591	0.529387	0.431082
	(2,1)	0.427613	0.515977	0.487046	0.487046	0.487046
	(2,1)	0.418140	0.418140	0.078409	0.418140	0.508899
	(1,2)	0.601821	0.490120	0.519899	0.542569	0.453750
	(2,2)	0.492400	0.622827	0.365255	0.591216	0.461434
	(3,2)	0.468770	0.580248	0.390041	0.580248	0.601366
0.2	(1,1)	0.581912	0.589157	0.530095	0.581431	0.581431
	(2,1)	0.580637	0.591317	0.525055	0.592110	0.620652
	(3,1)	0.418140	0.418140	0.078409	0.418140	0.578590
	(1,2)	0.622557	0.513338	0.508589	0.552535	0.508589
	(2,2)	0.492400	0.597441	0.560162	0.622290	0.594692
	(3,2)	0.468770	0.629235	0.353605	0.499358	0.641628
0.05	(1,1)	0.516833	0.558359	0.665717	0.622865	0.673153
	(2,1)	0.517449	0.659746	0.522808	0.541824	0.646898
	(3,1)	0.418140	0.418140	0.078409	0.418140	0.700334
	(1,2)	0.627588	0.581447	0.671456	0.591838	0.675745
	(2,2)	0.526068	0.531316	0.599709	0.610882	0.585324
	(3,2)	0.413540	0.623076	0.384262	0.623076	0.623076

Table 9.15: Worst-Case Performance for the DOPTW - Narrow TWs

customers that can be serviced, which causes the ineffectiveness.

We also consider the worst-case performance of these heuristic methods, with the lowest proportion of the maximum objective value obtained for a particular problem instance, given in Table 9.15. The table shows that the RRAT0.3 method obtains some poor solutions to problems with low arrival rates, and this suggests that this method is inappropriate for these problems. The NRED1.2 method appears to be particularly effective in avoiding poor solutions with high arrival rates, particularly with higher levels of the delay penalties. The highest minimum proportion found is for MAXGAIN, but this is due to the other methods obtaining some poor solutions to problems with low arrival rates. This method also appears to be particularly effective for problems with low delay penalties, and it seems that the most effective policy for problems with these low penalties is to accept all customers and to exclude customers at a later stage, if more profitable customers arrive.

9.4 Same Day Dynamic Subset Selection

We now consider models for problems where the customers are arriving while the servicing is taking place, as is the case with same day customer service. These problems are more similar to the form of dynamic vehicle routing problem that has previously been studied. Whereas most previous problems have required all customers to be serviced, we allow customers to be excluded from service, and this enables feasible solutions to be obtained for problem instances for which there are greater numbers of customer requests than can be handled by the server.

If there are constant rewards that are received for servicing the customers, then we have another dynamic version of the OP, but the static version of this problem is the Orienteering Problem with release times (or one-sided time windows). Different situations occur if we have customer reward being a function of the time at which customers are serviced, and we now define this problem which we call the Dynamic MCPTDR (DMCPTDR). This work is motivated by the similarities between decreasing reward functions and the penalties for delayed service that are a feature of the static DMP and TRP, and the DTRP as defined by Bertsimas and van Ryzin [14].

We define the DMCPTDR to consist of a problem where:

- customers arrive for service according to some given distribution
- the site where each request is carried out is located within a given section of the Euclidean plane
- each customer has an associated reward function, which is a function of the time between the customer's request being lodged and the time the customer is serviced
- each customer may have a penalty that is incurred if the customer is not serviced, and may correspond to the time at which the customer is notified of their non-service
- there is a given time limit over which the server is available, and the objective is to maximize the net reward that is collected for service within this time limit

We see that, if the time limit is effectively infinite, there are high penalties for non-service and each customer's reward function is linearly decreasing at the same rate, then the objective for the DMCPTDR will be the same as for the DTRP, i.e., minimize the sum of the delay times or time in the system while servicing every customer. A practical additional problem is one where the delay of service is penalized differently for each customer, in which case the objective is to minimize the weighted sum of the penalties. The difference between this problem definition and the DTRP, however, is that for the DMCPTDR we do not consider there to be service times. For the DTRP, the behaviour of the problems has been given as a function of $\rho = \lambda\bar{s}$, where λ is the customer arrival rate and $\bar{s}$ is the mean service time. For the system to be stable we require that the average servicing time does not exceed the inter-arrival time of customers, i.e., $\bar{s} + \bar{t} \leq \frac{1}{\lambda}$, where $\bar{t}$ is the average time of travel in a solution, and is method-dependent. Without service times, $\rho = 0$ and the system is stable if $\bar{t} \leq \frac{1}{\lambda}$.

For the DMCPTDR, we do not require that all customers are serviced, and so there is no requirement of stability. This enables more general problems to be considered, with the established theoretical results being no longer valid. The imposition of a binding time limit will also create different behaviour, as the attractiveness of a customer becomes a consideration in determining whether it will be serviced or not, whereas when all customers are serviced we are merely interested in the best time to service the customer.

We now consider how the results obtained by Bertsimas and van Ryzin [14, 15] can be applied to the DMCPTDR. Bertsimas and van Ryzin showed that an optimal policy for very light traffic was to service the customers in the order they arrived, returning to the median location when no customers are available. This result is intuitive, as being located at the median will minimize the time needed to travel to the next customer, and for optimality to be achieved we require that the server always reaches the median location before the next customer arrives. This policy would also be optimal in these conditions for the DMCPTDR, regardless of the rates at which the rewards are declining as, with very light traffic, the objective would be to minimize the average delay time of all customers. This leads to a necessary addition to our solution methods, *viz*: if the vehicle is empty we return to the median of the service area, which for the problems we consider is the location of the depot.

For heavy traffic, a policy which was proven by Papastavrou [131] to be asymptotically optimal, involves creating sets of customers within radial sections of the service area, and servicing the customers within each region in a first-come, first-served manner. This policy is not one that is practical for the DMCPTDR, as its optimality relies on all customers being serviced and will therefore not be effective for problems where we only service a subset of the customers. This policy is again one that has been devised for its tractability rather than its practicality, and it is unlikely to be useful in a practical method.

Since the existing theory for the DTRP is not applicable to the DMCPTDR, we develop our own solution methods for these problems. Previous practical methods for dynamic vehicle routing problems have fallen into two broad categories:

- restart techniques, where whole routes are re-created whenever a new customer request arrives
- local search techniques, where, if possible, a new customer is included within an adaptation of the current solution routes

The advantage of local search techniques is that they are generally quicker and can therefore be used for situations where there are frequent customer arrivals. The disadvantage is that these techniques are largely myopic, with the effectiveness of a new customer only assessed within the context of the current solution. The restart techniques are generally more effective with respect to the objective function, but the disadvantage is that a lot of computational effort is put into creating routes that may bare little resemblance to the routes that are actually carried out once further information has been obtained.

The approach that we are taking is to use fast local operations, i.e., the STEEPEST ASCENT method previously applied in Chapter 8, to update the current solution. This method thus enables quick assessment of the viability of the new customer, which enables decisions of acceptance or rejection of the customer to be made almost instantaneously. This is one of the major requirements that we have for dynamic subset selection problems.

Computational Results

We develop a simulation environment for the DMCPTDR over which to test the effectiveness of our solution methods. We generate 100 layouts, each consisting

of 200 locations uniformly distributed over the Euclidean square $[0, 1.5]^2$, with the depot located at the centre of the square. We randomly generate customer arrivals from a Poisson process, with given arrival rates, where for each customer we randomly assign one of the unused locations to be the customers location. We simulate the operation of a single vehicle during the course of an eight hour day, where the travel time, in hours, between locations is given by the Euclidean distance and the customers are able to arrive for service at any stage during the first six hours of operation.

We use three different arrival rates in our simulation, with these being 10, 5 and 2.5 customers per hour, and we found through preliminary testing that these arrival rates enabled respectively few, many and most of the available customers to be serviced within the vehicle route. We also use two different distributions of the maximum reward available, with these generated from the distributions $U(10, 100)$ and $U(10, 20)$ respectively. The reward for each customer is linearly declining in the deviation between the time the customer's request is lodged and the time that service is carried out, and we consider two distributions of the rate of decline in the customer's reward. These are given as a proportion of the customer's maximum reward, with each customer's proportion derived from the distributions $U(0, 0.2)$ and $U(0.25, 0.5)$ respectively. Therefore we obtain 12 classes of problems, through the different combinations of arrival rate, reward variability and variability in the decline of the rewards, which gives a total of 1200 problem instances.

We develop some heuristic methods for two different variations of the problem. In the first instance, we consider the case where there are no penalties for excluding a customer from service at a later date. This problem eliminates the need to guarantee service to any of the customers, so at any stage, all unserviced customers with positive reward value are candidates to be the next customer serviced. The second model we consider is where no customers may be excluded from service, so the decision of whether or not to service the customer must be carried out as soon as the customer request arrives. Another scenario, where there is a finite, positive penalty for excluding a customer after it has been provisionally accepted for service, is left for future consideration.

Algorithm 9.2 function SAME-DAY APPEND

```
// Initialize the solution route, T
T ← {0}
if (Next event is the arrival of customer i) then
    if (vehicle is currently empty) then
        Add customer i to the vehicle route
    else if (method allows for preemption) then
        if (servicing i next, is better than servicing current) then
            numavail ← numavail + 1, avail_numavail ← current
            Set current = i
        else
            numavail ← numavail + 1, avail_numavail ← i
        end
    else
        numavail ← numavail + 1, avail_numavail ← i
    end
else if (Next event is the servicing of customer current) then
    gain ← 0
    for (j = 1 to numavail) do
        δ_j ← the gain obtained by servicing avail_j next
        if (δ_j > gain) then
            gain ← δ_j, best_j ← j
        end
    end
    if (gain > 0) then
        Set current = avail_best_j, i.e., Head vehicle towards this customer
        numavail ← numavail − 1
        for (j = best_j to numavail) do
            avail_j ← avail_j+1
        end
    else
        Head vehicle towards the median location
    end
end

end
```

Case I: No Exclusion Penalties

We develop a number of solution methods for problems with no exclusion penalties. These methods are based on two separate approaches:

- append methods: we only consider the next customer that will be serviced, and all other candidate customers are stored in a list. The general form of these methods are described in Algorithm 9.2. Thus we either service the arriving customer or we put it in a list with the other unserviced customers. We consider adding an unserviced customer once we've finished servicing the current customer.

- route methods: we store all customers that are currently scheduled in a service route, with all other candidate customers included in other feasible solution routes. The general form of these methods are described in Algorithm 9.3. Thus we carry out only the first solution route, with the remaining routes used to group together the remaining customers, in order to enhance the effectiveness of the inter-route improvement routines.

With both types of methods, if there are no eligible customers when the service vehicle completes the service of a customer, then the vehicle heads towards the median of the service area (which, in the cases considered, is the location of the depot). When a customer request arrives, the solution route is updated, and, if required, the vehicle will wait at the median location until the next customer request is accepted.

The append methods that we consider are:

- B4BApp — we add the unserviced customer with the highest ratio of reward that will be claimed if the customer is included next, to the travel time from the current location of the vehicle. We only add a customer once we have finished servicing the currently scheduled customer, with any customer whose request arrives while the vehicle is busy, being added to the list of available customers.

- AggApp — we calculate the aggregate score $R(i)$, for each candidate customer i, which is given as $R(i) = \sum_{j \in S} r_{ij} * e^{-\mu t_{ij}}$, where r_{ij} is the reward that will be received for servicing customer j after customer i, t_{ij} is the travel time between the customer locations and S is the set of customers that can

Algorithm 9.3 function SAME-DAY ROUTE

```
// Initialize the solution route, r_1
r_1 ← {0}, size_1 ← 0, numroutes ← 1
if (Next event is the arrival of customer k) then
    go_on ← 1
    for ((i = 1 to numroutes)&(go_on)) do
        best ← 0
        for j = 1 to size_i + 1 do
            δ_j ← the change from adding k in i in position j
            if (δ_j > best) then
                best ← δ_j, best_j ← j
            end
        end
        if (best > 0) then
            go_on ← 0
            Add customer k to the route i in position best_j
        end
    end
    if (go_on > 0) then
        numroutes ← numroutes + 1, r_numroutes ← {0, k, 0}
        size_numroutes ← 1
    end
    Apply STEEPEST ASCENT to the routes
    Head vehicle towards the first customer in r_1
else if (Next event is the servicing of a customer from r_1) then
    Remove customer from r_1, i.e., size_1 = size_1 − 1
    if (r_1 is now empty) then
        Head vehicle towards the median location
    else
        Head vehicle towards the next customer in r_1
    end
end

end
```

be serviced after i has been included. The weighting factor μ is a user-defined parameter, which we found was most effective if set to a value of approximately 2. We append the customer with the highest ratio of aggregate score to the time from the current location and we again only alter the course when we service a customer.

- PREAGG — we carry out the same method as AGGAPP, except that when a new customer arrives for service we select to either service the currently scheduled customer, or to service the newly arrived customer. This allows the service to be preempted if a more attractive customer arrives. We include the new customer if its ratio of aggregate score to travel time from current location exceeds that for the currently scheduled customer, where the aggregate score is calculated from the current position. If the new customer is the better customer according to our criterion, we remove the currently scheduled customer and add it to the list of available customers. We again use the value of μ of 2.0.

- PRENEAR — the service vehicle travels towards the nearest non-serviced customer that will have positive reward when it is serviced. If a newly arriving customer is closer to the current location of the service vehicle, the vehicle changes course and heads towards the new customer.

For the routing methods, we add all customers to vehicle routes, with the solution consisting of the first route, i.e., the other routes are 'dummy' routes, that are just created in order to group together the customers that are not included in the current solution, so that the inter-route improvement routines may be applied. When a new customer request arrives, we update the current location of the solution vehicle and create a dummy node, corresponding to this location, which is considered to be the initial node in the first route. The new customer is considered for insertion in each of the routes, successively, and it is added to the first route where the insertion can be feasibly implemented and the inclusion leads to 'improvement' according to the criterion of the method. If the customer cannot 'improve' any of the current vehicle routes, a new dummy route, that just includes this customer, is created. The routing methods we consider are:

- B4BTOUR — we add the newly arriving customer to the first of the routes where increased reward is obtained. The customer is added to this route in

the position where the ratio of additional reward to additional travel time is maximized.

- AppendTour — we append the new customer to the first eligible route where the inclusion of the customer leads to increased reward for the route.
- CheapTour — we add the new customer to the first eligible route, where the insertion position is where the least additional travel time is added.
- MaxTour — we add the new customer to the first eligible route where increased reward can be obtained, with the insertion occurring in the position where the maximum change in reward is obtained.

The results of the testing over the 1200 problem instances are shown in Table 9.16, where Methods 1 through 8 are respectively B4BApp, AggApp, PreAgg, PreNear, B4BTour, AppendTour, CheapTour and MaxTour. The row entitled 'Unimp' gives the mean overall reward obtained by the methods, without the application of Steepest Ascent, while the row entitled 'Total' gives the mean final reward, where the routing methods include Steepest Ascent being applied each time a new customer is inserted in the routes. The column 'Types' gives the problem types we are considering, where A refers to the arrival rate (A = 1 has arrival rate 10.0, A = 2 has arrival rate 5.0 and A = 3 has arrival rate 2.5), R refers to the reward variability (R = 1 is high reward variability while R = 2 is low reward variability) and D refers to the rate of decline of the rewards (D = 1 has the low decline while D = 2 has greater decline).

From Table 9.16 we see that the routing methods, with Steepest Ascent applied, are clearly more effective than the append methods. For example, if we test PreAgg (the append method with the highest overall mean) against the improved solutions found by B4BTour (the tour method with highest overall mean), we find using the Wilcoxon Signed Rank Test to a 5% level, that the B4BTour method is significantly more effective for all 12 problem classes. We see from the low unimproved mean rewards for these routing methods, that it is not the effectiveness of the construction methods that leads to the improved results, rather it is the effectiveness of the improvement methods. For example, the AppendTour method obtains very poor solutions without the application of the improvement routines, but the mean improved reward is still much higher than that obtained by the append methods.

Types	Methods							
(A,R,D)	1	2	3	4	5	6	7	8
(1,1,1)	1533.54	1587.58	1638.02	1519.18	1744.79	1729.46	1740.83	1744.31
(1,1,2)	921.52	942.48	1044.69	935.65	1085.40	1092.48	1091.62	1079.03
(1,2,1)	393.05	414.01	427.50	408.39	449.87	450.27	448.35	449.30
(1,2,2)	232.28	245.66	263.97	247.94	275.80	272.82	274.61	275.92
(2,1,1)	953.88	989.29	1031.78	968.78	1091.66	1072.27	1088.02	1090.93
(2,1,2)	581.24	599.39	671.36	609.11	700.21	696.77	698.52	699.84
(2,2,1)	254.94	265.10	276.40	265.79	288.11	286.29	287.43	287.37
(2,2,2)	151.81	156.13	178.14	166.88	181.24	180.72	181.60	181.43
(3,1,1)	591.18	599.70	614.02	597.03	655.44	656.38	655.10	655.48
(3,1,2)	357.20	360.65	415.79	380.34	439.57	438.61	439.99	439.65
(3,2,1)	150.73	152.36	159.76	157.12	169.17	166.78	169.55	169.13
(3,2,2)	99.59	100.58	111.88	109.36	118.48	118.29	118.39	118.46
Total	518.41	534.41	569.44	530.46	599.98	596.76	599.50	599.24
Unimp	518.41	534.41	569.44	530.46	438.52	198.24	414.92	405.81

Table 9.16: Performance for the DMCPTDR - No Penalties

Of the append methods, PREAGG is clearly the most effective, and its superiority over AGGAPP indicates that the ability to divert from the currently scheduled customer when a superior customer arrives, is an important factor for creating reasonable solutions. These append methods, which use the aggregate scores to reduce the myopia involved, are still unable to compete with the improvement methods, which explicitly take into account all customers when making each decision, and so for this type of problem with no penalties for excluding customers, we would recommend the use of a routing heuristic. The total computational time required for the 1200 problem instances was approximately 6 minutes for each of the append heuristics while each of the routing heuristics took approximately 55 minutes to run. This corresponds to under 3 seconds to solve each problem instance, and so the requirement of minimal computational time appears to have been met by these methods.

In order to test the absolute effectiveness of these methods, we found a solution to the static version of this problem, i.e., the MCPTDR with time windows and strictly decreasing rewards. The method we used was the REWAPP method, that we previously applied to the MCPTDR, and the mean rewards for the 12 problem classes were 1870.12, 1328.16, 480.97, 333.33, 1175.79, 874.81, 310.95, 226.44,

715.58, 560.84, 188.23 and 150.47, with an overall mean of 684.64. These results show that, as we would expect, the discrepancy between the dynamic and static solutions is greatest when the rates of decline in rewards are higher.

The dynamic routing methods appear to be quite effective for the versions of the DMCPTDR considered here, as the difference between the static and dynamic solutions appears to be quite low. The requirement of decision-making in real-time will obviously lead to less effective solutions being found, but we feel that the difference in this case is acceptable, and therefore the adaptation of the static methods appears to be an adequate technique for this problem.

In order to more clearly test some of the attributes of the behaviour of the heuristics, we also create a second set of test problems, with non-uniform reward distribution. The distributions we consider are:

1. Customer rewards are uniformly distributed between 10 and 100, but there is a 1% probability that a reward will be uniformly distributed in the range (5000,10000). This problem set tests a heuristic's ability to cope with unusual customers which are highly attractive.

2. The reward for a customer is given as a linear function of the time at which the customer becomes available, with the rewards being given in the range (0,100). This tests for the effectiveness of the heuristic for dealing with early customers that are relatively unattractive.

3. A customer's reward is taken from a linear function of the travel time from the depot, where these rewards are again scaled to be in the range (0,100). This creates a form of equality where greater reward is obtained from the more distant customers.

We again use the three arrival rates and the two distributions of the variability in the decline of the rewards, as considered previously. Together with the three reward distributions, we obtain a total of 18 problem classes, for which we generate 50 problem instances, giving a total of 900 problem instances. The results of testing the heuristics on this data are given in Table 9.17.

We see again that the routing heuristics in general outperform the append heuristics. Again if we test the differences in the effectiveness of the method of each type with the highest overall mean, i.e., PREAGG and CHEAPTOUR, we find that CHEAPTOUR is significantly more effective for 17 of the 18 problem classes

Types	Methods							
(A,R,D)	1	2	3	4	5	6	7	8
(1,1,1)	4459.86	4530.28	4625.05	3632.31	4381.71	4522.31	4511.85	4492.78
(1,1,2)	4069.08	3886.90	4215.28	2269.30	4535.25	4554.09	4647.32	4593.15
(1,2,1)	1195.35	1194.24	1250.80	1185.43	1344.42	1341.28	1350.25	1352.92
(1,2,2)	777.99	818.87	856.50	813.04	920.37	908.23	922.14	920.32
(1,3,1)	1378.29	1365.62	1424.34	1357.78	1609.67	1589.92	1604.56	1607.90
(1,3,2)	796.19	795.83	844.06	793.98	932.19	920.25	938.14	929.66
(2,1,1)	3202.04	3384.83	3276.07	2634.55	3372.84	3384.88	3389.72	3391.31
(2,1,2)	1850.65	2135.16	2416.67	1274.10	2494.07	2462.57	2503.88	2460.09
(2,2,1)	739.18	732.78	779.72	728.10	855.46	851.94	856.27	858.54
(2,2,2)	479.84	493.97	565.07	512.44	609.16	617.63	608.43	610.49
(2,3,1)	885.59	899.33	930.81	906.37	988.43	987.15	988.42	985.80
(2,3,2)	530.14	535.51	596.85	557.81	624.56	624.19	625.13	621.56
(3,1,1)	1880.33	1887.01	1943.96	1888.50	2192.22	2192.81	2192.37	2192.19
(3,1,2)	1184.87	1231.29	1381.64	995.20	1461.36	1460.65	1459.22	1462.73
(3,2,1)	473.64	480.32	505.29	475.18	549.34	542.77	548.85	547.17
(3,2,2)	308.70	314.49	386.66	354.27	398.14	403.50	397.13	398.39
(3,3,1)	589.50	595.18	605.55	587.79	646.65	636.92	646.59	646.65
(3,3,2)	327.07	331.82	377.11	361.70	393.49	389.80	391.43	393.49
Total	1396.02	1422.97	1498.97	1184.88	1572.74	1577.27	1587.87	1581.40
Unimp	1396.02	1422.97	1498.97	1184.88	998.26	367.89	885.76	907.24

Table 9.17: Performance for the DMCPTDR with Non-uniform Reward - No Penalties

(when tested by the Wilcoxon Signed Rank Test at a 5% level). The tour methods are again very ineffective without the application of the improvement routines.

One interesting feature is that, with the highest arrival rates and with the first distribution of rewards, the tour methods all obtain higher mean rewards for the problems with the higher values of the rate of decline of the rewards, whereas the append methods all obtain lower rewards in this case. This shows that there happens to be a greater number of customers with the high reward values, for the problems with the higher rates of decline, and that the tour methods are better able to handle this situation through accommodating these crucial customers within the solution route.

We again found a measure of the absolute effectiveness of the methods through applying the static REWAPP method on these problems. The means obtained by this method for the problem classes are 5095.90, 6319.10, 1485.91, 1158.40, 1649.91

and 1158.79 with high arrival rates, 3762.09, 3237.42, 970.03, 787.06, 1085.75 and 797.58 with medium arrival rates and 2365.12, 2017.57, 625.67, 535.07, 721.53 and 524.80 with low arrival rates. We see that, for the problem classes that have high rates of decline and the occasional very large reward, the static mean obtained is far greater than that obtained by our dynamic methods, but this is an unavoidable problem trait, that cannot be overcome without having some prediction of future events. The dynamic tour methods again appear to be effective for these problems, and they appear to be effective for any form of the reward functions tested. Therefore for any dynamic problem with decreasing rewards and no penalties for excluding a customer from service, we would recommend the use of tour heuristics for finding a good solution to the current problem. More advanced solution methods will lead to better results being obtained, although the methods are required to work quickly enough to allow the routes to be updated in between the times new arrivals occur.

Case II: Infinite Penalties

We consider a variation of the single vehicle DMCPTDR described above, where there are infinite penalties for a customer being excluded from service at a later stage. This problem requires that customers be quickly processed as soon as they arrive, with notification then given as to whether or not the service will be carried out. We assume for this problem that there is no limit on the decline in the reward received, so a customer that has been scheduled can be serviced even at a time for which its reward received is negative. We feel that this is an important requirement to allow some flexibility of the routing, although consideration should be given to the level of service that is afforded, as this may allow unacceptable service. Different reward functions may be more appropriate, e.g., increasing rates of decline of reward as deviation from the requested time increases, but we are considering constant rates in this problem.

When considering solution methods for this problem, there are a number of features to note. Once a customer has been accepted for service, the service is guaranteed, and we consider that a customer needs to be processed immediately. Therefore it is not possible to provisionally include a customer, so there is no advantage to storing the eligible customers in a list. This was how we dealt with customers in our append methods with no delay penalties, so these append methods

cannot be directly applied to problems with infinite penalties. One possible update of these methods is to just consider one customer in the solution route at a time, but the problem with this method is that, for the test problems we consider, the customer arrivals take place over the interval (0, 6 hours), while the route is allowed to have a duration of up to 8 hours. It is therefore highly likely that the service vehicle would be unutilized at the end of the time period in this case. Another important feature that we found with append methods with no penalties, is the ability for preemption of service when a new customer arrives. Since the customer that is currently being headed towards will still require scheduling in this case, we need to create partial routes in order for the method to allow preemption while still servicing the currently scheduled customers. In this case, the append method is similar to a tour method and, because of this feature and the other mentioned difficulties for implementing append methods for these problems, we restrict our attention to tour methods.

For solution methods for these dynamic problems, we are unable to predict future events, but we believe that being able to allow for these later events is an important device that will allow more effective solutions to be obtained. Therefore we do not want to over-allocate resources, i.e., the vehicle's time, to customers that are not sufficiently profitable (at the expense of future events). Since there are a number of cases that may occur with this problem, with different scenarios promoting different methods, we seek to empirically identify some strengths and weaknesses of the approaches under some known situations. This will enable the selection of a given routine for specific circumstances that occur in practice, but we are also looking to obtain general robust techniques that will be effective for a broad range of problems, and can therefore be applied to problems where the situations are known with much less certainty.

For our methods, we insert the newly arriving customer in its best position in the solution route, or place it in a second route (if no improving position is available). An adapted version of Steepest Ascent is then applied to the routes, where we consider only the intra-route improvement routines and one version of the General Exchange routine (Move and Cross are subsets of General Exchange when there is only, at most, one customer outside of the solution route). Since no customer can be excluded from the solution route, we only consider the case where the customer is removed from the solution vehicle, the customer from the

second route is put in the solution vehicle, and the original customer is repositioned within the solution route. We accept the new solution as our incumbent solution if the solution sufficiently improves upon the current one, where the definition of 'sufficient' is the major component of the heuristics. The methods that we consider are:

- B4BTOUR — each customer is placed in the position where it maximizes the ratio of additional reward to additional time required, and Steepest Ascent is then applied. We accept the customer if the reward for the solution is increased. This method relies on the current knowledge of customers, and assumes that the knowledge we do have is far more important than speculative information of future events.

- THRESHREWARD — each customer is placed as above, and the inclusion is accepted if the change in reward in the improved solution exceeds a given threshold. We found through testing that this threshold is most effective if it is quite low, and so we selected a required reward change of $min + 0.1 * (max - min)$, where min and max are respectively the minimum and maximum initial rewards that may be received within a given problem. This method ensures that customers that only marginally improve the reward are not included, and it requires some knowledge (or estimate) of the range of the extreme reward values that may be available.

- THRESHB4B — the customers are placed as above, but the move is only accepted if the bang-for-buck ratio of change in reward divided by change in time, exceeds a given threshold. If the vehicle is empty, the customer is accepted if its bang-for-buck ratio exceeds the minimum of the initial available rewards divided by our estimated average insertion time; this seems to be a reasonable approach, since the minimum reward received is positive in the problems we consider. We adapt the improvement routines so that each method carries out the move that maximizes the ratio of reward change to time (rather than maximizing the reward change as is usually the case), and this is an attempt to consider the same quantities in the improvements as in the evaluation of the solution. This threshold was taken to be a given proportion of the reward available scaled by some measure of the time, and

much different behaviour was obtained with different thresholds. We consider here the thresholds to be of the form $(min + k * (max - min))/1.5$, where we implement the cases with $k = 0.3$ and 0.7. This method therefore considers both the reward and time changes in evaluating the effectiveness of a customer, and prevents adding a customer with high reward that causes unduly large additional travel times and thereby prevents future insertions from being possible.

- ARTIFTOUR — we create artificial time windows on customers and again add the customer with highest ratio of reward to time change. We set the time windows to ensure that a given proportion of the maximum available reward is received, and found that the minimum proportion of reward that is acceptable needed to be quite a low value, or else too many customers would be excluded. This is due to the rewards declining immediately from the time the customer's request is received, but with advanced requests we would probably be better suited to ensure greater rewards are collected. The latest time a customer could be serviced was set so that customer i receives a reward of at least $0.3 * max_i$, where max_i is the maximum reward available for this customer. Any customer that can be feasibly included and that leads to a reward being increased by at least $min + 0.2 * (max - min)$ is accepted. This method operates by ensuring that some guaranteed level of service is afforded, and it requires that a given level of improvement is attained.

We tested these methods on the 1200 problem instances described previously, and the results are given in Table 9.18. The methods are respectively B4BTOUR, THRESHREWARD, THRESHB4B with parameters 0.3 and 0.7, and ARTIFTOUR. The mean reward obtained by each method for each problem class is given in the Table, with the bottom three rows giving the overall mean reward, the average numbers of customers served and the total number of customers serviced such that negative reward is obtained.

We see that the least effective method overall is the ARTIFTOUR method, although it obtains the highest mean reward with high arrival rates and reward variability, and low values of the decline in reward. In this case, it appears that the threshold of required reward and the restriction on the minimum reward received from a serviced customer, leads to a more efficient use of the vehicle's time. Thus, a higher mean reward is received, although fewer customers are serviced than for

Types	Methods				
(A,R,D)	B4BT	ThRe0.1	ThRat0.3	ThRat0.7	Artif
(1,1,1)	1369.77	1433.01	1314.55	1411.43	1453.01
(1,1,2)	839.57	900.94	900.92	970.17	828.31
(1,2,1)	348.85	345.31	337.36	348.88	325.97
(1,2,2)	216.08	201.46	244.47	254.02	185.70
(2,1,1)	946.28	972.11	948.30	949.71	956.58
(2,1,2)	596.12	618.98	640.83	651.64	581.71
(2,2,1)	249.97	239.16	253.71	259.02	223.85
(2,2,2)	146.39	134.78	166.43	164.18	121.61
(3,1,1)	613.46	622.82	615.12	587.19	603.61
(3,1,2)	398.12	405.04	417.56	403.90	391.07
(3,2,1)	154.07	143.14	153.96	148.07	130.11
(3,2,2)	94.39	84.97	106.10	102.26	74.71
Total	497.76	508.48	508.28	520.87	489.69
Num Served	18.55	16.89	20.33	19.16	14.91
Negative	505	240	567	219	0

Table 9.18: Performance for the DMCPTDR - Infinite Penalties

the other methods. The mean number of customers served by this method is the lowest of the methods for each of the problem classes, as the imposition of time windows will restrict the allowed routing options. The time windows ensure that all selected customers are serviced while they have positive reward, and this leads to better customer service, but also results in lower values of the objective function considered. The method is less effective with the higher values of the rates of decline of rewards, as the time windows imposed are prohibitively narrow in this case.

The other three methods that reject some customers (the implementation of ThreshReward and the two implementations of B4BTour), result in a higher overall mean than the method that includes any feasible, improving customer. The implementation of the ThreshReward method, with the rejection criterion based purely on the additional reward received, obtains a higher mean reward over that obtained by B4BTour for all six problem classes with higher reward variability, whereas it obtains lower means for all six classes with low reward variability. This suggests that the rejection criterion used is more appropriate for problems with higher reward variability, as with lower reward variability there is less difference in the attractiveness of the customers based on the rewards, and so the travel

time for a customer is a more important concern. This method services the least number of customers of all the methods, apart from ARTIFTOUR, and the lack of consideration of travel times appears to reduce the numbers serviced which adversely affects the effectiveness of the method. An advantage, however, is that without including customers with low rewards, there are fewer marginal customers whose reward becomes negative when later customers are included.

The two implementations of the THRESHB4B methods (which rely on bang-for-buck ratios for rejection) appear to be effective in creating good solutions to these problems. Thus we compare the effectiveness of these implementations, with each other, as well as with the implementation of the B4BTOUR, via the Wilcoxon Signed Rank Test at a 5% level. In comparing THRAT0.3 and THRAT0.7, we see that the THRAT0.7 method is significantly more effective for the two problem classes with high arrival rates and high reward variability, whereas the THRAT0.3 method is significantly more effective for three problem classes, each of which have low arrival rates. Therefore, rejecting greater numbers of customers leads to improved solutions with high arrival rates (as the focus is on servicing customers that are the most beneficial) while with low arrival rates it is better to simply exclude the worst customers and thus too many customers are rejected with the higher parameter setting. The rejection of few customers by the THRAT0.3 method, leads to greater numbers of customers with negative rewards, which is a disadvantage for problems where the customer service is of major importance.

When we compare these two implementations against the B4BTOUR method via the Wilcoxon Signed Rank Test at a 5% level, we find that THRAT0.3 is more effective for five of the problem classes, with each of these having high rates of decline, whereas B4BTOUR is more effective for the problem class with high arrival rate, high reward variability and low decline rates. THRAT0.7 is more effective than B4BTOUR for five problem classes, each having high declines, and B4BTOUR is not more effective for any of the problem classes. This shows that the major advantage of the THRESHB4B methods is the increased effectiveness for problem classes with high rates of decline in the reward. For these problems, there is a greater impact of servicing poor customers, and the exclusion of greater numbers of these customers enables better solutions to be obtained.

We now consider the effectiveness of these heuristics on the other data set we previously considered in Table 9.17, and the results obtained are given in Table 9.19.

Types	Methods				
(A,R,D)	B4BT	ThRe0.1	ThRat0.3	ThRat0.7	Artif
(1,1,1)	3307.26	3147.45	2673.29	3171.49	3618.97
(1,1,2)	3475.67	3858.01	3691.28	4465.98	2509.46
(1,2,1)	783.18	783.18	819.10	970.38	851.57
(1,2,2)	625.32	625.32	634.63	813.01	620.48
(1,3,1)	1288.88	1288.88	1154.12	1243.94	1284.89
(1,3,2)	766.92	766.92	768.19	787.79	663.53
(2,1,1)	2470.18	2805.44	2302.78	2787.05	3013.51
(2,1,2)	2168.67	2352.30	2186.70	2404.93	1864.07
(2,2,1)	671.90	671.90	647.20	710.74	662.39
(2,2,2)	465.99	465.99	500.87	535.51	455.58
(2,3,1)	909.36	909.36	890.65	874.11	893.77
(2,3,2)	531.75	531.75	553.89	538.28	499.40
(3,1,1)	1709.17	1863.21	1871.15	1987.97	1850.25
(3,1,2)	1372.96	1395.87	1405.86	1457.87	1377.31
(3,2,1)	479.97	479.97	499.77	486.68	497.21
(3,2,2)	352.25	352.25	372.14	365.03	337.30
(3,3,1)	628.68	628.68	619.10	552.39	622.84
(3,3,2)	360.42	360.42	370.82	341.14	344.05
Total	1242.70	1293.72	1220.09	1360.79	1220.36
Num Served	20.18	19.53	20.44	18.39	16.57
Negative	721	592	716	222	0

Table 9.19: Performance for the DMCPTDR with Non-uniform Reward - Infinite Penalties

In these cases, we assumed that the minimum reward available for each problem instance was 0, and the maximum available was 100. If we considered the actual maximum reward available for reward type 1, which is equal to 10000, then none of the customers apart from those with the unusually high reward values will be considered. This would not be an effective approach, and so we exclude these high reward customers from consideration when we calculate the rejection criteria.

Table 9.19 shows that the ThRat0.7 method obtains by far the highest overall mean reward of the methods considered. The main advantage that this method holds, is in the problem instances with reward type 1 for which the exclusion of greater numbers of the standard customers allows greater room to include the customers with high reward values.

We illustrate this case through considering one test problem with high arrival

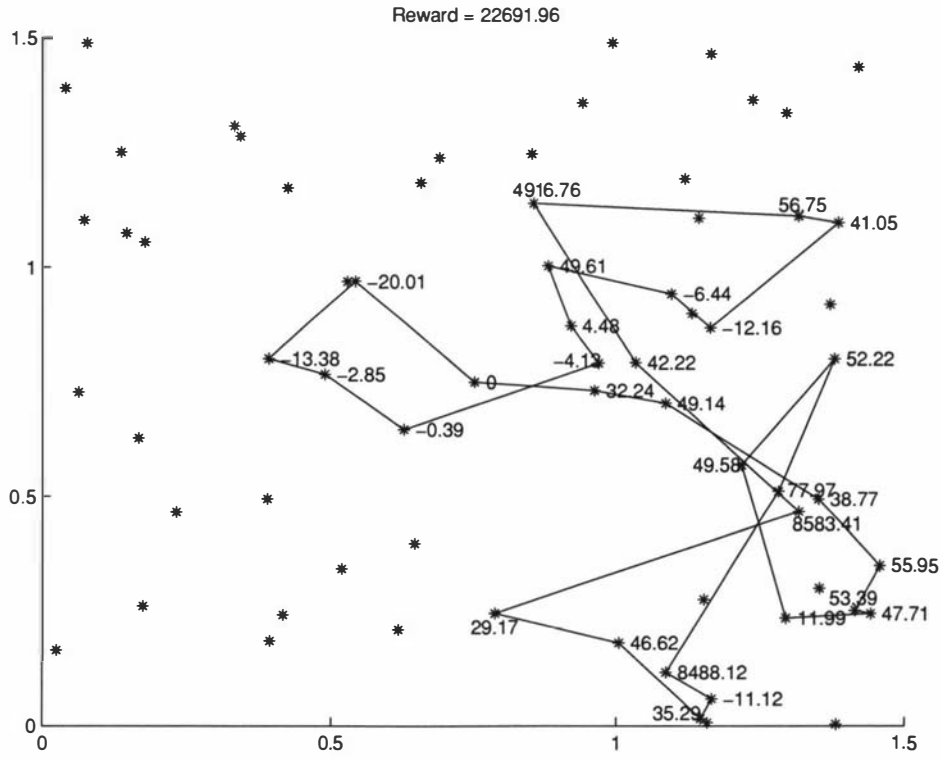

Figure 9.16: Solution Obtained by THRAT0.7 for Problem 74

rates and rates of decline, and reward type 1. The solution obtained by THRAT0.7 is given in Figure 9.16 and it has a total reward of 22691.96. The location of each of the available customers is noted by an asterisk, with the label for each customer in the tour being the reward that is obtained from this customer in the solution. The solution route begins by servicing the customers in the bottom right hand corner of the solution area, and then continues servicing the customers, with the three customers with high rewards being able to be serviced at times close to when they become available. Of note is that the insertion of the 58th customer, located at (0.8539,1.1384) and with initial reward of 5769.94, causes seven of the remaining customers to be serviced at a time when their reward is negative.

The solution obtained by B4BTOUR is shown in Figure 9.17, where the total reward received is equal to 12799.59. We see that the marked difference between the two solutions is that the B4BTOUR method is unable to include the crucial 54th customer, which is serviced with a total reward of 8583.41 in the solution obtained by THRAT0.7. The inclusion of too many distant customers leads to there being little flexibility in the routing allowed, and this prevents this important customer from being included.

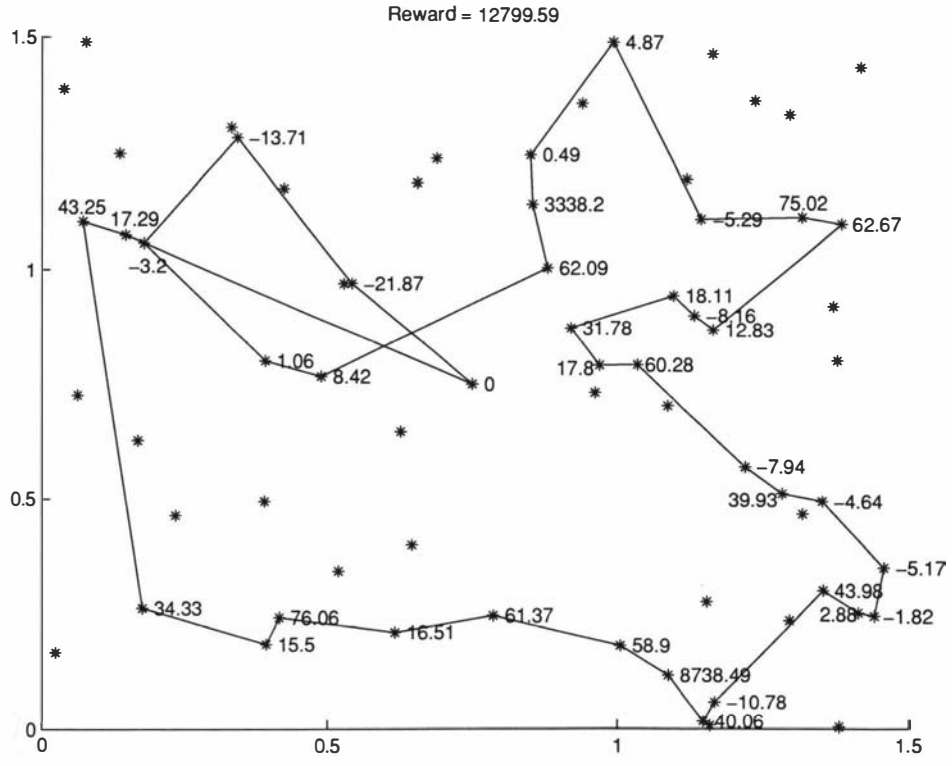

Figure 9.17: Solution Obtained by B4BTour for Problem 74

We again test the difference in the effectiveness of B4BTour and the ThreshB4B methods, by the Wilcoxon Signed Rank Test at a 5% level. Through this testing, we see that ThRat0.7 is significantly more effective than B4BTour for six of the problem classes, with each of these having the first two reward types, whereas the B4BTour method is not significantly more effective for any of the problem classes. ThRat0.7 is significantly more effective than ThRat0.3 for four of the problem classes, and seems to be the best of the methods for these problems. The B4BTour method is relatively ineffective with type 2 reward functions, as this case involves higher rewards for later arrivals, while the method looks to insert the earliest customers into the solution route.

The ArtifTour method is quite effective for the problem classes with low rates of decline in the rewards, particularly with the first reward type. This method appears to be relatively poor with high rates of decline, and so we wouldn't recommend its use for these cases. The advantage of guaranteed level of service, makes this method promising with low rates of decline, and so it is the method we would recommend in this case, particularly if the service of customers is an important concern.

We have therefore seen that there are major gains in customer service and profitability that may be made through rejecting customers from consideration, for certain problem classes for the DMCPTDR. A method that rejects many customers, according to the ratio of additional reward to additional time seems to obtain improved results, on average, compared to including all customers, for a range of problems with reasonably high arrival rates. The choice of the parameter settings for this method makes a large difference to the effectiveness of the method, so this decision needs to be tailored to the situation to be solved. A method that includes the imposition of artificial time windows is able to create improved solutions for a number of problems with low declines in the reward values, and this has the additional bonus of being able to give some guarantee of service time to a customer.

The rejection of customers that may be feasibly included in the solution route when there are no eligible customers currently available, is a deviation from the standard methodology of static routing problems. The standard approach of including the customer if it results in increased reward for the current solution, appears to be most effective for problems with uniform rewards, low rates of decline in the rewards and quite low arrival rates. For other problem classes, the rejection of eligible customers is an essential element that is required in order to create effective methods for dynamic subset selection routing problems. Thus we have identified further cases where new methods are required for dealing with dynamic problems.

9.5 Conclusions

In this chapter we have introduced dynamic versions of subset selection problems and considered some of the practical implications that arise with these problems. These problems call for the decisions of whether to include the customer or not to be made when the customer's request for service arrives. We have considered a number of different problem classes and determined some average case behaviour for different methods. All these methods were developed in order to obtain solutions very quickly, which we feel is an important requirement for practical methods.

For next day service, the specific models we considered were the DOP and DOPTW, where finite penalties are incurred for delayed notification of a customer not being serviced. For the DOP we found that the choice of appropriate techniques for rejecting customers from consideration may lead to improved results

with high delay penalties. For problems with uniformly distributed rewards and locations, an effective method was found that rejected customers purely based upon their location. This method is not appropriate for general cases and so we do not consider this to be robust enough to be considered for cases where information is not known with certainty. We found that methods based on the rejection of customers according to either their reward value or their ratio of reward to insertion time, may be effectively applied to a wide range of problems, with the selection of the appropriate rejection criteria being an important factor in the effectiveness of the methods. Better results are obtained through rejecting greater numbers of customers when the delay penalties and/or the arrival rates are higher.

With low delay penalties, we found that improved solutions may be obtained through using methods that delay the routing decisions until further information becomes available. Thus, methods incur penalties for any customers that cannot be included in the solution at the time the decisions are made, but with low delay penalties the gains in rewards outweigh the additional penalties incurred. With low arrival rates and most customers able to be serviced within the solution route, the best policy we found was to include the customers in the solution on a first-come, first-served basis, where a customer may replace others in the solution if it is beneficial to do so.

For the DOPTW we explored different practical techniques for creating solutions and we found that improved solutions could again be obtained for certain problem classes, through rejecting some customers, rather than inserting any eligible customer. These methods were of most benefit with high arrival rates and high delay penalties, and were beneficial for any the widths of time windows considered. The benefit of delayed decision making was reduced as the width of the time windows decreased, as the value of the additional information was less than the cost of delay penalties incurred. In general, for the DOPTW, there was less advantage derived from using the rejection criteria we considered, so further work in obtaining appropriate solution methods appears to be warranted.

We also considered the implications of same day servicing, and introduced a specific model for a problem we call the DMCPTDR. We considered two cases: where there was no penalty for excluding a customer from service at a later stage, and when there were infinite penalties. With no penalties, we found that methods which create routes through all the eligible customers are able to obtain what seem

to be effective solutions. With infinite penalties we found that the rejection of customers according to the ratio of additional reward to insertion time was able to find improved solutions over inserting all customers, for a number of problem classes, particularly with high delay penalties. One intriguing method that warrants further consideration involves the imposition of artificial time windows which restrict the time at which the customer is serviced. This method obtains reasonable solutions, but the major advantage is in the increased level of service that can be offered. The other methods, which obtained higher mean rewards over the problems we tested, rely on being able to poorly service some of the incumbent customers in order to be able to incorporate later customers which are more attractive.

The field of DSSRPs is one that appears to have many practical uses and, as such, more research into this area is merited. The additional considerations of the levels of customer service introduce new features to the problem, and lead to approaches that deviate from static methods being called for. Different versions of the problem involving multiple vehicles, and same day service with advanced requests and/or finite penalties may create problems for which different features of the problem need to be exploited, but we leave these as areas of future research.

CHAPTER 10

Conclusions and Recommendations

In this thesis we have investigated various models for dealing with vehicle routing problems in which not all customers need to be serviced. This is the field of subset selection routing problems, for which the prototypical problem type is the Orienteering Problem (OP). In the OP, the server can freely choose which customers to service, with the remaining customers simply disregarded. We introduced the concept of a "pure" subset selection routing problem to incorporate this situation, with "restricted" subset selection referring to situations in which some or all of the non-serviced customers are considered within the objective function of the problem. This form of problem is more relevant if customer service is an important consideration, as it is able to ensure that each customer is considered for service, with the implications of non-service being accounted for in the form of penalties in the objective function.

We studied two forms of time consideration that have been previously applied to the OP: time windows on when the customer may be serviced, and time-dependent rewards. In this way, the reward received is a function of the time of servicing, and this may be used to introduce temporal scheduling aspects, in conjunction with routing considerations. We discussed the implications of these features on the applicability of existing solution methods and considered how they may be used to model different situations.

In order to extend the scope of the OP further, we studied the effect of introducing task constraints to routing problems and discussed the implications and

practicalities involved with including general forms of tasks. Most previous research has considered tasks to consist of (origin, destination) pairs, although more general task definitions may enable different phenomena to be modelled. We created a model for the Task Routing Collection Problem (TRCP), which is a straightforward extension of the OP that includes tasks instead of single points to be visited, and we demonstrated, by use of an example, the potential improvements that may be made if the splitting of tasks is allowed. For the TRCP, the additional feature we considered was to allow tasks to consist of up to three subtasks to be visited and this required an update of existing methods to allow for forward and backward precedences for each subtask.

We created a combined model, which we call the Maximum Collection Problem (MCP), which incorporates the different methods for dealing with non-service, time constraints and time-dependent rewards and task constraints. One additional feature that arises within the MCP is in the situation where there are time-dependent rewards and task constraints. In this case, alternative forms of reward function may be appropriate and we investigated the case where the reward received is a function of both the time at which the customer is serviced and the duration for which the customer's load is in the service vehicle. This case requires two criteria to be met in order to offer acceptable service to the customer, with the reward that is received being a reducing function of the deviation from the desired service time, and the excess time for which the customer's load is in transit.

For time-dependent rewards applied to individual points, we considered two different forms of the reward function; non-increasing reward values and the piecewise linear reward function that was shown in Figure 6.2. With our piecewise linear reward function, gains in reward may be made by delaying the service of a customer in order to obtain a higher reward. In order to calculate the optimal waiting time to include within the current solution, we adapted the algorithm of Dumas, Soumis and Desrosiers [43], which had been developed for minimizing the inconvenience costs within a soft time window problem. We were able to apply this algorithm to our piecewise linear reward functions, and, after further modification, we were able to also apply the algorithm to task problems for which the reward received is purely a function of the time at which the customer is serviced, and to task problems with rewards affected by both the time service is completed and the time for which the load is in the vehicle.

We developed generic methods for solving any version of the MCP, with these adapted in order to efficiently deal with the constraints of the problem. We created an improvement routine, STEEPEST ASCENT, which contained intra-route improvement routines that were used to either reduce the distance travelled within the current route or to increase the reward received, and the inter-route improvement routines: Move, Generalized Exchange and Cross. The Generalized Exchange routine is relatively computationally expensive, but we found it to be very effective for our subset selection routing problems, as it enables intra-route improvements to take place within the search procedure. This routine incorporates the inter-route Move routine, but since we found that large increases in the computational time resulted from the Generalized Exchange routine being applied to unimproved solutions, we use the more efficient Move routine to find initial improvements, before applying the Generalized Exchange routine to create further improvements.

We developed a number of new techniques for generating problem instances, in order to obtain test problems with characteristics we identified as being important. In our preliminary testing, we focused on some properties of the problem and then randomized out a number of other features. Since there are so many possible features of the problems, we needed to concentrate on certain features in order to make any sort of headway, but the choices of which features to ignore is able to mask some of the structure of the problems. For example, during the preliminary testing for the OPTW (in Section 8.1.1), we ignored the influence of the form of the tour to which the time windows were applied, but in later testing (in Section 8.2.2) we found that our standard Tabu Search method was unable to obtain effective solutions for problems in which all the customers could be serviced and the time windows were based upon an effective TSP solution. Thus we were able to investigate the effect of the method for generating the time windows because we fixed the number of customers serviced.

We created test problems for four variants of the MCP and performed preliminary investigation into their behaviour. For the OPTW we developed problem classes according to the number of customers that could be serviced, the width of the time windows and the variability in the rewards. We found that the APPEND method, based on the subgravity approach, was able to be used to construct good initial solutions, although the method's lack of concern with placing the customer in its best position resulted in the CHEAP method, which focuses purely on the

insertion times, being the most effective with loosely constrained problems. We tested a number of different search routines on the test problems and found that our version of Tabu Search appeared to be quite successful over most of the problem classes considered. However, this method appeared to be least effective for very loosely constrained problems, and for the tightest time windows we considered, there was not enough flexibility to elicit distinguishability between the customers.

In later testing on the OPTW in Section 8.2.2, for problems for which we knew that all customers could be serviced within a single route, we found that our Tabu Search method was again least effective for problems with loose time window constraints, particularly with the constraints based upon TSP type tours. In these problems there are many options for routing the customers, although it appears to be more difficult to identify the optimal solutions. We found that the most difficult problems to solve included the time windows based upon TSP type tours and, in this case, we were able to obtain improved solutions by incorporating a TSP based construction method, which can better counter the weaknesses of the search method, rather than only using random initialization of the solution routes. For the tight scheduling OPTW problems of Section 8.2.4 our Tabu Search method was again quite effective, but we identified that improvements could be made by incorporating routines that focus on finding feasible positions to add the non-serviced customers, in this case. Therefore the introduction of temporary violations of the time window constraints would likely lead to improved results for our OPTW method.

We also carried out testing on the TRCP, a new problem which incorporates tasks within a subset selection problem. We identified three factors for testing the TRCP: the number of customers serviced, the size of the load carried and the variability in the reward function. We developed new methods for generating tasks for these problems, as most of the previous generation methods for task problems had randomly generated the tasks, thereby ignoring the potential structure present. We randomized out the methods for generating the tasks, in order to focus on the identified problem features, and in hindsight, we believe that neglecting the structure of the tasks misses some important information about the form of the problem.

Of the construction techniques we tested on the TRCP, there was little difference in the quality of the solutions obtained after the application of the STEEPEST

ASCENT method, despite very varied solutions being constructed. The CLUSDIST method, which creates clusters by combining tasks that are close together with regard to the distance required to service the tasks in a single route, seemed to be the most effective, since it created solutions that were amenable to improvement via the routines we consider.

We found in preliminary testing for the TRCP, that using insertion in the construction and improvement techniques was computationally expensive. We were able to obtain large improvements in the efficiency of our routines, at a later stage, which enabled more comprehensive testing to be carried out in Section 8.2.5. This improvement was obtained by limiting the number of times that the full neighbourhood of insertion was evaluated within the search routines. In this additional testing we found that our Tabu Search routine appears to be reasonably effective, although a higher mean reward was able to be obtained for certain problem classes, by otherwise applying a number of our construction methods to the problem instances.

Our main investigation in this section, however, was into the structure of the problems obtained, which we investigated with regard to the computational time required by our Tabu Search method. We found that, for each of the types of problems considered, the type of reward function used had a large effect on the computational time required, as longer times were necessary for problems for which the reward received was a function of the time required to service the task. With regard to the method of generating the tasks, we found that the form of the solution obtained was affected dramatically by the type of tasks available. With geographically close tasks, the subtasks of a customer were mostly serviced consecutively and, since the capacity constraint of the problem was not binding, we found that the computational time required for the problems with high loads and high numbers serviced was higher as there were larger neighbourhoods of insertion to evaluate. The loads in the vehicle were much higher when the subtasks were more dispersed, and this resulted in shorter computational times for these problems. Thus, the method of task generation may have a significant effect on the structure of the problem and the methods that should therefore be applied. With spatially close tasks, methods that place tasks consecutively, e.g., the Cross exchange routine, may be more relevant, and greatly reduce the time required compared with inserting each of the subtasks separately.

For the MCPTDR we created new types of problems by considering piecewise linear reward functions. These problems model customer dissatisfaction, where the interval over which the reward is at its maximum value corresponds to a desired service time, and the available reward is reduced as the deviation from the desired time increases. The MCPTDR had previously been defined by Brideau and Cavalier [20] to consist of purely decreasing rewards and we also considered this form of function. We include time windows on the reward functions, as this enabled a wider range of problems to be defined, and also enabled the slopes of the reward functions to be directly determined from the time windows.

In our preliminary testing on the MCPTDR the problem classes we investigated were concerned with: the number of customers that could be serviced, the width of the time windows of availability, the variability in the maximum reward values and the type of reward function. We found that the construction methods which were able to evaluate the effect of an insertion according to the potential reward that could be received (e.g., our REWAPP method and our BRIDPOT method — an adaptation of the method of Brideau and Cavalier [20]), rather than the current reward, obtained effective solutions to these problems. We found that the most effective solutions from our construction methods were obtained by the ARTIF method, which places an artificial time window so that a minimum proportion of the maximum available reward will be obtained. This method also took the least computational time, as restricting the positions in which the customer may be inserted reduces the time required for evaluating the potential insertion of a customer. We found that we needed to relax the time windows in order to fine-tune the solutions obtained, thereby permitting low rewards to be received from some of the customers. The ARTIF method was quite effective on the problems for which the customer rewards were strictly decreasing, which was a surprising result, as the artificial windows seem to be best suited to encompass the plateau of a reward function.

With time-dependent rewards we are able to improve upon the solutions obtained by applying intra-route improvements, and we found that these formed an important part of our STEEPEST ASCENT method, particularly when most of the customers are serviced. Our improvement routines all involve evaluating the effect of a move according to the reward that will be received if the customer is serviced as soon as the customer is arrived at, and so these methods are most appropriate

for problems for which no waiting time is required. We do allow the insertion of waiting time at the end of our search, but, since the moves are not evaluated according to their potential rewards, these methods may prove to be ineffective when applied in a straightforward manner to forms of problems for which waiting time is essential.

In order to evaluate the effect of each potential move exactly, we developed the method GWIMP, which constructed the solutions and carried out the STEEPEST ASCENT method by evaluating the potential reward, including non-compulsory waiting time, obtainable for each proposed perturbation of the solution. This method proved to be more effective than our other methods for a problem class with high time limits compared to the time required to service each of the customers, in which case the identification of the best time to service each customer becomes the most important concern, and for a problem class with tight time windows, where large improvements may be made by waiting for short periods of time. This method proved to be very computationally expensive, and it was not applicable to problems with strictly decreasing rewards. We again needed to apply a restricted version of Tabu Search in the preliminary phase, due to the computational time required being too high, and in this case we used the Exchange routine instead of Generalized Exchange. For the other problems considered with type I rewards, our Tabu Search method appeared to be effective, but with type II rewards, the mean reward obtained by the best of the construction heuristics on each instance is higher than that obtained by our Tabu Search method.

We tested many different versions of the MCPTDR in order to identify cases where our evaluative methods were more (or less) effective. We tested our REWAPP method against our implementations of the previously published heuristics of Brideau and Cavalier [20] and Erkut and Zhang [45], on a set of problems generated according to the same distribution as was used in Erkut and Zhang's paper. These problems were of a restricted form, with most customers able to be serviced while obtaining positive reward. We found that the method of Erkut and Zhang slightly outperformed our method for certain problem classes with most customers serviced, but, as fewer customers were able to obtain positive reward, our method became the more effective. We also included more comprehensive improvement routines than the 2-opt method that had been used by Brideau and Cavalier [20] and Erkut and Zhang [45], and found that much greater improvements could be

made, particularly by using the 1-Or opt method, which enabled effective localized improvements to be made.

In Section 8.2.2 we tested the effectiveness of our Tabu Search method in a situation for which it was possible to service all customers at a time where their maximum reward was able to be obtained. For these problems we found that our method was less effective for problems for which the width of the time window of availability was much greater than the width of the time window over which the optimum could be obtained. We found that modelling this problem as involving hard time windows led to much improved results in a far shorter amount of computational time, although this method would only be effective for a very restrictive set of problems for which we know that the customers can be serviced at a time when their maximum reward is obtained. We also found that small alterations to the width of the artificial time windows used by the ARTIF method can lead to large differences in the solution quality obtained, so these need to be set at appropriate levels for each problem instance considered.

We further tested our Tabu Search methods, in Section 8.2.3, on problems for which the initial and final reward obtained may both be positive and our methods seemed to be able to cope effectively with this situation. For problems for which the minimum reward that may be received is relatively high compared to the maximum available reward, then our methods will be more appropriate as waiting times become less relevant and the ability to service greater numbers of customers will be more important. We found also that for problems with large variability in the reward values, we can obtain effective solutions by setting hard artificial time windows. The advantages of this approach are that it can be used to ensure that the selected customers will be serviced at an adequate level (which may be a more important consideration than the actual reward received) and that the artificial time window can reduce the difficulties associated with evaluating the reward, as a significant reward will be received even if the customer is serviced at the beginning of its time window.

The most interesting of the MCPTDR findings were obtained for the problems of Section 8.2.4, for which waiting time had to be added in order to obtain effective solutions. We found that our Tabu Search method was very ineffective for these test problems, as it frequently serviced customers at the beginning of their time window, when no reward was obtained. We were able to obtain improved results

by applying Tabu Search with hard artificial time windows to these problems, although for the general problem class, for which the clusters did not need to be fully serviced, poor solutions were obtained. We created two new methods in order to deal with these situations; both of the methods gave promising results and warrant further consideration. The method TWAIT involved implementing Tabu Search, with Move being the only inter-route routine considered and with each potential move being evaluated exactly, according to the reward that would be received if the optimal waiting time were included in the proposed solution. This routine was very effective on the problems considered, although the computational time required increased dramatically for the problem instances in which larger neighbourhoods for insertion were present. The other method we considered, TADAPT, involved applying artificial time windows to the customers and adapting these throughout the course of the search. This approach seems to be very promising and it may overcome the problem of needing to know a lot about the problem instance in order to be able to effectively apply the artificial time windows. There are many adaptations that may be made to the TADAPT method and we believe that this is a worthwhile area of further research, as these artificial time windows are very general and may be applied to any sort of problem to further restrict the time at which a customer may be serviced. For example, artificial time windows may be applied to soft time window problems in order to limit the allowed deviation from the customer's desired service time during certain phases of a search method.

For the combined version of the MCP, containing tasks and time constraints, our preliminary testing involved varying the numbers of customers served, the tightness of the time windows, the size of the loads carried, the variability in the maximum reward value and the type of reward function considered. A total of 54 problem classes were created, but there were still a number of features (such as the number of subtasks within the tasks and the distance required to service each customer) which are likely to have a significant effect on the behaviour of the problems. Therefore we see the combined MCP as a very complicated problem for which there is a great deal of flexibility available.

We found in our preliminary testing that the time required to solve large numbers of problems for the MCP becomes restrictive on the types of methods that may be used. We saw that large increases in computational time may be required to handle time-dependent rewards and to handle tasks, but the inclusion of both of

these features leads to even larger increases in the time required. Because of this fact, we obtained few insights from our preliminary testing, as we found that the best of the construction heuristics, each improved by applying STEEPEST ASCENT, obtained higher mean rewards than the restricted versions of the search routines we applied.

We also carried out preliminary testing on a relaxation of the task constraints, so that there were two pick-ups that could be made, in either order. The effect of this relaxation was to greatly increase the computational times required within the improvement phase — as the precedence constraints become less restrictive, greater numbers of moves become available. The relaxation of the precedence constraints also required different methods to be employed for keeping track of when a customer was serviced, which was particularly relevant with our reward function that penalizes for excess time a customer's load is in the vehicle.

In the additional testing of Section 8.2.6 we found that our Tabu Search methods did not seem to be as effective as the heuristic methods we apply. The heuristic method, GREEDY, which positions the subtasks of a customer consecutively and includes waiting time at the start of a task, to ensure that the maximum reward is received from the customer, obtained more effective solutions than our Tabu Search method. The GREEDY method evaluates the reward exactly in the construction phase, and this results in the improved rewards being obtained. The Tabu Search method was ineffective because it did not explicitly consider waiting time when assessing the impact of each move, and so the development of methods to effectively deal with waiting time are important in order to enhance the applicability of our methods.

The culmination of the thesis was in the development of models for dynamic versions of subset selection routing problems, and we created models for both next day service, in which the selection of which customers to service takes place prior to the actual servicing taking place, and same day servicing, in which customer requests arrive while service is being carried out. These models relate to practical situations that are becoming more prevalent in the immediate-response society in which we live.

In the models we developed for next day service, there is a time limit on the duration of the vehicle's route and customer requests may be either rejected or provisionally accepted for service; if an accepted customer is excluded from service

at a later stage, a penalty is incurred. We developed fast methods for assessing a customer's viability, which may allow almost instantaneous decision making, so as to prevent excessive loss of goodwill that may result from poor service being provided. We found that a method which accepts all of the customers that are near to the origin obtained very effective solutions for problems for which the rewards and locations were uniformly distributed, although this method was not very resilient to non-uniform data. We found that methods based upon thresholds of acceptance, according to ratios of rewards to inclusion time, were effective for a wide range of problems, including situations for which time window constraints were present. The level of the threshold that obtains the most effective results needs to be altered according to the level of the arrival rates and the rates of the penalties for excluding customers from service after provisionally accepting them for service. If the delay penalties are low, we may obtain improved solutions by delaying the decision making until further information becomes available.

For same day service, we developed models for situations in which there are no penalties for excluding a provisionally accepted customer from service and where there are infinite penalties for excluding customers. When there are no penalties, we found that effective solutions may be obtained by placing all customers within routes and updating the routes when a new customer's request arrives. With infinite penalties we found that accepting a customer for service according to the ratio of the additional reward to additional time, resulted in effective solutions being obtained. One interesting phenomenon in this case was the collection of negative reward from some customers, and this may occur when the highly attractive customers subsequently arrive. We may prevent the poor service to a customer by using artificial time windows to offer a guaranteed level of service.

Future Work

We have explored a number of areas of subset selection routing problems in this thesis, and we have created a number of models for different practical situations. The general model we have developed for the MCP is extensive, and there are many areas for which future research may be rewarding. These include:

- the modelling of restricted subset selection problems, for which consideration

of the non-serviced customers is required. We feel that the modelling of non-service is an important concern that needs to be addressed and effectively handled within solution methods, in order for the practicality of the problem to be more thoroughly addressed.

- the investigation of the effect of multiple vehicles within the model. Our methods are all applicable to multiple vehicle situations, with only our method of evaluating the objective value needing to be altered, but there was enough material of interest within the MCP, without worrying about the interaction between different service vehicles, so we didn't test any multiple vehicle situations. The problem generation methods that we employed were based on the assumption that only one route at a time is being carried out, and so different techniques will be required to generate interesting and challenging problems with more than one route. Also with more than one route, different methods for constructing solutions are required, in order to take advantage of some form of partitioning of the problem space. The development of worst-case analysis of these problems and the categorization of different types of problems would also yield fruitful research.

- implementation of techniques that increase the efficiency of the routines. We found that we could make major gains in the efficiency of our routines for the TRCP, by assessing the potential gain that could be made for an insertion before actually attempting to carry out the insertion. With time-dependent rewards, the impact of a move is less straightforward to calculate as, for example, it is possible for the reward for a vehicle to increase if a customer is removed. The calculation of an effective bound on the effect of a move, may prevent unnecessarily attempting to insert a customer in a route, when the insertion can not result in improved solutions. Increased efficiency is required in the problem which contains both the time-dependent rewards and task constraints, as the computational time is potentially very restrictive for these problems.

- more effective techniques for handling time-dependent rewards. We have found that our methods of assessing the effect of a move according to the reward that is received when the customer is serviced immediately, may lead to poor results being obtained for problems for which non-compulsory waiting

times feature prominently in effective solutions. For the MCPTDR we have identified the use of adaptive artificial time windows and the development of a restricted version of Tabu Search for which the potential reward, including waiting times, is calculated for each proposed move, as being methods that may be used to effectively deal with problems requiring waiting times. We have seen that the development of effective methods for instances of the general version of the MCP is required, with some form of adaptive artificial time window a possibility.

- extension of the same day dynamic models. For the dynamic situations in which the service takes place while the customer requests are being issued, we only considered the case of immediate request customers with decreasing rewards. If the customer requests service in advance, then it may be more difficult to assess the impact of the customer on the routing, without information about subsequent requests being known. Also, the inclusion of finite penalties for excluding customers from service at a later stage, in the same day service model, will lead to different behaviour of the problems and will require the development of different solution methods.

- improvement of the Tabu Search routines. We found for some special problems, our Tabu Search routine was less effective than some iterative approaches, so the investigation of the reasons for this ineffectiveness and the development of improved implementations appear to be warranted.

APPENDIX A

Additional Results from Testing

Probs	Improvement Routines							
(N,L,R)	SA	MOVE	NOMV	CROSS	NOCR	EXCH	NOEX	OTH
(1,1,1)	21.00	20.36	20.96	20.96	20.43	20.96	20.96	20.88
(1,1,2)	87.61	83.40	87.60	86.79	86.34	87.16	86.33	87.63
(1,1,3)	341.89	322.76	342.17	335.56	336.69	338.59	334.46	342.40
(2,1,1)	60.08	59.35	60.07	60.27	59.41	60.02	60.02	59.99
(2,1,2)	246.38	238.87	246.16	244.49	244.97	245.40	244.00	245.47
(2,1,3)	965.53	932.35	965.42	954.98	958.32	960.63	952.81	963.42
(3,1,1)	95.86	95.51	95.83	95.98	95.61	95.76	95.76	95.86
(3,1,2)	383.94	380.27	383.93	383.15	383.51	383.93	382.76	383.94
(3,1,3)	1428.84	1416.64	1429.42	1424.25	1426.48	1428.75	1422.47	1428.84
(1,2,1)	21.91	21.55	21.92	21.50	21.58	21.88	21.88	21.99
(1,2,2)	93.20	90.08	93.40	88.41	92.45	92.73	90.89	93.44
(1,2,3)	363.83	345.44	365.18	343.12	362.02	360.62	349.33	364.88
(2,2,1)	58.67	57.80	58.54	56.75	58.35	58.12	58.12	58.62
(2,2,2)	242.50	235.12	242.88	228.55	242.50	240.81	236.13	242.88
(2,2,3)	934.98	897.00	937.47	870.57	933.56	934.09	905.03	933.45
(3,2,1)	95.66	95.02	95.92	93.78	95.51	95.22	95.22	95.66
(3,2,2)	384.52	379.49	384.07	375.35	383.77	383.45	380.91	384.52
(3,2,3)	1410.33	1396.14	1410.94	1373.04	1410.33	1409.31	1399.15	1410.33
(1,3,1)	23.85	23.58	23.85	22.45	23.65	23.79	23.79	24.08
(1,3,2)	100.53	97.41	100.78	93.15	99.53	99.86	98.84	100.63
(1,3,3)	395.04	383.12	396.44	364.94	392.47	393.39	386.42	398.36
(2,3,1)	58.89	58.49	59.02	50.05	58.71	58.81	58.81	59.13
(2,3,2)	243.15	233.33	243.72	200.06	242.50	241.98	234.92	243.83
(2,3,3)	964.26	925.03	967.27	783.88	962.44	962.83	929.07	960.83
(3,3,1)	93.72	93.47	93.12	75.50	93.72	93.47	93.47	93.75
(3,3,2)	370.46	367.17	369.83	300.10	370.46	370.46	367.20	370.46
(3,3,3)	1397.34	1384.92	1399.88	1128.06	1397.34	1395.91	1385.39	1397.34
Mean	403.11	393.84	403.55	373.17	401.95	402.15	396.82	403.06
Time	6:09:48	0:43:00	19:51:48	0:09:48	6:22:24	3:59:13	0:45:44	6:21:58

Table A.1: Testing Improvement Routines for TRCP Instances

Classes	Improvement Routines					
(T,N,R)	ASCENT	NOINTRA	NOMOVE	NOEXCH	NOCROSS	INTEXCH
(1,1,1)	19.08	18.74	19.10	18.84	19.01	19.05
(1,1,2)	177.55	175.53	176.73	175.49	177.16	176.35
(1,1,3)	690.48	676.84	693.14	683.63	684.24	686.90
(1,2,1)	47.17	46.07	47.34	45.46	46.99	47.12
(1,2,2)	428.15	415.55	430.29	413.38	425.88	427.91
(1,2,3)	1657.24	1621.14	1669.14	1610.47	1632.55	1643.05
(1,3,1)	60.37	57.97	60.34	57.99	60.15	60.22
(1,3,2)	545.48	531.51	547.54	529.15	544.58	546.93
(1,3,3)	2095.06	2059.80	2095.95	2004.97	2092.90	2093.97
(2,1,1)	24.11	23.40	24.10	23.54	24.01	24.03
(2,1,2)	216.73	210.77	217.79	213.22	216.16	217.05
(2,1,3)	837.78	815.36	841.75	824.34	833.88	838.74
(2,2,1)	55.99	53.86	56.01	54.47	55.66	55.67
(2,2,2)	508.74	491.84	510.11	489.78	508.19	509.49
(2,2,3)	1933.14	1880.10	1948.06	1858.93	1928.44	1944.46
(2,3,1)	78.65	71.85	78.66	76.84	78.62	78.73
(2,3,2)	712.00	657.67	715.60	694.44	710.46	715.60
(2,3,3)	2632.44	2424.68	2636.71	2566.42	2626.50	2629.96
(3,1,1)	27.95	27.10	28.04	27.40	27.74	27.69
(3,1,2)	247.73	240.65	247.35	240.98	246.63	246.40
(3,1,3)	953.95	923.33	955.84	934.47	950.83	952.17
(3,2,1)	58.62	55.84	58.70	56.79	58.52	58.59
(3,2,2)	547.93	519.05	547.31	523.36	546.63	546.86
(3,2,3)	2016.66	1937.48	2029.08	1947.57	2016.17	2024.78
(3,3,1)	89.11	73.44	89.13	89.00	89.11	89.13
(3,3,2)	801.02	662.85	801.03	797.22	801.02	801.03
(3,3,3)	2933.43	2360.66	2939.81	2923.50	2933.43	2939.81
Mean	755.43	704.93	757.95	736.36	753.17	755.62

Table A.2: Improvement Routines for MCPTDR Instances – Type I

Classes	Improvement Routines					
(T,N,R)	ASCENT	NOINTRA	NOMOVE	NOEXCH	NOCROSS	INTEXCH
(1,1,1)	16.79	16.73	16.79	16.57	16.61	16.61
(1,1,2)	152.64	151.71	152.64	150.58	150.66	150.50
(1,1,3)	611.70	606.65	612.54	609.10	603.72	604.69
(1,2,1)	47.01	46.17	47.09	46.66	46.29	46.27
(1,2,2)	428.17	421.78	428.03	426.21	421.07	421.43
(1,2,3)	1621.64	1602.23	1627.48	1616.35	1590.27	1598.10
(1,3,1)	60.30	59.61	60.35	59.98	59.99	60.08
(1,3,2)	549.54	542.13	550.39	546.47	544.68	545.48
(1,3,3)	2113.88	2090.22	2114.30	2099.70	2100.21	2107.18
(2,1,1)	19.55	19.27	19.56	19.38	19.44	19.46
(2,1,2)	176.52	174.11	176.60	174.94	174.44	174.52
(2,1,3)	678.48	674.12	678.40	676.85	674.22	674.67
(2,2,1)	53.15	51.90	53.26	52.81	52.73	52.74
(2,2,2)	480.22	470.95	480.24	475.35	478.17	478.43
(2,2,3)	1808.69	1787.45	1813.37	1803.83	1792.41	1798.85
(2,3,1)	72.98	71.04	73.13	72.78	72.97	73.09
(2,3,2)	663.18	643.29	662.80	659.00	663.65	662.44
(2,3,3)	2475.42	2428.98	2485.72	2464.88	2474.30	2484.76
(3,1,1)	22.46	22.08	22.44	22.37	22.31	22.30
(3,1,2)	205.45	200.56	204.93	204.77	203.73	203.99
(3,1,3)	791.89	784.95	791.49	786.67	787.19	786.82
(3,2,1)	55.07	53.85	55.18	54.57	54.96	55.07
(3,2,2)	493.86	483.24	495.15	489.80	493.43	494.49
(3,2,3)	1905.35	1871.27	1906.53	1891.45	1898.88	1902.33
(3,3,1)	79.54	76.22	79.54	79.54	79.54	79.54
(3,3,2)	711.51	681.93	711.51	711.51	711.51	711.51
(3,3,3)	2658.39	2544.68	2658.39	2658.39	2658.39	2658.39
Mean	701.98	688.04	702.88	698.91	697.99	699.40

Table A.3: Improvement Routines for MCPTDR Instances – Type II

Classes	Methods					
(T,N,L,R)	ASCENT	MOVE	CROSS	NOINTRA	NOMOVE	EXCH
(1 1 1 1)	75.30	71.31	74.07	74.73	75.20	74.01
(1 1 1 2)	308.32	289.63	303.55	307.33	305.10	302.22
(1 2 1 1)	265.21	240.02	253.69	262.07	264.91	261.50
(1 2 1 2)	1005.53	892.80	957.42	988.449	1015.65	999.55
(1 3 1 1)	290.16	286.31	292.01	282.00	295.27	293.73
(1 3 1 2)	1109.05	1053.76	1086.86	1057.29	1102.90	1089.66
(2 1 1 1)	92.58	88.25	91.33	90.17	92.93	92.45
(2 1 1 2)	363.03	337.96	352.03	354.61	365.75	353.91
(2 2 1 1)	279.82	254.94	271.90	273.82	281.77	276.15
(2 2 1 2)	1099.83	992.62	1062.14	1071.63	1099.85	1085.57
(2 3 1 1)	339.60	331.79	334.32	302.82	339.03	335.01
(2 3 1 2)	1305.69	1283.49	1296.08	1175.78	1302.665	1293.53
(3 1 1 1)	106.076	101.13	102.95	102.98	105.91	104.25
(3 1 1 2)	399.827	363.68	389.44	392.29	402.11	385.79
(3 2 1 1)	297.57	274.00	286.38	286.09	296.52	291.27
(3 2 1 2)	1117.14	1011.73	1086.14	1091.50	1125.74	1100.36
(3 3 1 1)	356.53	353.29	353.21	297.88	354.22	353.68
(3 3 1 2)	1332.93	1331.79	1336.13	1138.49	1343.69	1335.39
(1 1 2 1)	72.40	67.99	66.92	72.05	73.63	70.39
(1 1 2 2)	287.78	265.82	262.21	281.99	287.82	276.13
(1 2 2 1)	255.78	211.05	174.956	249.98	256.14	235.72
(1 2 2 2)	988.73	778.65	650.46	961.02	1001.22	951.30
(1 3 2 1)	314.66	296.72	273.28	287.43	316.60	308.14
(1 3 2 2)	1201.13	1129.18	1006.86	1090.02	1218.15	1182.26
(2 1 2 1)	92.02	84.96	83.66	88.96	91.68	90.98
(2 1 2 2)	349.56	313.21	310.35	339.29	346.55	335.51
(2 2 2 1)	269.15	214.90	186.55	265.68	268.32	248.39
(2 2 2 2)	1014.83	822.12	684.19	998.40	1029.49	960.25
(2 3 2 1)	351.80	342.49	305.05	313.64	355.42	347.73
(2 3 2 2)	1328.00	1282.99	1130.29	1203.47	1345.61	1312.21
(3 1 2 1)	100.3	91.55	90.37	97.37	99.75	96.32
(3 1 2 2)	384.18	351.91	348.93	384.73	391.87	375.22
(3 2 2 1)	276.53	232.00	186.30	265.72	277.24	258.54
(3 2 2 2)	1065.96	834.98	680.37	1007.08	1063.51	983.69
(3 3 2 1)	377.95	372.62	337.97	324.56	382.17	377.97
(3 3 2 2)	1387.34	1358.66	1228.78	1205.28	1396.67	1379.75
Total	562.8	516.95	498.26	527.41	565.86	550.52

Table A.4: Testing Improvement Routines for Reward Type I for Combined MCP

Classes	Methods					
(T,N,L,R)	ASCENT	MOVE	CROSS	NOINTRA	NOMOVE	EXCH
(1 1 2 1)	62.02	60.38	59.34	61.91	61.79	61.05
(1 1 2 2)	242.70	227.08	227.44	242.43	241.04	233.28
(1 2 2 1)	191.66	163.71	162.19	188.10	193.43	188.39
(1 2 2 2)	732.27	610.20	613.82	732.79	744.86	713.32
(1 3 2 1)	235.78	220.24	218.14	212.70	243.76	232.05
(1 3 2 2)	889.25	807.59	793.70	787.97	925.23	880.53
(2 1 2 1)	78.29	73.83	74.23	75.41	77.82	77.34
(2 1 2 2)	300.55	285.10	289.68	290.14	301.68	299.01
(2 2 2 1)	206.91	174.68	175.76	202.18	205.71	194.36
(2 2 2 2)	797.91	670.39	707.18	796.94	807.70	783.62
(2 3 2 1)	291.88	287.46	286.93	276.27	296.73	292.32
(2 3 2 2)	1120.19	1090.82	1095.33	1050.77	1136.37	1116.17
(3 1 2 1)	85.43	82.86	84.46	84.04	86.85	84.91
(3 1 2 2)	344.22	317.33	329.34	334.94	343.60	341.19
(3 2 2 1)	208.37	176.24	180.55	207.32	207.95	202.04
(3 2 2 2)	798.58	685.31	696.70	788.09	813.00	773.84
(3 3 2 1)	300.09	299.36	300.34	256.80	304.04	301.29
(3 3 2 2)	1150.32	1150.61	1145.78	981.06	1170.21	1152.64
Total	446.47	410.18	413.38	420.55	453.43	440.41

Table A.5: Testing Improvement Routines for Reward Type II for Combined MCP

			Methods	
Phase	Class		BRIDEAU	REWAPP
UNIMP	$U(1,11)$	ERKUT	0.0439	-0.1032
		BRIDEAU		-0.0001
	$U(1,16)$	ERKUT	0.0000	0.0856
		BRIDEAU		-0.0000
2-OPT	$U(1,11)$	ERKUT	0.0014	-0.6774
		BRIDEAU		-0.0001
	$U(1,16)$	ERKUT	0.0000	-0.2975
		BRIDEAU		-0.0000
ASCENT	$U(1,11)$	ERKUT	0.3674	-0.5768
		BRIDEAU		-0.4020
	$U(1,16)$	ERKUT	0.0515	0.8697
		BRIDEAU		-0.3590

Table A.6: P-Values for Differences on Erkut's Data - Set I

			Methods	
Phase	Class		BRIDEAU	REWAPP
UNIMP	$U(1,26)$	ERKUT	0.0000	0.0007
		BRIDEAU		-0.0000
	$U(1,51)$	ERKUT	0.0000	0.0000
		BRIDEAU		-0.0000
	$U(1,101)$	ERKUT	0.0000	-0.4535
		BRIDEAU		-0.0000
	$U(1,201)$	ERKUT	0.0010	-0.0013
		BRIDEAU		-0.0000
	$U(1,401)$	ERKUT	0.0001	0.2009
		BRIDEAU		-0.0000
2-OPT	$U(1,26)$	ERKUT	0.0000	0.2120
		BRIDEAU		-0.0000
	$U(1,51)$	ERKUT	0.0000	0.0000
		BRIDEAU		-0.0151
	$U(1,101)$	ERKUT	0.0000	0.7181
		BRIDEAU		-0.0000
	$U(1,201)$	ERKUT	0.0096	-0.0009
		BRIDEAU		-0.0000
	$U(1,401)$	ERKUT	0.0035	-0.0288
		BRIDEAU		-0.0000
ASCENT	$U(1,26)$	ERKUT	0.3420	0.3912
		BRIDEAU		-0.4278
	$U(1,51)$	ERKUT	0.8192	-0.8500
		BRIDEAU		-0.8867
	$U(1,101)$	ERKUT	-0.7891	-0.4445
		BRIDEAU		-0.6244
	$U(1,201)$	ERKUT	-0.2344	-0.0216
		BRIDEAU		-0.0523
	$U(1,401)$	ERKUT	-0.9167	-0.0171
		BRIDEAU		-0.0108

Table A.7: P-Values for Differences on Erkut's Data - Set II

Class	Construction Techniques						
(W,T,V)	RApp	Artif	ArtM	AtMax	NewM	TImp	TInt
(1,1,1)	69	67	99	85	100	100	99
(1,1,2)	62	66	99	97	100	98	100
(1,1,3)	53	56	100	89	100	99	100
(1,2,1)	23	25	91	41	93	96	94
(1,2,2)	19	24	94	44	92	75	90
(1,2,3)	10	27	94	40	93	84	87
(1,3,1)	32	44	96	55	99	95	100
(1,3,2)	34	39	96	58	94	93	97
(1,3,3)	31	42	97	54	95	82	96
(2,1,1)	45	53	100	84	97	99	99
(2,1,2)	45	48	100	88	99	96	100
(2,1,3)	47	48	99	87	99	96	99
(2,2,1)	14	15	92	52	91	83	71
(2,2,2)	9	19	95	51	94	65	76
(2,2,3)	7	9	91	48	92	60	55
(2,3,1)	16	16	91	53	94	92	92
(2,3,2)	11	17	96	52	94	82	92
(2,3,3)	9	18	94	41	89	63	87
(3,1,1)	67	65	95	75	90	100	98
(3,1,2)	62	66	93	71	92	99	99
(3,1,3)	73	70	89	72	88	100	98
(3,2,1)	60	61	89	66	85	97	96
(3,2,2)	62	63	91	59	88	99	97
(3,2,3)	63	57	85	62	87	98	94
(3,3,1)	89	94	100	98	100	100	99
(3,3,2)	90	97	100	96	100	100	100
(3,3,3)	84	84	100	97	100	99	100
Total	1186	1290	2566	1815	2545	2450	2515
Time	0:35:13	0:12:18	0:12:37	0:22:50	0:11:57	2:22:20	2:41:07

Table A.8: Number of Optimal Solutions Found — n = 40

Class	Construction Techniques						
(W,T,R)	RApp	Artif	ArtM	AtMax	NewM	TImp	TInt
(1,1,1)	0.99095	0.99234	0.99975	0.99685	1.00000	1.00000	0.99998
(1,1,2)	0.98863	0.99011	0.99990	0.99969	1.00000	0.99983	1.00000
(1,1,3)	0.98815	0.99016	1.00000	0.99868	1.00000	0.99995	1.00000
(1,2,1)	0.98353	0.98963	0.99867	0.99179	0.99908	0.99984	0.99963
(1,2,2)	0.97943	0.98955	0.99892	0.99299	0.99875	0.99930	0.99911
(1,2,3)	0.98236	0.99149	0.99935	0.99375	0.99947	0.99937	0.99947
(1,3,1)	0.97808	0.98976	0.99898	0.99104	0.99992	0.99958	1.00000
(1,3,2)	0.97882	0.98858	0.99945	0.99244	0.99932	0.99906	0.99962
(1,3,3)	0.97465	0.99144	0.99990	0.99246	0.99966	0.99830	0.99969
(2,1,1)	0.99053	0.99102	1.00000	0.99882	0.99985	0.99997	0.99999
(2,1,2)	0.99290	0.99369	1.00000	0.99881	0.99972	0.99988	1.00000
(2,1,3)	0.99211	0.99373	0.99989	0.99912	0.99989	0.99998	0.99999
(2,2,1)	0.98304	0.98698	0.99931	0.99660	0.99966	0.99970	0.99898
(2,2,2)	0.98477	0.98848	0.99956	0.99656	0.99952	0.99912	0.99917
(2,2,3)	0.98295	0.98829	0.99963	0.99652	0.99963	0.99890	0.99810
(2,3,1)	0.98688	0.98906	0.99904	0.99521	0.99966	0.99978	0.99961
(2,3,2)	0.98494	0.98799	0.99958	0.99498	0.99949	0.99920	0.99955
(2,3,3)	0.98395	0.98996	0.99964	0.99407	0.99910	0.99861	0.99924
(3,1,1)	0.99529	0.99490	0.99978	0.99764	0.99888	1.00000	0.99998
(3,1,2)	0.99429	0.99534	0.99936	0.99750	0.99979	0.99998	0.99999
(3,1,3)	0.99619	0.99730	0.99901	0.99849	0.99886	1.00000	0.99998
(3,2,1)	0.99733	0.99826	0.99941	0.99818	0.99935	0.99999	0.99984
(3,2,2)	0.99752	0.99828	0.99977	0.99866	0.99950	0.99996	0.99983
(3,2,3)	0.99801	0.99878	0.99978	0.99880	0.99958	0.99997	0.99990
(3,3,1)	0.99947	0.99983	1.00000	0.99978	1.00000	1.00000	0.99987
(3,3,2)	0.99910	0.99990	1.00000	0.99958	1.00000	1.00000	1.00000
(3,3,3)	0.99820	0.99967	1.00000	0.99985	1.00000	0.99995	1.00000
Total	0.98880	0.99305	0.99962	0.99671	0.99958	0.99957	0.99965

Table A.9: Average Proportion of Optimal Objective Value Found — n = 40

Class	Construction Techniques						
(W,T,V)	RApp	Artif	ArtM	AtMax	NewM	TImp	TInt
(1,1,1)	25	36	99	63	97	99	97
(1,1,2)	28	31	98	61	97	90	92
(1,1,3)	24	17	96	69	93	89	93
(1,2,1)	4	5	87	13	85	37	27
(1,2,2)	4	4	88	10	81	24	25
(1,2,3)	0	2	86	15	86	22	14
(1,3,1)	2	4	78	7	72	18	33
(1,3,2)	0	2	83	9	71	19	30
(1,3,3)	0	7	80	8	80	12	23
(2,1,1)	34	28	98	68	96	88	92
(2,1,2)	20	21	100	65	98	80	90
(2,1,3)	18	24	98	64	97	74	81
(2,2,1)	1	2	76	13	68	34	11
(2,2,2)	0	3	82	12	79	11	7
(2,2,3)	0	1	81	13	76	10	4
(2,3,1)	0	0	74	8	76	17	25
(2,3,2)	1	0	82	11	79	15	22
(2,3,3)	2	1	85	7	75	6	12
(3,1,1)	68	60	84	61	82	99	96
(3,1,2)	58	65	82	65	80	96	99
(3,1,3)	52	51	84	58	76	96	95
(3,2,1)	55	51	89	62	82	99	99
(3,2,2)	51	63	94	67	84	96	98
(3,2,3)	54	53	90	54	77	97	98
(3,3,1)	96	96	100	99	100	96	97
(3,3,2)	89	96	100	100	100	97	98
(3,3,3)	91	95	100	100	100	96	97
Total	777	818	2394	1182	2287	1617	1655
Time	03:10	00:50	01:02	02:05	01:04	21:49	27:21

Table A.10: Proportion of Optimal Solutions Found — n = 60

Class	Construction Techniques						
(W,T,V)	RApp	Artif	ArtM	AtMax	NewM	TImp	TInt
(1,1,1)	0.98818	0.98968	0.99992	0.99612	0.99984	0.99999	0.99996
(1,1,2)	0.98706	0.98688	0.99975	0.99572	0.99948	0.99967	0.99975
(1,1,3)	0.98630	0.98893	0.99975	0.99712	0.99971	0.99980	0.99979
(1,2,1)	0.98222	0.98774	0.99858	0.99045	0.99875	0.99902	0.99776
(1,2,2)	0.97884	0.98812	0.99906	0.98834	0.99848	0.99821	0.99727
(1,2,3)	0.98205	0.98719	0.99902	0.99014	0.99912	0.99762	0.99752
(1,3,1)	0.97635	0.98389	0.99711	0.98517	0.99574	0.99611	0.99589
(1,3,2)	0.97886	0.98278	0.99760	0.98307	0.99614	0.99532	0.99589
(1,3,3)	0.97973	0.98725	0.99840	0.98699	0.99810	0.99407	0.99559
(2,1,1)	0.99177	0.99124	0.99982	0.99864	0.99988	0.99986	0.99989
(2,1,2)	0.98804	0.99240	1.00000	0.99824	0.99994	0.99969	0.99989
(2,1,3)	0.99207	0.99264	0.99989	0.99888	0.99983	0.99975	0.99973
(2,2,1)	0.98320	0.98766	0.99887	0.99412	0.99884	0.99930	0.99668
(2,2,2)	0.98645	0.98704	0.99945	0.99432	0.99956	0.99839	0.99700
(2,2,3)	0.98393	0.98879	0.99951	0.99581	0.99900	0.99699	0.99592
(2,3,1)	0.98957	0.99055	0.99789	0.99175	0.99830	0.99822	0.99782
(2,3,2)	0.98816	0.99049	0.99892	0.99174	0.99870	0.99775	0.99783
(2,3,3)	0.98578	0.99078	0.99913	0.99284	0.99857	0.99679	0.99722
(3,1,1)	0.99773	0.99675	0.99971	0.99822	0.99921	1.00000	0.99997
(3,1,2)	0.99654	0.99835	0.99958	0.99777	0.99939	0.99997	1.00000
(3,1,3)	0.99744	0.99747	0.99966	0.99817	0.99924	0.99996	0.99996
(3,2,1)	0.99799	0.99848	0.99973	0.99894	0.99944	0.99998	0.99999
(3,2,2)	0.99769	0.99868	0.99985	0.99902	0.99963	0.99997	0.99999
(3,2,3)	0.99820	0.99900	0.99967	0.99893	0.99953	0.99997	0.99999
(3,3,1)	0.99950	0.99979	1.00000	1.00000	1.00000	0.99943	0.99953
(3,3,2)	0.99905	0.99973	1.00000	1.00000	1.00000	0.99971	0.99974
(3,3,3)	0.99927	0.99973	1.00000	1.00000	1.00000	0.99985	0.99984
Total	0.98939	0.99214	0.99935	0.99510	0.99912	0.99856	0.99848

Table A.11: Average Proportion of Optimal Objective Value Found — n = 60

Class	Construction Techniques				
(W,T,R)	RewApp	Artif	ArtMax	AtMax	NewMax
(1,1,1)	4	4	83	21	85
(1,1,2)	5	4	84	15	76
(1,1,3)	3	5	87	13	88
(1,2,1)	0	0	57	0	58
(1,2,2)	0	0	63	1	53
(1,2,3)	0	0	63	0	54
(1,3,1)	0	0	43	1	48
(1,3,2)	0	0	46	0	42
(1,3,3)	0	0	58	0	50
(2,1,1)	2	3	86	30	90
(2,1,2)	3	1	92	25	87
(2,1,3)	6	4	90	25	89
(2,2,1)	0	0	55	2	50
(2,2,2)	0	0	60	1	57
(2,2,3)	0	0	54	0	54
(2,3,1)	0	0	42	0	43
(2,3,2)	0	0	51	0	42
(2,3,3)	0	0	47	1	36
(3,1,1)	46	56	71	58	63
(3,1,2)	52	54	76	42	61
(3,1,3)	48	63	71	38	54
(3,2,1)	65	66	93	67	87
(3,2,2)	72	66	95	67	85
(3,2,3)	59	67	95	53	91
(3,3,1)	85	96	100	100	100
(3,3,2)	83	98	100	100	100
(3,3,3)	80	97	100	100	100
Total	613	684	1962	760	1843
Time	≈33 hours	07:18:06	10:18:50	18:36:46	09:56:47

Table A.12: Proportion of Optimal Solutions Found — n = 100

Class	Construction Techniques				
(W,T,R)	RewApp	Artif	ArtMax	AtMax	NewMax
(1,1,1)	0.98681	0.98923	0.99937	0.99650	0.99937
(1,1,2)	0.98691	0.98767	0.99928	0.99580	0.99882
(1,1,3)	0.98770	0.98951	0.99957	0.99548	0.99964
(1,2,1)	0.97781	0.98509	0.99613	0.98583	0.99609
(1,2,2)	0.97980	0.98567	0.99672	0.98727	0.99598
(1,2,3)	0.98193	0.98807	0.99790	0.98851	0.99722
(1,3,1)	0.98512	0.98798	0.99406	0.98733	0.99437
(1,3,2)	0.98565	0.98833	0.99505	0.98730	0.99467
(1,3,3)	0.98535	0.99015	0.99732	0.98765	0.99657
(2,1,1)	0.98915	0.99051	0.99964	0.99843	0.99977
(2,1,2)	0.98991	0.99081	0.99986	0.99812	0.99977
(2,1,3)	0.99161	0.99279	0.99992	0.99820	0.99981
(2,2,1)	0.98415	0.98876	0.99767	0.99459	0.99798
(2,2,2)	0.98472	0.98919	0.99808	0.99427	0.99791
(2,2,3)	0.98458	0.98910	0.99825	0.99424	0.99821
(2,3,1)	0.99339	0.99434	0.99689	0.99457	0.99724
(2,3,2)	0.99271	0.99465	0.99748	0.99339	0.99703
(2,3,3)	0.99193	0.99408	0.99794	0.99372	0.99687
(3,1,1)	0.99807	0.99850	0.99901	0.99849	0.99873
(3,1,2)	0.99716	0.99736	0.99936	0.99740	0.99895
(3,1,3)	0.99709	0.99860	0.99900	0.99649	0.99780
(3,2,1)	0.99958	0.99961	0.99985	0.99966	0.99980
(3,2,2)	0.99955	0.99949	0.99994	0.99965	0.99992
(3,2,3)	0.99966	0.99967	0.99998	0.99936	0.99990
(3,3,1)	0.99854	0.99988	1.00000	1.00000	1.00000
(3,3,2)	0.99821	0.99986	1.00000	1.00000	1.00000
(3,3,3)	0.99847	0.99987	1.00000	1.00000	1.00000
Total	0.99072	0.99318	0.99862	0.99489	0.99833

Table A.13: Average Proportion of Optimal Objective Value Found — n = 100

Class	Construction Techniques						
(W,T,V)	RApp	Artif	ArtM	AtMax	NewM	TImp	TInt
(1,1,1)	1	1	20	5	23	12	11
(1,1,2)	3	2	21	1	19	8	11
(1,1,3)	1	3	19	3	24	5	7
(1,2,1)	0	0	13	0	15	1	0
(1,2,2)	0	0	14	0	14	0	0
(1,2,3)	0	0	16	0	14	0	0
(1,3,1)	0	0	9	1	10	0	0
(1,3,2)	0	0	11	0	9	0	0
(1,3,3)	0	0	16	0	12	0	0
(2,1,1)	0	1	22	10	24	17	6
(2,1,2)	2	1	24	7	22	11	9
(2,1,3)	1	0	23	6	22	7	4
(2,2,1)	0	0	15	0	13	1	0
(2,2,2)	0	0	16	0	15	0	0
(2,2,3)	0	0	15	0	11	0	0
(2,3,1)	0	0	7	0	6	1	0
(2,3,2)	0	0	12	0	11	0	0
(2,3,3)	0	0	9	0	7	0	0
(3,1,1)	11	15	18	12	15	25	24
(3,1,2)	13	13	20	12	16	24	24
(3,1,3)	11	16	21	10	13	24	25
(3,2,1)	14	16	23	15	24	25	25
(3,2,2)	18	19	23	16	23	25	25
(3,2,3)	12	18	24	14	22	25	25
(3,3,1)	22	24	25	25	25	17	17
(3,3,2)	21	24	25	25	25	19	21
(3,3,3)	20	25	25	25	25	19	20
Total	150	178	486	187	459	266	254

Table A.14: Number of Optimal Solutions Found in Subset — n = 100

Class	Construction Techniques						
(W,T,V)	RApp	Artif	ArtM	AtMax	NewM	TImp	TInt
(1,1,1)	0.98588	0.99037	0.99882	0.99582	0.99956	0.99956	0.99928
(1,1,2)	0.98832	0.98985	0.99961	0.99574	0.99869	0.99929	0.99934
(1,1,3)	0.98899	0.99214	0.99874	0.99602	0.99968	0.99927	0.99871
(1,2,1)	0.98320	0.98579	0.99565	0.98418	0.99697	0.99735	0.99479
(1,2,2)	0.98036	0.98723	0.99641	0.98578	0.99438	0.99575	0.99301
(1,2,3)	0.98257	0.98532	0.99812	0.98907	0.99706	0.99554	0.99299
(1,3,1)	0.98627	0.98708	0.99349	0.98865	0.99386	0.99525	0.99457
(1,3,2)	0.98631	0.98811	0.99489	0.98666	0.99471	0.99463	0.99435
(1,3,3)	0.98551	0.99109	0.99701	0.98860	0.99587	0.99401	0.99337
(2,1,1)	0.98601	0.99142	0.99987	0.99849	0.99976	0.99986	0.99882
(2,1,2)	0.99118	0.99095	0.99999	0.99841	0.99989	0.99953	0.99952
(2,1,3)	0.99167	0.99307	0.99995	0.99683	0.99979	0.99870	0.99912
(2,2,1)	0.98619	0.99010	0.99793	0.99471	0.99755	0.99890	0.99650
(2,2,2)	0.98691	0.99072	0.99860	0.99439	0.99722	0.99697	0.99601
(2,2,3)	0.98390	0.99112	0.99829	0.99307	0.99767	0.99473	0.99536
(2,3,1)	0.99267	0.99481	0.99607	0.99436	0.99571	0.99783	0.99737
(2,3,2)	0.99035	0.99380	0.99698	0.99283	0.99694	0.99778	0.99758
(2,3,3)	0.99236	0.99364	0.99752	0.99390	0.99592	0.99748	0.99682
(3,1,1)	0.99870	0.99853	0.99859	0.99837	0.99845	1.00000	0.99997
(3,1,2)	0.99890	0.99716	0.99872	0.99625	0.99856	0.99999	0.99994
(3,1,3)	0.99753	0.99870	0.99926	0.99490	0.99622	0.99999	1.00000
(3,2,1)	0.99949	0.99963	0.99999	0.99967	0.99994	1.00000	1.00000
(3,2,2)	0.99985	0.99967	0.99993	0.99965	0.99998	1.00000	1.00000
(3,2,3)	0.99957	0.99974	0.99997	0.99917	0.99991	1.00000	1.00000
(3,3,1)	0.99883	0.99997	1.00000	1.00000	1.00000	0.99691	0.99697
(3,3,2)	0.99817	0.99974	1.00000	1.00000	1.00000	0.99836	0.99881
(3,3,3)	0.99882	1.00000	1.00000	1.00000	1.00000	0.99947	0.99938
Total	0.99110	0.99355	0.99848	0.99464	0.99797	0.99790	0.99741

Table A.15: Average Proportion of Optimal Objective Value Found in Subset — n = 100

Class	Solution Methods								
(W,T,R)	1	2	3	4	5	6	7	8	9
(1,1,1)	96	99	95	98	94	100	100	100	100
(1,1,2)	95	100	96	99	99	100	100	100	100
(1,1,3)	96	99	97	99	97	100	100	100	100
(1,2,1)	88	91	89	93	87	98	98	99	100
(1,2,2)	91	96	95	98	86	99	100	100	100
(1,2,3)	84	94	84	95	82	99	99	100	100
(1,3,1)	92	89	95	98	95	100	100	100	100
(1,3,2)	97	95	97	98	95	100	100	99	100
(1,3,3)	93	95	94	99	96	100	100	99	100
(2,1,1)	96	100	98	99	96	100	100	100	100
(2,1,2)	96	100	98	99	97	100	100	100	100
(2,1,3)	97	100	99	99	100	100	100	100	100
(2,2,1)	93	89	85	93	90	99	100	100	100
(2,2,2)	94	94	92	97	92	99	100	99	100
(2,2,3)	94	92	94	94	91	99	99	99	100
(2,3,1)	89	88	93	98	93	100	100	100	100
(2,3,2)	94	92	98	98	90	100	100	100	100
(2,3,3)	93	91	93	98	90	100	100	100	100
(3,1,1)	65	100	61	66	68	92	98	100	100
(3,1,2)	67	100	63	72	62	92	99	100	100
(3,1,3)	70	100	66	73	66	96	98	100	100
(3,2,1)	82	69	73	84	68	96	98	99	100
(3,2,2)	85	76	85	86	68	100	100	99	99
(3,2,3)	86	73	74	87	74	98	99	99	100
(3,3,1)	100	100	100	100	100	100	100	100	100
(3,3,2)	100	100	100	100	99	100	100	100	100
(3,3,3)	100	100	100	100	100	100	100	100	100
Total	2433	2522	2414	2520	2375	2667	2688	2692	2699

Table A.16: Number of Optimal OPTW Solutions Obtained - n = 40

Class	Solution Methods								
(W,T,R)	1	2	3	4	5	6	7	8	9
(1,1,1)	89	100	92	96	98	100	100	100	100
(1,1,2)	92	100	95	97	93	98	100	100	100
(1,1,3)	90	100	94	97	96	100	100	100	100
(1,2,1)	81	72	81	91	81	98	100	97	100
(1,2,2)	86	84	86	93	78	100	100	97	99
(1,2,3)	79	86	83	94	82	99	100	98	99
(1,3,1)	80	66	87	91	76	98	100	100	100
(1,3,2)	82	83	87	94	77	98	100	100	100
(1,3,3)	83	78	82	94	82	100	100	99	100
(2,1,1)	87	100	86	96	88	99	100	100	100
(2,1,2)	89	100	89	98	90	100	100	100	100
(2,1,3)	90	100	88	97	95	100	100	100	100
(2,2,1)	71	70	82	86	69	100	100	99	100
(2,2,2)	82	78	85	92	74	100	100	97	100
(2,2,3)	72	80	80	93	73	96	98	97	99
(2,3,1)	83	83	87	90	83	99	100	97	100
(2,3,2)	91	90	94	93	84	99	99	99	100
(2,3,3)	86	90	91	94	82	97	100	98	100
(3,1,1)	40	99	31	52	49	83	93	99	100
(3,1,2)	51	99	34	56	45	83	96	100	100
(3,1,3)	40	99	36	57	53	76	90	99	100
(3,2,1)	80	65	72	79	62	95	97	100	100
(3,2,2)	84	67	75	84	67	96	98	99	100
(3,2,3)	79	72	76	80	64	97	98	99	100
(3,3,1)	100	100	100	100	100	100	100	100	100
(3,3,2)	100	100	100	100	100	100	100	100	100
(3,3,3)	100	100	100	100	100	100	100	100	100
Total	2187	2361	2193	2394	2141	2611	2669	2674	2697

Table A.17: Number of Optimal OPTW Solutions Obtained - n = 60

Class	Solution Methods								
(W,T,R)	1	2	3	4	5	6	7	8	9
(1,1,1)	57	99	53	77	67	94	100	100	100
(1,1,2)	59	100	62	84	69	90	99	100	100
(1,1,3)	59	100	59	85	73	94	98	100	100
(1,2,1)	48	35	39	75	44	88	95	88	94
(1,2,2)	58	55	56	80	60	84	99	93	100
(1,2,3)	52	53	46	81	48	86	94	94	97
(1,3,1)	43	49	52	86	47	93	98	93	97
(1,3,2)	49	58	59	88	52	90	100	92	99
(1,3,3)	59	59	47	87	65	94	97	96	99
(2,1,1)	60	100	62	88	70	93	97	100	100
(2,1,2)	70	100	70	93	73	97	100	100	100
(2,1,3)	71	100	75	94	81	97	98	100	100
(2,2,1)	44	38	38	69	51	86	99	88	98
(2,2,2)	48	48	51	78	52	87	96	87	98
(2,2,3)	48	51	52	81	52	89	96	95	98
(2,3,1)	42	39	43	76	44	88	96	87	93
(2,3,2)	53	58	48	82	54	91	99	93	98
(2,3,3)	53	56	41	81	49	91	98	93	97
(3,1,1)	13	99	13	26	22	42	61	99	99
(3,1,2)	13	97	13	31	24	45	62	98	98
(3,1,3)	17	98	13	30	21	47	68	99	99
(3,2,1)	83	64	69	76	63	94	97	97	98
(3,2,2)	83	70	76	79	69	97	99	98	99
(3,2,3)	82	70	74	82	66	97	98	99	99
(3,3,1)	100	100	100	100	100	100	100	100	100
(3,3,2)	100	100	100	100	100	100	100	100	100
(3,3,3)	100	100	100	100	100	100	100	100	100
Total	1564	1996	1511	2109	1616	2354	2544	2589	2660

Table A.18: Number of Optimal OPTW Solutions Obtained - n = 100

APPENDIX B

Description of Statistical Methods

Throughout the analysis of our results, we have used non-parametric statistical tests. These techniques are not as powerful as parametric tests, but they have the advantage of still being applicable, even when the assumptions of the parametric tests are not valid. For the comparison of two means, the standard parametric test is the t-test, and the assumptions required for this test are:

- the populations come from a normal distribution
- the variances of the distributions are equal

We explore the validity of these assumptions, by looking at the data for the OPTW that is shown in Table 8.1. Our tests are being carried out over a broad range of test problems, which we have grouped together into the problem classes that we deemed to be relevant. In these cases the problem classes took into account the number of customers serviced, the tightness of the time windows and the variability of the reward functions. The other factors, e.g., the distributions of the locations and the method for applying the time windows, were ignored when defining the problem classes, although these are likely to form sub-groups of problems within the problem classes, which will in turn lead to multi-modality and thus, non-normality of the results.

We consider the results obtained by the APPEND method for the second class of these problems, i.e., those with N=T=1 and R=2, and the histogram and normal

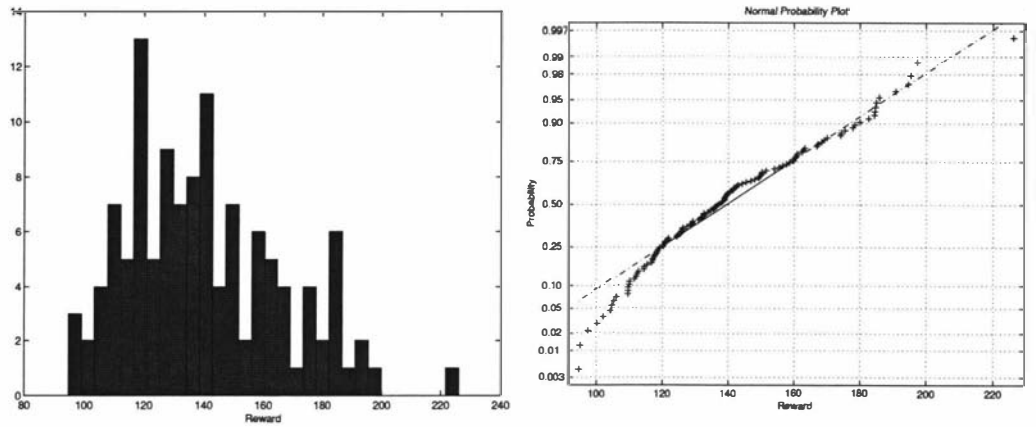

Figure B.1: Histogram and Normal Plot for Class 2

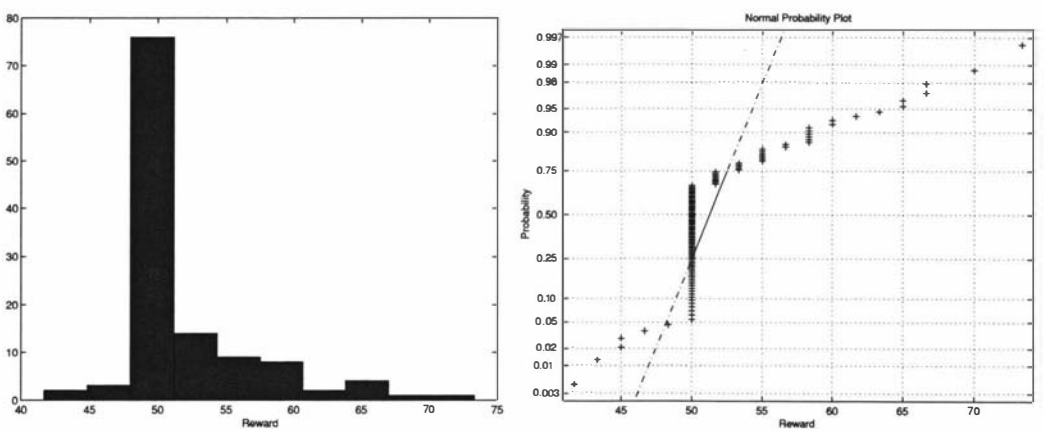

Figure B.2: Histogram and Normal Plot for Class 4

plot for this data are shown in Figure B.1. These plots show that the results deviate from the required assumptions for normality, as the restricted length of the tails, due to the form of the data being used, deviates from the required assumptions. A more dramatic example of non-normality is shown in Figure B.2 for the problem class with N=R=2 and T=1. In this case the majority of the observations fall at the same value, and thus the relatively few observations in the tails indicates the non-normality of the data. Since the observed data, and the actual forms of the data, do not satisfy the required assumptions for normality, we were required to use non-parametric tests for the analysis of the data.

The advantage of non-parametric methods is the fewer assumptions that are required in order for the tests to be applicable. The non-parametric test we used in our preliminary investigations is the Kruskal-Wallis test, which is a non-parametric version of the classical one-way analysis of variance and is used for testing whether

or not there is a significant difference in the medians from a number of data sets. For our data, we implement the Kruskal-Wallis test, using the Statistical Toolbox of Matlab 6.0 [1], to compare the results obtained by the different heuristics on the same data set. Our null hypothesis is that the different heuristics are of equal effectiveness for the data set given, in which case the median of the differences will be zero. The alternative hypothesis is that one of the methods is more effective than the other.

We initially test whether there is a significant difference between any of the heuristics, and if so, we carry out pairwise tests to check where these differences arise. Since we are carrying out multiple, dependent tests on the data set, we use the Bonferroni correction in order to ensure that the total probability of a Type I error is less than the required standard. Thus we use the *multcomp* routine in Matlab, with the Bonferroni correction, in order to correct for the probability of error in these tests.

The Bonferroni correction states that if we carry out n hypothesis tests on a set of data, then in order to keep the probability of error to at most α, we set the α_i for each test to be equal to $\frac{\alpha}{n}$. In the results Tables, a number in bold indicates that the method is significantly more effective than each of the other methods for the problem class, which is the case when the p-value of the individual test is less than $\frac{0.05}{n}$, where n is the number of tests we are implicitly considering.

For more specific testing, we apply the Wilcoxon-Signed Rank Test on the results obtained by pairs of methods. This method tests whether the medians of the difference between the data sets is equal to zero, by taking the ranks of the differences and summing these, and comparing the value obtained with the theoretical t-distribution for these values. Again, if we carry out n tests, we require the p-value of each test to be less than $\frac{0.05}{n}$.

Bibliography

[1] Matlab 6.0.0.88, version 12. `http://www.mathworks.com`.

[2] B.-H. Ahn and J.-Y. Shin. Vehicle-routeing with time windows and time-varying congestion. *Journal of the Operational Research Society*, 42(5):393–400, 1991.

[3] E. Arkin, J. Mitchell, and G. Narasimhan. Resource-constrained geometric network optimization. In *Proceedings of ACM Symposium on Computational Geometry*, 1998.

[4] N. Ascheuer, M. Jünger, and G. Reinelt. A branch and cut algorithm for the asymmetric traveling salesman problem with precedence constraints. Technical Report SC97-70, ZIB, 1997.

[5] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala. New approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. *SIAM Journal on Computing*, 28(1):254–262, 1998.

[6] E. Baker. An exact algorithm for the time-constrained traveling salesman problem. *Operations Research*, 31:938–945, 1983.

[7] N. Balakrishnan. Simple heuristics for the vehicle routeing problem with soft time windows. *Journal of the Operational Research Society*, 44(3):279–287, 1993.

[8] E. Balas. The prize collecting traveling salesman problem. *Networks*, 19:621–636, 1989.

[9] E. Balas. he prize collecting travelling salesman problem: II Polyhedral results. *Networks*, 25:199–216, 1995.

[10] J. Bartholdi and L. Platzman. Heuristics based on spacefilling curves for combinatorial problems in euclidean space. *Management Science*, 34(3):291–305, 1988.

[11] J. Beardwood, J. Halton, and J. Hammersley. The shortest path through many points. *Proceedings of the Cambridge Philosophical Society*, 55:299–328, 1959.

[12] J. Beasley and E. Nascimento. The vehicle routing-allocation problem : a unifying framework. Technical report, Imperial College, London, 1995.

[13] W. Bell, L. Dalberto, M. Fisher, A. Greenfield, R. Jaikumar, P. Kedia, R. Mack, and P. Prutzman. Improving the distribution of industrial gases with an online computerized routing and scheduling optimizer. *Interfaces*, 13(6):4–23, 1983.

[14] D. Bertsimas and G. van Ryzin. A stochastic and dynamic vehicle routing problem in the euclidean plane. *Operations Research*, 39(4):601–615, 1991.

[15] D. Bertsimas and G. van Ryzin. Stochastic and dynamic vehicle routing in the euclidean plane with multiple capacitated vehicles. *Operations Research*, 41(1):60–76, 1993.

[16] D. Bertsimas and G. van Ryzin. Stochastic and dynamic vehicle routing with general demand and interarrival time distributions. *Advances in Applied Probability*, 25:947–978, 1993.

[17] L. Bianco, A. Mingozzi, S. Ricciardelli, and M. Spadoni. Exact and heuristic procedures for the travelling salesman problem with precedence constraints, based on dynamic programming. *INFOR*, 32(1):19–31, 1994.

[18] D. Bienstock, M. Goemans, D. Simchi-Levi, and D. Williamson. A note on the prize collecting traveling salesman problem. *Mathematical Programming*, 59:413–420, 1993.

[19] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[20] R. Brideau III and T. Cavalier. The maximum collection problem with time-dependent rewards. presented at the TIMS conference, 1994.

[21] S. Butt and T. Cavalier. A heuristic for the multiple tour maximum collection problem. *Computers in Operations Research*, 21(1):101–111, 1994.

[22] S. Butt and D. Ryan. Using column generation to solve the multiple tour maximum collection problem. In *Proceedings of the 32nd Annual ORSNZ Conference*, pages 143–148, 1996.

[23] S. Butt and D. Ryan. An optimal procedure for the multiple tour maximum collection problem using collection generation. *Computers and Operations Research*, 427–441, 1999.

[24] M. Byrne. *Aspects of the Vehicle Routing Problem with Pickup and Delivery.* PhD thesis, Massey University, 1993.

[25] I.-M. Chao. *Algorithms and Solutions to Multi-level Vehicle Routing Problems.* PhD thesis, University of Maryland, 1993.

[26] I.-M. Chao, B. Golden, and E. Wasil. A fast and effective heuristic for the orienteering problem. *European Journal of Operational Research*, 88:475–489, 1996.

[27] I.-M. Chao, B. Golden, and E. Wasil. The team orienteering problem. *European Journal of Operational Research*, 88:464–474, 1996.

[28] W.-C. Chiang and R. Russell. A reactive tabu search metaheuristic for the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 9(4):417–430, 1997.

[29] G. Clarke and W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12:568–581, 1964.

[30] J. Current and D. Schilling. The covering salesman problem. *Transportation Science*, 23(3):208–213, 1989.

[31] R. Deitch and S. Ladany. The one-period bus touring problem: solved by an effective heuristic for the orienteering problem and improvement algorithm. *European Journal of Operational Research*, 127(1):69–77, 2000.

[32] M. Desrochers, J. Desrosiers, and M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2):342–354, 1992.

[33] M. Desrochers, C. Jones, J. Lenstra, M. Savelsbergh, and L. Stougie. Towards a model and algorithm management system for vehicle routing and scheduling problems. *Decision Support Systems*, 25(2):109–133, 1999.

[34] M. Desrochers, J. Lenstra, and M. Savelsbergh. Vehicle routing with time windows : optimization and approximation. In B. L. Golden and A. A. Assad, editors, *Vehicle Routing : Methods and Studies*, pages 65–84. North Holland, 1988.

[35] M. Desrochers, J. Lenstra, and M. Savelsbergh. A classification scheme for vehicle routing and scheduling problems. *European Journal of Operational Research*, 46:322–332, 1990.

[36] M. Diaby and R. Ramesh. The distribution problem with carrier service: a dual based penalty approach. *ORSA Journal on Computing*, 7(1):24–35, 1995.

[37] M. Dorigo and L. Gambardella. Ant colonies for the traveling salesman problem. *BioSystems*, 43:73–81, 1997.

[38] M. Dror and P. Trudeau. Savings by split delivery routing. *Transportation Science*, 23(2):141–145, 1989.

[39] M. Dror and P. Trudeau. Split delivery routing. *Naval Research Logistics Quarterly*, 37:383–402, 1990.

[40] G. Dueck and T. Scheuer. Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, 90:161–175, 1990.

[41] Y. Dumas, J. Desrosiers, E. Gelinas, and M. Solomon. An optimal algorithm for the traveling salesman problem with time windows. *Operations Research*, 43(2):367–371, 1995.

[42] Y. Dumas, J. Desrosiers, and F. Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54:7–22, 1991.

[43] Y. Dumas, F. Soumis, and J. Desrosiers. Optimizing the schedule for a fixed vehicle path with convex inconvenience costs. *Transportation Science*, 24(145–152), 1990.

[44] S. Eilon, C. Watson-Gandy, and N. Christofides. *Distribution Management*. Charles Griffin and Company, London, 1971.

[45] E. Erkut and J. Zhang. The maximum collection problem with time-dependent rewards. *Naval Research Logistics*, 43:749–763, 1996.

[46] L. Escudero. An inexact algorithm for the sequential ordering problem. *European Journal of Operational Research*, 37:236–253, 1988.

[47] R. Fahrion and M. Wrede. On a principle of chain-exchange for vehicle routeing problems (1-VRP). *Journal of the Operational Research Society*, 41(9):821–827, 1990.

[48] T. Feo and M. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[49] J. Ferland and L. Fortin. Vehicles scheduling with sliding time windows. *European Journal of Operational Research*, 38:213–226, 1989.

[50] M. Fischetti, J. Gonzàlez, and P. Toth. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, 45(3):378–394, 1997.

[51] M. Fischetti, J. González, and P. Toth. Solving the orienteering problem through branch-and-cut. *INFORMS Journal on Computing*, 10(2):133–148, 1998.

[52] M. Fischetti, G. Laporte, and S. Martello. The delivery man problem and cumulative matroids. *Operations Research*, 41(6):1055–1064, 1993.

[53] M. Fischetti and P. Toth. An additive approach for the optimal solution of the prize-collecting traveling salesman problem. In B.L. Golden and A.A. Assad, editors, *Vehicle Routing: Methods and Studies*, pages 319–343. North Holland, 1988.

[54] M. Fisher and R. Jaikumar. A generalized assignment heuristic for the vehicle routing problem. *Networks*, 11:109–124, 1981.

[55] G. Frederickson and D. Guan. Preemptive ensemble motion planning on a tree. *SIAM Journal on Computing*, 21(6):178–193, 1978.

[56] G. Frederickson, M. Hecht, and C. Kim. Approximation algorithms for some routing problems. *SIAM Journal on Computing*, 7(2):178–193, 1978.

[57] P. Frizzell. *Aspects of the Vehicle Routing and Scheduling Problem.* PhD thesis, Massey University, 1993.

[58] B. Gaboune, G. Laporte, and F. Soumis. Expected distances between two uniformly distributed random points in rectangles and rectangular parallelepipeds. *Journal of the Operational Research Society*, 44(5):513–519, 1993.

[59] M. Gendreau, F. Guertin, J.-Y. Potvin, and E. Taillard. Parallel tabu search for real-time vehicle routing and dispatching. *Transportation Science*, 33(4):381–390, 1999.

[60] M. Gendreau, A. Hertz, and G. Laporte. New insertion and post-optimization procedures for the traveling salesman problem. *Operations Research*, 40:1086–1094, 1992.

[61] M. Gendreau, A. Hertz, and G. Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40(10):1276–1290, 1994.

[62] M. Gendreau, A. Hertz, G. Laporte, and M. Stan. A generalized insertion heuristic for the traveling salesman problem with time windows. *Operations Research*, 43(3):330–335, 1998.

[63] M. Gendreau, G. Laporte, and F. Semet. Solving an ambulance location model by tabu search. *Location Science*, 5(2):75–88, 1997.

[64] M. Gendreau, G. Laporte, and F. Semet. A branch-and-cut algorithm for the undirected selective traveling salesman problem. *Networks*, 32:263–273, 1998.

[65] M. Gendreau, G. Laporte, and D. Vigo. Heuristics for the traveling salesman problem with pickup and delivery. *Computers and Operations Research*, 26:699–714, 1999.

[66] D. Gensch. An industrial application of the traveling salesman's subtour problem. *AIIE Transactions*, 10(4):362–370, 1978.

[67] B. Gillett and L. Miller. A heuristic algorithm for the vehicle dispatching algorithm. *Operations Research*, 22:340–349, 1974.

[68] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers in Operations Research*, 13:533–549, 1986.

[69] F. Glover. Tabu search : a tutorial. *Interfaces*, 20(4):74–94, 1990.

[70] M. Goemans and D. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24:296–317, 1995.

[71] B. Golden, A. Assad, and R. Dahl. Analysis of a large scale vehicle routing problem with an inventory component. *Large Scale Systems*, 7:181–190, 1984.

[72] B. Golden, L. Levy, and R. Dahl. Two generalizations of the traveling salesman problem. *OMEGA*, 9(4):439–441, 1981.

[73] B. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval Research Logistics*, 34:307–318, 1987.

[74] B. Golden, Q. Wang, and L. Liu. A multifaceted heuristic for the orienteering problem. *Naval Research Logistics*, 35:359–366, 1988.

[75] L. Gomes, V. Diniz, and C. Martinhon. An hybrid GRASP+VNS metaheuristic for the prize-collecting traveling salesman problem. `ftp://jambo.caa.uff.br/pub/docs/reltec/rt_06-00.ps.gz`, 2000.

[76] M. Göthe-Lundgren, K. Jörnsten, and P. Värbrand. On the nucleolus of the basic vehicle routing game. *Mathematical Programming*, 72:83–100, 1996.

[77] M. Grötschel, D. Hauptmeier, S. Krumke, and J. Rambau. Simulation studies for the online dial-a-ride problem. Technical Report SC-99-09, Konrad-Zuse-Zentrum für Informationtechnik Berlin, 1999.

[78] D. J. Guan. Routing a vehicle of capacity greater than one. *Discrete Applied Mathematics*, 81(1):41–57, 1998.

[79] R. Hall. The fastest path through a network with random time-dependent travel times. *Transportation Science*, 20(3):182–188, 1986.

[80] D. Hauptmeier, S. Krumke, and J. Rambau. The online dial-a-ride problem under reasonable load. Technical Report SC-99-08, Konrad-Zuse-Zentrum für Informationtechnik Berlin, 1999.

[81] D. Hauptmeier, S. Krumke, J. Rambau, and H.-C. Wirth. Euler is standing in line. Technical Report SC-99-06, Konrad-Zuse-Zentrum für Informationtechnik Berlin, 1999.

[82] P. Healy and R. Moll. A new extension of local search applied to the dial-a-ride problem. *European Journal of Operational Research*, 83:83–104, 1995.

[83] A. Hill, V. Mabert, and D. Montgomery. A decision support system for the courier vehicle scheduling problem. *OMEGA*, 16(4):333–345, 1988.

[84] J. Homberger. Extended Solomon's VRPTW instances. `http://www.fernuni-hagen.de/WINF/touren/menuefrm/probinst.htm`.

[85] J. Hooker. Testing heuristics: we have it all wrong. *Journal of Heuristics*, 1:33–42, 1995.

[86] S. Ichoua, M. Gendreau, and J.-Y. Potvin. Diversion issues in real-time vehicle dispatching. *Transportation Science*, 34(4):426–438, 1999.

[87] G. Lapalme J.-Y. Potvin and J.-M. Rousseau. A generalized k-opt exchange procedure for the MTSP. *INFOR*, 27(4):474–481, 1989.

[88] P. Jaillet. *The Probabilistic Traveling Salesman Problem.* PhD thesis, Department of Civil Engineering, MIT, 1985.

[89] J.-J. Jaw, A. Odoni, H. Psaraftis, and N. Wilson. A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research - Series B*, 20B(3):243–257, 1986.

[90] M. Johnston. *Dynamic Routing with Competition: Foundations and Strategies.* PhD thesis, Massey University, 1999.

[91] M. Kantor and M. Rosenwein. The orienteering problem with time windows. *Journal of the Operational Research Society*, 43(6):629–635, 1992.

[92] S. Kataoka and S. Morito. An algorithm for single constraint maximum collection problem. *Journal of the Operational Research Society of Japan*, 31(4):515–531, 1988.

[93] S. Kataoka, T. Yamada, and S. Morito. Minimum directed 1-subtree relaxation for score orienteering problem. *European Journal of Operational Research*, 104:139–153, 1998.

[94] C. Keller. *Multiobjective Routing through Space and Time.* PhD thesis, University of Western Ontario, 1985.

[95] C. Keller. Algorithms to solve the orienteering problem : a comparison. *European Journal of Operational Research*, 41:224–231, 1989.

[96] C. Keller and M. Goodchild. The multiobjective vending problem : a generalization of the travelling salesman problem. *Environment and Planning B : Planning and Design*, 15:447–460, 1988.

[97] A. Klegweyt and J. Papastavrou. Acceptance and dispatching policies for a distribution problem. *Transportation Science*, 32(2):127–141, 1998.

[98] A. Klegweyt and J. Papastavrou. The dynamic and stochastic knapsack problem. *Operations Research*, 46(1):17–35, 1998.

[99] Y. Koskosidis, W. Powell, and M. Solomon. An optimization-based heuristic for vehicle routing and scheduling with soft time window constraints. *Transportation Science*, 26(2):69–85, 1992.

[100] S. Krumke. News from the online travelling repairman. Technical Report SC-00-08, Konrad-Zuse-Zentrum für Informationtechnik Berlin, 2000.

[101] M. Kubo and H. Kasugai. Heuristic algorithms for the single vehicle dial-a-ride problem. *Journal of the Operations Research Society of Japan*, 33(4):354–365, 1990.

[102] M. Kubo and H. Kasugai. The precedence constrained traveling salesman problem. *Journal of the Operations Research Society of Japan*, 34(2):152–172, 1991.

[103] M. Kubo and H. Kasugai. On symmetric subtour problems. *Japan Journal of Industrial and Apllied Mathematics*, 9(3):383–396, 1992.

[104] S. Ladany. Optimal tourist bus tours. *Tourism Economics*, 5(2):175–190, 1999.

[105] M. Laguna, J. Barnes, and F. Glover. Tabu search methods for a single machine scheduling problem. *Journal of International Manufacturing*, 2:63–73, 1991.

[106] A. Langevin, M. Desrochers, J. Desrosiers, E. Gelinas, and F. Soumis. A two-commodity flow formulation for the traveling salesman and the makespan problems with time windows. *Networks*, 23:631–640, 1993.

[107] G. Laporte, A. Asef-Vaziri, and C. Sriskandarajah. Some applications of the generalized travelling salesman problem. *Journal of the Operational Research Society*, 47:1461–1467, 1996.

[108] G. Laporte and S. Martello. The selective traveling salesman problem. *Discrete Applied Mathematics*, 26:193–207, 1990.

[109] G. Laporte and Y. Nobert. Generalized travelling salesman problems through n sets of nodes : an integer programming approach. *INFOR*, 21:61–75, 1983.

[110] G. Laporte and F. Semet. Computational evaluation of a transformation procedure for the symmetric generalized traveling salesman problem. *INFOR*, 37(2):114–120, 1999.

[111] J. Larsen. Vehicle routing with time windows - finding optimal solutions efficiently. `http://citeseer.nj.nec.com/larsen99vehicle.html`, 1999.

[112] A. Leifer and M. Rosenwein. Strong linear programming relaxations for the orienteering problem. *European Journal of Operational Research*, 73:517–523, 1994.

[113] S. Lin. Computer solutions of the traveling salesman problem. *Bell Systems Technical Journal*, 44:2245–2269, 1969.

[114] S. Lin and B. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21:498–516, 1973.

[115] M. Lübbecke. *Engine Scheduling by Column Generation.* PhD thesis, Technischen Universität Braunschweig, 2001.

[116] A. Lucena. Time-dependent traveling salesman problem - the deliveryman case. *Networks*, 20:753–763, 1990.

[117] K. Lund, O. Madsen, and J. Rygaard. Vehicle routing with varying degrees of dynamism. Technical Report IMM-REP-1996-1, IMM, Technical University of Denmark, 1996.

[118] M. and D. Johnson. *Computers and Intractability : a Guide to the theory of NP-Completeness.* W.H. Freeman, San Francisco, 1979.

[119] O. Madsen, H. Ravn, and J. Rygaard. A heuristic algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. *Annals of Operations Research*, 60:193–208, 1995.

[120] C. Malandraki and M. Daskin. Time dependent vehicle routing problems : formulations, properties and heuristic algorithms. *Transportation Science*, 26(3):185–200, 1992.

[121] C. Malandraki and M. Daskin. The maximum benefit Chinese postman problem and the maximum benefit traveling salesman problem. *European Journal of Operational Research*, 65:218–234, 1993.

[122] H. Millar. Planning fish scouting activity in industrial fishing. *Fisheries Research*, 25:63–75, 1996.

[123] H. Millar and M. Kiragu. A time-based formulation and upper bounding scheme for the selective travelling salesperson problem. *Journal of the Operational Research Society*, 48(5):511–518, 1997.

[124] A. Mingozzi, L. Bianco, and S. Ricciardelli. Dynamic programming strategies for the traveling salesman problem with time window and precedence constraints. *Operations Research*, 45(3):365–377, 1997.

[125] J. Mittenthal and C. Noon. An insert/delete heuristic for the traveling salesman subset-tour problem with one additional constraint. *Journal of the Operational Research Society*, 43(3):277–283, 1992.

[126] P. Moscato and M. Norman. A memetic approach for the traveling salesman problem implementation of a computational ecology for combinatorial optimization on message-passing systems. In M. Valero, E. Onate, M. Jane, J. Larriba, and B. Suarez, editors, *Parallel Computing and Transputer Applications*, pages 177–186. IOS Press, 1992.

[127] P. Mullaseril, M. Dror, and J. Leung. Split-delivery routeing heuristics in livestock food distribution. *Journal of the Operational Research Society*, 48:107–116, 1997.

[128] W. P. Nanry and J. W. Barnes. Solving the pickup and delivery problem with time windows using reactive tabu search. *Transportation Research Part B*, 34:107–121, 2000.

[129] I. Or. *Traveling Salesman-Type Combinatorial Problems and their Relation to the Logistics of Blood Banking.* PhD thesis, Department of Industrial Engineering and Management Science, Northwestern University, 1976.

[130] I. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41:421–451, 1993.

[131] J. Papastavrou. A heavy traffic lower bound for a stochastic and dynamic vehicle routing problem. Technical report, Purdue University, 1994.

[132] J. Papastavrou. A stochastic and dynamic routing policy using branching processes with state dependent immigration. *European Journal of Operational Research*, 95(1):167–177, 1996.

[133] J. Papastavrou, S. Rajogopalan, and A. Klegweyt. The dynamic and stochastic knapsack problem with deadlines. *Management Science*, 42(12):1706–1718, 1996.

[134] J. Partyka and R. Hall. On the road to service. *OR/MS Today*, 27(4):26–35, 2000.

[135] J. Pekny, D. Miller, and G. McRae. An exact parallel algorithm for scheduling when production costs depend on consecutive system states. *Computers and Chemical Engineering*, 14(9):1009–1023, 1990.

[136] J.-C. Picard and M. Queyranne. The time-dependent traveling salesman problem and its applications to the tardiness problem in one-machine scheduling. *Operations Research*, 26(1):86–110, 1978.

[137] J.-Y. Potvin. The traveling salesman problem: A neural network perspective. *ORSA Journal on Computing*, 5:328–348, 199.

[138] J.-Y. Potvin and S. Bengio. The vehicle routing problem with time windows part II: genetic search. *INFORMS Journal on Computing*, 8(2):165–172, 1996.

[139] J.-Y. Potvin, T. Kervahut, B.-L. Garcia, and J.-M. Rousseau. The vehicle routing problem with time windows part I: tabu search. *INFORMS Journal on Computing*, 8(2):158–164, 1996.

[140] J.-Y. Potvin and J.-M. Rousseau. An exchange heuristic for routeing problems with time windows. *Journal of the Operational Research Society*, 46:1433–1446, 1995.

[141] H. Psaraftis. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14(2):130–154, 1980.

[142] H. Psaraftis. Analysis of an $O(n^2)$ heuristic for the single vehicle many-to-many Euclidean dial-a-ride problem. *Transportation Research - Series B*, 17B:133–145, 1983.

[143] H. Psaraftis. k-Interchange procedures for local search in a precedence-constrained routing problem. *European Journal of Operational Research*, 13:391–402, 1983.

[144] H. Psaraftis. Scheduling large-scale advance-request dial-a-ride problems. *American Journal of Mathematical and Management Sciences*, 6:327–367, 1986.

[145] H. Psaraftis. Dynamic vehicle routing problems. In B.L. Golden and A.A. Assad, editors, *Vehicle Routing: Methods and Studies*, pages 223–248. North Holland, 1988.

[146] H. Psaraftis. Dynamic vehicle routing : status and prospects. *Annals of Operations Research*, 61:143–164, 1995.

[147] T. Ralphs, L. Kopmanz, W. Pulleyblank, and L. Trotter. On the capacitated vehicle routing problem. `http://www.lehigh.edu/~tkr2/research/papers/vrp.pdf`, 2001.

[148] R. Ramesh and K. Brown. An efficient four-phase heuristic for the generalized orienteering problem. *Computers in Operations Research*, 18(2):151–165, 1991.

[149] R. Ramesh, Y.-S. Yoon, and M. Karwan. An optimal algorithm for the orienteering tour problem. *ORSA Journal on Computing*, 4(2):155–165, 1992.

[150] C. Rivers. On route re-distribution of pay load. In *ORSNZ Conference, Hamilton*, 1999.

[151] Y. Rochat and E. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1:147–167, 1995.

[152] P. Rogerson. Spatial search for the lowest price. *Geographical Analysis*, 22(4):336–347, 1990.

[153] J.-M. Rousseau, G. Pesant, M. Gendreau, and J.-Y. Potvin. On the flexibility of constraint programming models: from single to multiple time windows for the traveling salesman problem. *European Journal of Operational Research*, 117(2):253–263, 1999.

[154] M. Savelsbergh. Local search in routing problems with time windows. *Annals of Operations Research*, 4:285–305, 1985.

[155] M. Savelsbergh. An efficient implementation of local search algorithms for constrained routing problems. *European Journal of Operational Research*, 47:75–85, 1990.

[156] M. Savelsbergh. Computer aided routing. CWI Tract 75, CWI, Amsterdam, 1992.

[157] M. Savelsbergh. The vehicle routing problem with time windows: minimizing route duration. *ORSA Journal on Computing*, 4(2):146–154, 1992.

[158] M. Savelsbergh and M. Sol. The general pickup and delivery problem. *Transportation Science*, 29(1):17–29, 1995.

[159] M. Savelsbergh and M. Sol. DRIVE: Dynamic routing of independent vehicles. *Operations Research*, 46(4):474–490, 1998.

[160] T. Sexton and L. Bodin. Optimizing single vehicle many-to-many operations with desired delivery times : I Scheduling. *Transportation Science*, 19(4):378–410, 1985.

[161] T. Sexton and L. Bodin. Optimizing single vehicle many-to-many operations with desired delivery times : II Routing. *Transportation Science*, 19(4):411–435, 1985.

[162] T. Sexton and Y.-M. Choi. Pickup and delivery of partial loads with "soft" time windows. *American Journal of Mathematical and Management Sciences*, 6:369–398, 1986.

[163] D. Skorin-Kapov and J. Skorin-Kapov. On tabu search for the location of interacting hub facilities. *European Journal of Operational Research*, 73(3):502–509, 1994.

[164] P. Sokkappa. *The Cost-Constrained Traveling Salesman Problem.* PhD thesis, University of California, Livermore, CA, 1990.

[165] M. Solomon. On the worst-case performance of some heuristics for the vehicle routing and scheduling problem with time window constraints. *Networks*, 16:161–174, 1986.

[166] M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.

[167] M. Solomon, E. Baker, and J. Schaffer. Vehicle routing and scheduling with time window constraints : efficient implementations of solution improvement procedures. In B. L. Golden and A. A. Assad, editors, *Vehicle Routing: Methods and Studies*, pages 85–105. North Holland, 1988.

[168] D. Stein. An assymptotic probabilistic analysis of a routing problem. *Mathematics of Operations Research*, 3(2):89–101, 1978.

[169] W. Stewart and B. Golden. Stochastic vehicle routing: a comprehensive approach. *European Journal of Operational Research*, 14:371–385, 1985.

[170] M. Swihart and J. Papastavrou. A stochastic and dynamic model for the single-vehicle pick-up and delivery problem. *European Journal of Operational Research*, 114(3):447–464, 1999.

[171] E. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J.-Y. Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31(2):170–186, 1997.

[172] S. Thangiah, I. Osman, R. Vinayagamoorthy, and T. Sun. Algorithms for the vehicle routing problems with time deadlines. *American Journal of the Mathematical and Management Sciences*, 13(3&4):323–355, 1993.

[173] S. Thangiah, J.-Y. Potvin, and T. Sun. Heuristic approaches to vehicle routing with backhauls and time windows. *Computers and Operations Research*, 23(11):1043–1058, 1996.

[174] P. Thompson and H. Psaraftis. Cyclic transfer algorithms for multivehicle routing and scheduling problems. *Operations Research*, 41(5):935–946, 1993.

[175] P. Toth and D. Vigo. Fast local search algorithms for the handicapped persons transportation problem. In I. Osman and J. Kelly, editors, *Metaheuristics*, chapter 41, pages 677–690. Kluwer Academic, Amsterdam, 1996.

[176] T. Tsiligirides. Heuristic methods applied to orienteering. *Journal of the Operational Research Society*, 35(9):797–809, 1984.

[177] J. Tsitsiklis. Special cases of traveling salesman and repairman problems with time windows. *Networks*, 22:263–282, 1992.

[178] V. Valls, A. Perez, and S. Quintanilla. A tabu search approach to machine scheduling. *European Journal of Operational Research*, 106(2/3):277–300, 1998.

[179] A. van Breedam. A parametric analysis of heuristics for the vehicle routing problem with side-constraints. *European Journal of Operational Research*, 137:83–104, 2002.

[180] L. van der Bruggen, J. Lenstra, and P. Schuur. Variable-depth search for the single-vehicle pickup and delivery problem with time windows. *Transportation Science*, 27(3):298–311, 1993.

[181] R. van der Wiel and N. Sahinidis. Heuristic bounds and test problem generation for the time-dependent traveling salesman problem. *Transportation Science*, 29(2):167–183, 1995.

[182] H. van Landeghem. A bi-criteria heuristic for the vehicle routing problem with time windows. *European Journal of Operational Research*, 36:217–226, 1988.

[183] B. Verweij and K. Aardal. The merchant subtour problem. `ftp://ftp.cs.uu.nl/pub/RUU/CS/techreps/CS-2000/2000-35.ps.gz`, 2000.

[184] T. Volgenant and R. Jonker. On some generalizations of the travelling-salesman problem. *Journal of the Operational Research Society*, 38(11):1073–1079, 1987.

[185] Q. Wang, X. Sun, B. Golden, and J. Jia. Using artificial neural networks to solve the orienteering problem. *Annals of Operations Research*, 61:111–120, 1995.

[186] C. Yang. A dynamic programming algorithm for the travelling repairman problem. *Asia Pacific Journal of Operational Research*, 6:192–206, 1989.