GRAPH THEORETIC FACILITY LAYOUT DESIGN AND EVALUATION: THEORETICAL AND PRACTICAL CONSIDERATIONS

A THESIS PRESENTED IN PARTIAL FULFILMENT
OF THE REQUIREMENTS FOR
THE DEGREE OF PH.D. IN
OPERATIONS RESEARCH AT
MASSEY UNIVERSITY

Kelvin Watson

January 1996

658.23015118
Wat

20 DL

Abstract

In this thesis we examine the Graph Theoretic Facility Layout Problem (GTFLP). The GTFLP is concerned with designing a building layout, with a specified number of facilities, with data relating to the facilities' areas, and proximity desirability ratings or material flows between the facilities. The objective is to design an efficient layout which incorporates these issues, by attempting to minimise the transportation cost of material flow between facilities, or maximise the desirability ratings, and designing *regularly* shaped facilities which allow effectiveness of the layout.

The GTFLP proceeds as a two phase process; the first generates a highly weighted (maximal) planar graph, called an adjacency graph, which specifies the relative spatial location of each facility, with respect to its adjacent facilities. This phase has been extensively studied, and although not a focus of this thesis, we address adjacency graph generation and provide a worst case analysis of the so-called TESSA method.

The main thrust of this thesis addresses the second phase of the GTFLP, where we examine the construction of the layout in light of the information given by the first phase. We review previous literature in this area, and extend this work by a series of enhancements to existing methods, and introduction of new techniques including: introducing the Vertex Splitting Algorithm, the Tiling Algorithm, and the SIMPLE Algorithm; analysis of previous methods, by completing the theory of the Deltahedron and Contraction Layout Algorithms for instance. Initial steps in characterising adjacency graphs, which by their structure allow the easy construction of a corresponding layout, is introduced, by providing a series of template layouts; furthermore we compare and contrast algorithms which force an overlying grid structure against those more generic methods, which do not impose this rigidity; and introduce some simple procedures for improving the regularity of a layout.

We formally define the concept of *regularity*, by presenting a series of quantifiable measures, which can be calculated to give an evaluation of the effectiveness of a layout. Thereby we attempt to quantifiably compare and rank the layout generation methods, by evaluating the regularity measures over a set of test problems. The effects of the various layout improvements, and initialisation processes will be shown within this computational process. We also examine the incorporation of a Material Handling System (MHS) within a layout. The calculation of the transportation costs involved in the implementation of each layout, via the Material Handling System, provides another mechanism for ranking the layout algorithms. Directions for future work are provided in the area of the Material Handling System. Indeed our work in this area only highlighted the importance of modelling this concept.

The final contribution of this thesis is the generation of a framework which attempts to look beyond the more theoretical GTFLP model. By invoking a three phase process, which allows the decomposition of the adjacency graph, interaction with a decision planner, and the ability to perturb the problem constraints, we can produce a range of alternative layout scenarios, since there is no *right* answer to this second phase, and indeed an infinite number of different layouts satisfy the problem constraints. This allows the design process to be directed in a more meaningful way, by exploiting structure within the adjacency graph and the working knowledge of a decision planner, providing a basis whereby the GTFLP can be effectively used within any building design process.

We conclude that the GTFLP model is an important concept within the more general Facility Layout Problem. We provide evidence that the standard Graph Theoretic model is perhaps overly restrictive. Indeed we shall see that the generation of a good adjacency graph does not in general correspond to obtaining a practical layout. With this in mind, we have identified the strengths and weaknesses of the various concepts and ideas used within Graph Theoretic Facility Layout Design, and consequently have created an integration of the adjacency graph and layout phases of this problem. This has provided a unification of the GTFLP into a more malleable form, which provides enough flexibility to accurately model the mechanics behind the design process.

Acknowledgements

This work would not have been possible without the support and encouragement of a number of people. Firstly I would like to thank my Lord Jesus, who saved me, and gave my life direction and vision. I am indebted to my supervisor, Dr John Giffin (who probably stopped reading after the previous sentence), who has taught me everything I know about Operations Research and in particular the Graph Theoretic Facility Layout Problem. He has doubled as my English teacher, correcting many grammatical errors and replacing my *fluffisms* with what I really meant. My wife, Veronica has been a constant source of love and encouragement, and I would like to thank her not only for this, but also for her perserverence and understanding. I would like to thank Mum and Dad for their support, and for fostering my interest in learning in my early years. Many thanks to Mark Johnston and Richard Rayner who managed to teach me enough about computers to make me dangerous. I would also like to acknowledge the support of John and Sandra Griffin through prayer, and finally thanks go to Placemakers without whom I could never have learnt the *practical aspects of facility layout.*

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The Facility Layout Problem (FLP) seeks the best spatial arrangement of a set of facilities, which minimises the total transportation cost of material flow between facilities, or maximises total facility adjacency benefits. The FLP has applications in arranging rooms within a building floorplan, placing machines on a factory floor, controls on an instrument panel, or components on a circuit board. For this reason, the FLP is an important problem within Industrial Design.

The FLP has been modelled as a Quadratic Assignment Problem [63], and a Graph Theoretic Problem [33]. Other models have been developed [72, 97] for instance, but essentially are based on either of these two models. We concentrate on the second model; the Graph Theoretic Facility Layout Problem (GTFLP). It was first proposed by Foulds and Robinson [33], as an alternative to the QAP; we will discuss more fully the contrasts between these two models in Chapter 2. The GTFLP is composed of two phases. The first phase is called the *Adjacency Problem*, where we seek to determine the spatial relationships between the facilities by generating an *adjacency* graph, in which each facility is represented by a vertex, including the *outside*, or *exterior* region, with an edge connecting two vertices of this graph if it is deemed desirable to have the corresponding facilities sharing a common wall within the layout. The adjacency graph must be planar for the second phase to be feasible, and in an optimization context this translates to maximal planarity, requiring that the Adjacency Graph is a Maximal Planar Graph (MPG).

The second phase is called the *Layout Problem*, where we attempt to determine an *orthogonal geometric dual* to the adjacency graph, while also incorporating area specifications on the facilities. This second phase is the main focus of this thesis: the construction of a practical layout, attaining certain regularity, efficiency, and aesthetic conditions. Somewhat surprisingly this second phase has attracted far less attention than the first phase in the literature. The reasons for this are not clear, perhaps due to the somewhat *fuzzy* nature of the problem, in that there exists no predefined objective guiding the construction of the layout (other than consistency with the first phase), and indeed, as we will see, there seldom exists a *right* or *best* answer. Rather the development of the layout must be able incorporate any qualitative measures, which are in general difficult to describe, let alone measure, and hence this phase can be very subjective. In pursuit of these goals, we do not only attempt to unify the literature directed at this phase, and indeed the GTFLP as a whole, but also to develop from this a mechanism whereby the subjective nature of the problem can be fully accommodated within the GTFLP model, developing the theoretical model into one which could prove to be a powerful tool within industrial design.

Background reading in the form of survey articles includes Foulds [30], Hassan and Hogg [51], Kusiak and Heragu [68], and Meller and Gau [84]. These papers provide summaries of the most common formulations of the FLP, as well as providing descriptions of the more widely utilised techniques in this area. A working knowledge of Graph Theoretic techniques and ideas is assumed; see for example Bondy and Murty [11].

Coda

In the next chapter we proceed to formally define the GTFLP. We present the definitions and descriptions that will be extensively used throughout this thesis, as well as providing the formulation of the GTFLP, and details of the more classical approach, the QAP. We also define the concept of *regularity*, which will become an important tool as we attempt to provide an extensive analysis of the GTFLP model. In Chapters 3 and 4, we review the two phases of the GTFLP, respectively, including new ideas and motivations, leading to the development of new approaches, for the second phase especially. Following these chapters, we embark upon an attempt to set the GTFLP within a more interactive design process, in order to show that the GTFLP is an important tool for any design process, and to provide the mechanism whereby this can be achieved.

Chapter 2

Problem Description

This chapter lays the foundation for the work that will be undertaken throughout this thesis. We begin by defining some standard notation and definitions for the GTFLP. We then examine some different formulations of the Facility Layout Problem, including the evolution and justification of the GTFLP. Finally we briefly critique the relevant previous literature, and define the concept of regularity.

2.1 Nomenclature and Definitions

In this section we set up some standard Graph Theoretic nomenclature, and provide some of the more important definitions and descriptions that we will require specifically for the GTFLP.

G	The graph G which is an ordered triple with vertex set V, edge set E and weights on the edges w_e, $e \in E$
n	The number of facilities, $\|V\|$. This includes a facility to represent the exterior of the layout, generally labelled e, or *exterior*
(i,j)	The edge (i,j), $(i,j) \in E$.
G_p	A subgraph of G having vertex set V_p ($\subseteq V$), edge set E_p, where $(i,j) \in E_p$ if $\{i,j\} \subseteq V_p$, and $\{(i,j)\} \in E$, and weights on the edges w_e, $e \in E_p$.
d_i	The degree of vertex i, that is the number of edges incident on i.
K_n	The *complete* graph on n vertices, with all pairs of vertices joined by an edge.

Maximal Planar Graph A maximal planar graph (MPG) is a graph which can be drawn in the plane with no edges crossing (planarity), and such that no more edges can be added to this graph without violating this planarity (maximality). Every face of an MPG is a triangle. This set of faces (or triangles) is denoted by T. A face bounded by edges (i, j), (i, k), and (j, k) is represented by its three vertices *i.e.* (i, j, k). Note that T is usually used as the data structure to represent an MPG. Euler showed that in every MPG, $|E| = 3n - 6$, and $|T| = 2n - 4$ (Euler Polyhedral Formulae).

Layout An orthogonal geometric dual of an MPG in which each vertex of the MPG corresponds to a facility, and each edge (i, j) of the MPG represents the requirement of a common length of wall, between i and j. A layout is also known as a block plan.

Benefit Matrix The matrix W where each entry w_{ij} represents the benefit of placing facilities i and j adjacent. The benefit matrix is also referred to as an adjacency matrix, relationship chart, relationship matrix or REL chart. In some instances we refer to the cost/flow matrix W where each entry w_{ij} represents the material flow between facilities i and j.

AEIOUX Scheme An ordinal scale which provides qualitative importance of an adjacency in the benefit matrix, as an alternative to quantitative cost/flow data. This scale and some commonly used numerical values associated with them are given in Table 2.1.

Symbol	Meaning	Weights	
A	Absolutely Essential	64	5
E	Essential	16	4
I	Important	4	3
O	Optional	1	2
U	Unimportant	0	1
X	Forbidden	-256	0

Table 2.1: The AEIOUX Scheme

Exterior Face The infinite region of any graph is referred to as the exterior or boundary face.

Adjacent, Non-Adjacent Two facilities in the layout are adjacent if they share a common length of wall in the layout; conversely they are non-adjacent if there is no common length of wall between them in the layout. In an MPG, two vertices are adjacent if they are connected by an edge, and non-adjacent if there is no edge connecting them.

Area Requirements A facility i has an area requirement, also called an area specification, denoted by a_i. a_i is the area of facility i in the layout. In an MPG the area specifications correspond to vertex weights.

Layout Perimeter The layout perimeter is the boundary of a rectangular polygon, whose area is equal to the sum of the facility areas to be placed in the layout. The layout perimeter is also known as the layout boundary.

3-Joint, 2-Joint An important artefact of any layout is the confluence of three walls called a 3-joint, and the confluence of two walls called a 2-joint. Each joint is a corner coordinate of at least two facilities. 2-joints are created as part of the orthogonalisation process. The four corners of the layout perimeter are all 2-joints. We also refer to 4-joints, which are similarly defined. Note that 4-joints can never occur in a layout with an underlying MPG adjacency structure.

Duality A Duality exists between an MPG, and its corresponding layout. The MPG is easily obtainable from the layout, but given an MPG it is usually not easy to find an orthogonal geometric dual. The vertices of the MPG correspond to the facilities of the layout, the edges of the MPG correspond to the sharing of positive wall length in the layout, and the faces of the MPG correspond to the 3-joints in the layout.

Separating Triangle A 3-cycle in an MPG which is not a face of the MPG is called a separating triangle, *i.e.* for a separating triangle (i, j, k), $\{(i, j), (i, k), (j, k)\} \subset E$, $\{(i, j, k)\} \notin T$. Separating Triangles are sometimes referred to as Complex Triangles.

Cyclic Ordering The cyclic ordering of a vertex x is a cyclic permutation of the list of edges adjacent to x. We denote this by $cyc_x(k)$ where k is the first vertex in the cyclic ordering. Unless otherwise stated we assume a clockwise orientation of the ordering. We sometimes refer to cyc_x when the starting vertex is unimportant.

Distance Class A vertex is in distance class i (denoted D_i) if it is of shortest edge distance (in terms of the number of edges) i from a specified root vertex. We also define D_{max}, the distance class of vertices farthest from the root vertex. This root vertex is normally the exterior facility, and unless otherwise stated this will be assumed throughout, *i.e.* $D_0 = \{e\}$.

Partial Layout A layout in which only a subset of facilities have been placed is said to be a partial layout. Since there could exist a number of partial layouts before the final completion of a layout, we commonly refer to the *current* partial layout.

Tetrahedron, Regular Octahedron, Regular Dodecahedron These are three special MPGs which we refer to extensively. Each of these three graphs is the only MPG which exists with all vertices of degree 3 (Tetrahedron), degree 4 (Regular Octahedron), and degree 5 (Regular Dodecahedron), respectively. Further by examining Euler's Polyhedral Formulae, these are the only three MPGs on $n \geq 4$ vertices with all vertices having the same degree. The Tetrahedron is shown in Figure 2.1, the Regular Octahedron in Figure 2.2, and finally the Regular Dodecahedron in Figure 2.3.

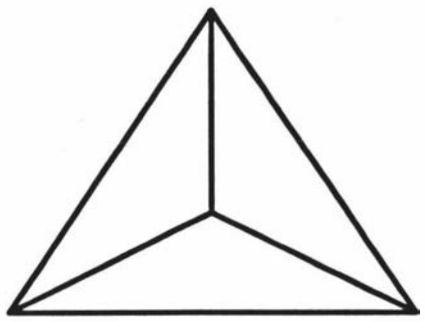


Figure 2.1: The Tetrahedron

Rectanguloid Each facility in the layout is defined by a rectanguloid, which is a simple closed curve whose boundary is made up of straight line segments

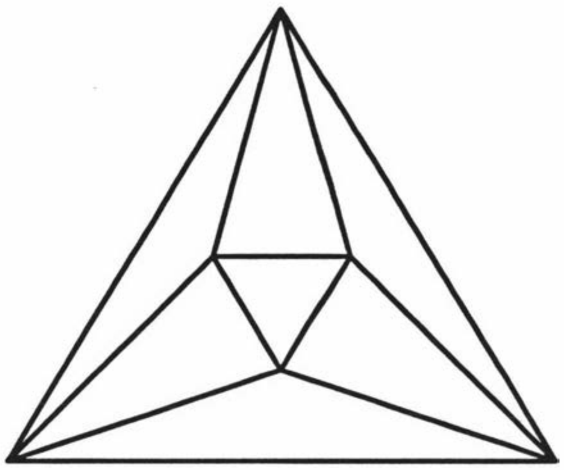


Figure 2.2: The Regular Octahedron

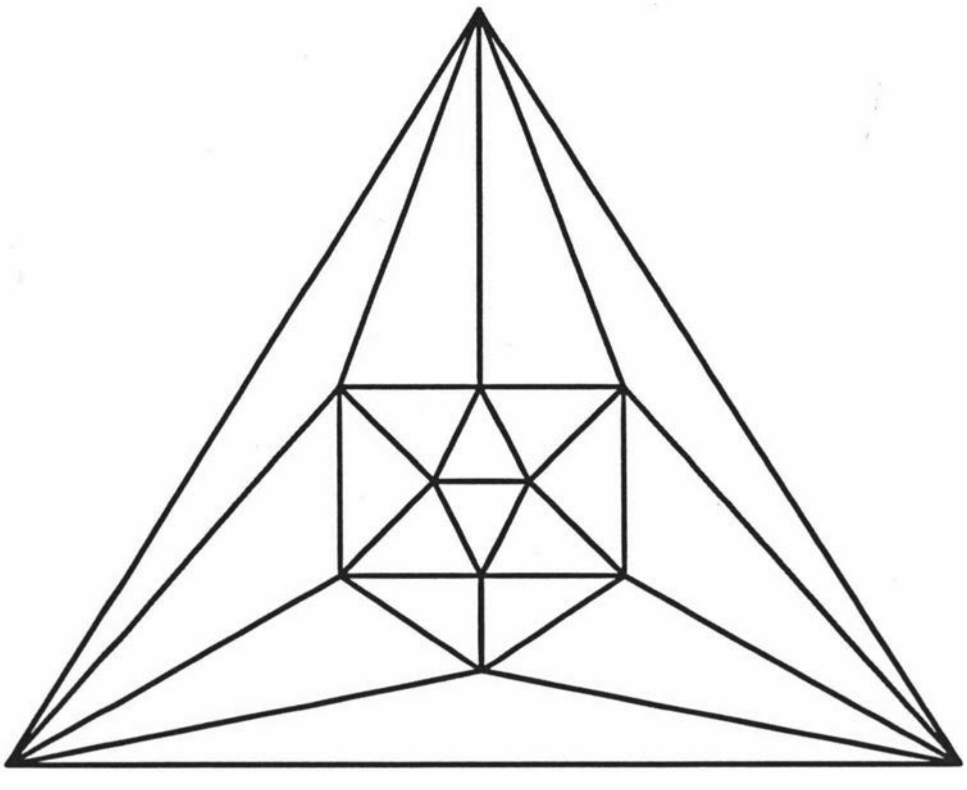


Figure 2.3: The Regular Dodecahedron

each parallel to a wall of the layout perimeter, with every corner being a right angle. It proves useful to define a facility by a list of its corner coordinates. Each rectanguloid can be described by a *turn sequence*, where each corner is either an L(left) or R(right) turn as we proceed around the boundary of the rectanguloid in a clockwise direction. Right turns correspond to interior 90° angles (convex corners), and left turns to interior 270° angles (reflex corners). Note that two turn sequences describe the same rectanguloid, if they are a cyclic permutation of each other.

Facility Shape Classification Facility shapes are determined by their turn sequences. A rectangular facility, labelled as an I-shaped facility, has turn sequence RRRR. Other common facility shapes are the L-shaped facility with turn sequence RRRLRR, T-shaped RRLRRLRR, S-shaped RRRLRRRL, Y-shaped RRRLRRL-RRL, and X-shaped RRLRRLRRLRRL. We refer to a facility as an X facility, for example, if it is X-shaped. Furthermore we define a layout to be an L-layout, for example, if the worst facility shape in the layout is an L facility. Other facility shapes exist, such as the U-shaped facility with turn sequence RRRRRRLL, and these will be described as they are encountered.

Block of Facilities A block of facilities is a rectangular substructure within a layout which wholly contains a subset of the facilities.

Faultline A faultline in a layout is any wall in the layout which has at least two walls perpendicular to it, one on each side of the wall. This is better described by referring to Figure 2.4.

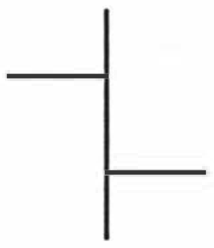


Figure 2.4: An Example of a (Vertical) Faultline

Dimensioned Layout, Undimensioned Layout A layout is said to be dimensioned if all the area specifications of the facilities are met in the layout, proportional to some scale factor. An undimensioned layout does not consider area specifications.

Dimensionalisable Layout A layout is said to be dimensionalisable if there exists no faultline within it. Dimensionaliasable layouts can be dimensioned regardless of the area specifications, and their structure is unaffected by any dimensioning.

Entering Facility The entering facility is the facility which is being placed in the current partial layout.

Placement Host A placement host is a facility within which an entering facility is entirely placed.

Top Facility The top facility at a 3-joint is defined to be the facility which does not have a corner point at that 3-joint. Occasionally we refer to the left and right facilities at a 3-joint with respect to the top facility at that 3-joint. Where there is no ambiguity we simply refer to the top facility. This is better described by referring to Figure 2.5, where we see that facility t is the top facility, l is the left facility and r the right facility.

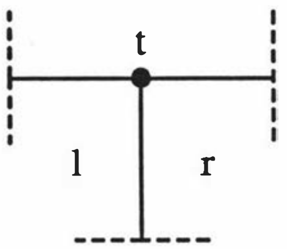


Figure 2.5: A 3-joint showing Top (t), Left (l), and Right (r) Facilities

Maximal Outerplanar Graph An outerplanar graph can be embedded in the plane so that all vertices lie on the exterior face. A maximal outerplanar graph is an outerplanar graph to which no more edges can be added without violating the outerplanarity, and is a triangulation of a polygon.

2.2 Formulations of the Facility Layout Problem

In this section we formally present the GTFLP, and discuss it in relation to its alternative formulation, the Quadratic Assignment Problem (QAP). Foulds [30] and more extensively Kusiak and Heragu [68] survey the work done on the FLP, on both the classical and graph-theoretic models, giving alternative formulations. The first formulation of the FLP was due to Koopmanns and Beckman [63], as a QAP. Each facility is considered as being composed of a number of unit subfacility modules. The building perimeter is divided into an orthogonal grid, with each cell of the grid having area equal to the area of a subfacility module. The QAP has the objective of minimising the relative location of the subfacilities so as to minimise total transportation cost throughout the layout, while maintaining the contiguity of each facility. We describe a variation on the original formulation in Formulation 1, as given by Hillier and Conners [58].

Formulation 1 *QAP*

N	$= \{1, 2, \ldots, n\}$, the set of subfacilities
M	$= \{1, 2, \ldots, m\}$, the set of grid locations
c_{ij}	The cost per unit time period of assigning subfacility i to location j.
d_{ij}	The distance from location i to location j, where the distance is an appropriate measure of the travel cost between i and j.
f_{ij}	The material flow from subfacility i to subfacility j.

$$a_{ijkr} = \begin{cases} f_{ij}d_{jr} & i \neq k \; or \; j \neq r \\ f_{ii}d_{jj} + c_{ij} & i = k \; or \; j = r \end{cases}$$

$$x_{ij} = \begin{cases} 1 & \text{if subfacility } i \text{ is assigned to location } j \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{array}{lllll} Min & \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{n}\sum_{r=1}^{m} a_{ijkr}x_{ij}x_{kr} & & & \\ s.t. & \sum_{i=1}^{n} x_{ij} & = & 1 & j \in M \\ & \sum_{j=1}^{m} x_{ij} & = & 1 & i \in N \\ & x_{ij} & = & 0 \; or \; 1 & i \in N, j \in M \end{array}$$

If $n < m$, we can introduce $m - n$ dummy subfacilities, with zero c_{ij} and f_{ij} values. The division of each facility into an integral number of modules circumvents the assumption that each facility is represented as a single object. The cost of this however, can result in a significant increase in the computational burden through the expansion of the problem size. The QAP formulation has been shown to be NP-complete [77, 99], so the optimal solution can be found for problems with fewer than 20 subfacilities in a *reasonable* time. In Section 2.4, we briefly outline heuristic methods which have been developed in order to solve the QAP approximately. The QAP and the work related to it is also known as Classical Layout.

We now consider an alternative formulation of the FLP within a Graph Theoretic framework. In this instance each facility is represented as a vertex in the graph. We assume that a relationship matrix is given, which identifies the desirability of placing pairs of facilities adjacently. This adjacency between two facilities is represented by an edge between the two corresponding vertices. The graph representing the adjacencies is planar if and only if the layout is planar [101], and in fact, provided no facility is nested entirely within another and all wall junctions are 3-joints, the graph must be maximally planar. The GTFLP formulation is given in Formulation 2, due to Foulds and Robinson [33]. Giffin [39] proves that we can assume the w_{ij} values are non-negative, and that the minimisation form of the objective, when using cost/flow data is equivalent to the objective of Formulation 2.

Formulation 2 *GTFLP*

w_{ij}	The closeness rating for placing facilities i and j adjacently.
N	The set of pairs of adjacencies which must be adjacent in any feasible solution
F	The set of pairs of facilities which must not be adjacent in any feasible solution.
E'	$= \{(i,j) : x_{ij} = 1, (i,j) \in E\}$
x_{ij}	$= \begin{cases} 1 & \text{if facility } i \text{ is adjacent to facility } j \\ 0 & \text{otherwise} \end{cases}$

$$\begin{array}{lrcll} Max & \multicolumn{4}{l}{\sum_{i \in E} \sum_{j \in E} w_{ij} x_{ij}} \\ s.t. & x_{ij} & = & 1 & (i,j) \in N \\ & x_{ij} & = & 0 & (i,j) \in F \\ & \multicolumn{4}{l}{(V, E' \cup N) \textit{ is a planar graph}} \end{array}$$

2.3 Justification of the Graph Theoretic Approach

Comparing the QAP and GTFLP formulations, we see that the GTFLP produces an adjacency prescription only, implying the necessity of a second phase to develop the actual layout. The location assignment costs c_{ij} of the QAP are assumed to be zero by the GTFLP, implying that the GTFLP is more applicable to a new layout design, rather than modification of an existing one, where the QAP, could be applied in either instance. The GTFLP does not consider non-adjacent facilities, so the overall transportation cost of the layout is not explicitly considered; however, Giffin and Foulds [31, 40] present formulations to remedy this, while not compromising the GTFLP formulation. Therefore concerns regarding the correspondences among transportation cost, material flow, and desired adjacency are alleviated.

A major advantage of the GTFLP over the QAP is the increased freedom allowed in the design of the layout. Rather than a rigid grid, the planner may begin with a blank area within which irregularly shaped facilities of differing areas may be manipulated. The GTFLP also allows provision for alternative objective functions as shown by Giffin and Foulds, with few algorithmic modifications required in moving from one objective to the other for the first phase, and indeed the second phase proceeds independently of the objective function. We will see in later chapters, however, that we can impose a second objective on the second phase in order to increase the likelihood of efficient and practical layout designs.

Hassan and Hogg [51, 53] provide an in-depth review of the application of Graph Theory to the FLP, including a synopsis of what the authors consider are drawbacks of the approach. They state the following advantages of the graph theoretic approach: the establishment of the upper bound of the sum of the $3n - 6$ largest edge weights; purported improvement in objective function value over conventional methods; and ability to specify exterior facilities. Hassan and Hogg attempt to discredit these last two advantages. They state that the purported improvement in the objective function shown by Hammouche and Webster [48] is of minimal value, as they do not compare the correct objectives. Reexamination of [48] however renders this statement incorrect as Hammouche and Webster did in fact make the correct comparisons. Hassan and Hogg state that the exterior facility could be added to conventional methods by delaying the addition of some facilities, which by default then be come adjacent to the layout exterior. However the inclusion of the exterior is not adequately represented within classical approaches by simply delaying the placement of certain facilities, as this assumes a 0/1 type of relationship between facilities and the exterior, while in reality there may exist relevant transportation data between facilities and the exterior which must be accommodated within the formulation. Hassan and Hogg also present the following reasons why the Graph Theoretic approach is not a adequate model: the umbrella effect; ignoring the minimum length of common borders between adjacent facilities; and having to be careful about the placement of the exterior so that it does not become interior. In the following chapters it will become apparent that these conclusions are weak. We will see that the only significant disadvantage is the ignoring of the common wall length between adjacent facilities, as the umbrella effect can be overcome, and the incorrect placement of the exterior inside the layout, comes from a lack of appreciation of the consequences of the arbitrary embedding of any MPG, of which one can always be obtained with the exterior on the outside.

Furthermore, Bozer and Meller [16] present a paper in which they show that the QAP formulation for the FLP is not necessarily realistic, and unnecessarily constrains the flexibility of the resulting layout. They show that the usual centroid-to-centroid distance measure is usually somewhat conservative, and present a new measure which includes a correction factor to more accurately determine the flow cost between facilities.

2.4 Related Work in Classical Layout

The heuristics that resulted from an attempt to approximate the optimal solution to the QAP are the initial layout procedures, on which much of the Graph Theoretic second phase work is based. The first attempt at computerized layout was the improvement procedure CRAFT [6]. Starting from a given feasible layout CRAFT considers departmental interchanges that are of equal area, or share a common border, in an effort to reduce the transportation cost of the layout. Malmborg [83] extends the CRAFT procedure to incorporate further parameters such as material handling system fleet size, in order to broaden the range of factors considered in swapping facilities. CORELAP [73] and ALDEP [100], are the most commonly quoted construction methods. CORELAP assigns the facility with the highest column sum in the relationship matrix, and places it in the centre of the layout. Subsequent facilities are selected on the basis of having the maximum adjacency with a facility in the partial layout. The entering facility is placed by considering all possible positions in which the facility could be placed, and selecting the one which maximises the closeness rating between the entering facility and its neighbours. ALDEP places facilities columnwise from left to right, going first down and then up, as if ploughing a field. A so-called sweep width is defined, which specifies how wide the columns are. The first facility is chosen randomly, and is placed in the north-west corner. Subsequent facilities are chosen on the basis of having the maximum relationship with the facility placed previously. The facility is placed from where the previous facility finished. Other construction methods such as PLANET [24], and improvement procedures such as COFAD [104], for example, have also been implemented, using much the same mechanisms as CORELAP, ALDEP and CRAFT. Whitehead and Eldars [107] presented a method essentially the same as CORELAP. This method was used in a case study by Agraa and Whitehead [2] to lay out a school building. This work was then extended by these same authors in [1] to incorporate nuisance restrictions, by automatically rejecting positions for the entering facility that do not satisfy the nuisance restrictions. Lewis and Block [79] conducted an experiment between some of these classical approaches, and some experienced layout planners. The results showed that as the problem size increased, computer implementation became more efficient than human experience.

Hassan, Hogg and Smith [54] presented SHAPE, which is essentially another extension of CORELAP, but with a rather more intelligent and intuitive foundation. Rather than examining every position where the entering facility could be placed as in CORELAP, a distance function was developed to rank potential candidate positions, by considering placing the entering facility on each side of the layout, and evaluating the movement cost for each candidate position with the facilities in the current partial layout. The method is compared to PLANET, and the results obtained showed SHAPE compared favourably. Hassan and Hogg [53] presented a paper which attempted to combine the classical approaches with graph-theoretic methods, but essentially produced a method very similar to CORELAP. We defer discussion of this approach until Section 4.6.4.

Heragu and Kusiak [56] present two mixed integer formulations for the FLP, which are not based on a linearisation of the QAP. Unlike the QAP formulation, this formulation does not require that the location of the sites need to be known *a* *priori.* A knowledge-based approach is introduced in order to heuristically determine layouts under these formulations. No experience is given as to how efficient the resulting layouts are. In a subsequent paper [57] Heragu and Kusiak, transform these formulations into linear continuous formulations with absolute values in the constraints and objective function, and also into nonlinear formulations which are solved heuristically using the penalty method of non-linear optimization, and Powell's method for unconstrained minimization. Heragu and Alfa [55] present a heuristic using the generic Simulated Annealing heuristic to solve the QAP, using two-opt and three-opt facility exchanges. The computational experience given shows that this is an effective approach, yielding highly weighted layout solutions in reasonable computational time. Jajodia, Minis, Harhalakis and Proth [61] introduced CLASS, another extension of CRAFT. The model used allows facilities to have unequal areas, by considering a facility as being divided into submodules, having large flow between them. CLASS uses Simulated Annealing strategies to improve a randomly chosen initial solution. Bozer, Meller and Erlebacher [17] introduce the the use of spacefilling curves in facility layout. By using space filling curves, any two facilities can be exchanged, making it a more powerful tool than CRAFT. Raoot and Rakshit [93] give a comparison of various alternative procedures for solving the QAP. This comparison examines the generic placement methodologies used, and concludes that methods such as CORELAP perform better than routines with more myopic placement mechanisms.

As we have discussed already Hammouche and Webster [48] showed that these classical approaches do not perform as well as the Graph Theoretic techniques. Some of the classical ideas can be utilised within the Graph Theoretic model, and these will be developed as they occur. On their own however the classical approaches are superseded by the Graph Theoretic methods.

2.5 Regularity

An integral part of the FLP that has been largely unexplored, is the measuring of how regular, usable or efficient a given layout is. A number of authors, for example Hassan and Hogg [52], have recognised the importance of regularity considerations, however have provided no details as to how this might be accommodated. The only insights into this problem were due to Raoot and Rakshit [92], and Bozer, Meller and Erlebacher [17] who each provided an evaluator of how regularity could be measured. We will describe these two measures along with some new possible evaluators shortly. Previously it has been left to the layout planner to make alterations by hand, with the only criterion being that it *looks good.* In this chapter we discuss ways of measuring regularity in a layout, by considering a variety of so called *regularity measures* and objectives.

Given a scenario of possible feasible layouts, we may subjectively choose the one which we think is best, however our choice may not be the same as someone else's, indeed our criteria for choice may be significantly different. In an effort to standardise the basis for comparison we will define regularity measures which attempt to objectively quantify how regular a layout is. Consider the following definitions of a facility i:

R_i	The area of the largest rectangle contained wholly within i
G_i	The area of the largest golden rectangle contained wholly within i
S_i	The area of the largest square contained wholly within i
$\bar{R}_i$	The area of the smallest rectangle that wholly contains i
$\bar{G}_i$	The area of the smallest golden rectangle that wholly contains i
$\bar{S}_i$	The area of the smallest square that wholly contains i
C_i	The number of corners of a facility i
P_i	The length of the perimeter of facility i

The concept of the Golden Rectangle requires more explanation. A Golden Rectangle has a width-to-length ratio of $1 : (1+\sqrt{5})/2$, called the Golden Ratio, *i.e.* the same width-to-length ratio as this page. This ratio is found from solving the quadratic equation $x^2 + x - 1 = 0$. It is widely accepted that the Golden Ratio was a term of proportional beauty administered in classical architecture. The Golden Ratio was included in the building instructions for Moses' Tabernacle in the Midian desert, and in the planning of the Great Pyramid of Egypt [18, 19]. It is useful to consider the Golden Rectangle within this work on regularity, as we will see it provides a useful intermediate polygon between the restrictive square, and the overly flexible rectangle with arbitrary width-to-length ratio.

We can define a number of measures which could provide a quantitative value for each facility, for example:

R_i/a_i	The ratio of the area of the largest rectangle contained in i to the area of i	(2.1)

G_i/a_i	The ratio of the area of the largest golden rectangle contained in i to the area of i	(2.2)
S_i/a_i	The ratio of the area of the largest rectangle contained in i to the area of i	(2.3)
$a_i/\bar{R}_i$	The ratio of the area of i to the area of the smallest rectangle that will enclose i (Raoot and Rakshit [92])	(2.4)
$a_i/\bar{G}_i$	The ratio of the area of i to the area of the smallest golden rectangle that will enclose i	(2.5)
$a_i/\bar{S}_i$	The ratio of the area of i to the area of the smallest square that will enclose i	(2.6)
$R_i/\bar{R}_i$	The ratio of the area of the greatest rectangle in i to the area of the smallest rectangle that will enclose i	(2.7)
$G_i/\bar{G}_i$	The ratio of the area of the greatest golden rectangle in i to the area of the smallest golden rectangle that will enclose i	(2.8)
$S_i/\bar{S}_i$	The ratio of the area of the greatest square in i to the area of the smallest square that will enclose i	(2.9)
C_i	The number of corners facility i has	(2.10)
$4\sqrt{a_i}/P_i$	The ratio of the optimal perimeter of a square facility i to the perimeter of i (Bozer, Meller and Erlebacher [17])	(2.11)

Measures 2.1, 2.2 and 2.3 attempt to measure the useable space available within a facility, by considering the ratio of the area of the largest polygon specified to the area of the facility. These are known as the enclosed polygon ratios. Measures 2.4, 2.5 and 2.6 attempt to measure the spread of a facility by considering the ratio of the area of the smallest polygon specified that will enclose the facility to the area of the facility. These are known as the bounding polygon ratios. Measures 2.7, 2.8 and 2.9 are measures which multiply the corresponding pairs from measures 2.1 - 2.3 and 2.4 - 2.6. This set of measures is the ratio of the useable space to the spread of the facility. Measure 2.10 is a simple yet effective measure which measures how many corners a facility has. Measure 2.11 assumes that the optimal shape for a facility is a square, which is where the perimeter is minimised *i.e.* $P_i^* = 4\sqrt{a_i}$.

We do not in general use Measures 2.7 - 2.9, as consider if the bounding rectangle and enclosed rectangle ratios were both 0.1, this would give a value of 1 to Measure 2.7. Measures 2.4 and 2.10 are known as shape based measures, while the

Shape	Score 1	Score 2
Rectangle	1	4
L-shape	0.5	1
S-shape	0.25	0
T-shape	0.25	0
Other	0	-1

Table 2.2: Possible nominal scales for facility regularity

other measures are known as usable area based measures.

Of course we can also include more qualitative measures such as a nominal scale for room shapes, as shown by Table 2.2. Unfortunately scales of this type are very subjective, and finding the right values to assign to each shape is hard. This problem is similar to that of what values to assign the AEIOUX scores in relationship charts.

Objectives could either be to maximise total regularity, also the same as maximising the average facility regularity, or to maximize the minimum facility regularity. We will see in the course of this thesis that it is prudent to consider both objectives in considering layout alternatives, as both omit important information which the other can better encapsulate.

Coda

In this chapter we have outlined the basis of the GTFLP process. We have compared the GTFLP with the QAP, and have concluded that the GTFLP is certainly a valid representation of the FLP, and hence worthwhile exploring. We will see as we progress the solidifying of this conclusion. We have briefly outlined the work that was previously completed in Classical Layout. Furthermore we have defined and discussed the concept of regularity.

In the next chapter, we examine the generation of a highly weighted MPG, in order to complete the first phase of the GTFLP. This comprises a literature review of the relevant heuristics, together with some theorems related to the performance of these heuristics. Chapter 4 provides the main foundational thrust for the remainder of this thesis, describing fully the second phase of the GTFLP: the design of the layout.

Chapter 3

Obtaining the Adjacency Graph

In this chapter, we examine the first phase of the GTFLP: the construction of a highly weighted MPG. This problem has been shown by Garey and Johnson [37], and Liu and Geldmacher [80] to be NP-complete, therefore it is unlikely that there will ever exist computationally efficient algorithms for generating the optimally weighted MPG for realistically sized problems. For this reason, we resort to heuristics, which attempt to build up the MPG so that its final weight is as large as can be found in a *reasonable* time. We shall see however that obtaining the best MPG, seldom corresponds to obtaining the most practical layout. This does not nullify the importance of obtaining highly weighted MPGs, as in most cases it provides a useful starting point from which important information can still be gleaned. We will address a more beneficial role of the best MPG in later chapters. The construction methods for obtaining a highly weighted MPG can be divided into two groups: those which require a planarity testing routine in order to maintain planarity of the adjacency graph are discussed in Section 3.1, and those, more widely used and discussed, which avoid planarity testing by utilising the underlying structure of the MPG, are examined in Section 3.2. Finally in this chapter we explore improvement routines which have been developed in order to attempt to improve upon the value of an initial MPG construction. Throughout this chapter we assume that in finding an MPG, we are finding one of as large weight as possible, unless otherwise stated.

3.1 Heuristic Methods that Require Planarity Testing

Foulds, Gibbons and Giffin [31] describe a Greedy Heuristic for obtaining an MPG, which we simply refer to as Greedy. Their heuristic orders the set of possible edges in non-increasing order, and then adds the edge at the top of the list to the current subgraph, as long as planarity is maintained until $3n - 6$ edges have been added. Foulds *et al.* use a series of filters described by Foulds and Robinson [33] in order to reduce the computational burden of the planarity testing algorithm of Hopcroft and Tarjan [59], which is used to ensure that the edge being added does not violate the planarity of the current subgraph. Unfortunately, even though the planarity testing algorithm of Hopcroft and Tarjan has a computational complexity of $O(n)$, it is extremely difficult to implement. We provide the computational complexity and worst case analysis of Greedy in Theorems 3.1.1 and 3.1.2. Firstly however, we provide a background description on the worst case analysis of heuristics due to Dyer, Foulds and Frieze [26]. Let q denote a problem instance, and let H be a heuristic applied to the problem q. Let $E_{H,q}$ denote the set of edges chosen by H and let $E_{*,q}$ denote the set of edges in the optimal solution. Let $w(S)$ be the total weight of the edges in the set S. Then the ratio $R_{H,q}$ given in Equation 3.1 is a measure of the quality of the solution produced by H.

$$R_{H,q} = \frac{w(E_{H,q})}{w(E_{*,q})} \tag{3.1}$$

The worst case ratio ρ_H is defined by Equation 3.2. This value gives us a guaranteed value for the quality of our solution in comparison with the optimum.

$$\rho_H = \inf_q(R_{H,q}) \tag{3.2}$$

Theorem 3.1.1 *The computational complexity of Greedy is $O(n^3)$*

Proof: See [31] □

Theorem 3.1.2 $\rho_{Greedy} = \frac{1}{3}$

Proof: See [26] □

A second approach is due to Carrie, Moore, Roczniak, and Seppanen [21]. This approach starts by constructing a Maximal Spanning Tree, and then adding edges to the subgraph, in decreasing order of their weight, as long as planarity is not violated, until the graph is maximally planar on the n vertices. Carrie *et al.* also devise a set of rules which are intended to place the edges as intelligently as possible, but the planarity testing phase is still required.

Hashimshony, Shaviv and Wachman [49] present a method of obtaining an MPG, by considering the complete graph on n facilities and deleting edges until planarity is obtained. Two methods are used, cancelling one, possibly highly weighted edge, and deleting a number of low weighted edges. The method uses the planarity testing algorithm of Lempel, Even and Cederbaum [76].

Giffin [39] presented the Greedy-Hamiltonian heuristic, which reduced the burden of the planarity testing phase. Initialising with a greedy Hamiltonian circuit, the highest weighted edge not violating planarity was added. The test for planarity simply coloured the vertices of a so-called *auxiliary graph* to assign the edges not on the Hamiltonian circuit to either the *inside* or *outside* of the current planar graph. A 2-colouring was used, and if every vertex in the auxiliary graph could be coloured such that no other vertex of the same colour was adjacent to it, the new edge could be accepted.

3.2 Heuristic Methods Avoiding Planarity Testing

This section describes the most widely used construction heuristics for generating an MPG. These methods are motivated by a desire to remove the difficulty of the planarity testing of the methods of Section 3.1. The first heuristic is Deltahedron which, due to its simple placement operation, has been widely explored, and variants of the objective function and alternative placement operations have been discussed at length by various authors. Section 3.2.2 discusses some fundamental extensions of Deltahedron. Finally we examine TESSA, which is a comparatively new heuristic, but has the advantage of being able to generate every MPG on n vertices.

3.2.1 Deltahedron

Foulds and Robinson [34] were the first to present a method for obtaining an MPG which required no planarity testing. Their method initially chooses four mutually adjacent vertices to create the initial MPG, K_4. Subsequent vertices are chosen, and placed in the MPG, by inserting them one at a time within a face of the current subgraph, so that the *entering vertex* (the vertex being inserted in the subgraph) is adjacent to each of the vertices of the face the entering vertex is inserted in. This basic vertex placement operation, of vertex x being inserted in face (p, q, r) is shown in Figure 3.1, with corresponding increase in benefit $B(Delta; p, q, r, x)$ shown in Equation 3.3. Foulds and Robinson called this method the Deltahedron Heuristic, but other authors have referred to it as Face Triangulation. The basic steps of the Deltahedron Heuristic are described in Algorithm 3.1; where Constructions A and B therein, are described below.

$$B(Delta; p, q, r, x) = w_{px} + w_{qx} + w_{rx} \tag{3.3}$$

Algorithm 3.1 *Deltahedron*

```
    Input: Adjacency Matrix
    Output: Deltahedron MPG G
    Choose 4 vertices $(a, b, c, d)$, via Construction A
    $V_G \leftarrow \{a, b, c, d\}$
    $E_G \leftarrow \{(a,b),(a,c),(a,d),(b,c),(b,d),(c,d)\}$
    $T_G \leftarrow \{(a,b,c),(a,b,d),(a,c,d),(b,c,d)\}$
    while $(|V_G| < n)$ do
        Choose vertex $x$ to be placed in face $(p, q, r)$ via Construction B
        $V_G \leftarrow V_G \cup \{x\}$
        $E_G \leftarrow E_G \cup \{(p,x),(q,x),(r,x)\}$
        $T_G \leftarrow T_G \cup \{(p,q,x),(p,r,x),(q,r,x)\} \backslash (p,q,r)$
    end
end
```

A number of possible versions for Constructions A and B have been suggested in the literature. Foulds and Robinson outline two in the original paper, the S-

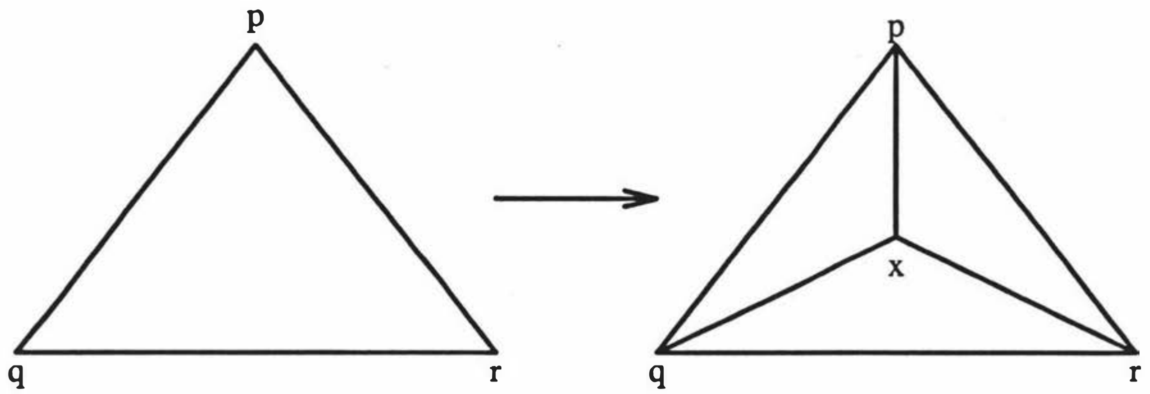


Figure 3.1: Deltahedron Insertion of a Vertex x into Face (p, q, r)

and R-constructions. The S-construction calculates $S(i)$, as shown in Equation 3.4, and orders the vertices in non-increasing order of their $S(i)$ values.

$$S(i) = \sum_{j=1}^{n} w_{ij} \tag{3.4}$$

Under this mechanism Construction A chooses the four vertices of highest $S(i)$ value (called SELECT by Foulds *et al.* [31]) while Construction B, chooses the vertex i which has the largest $S(i)$ value of all unplaced vertices, and inserts i in the face which will increase the overall benefit maximally. The R-construction uses the Tetrahedron in which the least benefit edge is maximised, for Construction A, and then for Construction B, the increase in benefit of placing each unplaced vertex in each face of the current subgraph is calculated. The vertex to be inserted in the MPG is one which has the largest difference between its first and second choice of possible faces; it is inserted in its first choice face, giving the highest increase in benefit. Foulds *et al.* give another A Construction, called AVARICE, which chooses the edge (a, b) of highest weight, then chooses a third vertex c such that the benefit of triangle (a, b, c) is maximised, and then chooses a fourth vertex d such that the weight of the Tetrahedron (a, b, c, d) is maximised. Green and Al-Hakim [47] define Construction A to choose the triangle K_3 of maximum benefit, and then add to this triangle, the vertex which gives the largest increase in benefit. They also define a greedy B Construction, where the benefit derived from placing each unplaced vertex in each face is evaluated, and the vertex/face pair giving the best increase in total benefit is chosen. Giffin [39] tested the most obvious construction, that of evaluating all possibilities for the tetrahedron, and choosing that of highest weight, for Construction A, and then using the Construction B of Green and Al-Hakim. Boswell [13] provides a computational experiment of the various A and B Constructions. The results show overwhelmingly that the greedy vertex insertion of Green and Al-Hakim was the best B construction, providing the best solutions in 80% of the problems solved. The experiment on the A Construction was less conclusive, with the greedy tetrahedron construction of Giffin having the edge over both the SELECT and AVARICE constructions, all of which were clearly superior to the Green and Al-Hakim A Construction. Boswell noted that as the problem size increased, the choice of A Construction became less important, as the tetrahedron contributed a smaller proportion to the final benefit of the MPG. Boswell recommended that the greedy constructions of Giffin, and Green and Al-Hakim should be used for the A and B Constructions respectively, and further noted that the computational times were minimal, being less than two seconds on forty facility problems. Theorems 3.2.1 - 3.2.3 due to Giffin, Foulds and Robinson, and Green and Al-Hakim provide the computational complexity of the Deltahedron heuristic for the various A and B constructions. Furthermore Dyer *et al.* provide a series of theorems on the worst case performance of some of the different variations on the Deltahedron Constructions. These are provided in Theorems 3.2.4 and 3.2.5, and Corollary 3.2.1; the reader is referred to Dyer *et al.* for proofs. We will see in Section 3.3 that although theoretically important, these results are in general overcome by powerful improvement techniques.

Theorem 3.2.1 *The constructions of Foulds and Robinson have computational complexity of* $O(n^2)$

Proof: See [31, 39] □

Theorem 3.2.2 *The construction of Green and Al-Hakim has computational complexity of* $O(n^3)$

Proof: See [47] □

Theorem 3.2.3 *The greedy construction of Giffin has computational complexity of* $O(n^4)$

Proof: See [39] □

Theorem 3.2.4 $\rho_{Delta} = 0$, *using the S construction*

Corollary 3.2.1 $\frac{1}{6} \leq \rho_{Delta} \leq \frac{2}{9}$, *using the S construction, and where all edge weights are either 0 or 1*

Theorem 3.2.5 $\rho_{Delta} = \frac{1}{6}$, *using the S construction, where* $S(i) = \max_j w_{ij}$

Giffin and Foulds [40] present adaptations of the Deltahedron, and the Greedy-Hamiltonian heuristics to incorporate near adjacency. Near adjacency is defined in terms of the number of boundary crossings required in the layout to travel between i and j say, and hence the minimum path, in terms of the number of edges, between i and j in the MPG. The two methods use a tailored version of Dantzig's shortest path algorithm [23], in order to extend the objective function to include benefits for near adjacency. The tailoring of Dantzig's algorithm reduces the computational complexity from $O(n^5)$ to $O(n^3)$ for the Deltahedron heuristic, however the Greedy-Hamiltonian method only reduces from $O(n^6)$ to $O(n^5)$.

Foulds and Giffin [32] present another variant on the Deltahedron heuristic, which attempts to minimize the total transportation cost between facilities. The S-construction, and AVARICE, are used, with a rectilinear distance norm in the objective function to approximate the transportation distances, by assuming that all facilities are square in shape. The S-construction is shown to have $O(n^3)$ computational complexity, while the AVARICE construction has $O(n^4)$ complexity. We will return to these two modifications by Giffin and Foulds in Chapter 10.

This completes the basic discussion of the Deltahedron heuristic. In the next section, we examine some extensions of Deltahedron which attempt to allow the insertion of more general structures into the MPG subgraph, whilst still maintaining planarity.

3.2.2 Deltahedron Extensions

The extensions of the Deltahedron that we will discuss can be divided into two groups. The first is the generalisation of placement of a vertex into more than one face, which we will see through the Wheel Generation Heuristic, and more generally through the Wheel Expansion Heuristic. The second type of generalisation is the generation of more complex substructures which can be placed within a face of the current subgraph.

The Wheel Generation and Wheel Expansion Heuristics

The motivation behind wanting to somehow generalise the Deltahedron Heuristic lies in the inability of Deltahedron to be able to generate any MPG on n facilities. This was noticed by Foulds and Robinson [34], who cited the Regular Octahedron as an MPG which has no vertex of degree 3, and hence cannot be created by Deltahedron. The Wheel Generation Heuristic was motivated by a theorem of Bowen and Fisk [15], which simply stated that the three operations of Figure 3.2, are sufficient to generate all MPGs from K_4.

Giffin stated that the three operations of Figure 3.2 need only be applied locally to the current subgraph, and hence was able to develop the greedy Wheel Generation Heuristic. The three operations yield the increases in benefit given in Equations 3.5 - 3.7, where each operation involves placing a vertex x in the appropriate triangle (p, q, r), quadrilateral (p, q, r, s), or fan (p, q, r, s, t), respectively.

$$\begin{aligned} B(WG: p,q,r,x) &= w_{px} + w_{qx} + w_{rx} && (3.5)\\ B(WG: p,q,r,s,x) &= w_{px} + w_{qx} + w_{rx} + w_{sx} - w_{pr} && (3.6)\\ B(WG: p,q,r,s,t,x) &= w_{px} + w_{qx} + w_{rx} + w_{sx} + w_{tx} - w_{ps} - w_{qs} && (3.7) \end{aligned}$$

The modification to the original Deltahedron heuristic then, was to initialise using any suitable A Construction, and then for each vertex/triangle, vertex/quadrilateral, and vertex/fan pairing, to calculate the increase in benefit of that pairing, and to perform the one giving the maximal increase, until all vertices have been inserted into the subgraph.

Eades, Foulds and Giffin [27] extend the Wheel Generation Heuristic to the Wheel Expansion Heuristic, which is a more general form, of which the Wheel Generation Heuristic becomes a special case. The method uses the fact that every vertex x in an MPG is the hub of a wheel W_x with all of its adjacent vertices forming the rim, which Eades *et al.* prove for each vertex is unique. The wheel expansion approach was first outlined by Baybars and Eastman [8], who proved some theoretical results about the underlying wheels in MPGs.

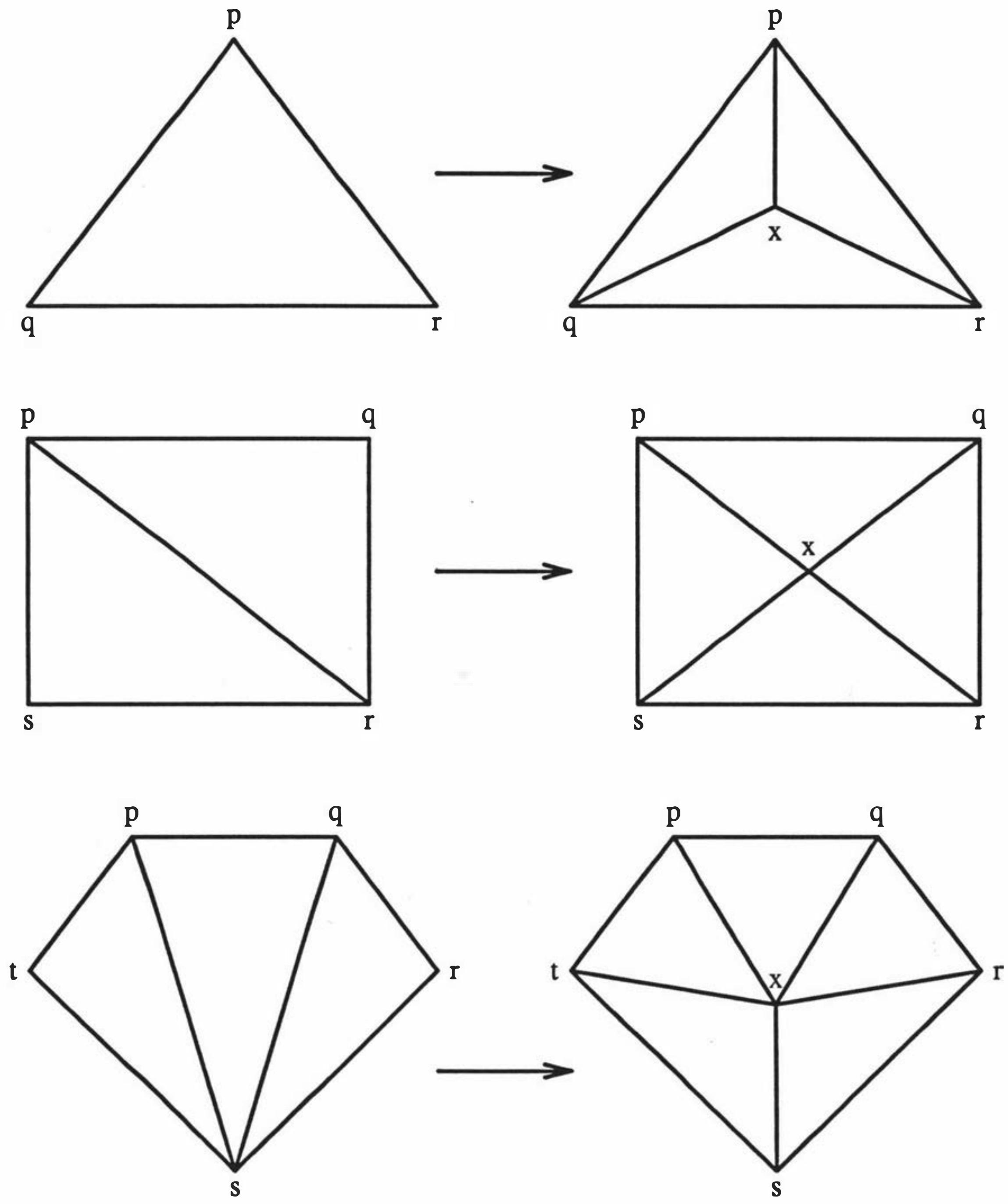


Figure 3.2: Bowen and Fisk Sufficiency Operations for Generating all MPGs from K_4

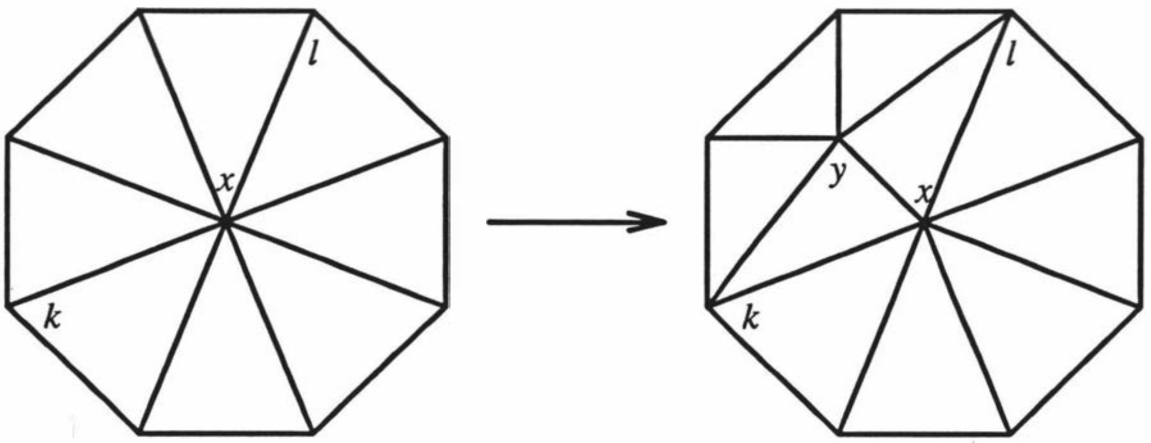


Figure 3.3: Wheel Expansion Operation placing vertex y in W_x

The Wheel Expansion operation considers a vertex x in the current partial solution, together with a vertex y not in the current partial solution. Two vertices k and l which are on the rim of W_x are chosen, and y is placed such that k and l are both on the rim of W_y and the new W_x. This operation is shown in Figure 3.3, with corresponding benefit in Equation 3.8, where the set P is the set of vertices on W_x forming a chain having k and l as its endpoints. Note that there are two such chains, and we distinguish between them by defining that the chain must move clockwise from k to l. We can easily see that Wheel Generation is a special case of Wheel Expansion, where we only consider k and l on W_x whose chain clockwise from k to l on W_x is of length three or less.

$$B(WE : x, k, l, y) = w_{xy} + w_{ky} + w_{ly} + \sum_{j \in P}(w_{jy} - w_{ky}) \tag{3.8}$$

Theorem 3.2.6 *Wheel Expansion and Wheel Generation have computational complexity of $O(n^4)$*

Proof: See [27, 39] □

The Extended Deltahedron Heuristic

Leung [78] provides a different type of generalisation of Deltahedron, in which as well as the single insertion of a vertex in a face, insertion of a triple of vertices into a face is also permissible, providing that the subgraph of the three entering vertices,

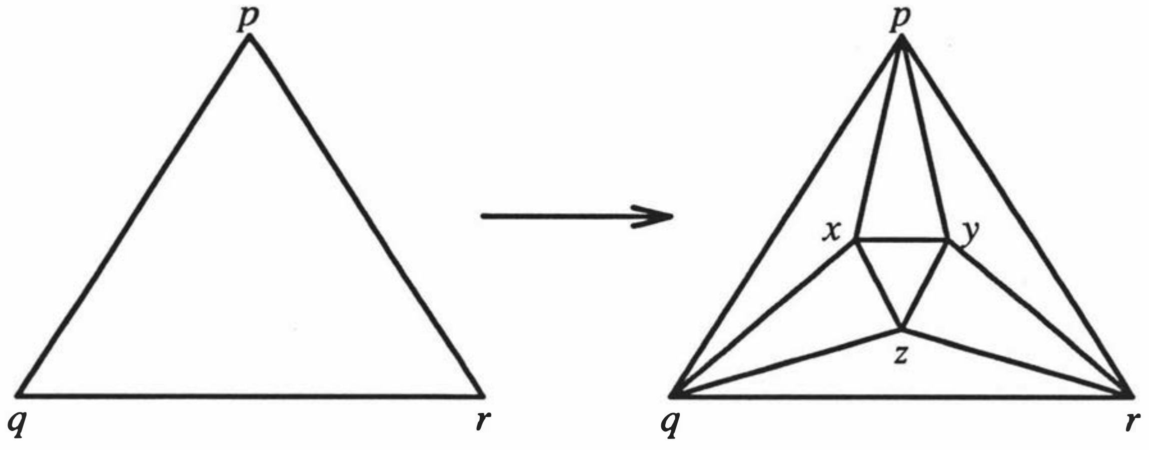


Figure 3.4: The Additional Operation for Extended Deltahedron

and the three vertices which make up the face the triple is placed in, form a regular octahedron. This operation is shown in Figure 3.4, where we insert vertices x, y and z in face (p, q, r), and the benefit of this operation is given by Equation 3.9. The method uses the greedy construction which chooses the operation which gives the best *average* increase, by dividing each $B(ExDelta : p, q, r, x, y, z)$ by three in order to make a valid comparison, between it and the standard Deltahedron operation. Leung initialises with a Tetrahedron, but this prohibits a six facility problem from generating the Regular Octahedron. Initialisation with the triangle K_3 overcomes this caveat, and also allows MPGs created entirely from the second operation to be formed. However, even with this slight generalisation, the Extended Deltahedron heuristic, cannot generate all MPGs, as there is no way of attaining, for example, the Regular Dodecahedron.

$$B(ExDelta : p, q, r, x, y, z) = w_{px}+w_{qx}+w_{py}+w_{ry}+w_{qz}+w_{rz}+w_{xy}+w_{xz}+w_{yz} \quad (3.9)$$

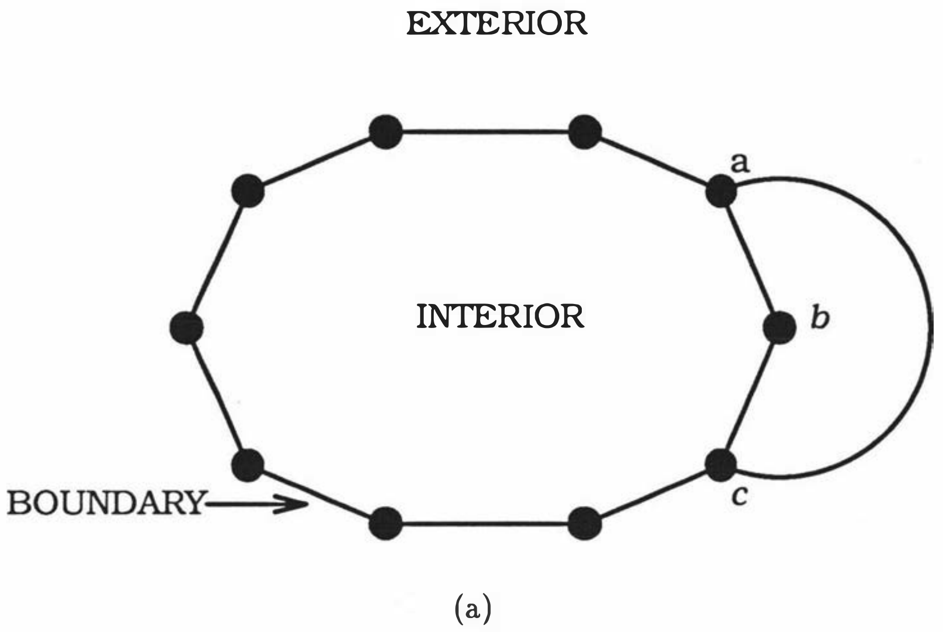
Theorem 3.2.7 *The Extended Deltahedron Heuristic has computational complexity of $O(n^4 \log n)$.*

Proof: See [78] □

3.2.3 TESSA

The TESSA heuristic was devised by Boswell [14] as an alternative method for obtaining a highly-weighted MPG. The benefits of TESSA are not only that it requires no planarity testing, but is capable of generating any MPG on n vertices, using face augmentation. Boswell used the fact that an MPG with n vertices can be defined by a list of its $2n-4$ faces. If we define the benefit of a face to be the sum of the benefits of its three edges, the sum of the face benefits of a given MPG is twice the sum of its edge benefits. Hence maximizing the face benefit is equivalent to maximizing the edge benefit sum. TESSA begins by selecting, from a list of all possible triangular faces on n vertices, the face with the greatest benefit as the initial partial solution. It then repeatedly adds faces with highest benefit to the boundary of the partial solution, where the boundary of a partial solution is defined to be the set of edges and vertices which are still available for inclusion in entering faces, without destroying its planarity, until the MPG is complete. Two operations exist, as shown in Figure 3.5, where we either add a face which contains two adjacent vertices on the boundary, or, add a face containing three consecutive vertices on the boundary.

By placing the $n(n-1)(n-2)/6$ possible faces in a list of unused faces in decreasing order of benefit, deciding ties arbitrarily, we are able to simply find the first face in the list that can be inserted, update the list, and repeat until $2n-4$ faces exist in the layout. Note that the explicit inclusion of the last face is unnecessary, as it will be the boundary of the current partial solution on $2n-5$ faces. If all three vertices of the face are consecutive on the boundary of the current partial solution, as in Figure 3.5(a), and the boundary has more than three edges, the length of the boundary is reduced by one, and the vertex b has moved from the boundary into the interior of the new partial solution. Therefore no other faces containing b can be added. Note that if the boundary of the current partial subgraph has three vertices, application of this operation would prematurely end the heuristic, and hence is disallowed. If two of the vertices of the face are consecutive on the boundary, and the third is not in the current partial solution, as in Figure 3.5(b), the length of the boundary is increased by one, and the edge de has moved from the boundary into the interior of the new partial solution. Thus no other faces containing the edge de can be added. A full description of TESSA is given in Algorithm 3.2, where the set F is the set of faces in the current solution, B_V is



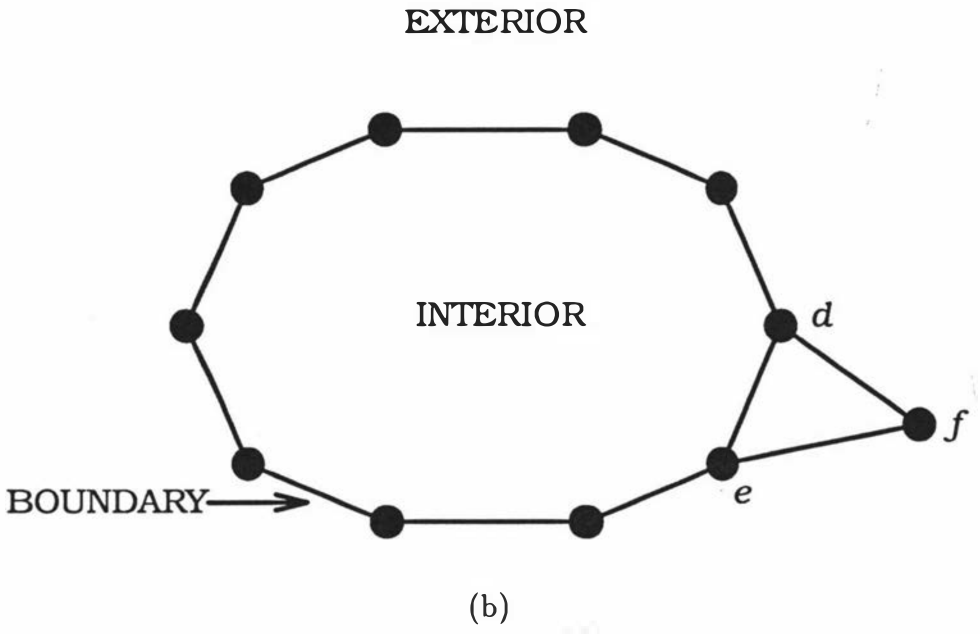


Figure 3.5: The TESSA Operations: (a) Adding a new face *abc* to the current partial solution; (b) Adding a new face *def* to the current partial solution

the set of vertices on the boundary of the current solution, and B_E is the set of edges on the boundary of the current solution.

Algorithm 3.2 *TESSA*

```
    **Input:** *Adjacency Matrix*
    **Output:** *TESSA MPG G*
    *Order faces* $n(n-1)(n-2)/6$ *in non-increasing order, denote ith face of this list as the graph* $M_i$*, with vertices* $m_{i1}, m_{i2}$*, and* $m_{i3}$
    $V_G \leftarrow V_{M_1}$
    $E_G \leftarrow E_{M_1}$
    $T_G \leftarrow T_{M_1}$
    $F \leftarrow \{1\}$
    $B_V \leftarrow V_{M_1}$
    $B_E \leftarrow E_{M_1}$
    **while** $(|T_G| < 2n-5)$ **do**
        **for** $i = 2$ **to** $n(n-1)(n-2)/6$ **do**
            **if** $(i \notin F$ **and** $\{(m_{i1}, m_{i2})\} \in B_E$ **and** $m_{i3} \notin V_G)$ **then**
            **Relabel vertices of* $M_i$ *if necessary to obtain all perturbations**
                $V_G \leftarrow V_G \cup V_{M_i}$
                $E_G \leftarrow E_G \cup E_{M_i}$
                $T_G \leftarrow T_G \cup T_{M_i}$
                $F \leftarrow F \cup \{i\}$
                $B_V \leftarrow B_V \cup V_{M_i}$
                $B_E \leftarrow (B_E \cup E_{M_i}) \backslash \{(m_{i1}, m_{i2})\}$
                $i \leftarrow 1$
            **end**
            **if** $(i \notin F$ **and** $\{(m_{i1}, m_{i2}), (m_{i2}, m_{i3})\} \in B_E$ **and** $|B_V| > 3)$ **then**
            **Relabel vertices of* $M_i$ *if necessary to obtain all perturbations**
                $V_G \leftarrow V_G \cup V_{M_i}$
                $E_G \leftarrow E_G \cup E_{M_i}$
                $T_G \leftarrow T_G \cup T_{M_i}$
                $F \leftarrow F \cup \{i\}$
                $B_V \leftarrow B_V \backslash \{m_{i2}\}$
                $B_E \leftarrow B_E \cup \{(m_{i1}, m_{i3})\} \backslash \{(m_{i1}, m_{i2}), (m_{i2}, m_{i3})\}$
                $i \leftarrow 1$
            end
            $i \leftarrow i + 1$
        end
    end
    for $i = 1$ to $n(n-1)(n-2)/6$ do
        if ($i \notin F$ and $|V_{M_i} \cap B_V| = 3$) then
            $T_G \leftarrow T_G \cup T_{M_i}$
            $i \leftarrow n(n-1)(n-2)/6 + 1$
        end
    end

end
```

Theorem 3.2.8 *The computational complexity of TESSA is* $O(n^5)$

Proof: See [13] □

Al-Hakim [5] provided the first hint that there may exist pathological problems for TESSA, by providing an example where TESSA would generate an infeasible solution. This is not entirely correct as the solution provided is in fact feasible, but has a very poor objective function value. Theorem 3.2.9 confirms the misgivings outlined by Al-Hakim, by providing an example where the worst case performance is achieved.

Theorem 3.2.9 $\rho_{TESSA} = 0$

Proof: Consider the n facility layout problem, with relationship chart as given in Table 3.1. In this problem only four faces have weight three; obviously no other face has weight equal to or greater than two. If face $(1,2,3)$ is chosen as the initial face, then the subsequent three insertions will be $(1,3,4), (1,4,5)$ and $(1,2,5)$, since no other face has weight three. This gives the partial solution of weight 12. Obviously, regardless of the order in which the first four faces are added, the partial solution will take the form of Figure 3.6.

Note that no other face which includes vertex 1 can be added to this partial graph, as vertex 1 has become interior. Since there remain no more faces of weight three, the next four faces in the list are all of weight two; no other face has weight greater than one. Only one of the faces of weight two, i.e. $(2,3,4), (3,4,5), (2,4,5)$

Edge	Weight
$(1,i)\ \forall i \neq 1$	1
$(2,3)$	1
$(3,4)$	1
$(4,5)$	1
$(2,5)$	1
All other edges	0

Table 3.1: Relationship Chart for the Pathological TESSA Example

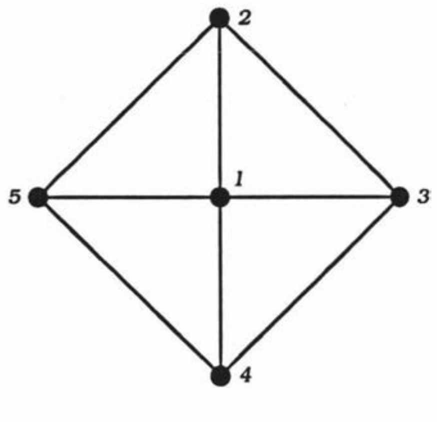


Figure 3.6: The partial solution after the addition of the first four faces

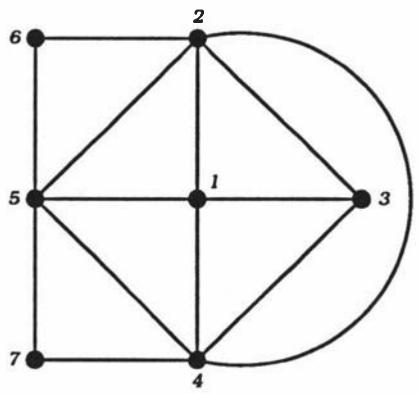


Figure 3.7: A partial solution after all possible non-zero weight faces are added by TESSA

and $(2,3,5)$, can be added, as the addition of any one of these faces leads to a boundary of three vertices. Subsequently, of the three faces not inserted, two are now unavailable because one of the vertices 2,3,4 or 5 is now interior. The other face is the current boundary and cannot be added; therefore the next insertion will make one of the edges of this face interior. The addition of any one of these faces will be the fifth insertion TESSA makes, and leads to the objective value being incremented by 2. Of the three edges now on the boundary, two have weight one, whilst the other has weight zero. Now, while there are still $2(n-5)$ faces with weight one remaining to be added, they all include either of the boundary edges of weight one. TESSA can add only one face adjacent to each of these edges, as the edges will then become interior. These two insertions will increment the objective value by two, and each will be a case (b) insertion. The remaining $n-7$ vertices will not increase the value of the MPG when they are added to the partial solution, as no non-zero weight faces remain which can be added to the partial solution, and so the value of the TESSA MPG is 16. A possible partial TESSA MPG could be that shown in Figure 3.7 where face $(2,3,4)$ was chosen, followed by $(2,5,6)$ and $(4,5,7)$.

Consider now an alternative solution to this problem, which includes all vertices being adjacent to facility 1, for example faces $(1,2,3),(1,3,4),(1,4,5),\ldots,(1,n-1,n),(1,n,2)$, which gives weight $2(n-2)+3$. We could also add faces $(2,3,4),(2,4,5)$ and $(2,5,6)$ to increment the solution by 5, to give a solution value of $2n+4$, with additional edges added to obtain maximal planarity. Finally Equation 3.10 completes the proof.

$$\rho_{TESSA} = \lim_{n\to\infty}\left(\frac{16}{2n+4}\right) = 0 \tag{3.10}$$

□

3.3 Improvement Procedures

Finally in this chapter, we examine the concept of improvement procedures, which attempt to make local changes to a heuristically constructed MPG, in an attempt to improve upon the total adjacency benefit of the MPG. The most simple local improvement is the diagonal swap which has been described by various authors [3, 28, 34, 35, 90], although it is not always referred to as the diagonal swap. The diagonal swap considers two faces (a, b, c), and (a, b, d), which share the edge (a, b). The *diagonal* of (a, b) is (c, d), and (c, d) may or may not appear in the current MPG. If (c, d) is in the MPG, then edge (a, b) is said to be *braced*, while if (c, d) does not appear in the MPG, (a, b) is said to be *unbraced*. Eggleton, Al-Hakim and MacDougall [29] showed that at most two thirds of the edges in an MPG can be braced. The diagonal swap works on an unbraced edge, replacing the unbraced edge with its diagonal. This operation is shown in Figure 3.8.

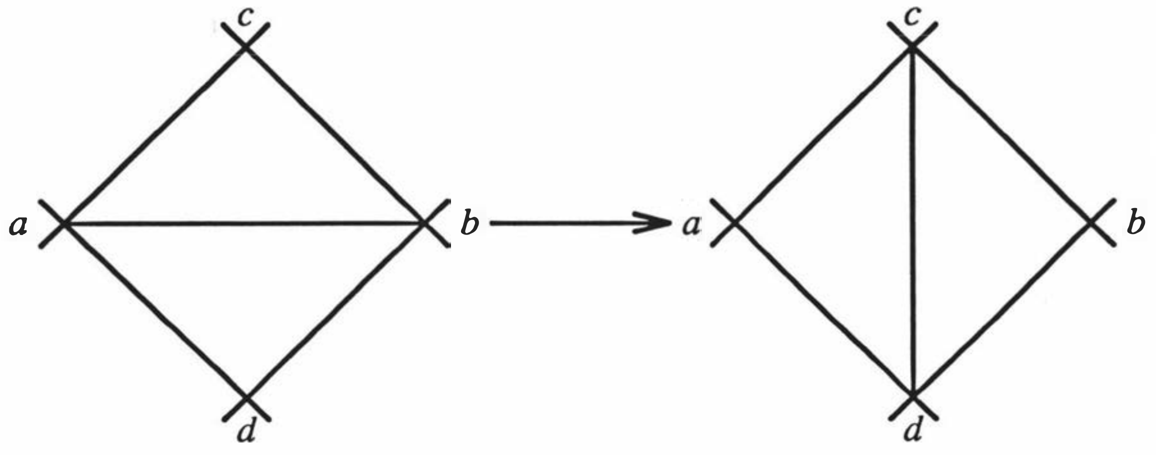


Figure 3.8: The Diagonal Swap Operation $(a, b) \to (c, d)$

Foulds and Robinson [35] introduced the α-operation, which consists of two cases, the first of which is the diagonal swap, of Figure 3.8, and the second case

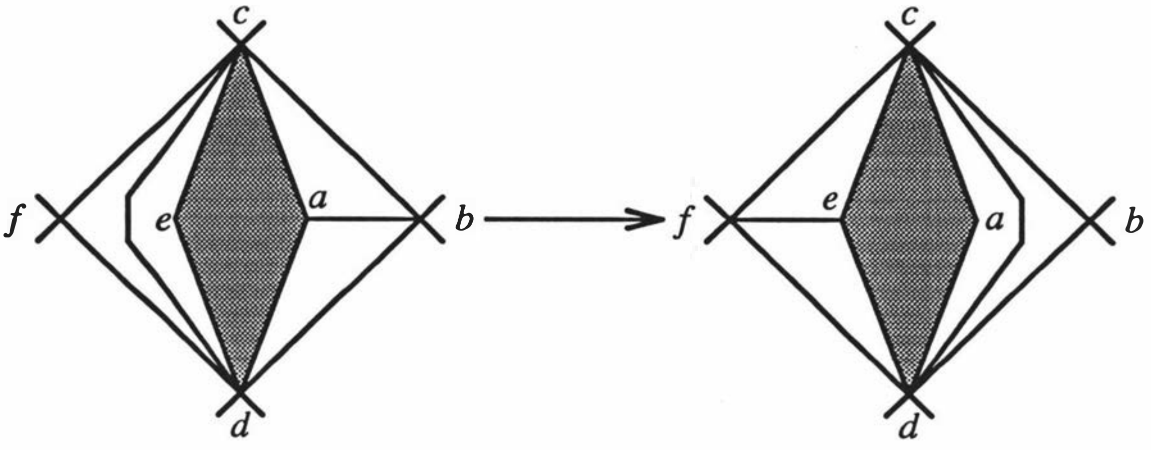


Figure 3.9: The Second Case of the α-Operation $(a, b) \to (e, f)$

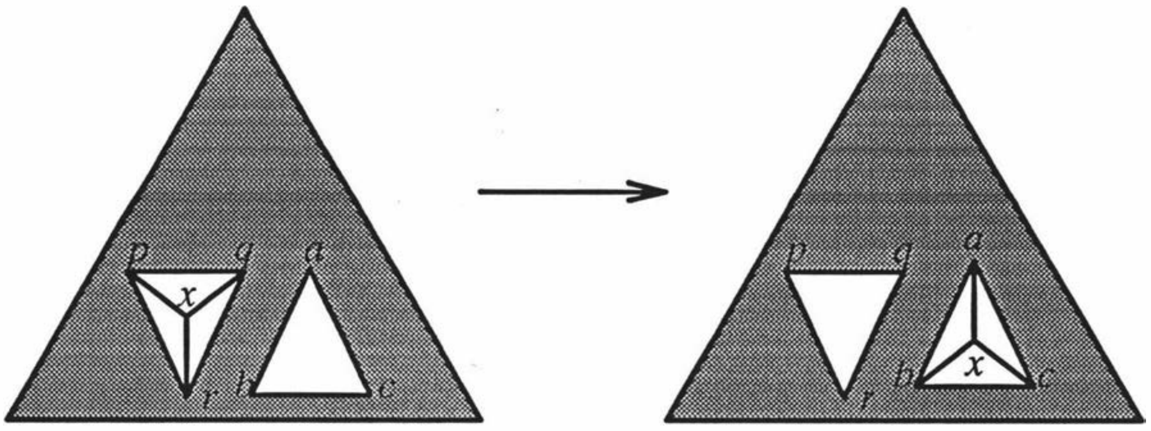


Figure 3.10: The Beta Operation $x \to (a, b, c)$

can be applied to a braced edge (a, b) as shown in Figure 3.9, and is equivalent to the two diagonal swaps $(c, d) \to (e, f)$, followed by $(a, b) \to (c, d)$.

Foulds and Robinson [34, 35] also considered the repositioning of a vertex of degree three, termed the β-operation, shown in Figure 3.10, where a vertex x is taken from face (p, q, r), and placed in face (a, b, c).

Foulds and Robinson proved that the α and β operations are sufficient to transform any MPG on n vertices to any other MPG on n vertices by a finite sequence of these operations. The authors also conjectured that the β-operation may in fact be redundant. The second case of the α-operation can be decomposed into first case α-operations, or diagonal swaps, hence the conjecture of Foulds and Robinson reduces to the diagonal swap alone. Lehel [74] provided a proof of this conjecture, however Eggleton and Al-Hakim [28] showed that Lehel's first assumption was incorrect. A proof was fully completed by Ning [89] who showed the β-operation is indeed redundant, and hence the α-operation alone is sufficient to transform any MPG on n vertices to any other. Eggleton and Al-Hakim provided another proof by firstly showing the flaw in Lehel's proof, then providing a correction to complete Lehel's proof.

Al-Hakim [3] presents the Γ-operation, which is an extension of the diagonal swap, but overcomes the problem of braced edges. However Boswell [12] presents the Γ-operation in a somewhat more elegant fashion. The following description of the Γ-operation is based on that of Boswell.

The Γ-operation possesses three cases, the first of which is the standard diagonal swap of Figure 3.8. The second case considers an edge (a, b), braced by its diagonal (c, d). With reference to Figure 3.11, showing this operation, the diagonal of (c, d) is (a, e). So the operation is that of $(a, b) \rightarrow (a, e)$, overcoming the difficulty of the fact that (a, b) is braced.

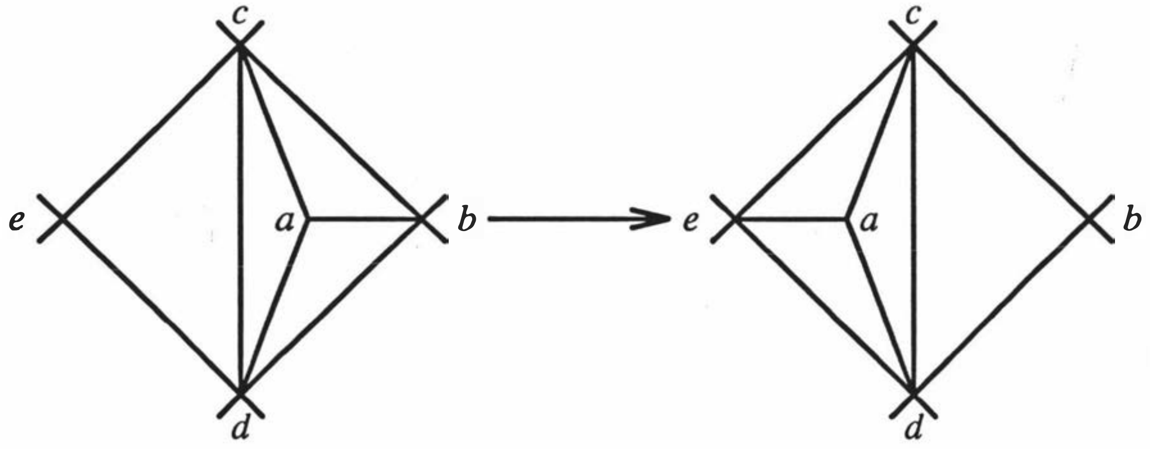


Figure 3.11: The Second Case of the Γ-Operation $(a, b) \rightarrow (a, e)$

The third case of the Γ-operation is where edge (a,b) is braced by (c, d), which has diagonal (e, f), where e and f are distinct from a, and b. There is a path between vertex a, and e or f, which does not pass through b, c, or d. The edge (a, b), may be replaced by one of edges (a, f), (b, e), or (e, f). Figure 3.12 shows these three cases. The shaded areas of Figure 3.12 indicate an arbitrary subgraph which contains a path between a and e, and there also exists a path between b and

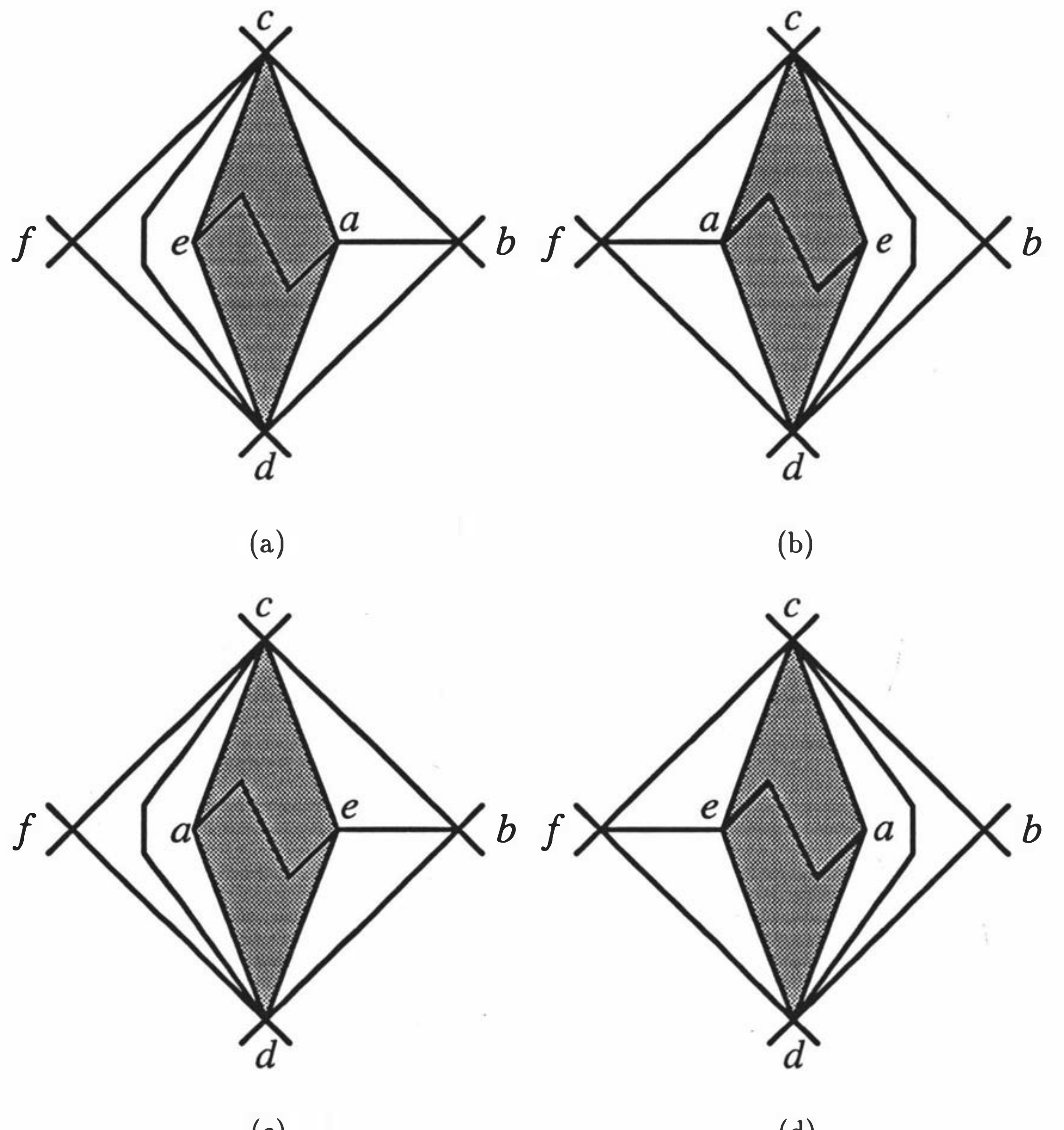


Figure 3.12: The Third Case of the Γ-Operation (a):(a, b) → (b):(a, f), (c):(b, e), (d):(e, f)

f. If there is no path between a and e, then paths between a and f, and b and e exist, and (a, b) may be replaced by (a, e), (b, f), or (e, f).

Boswell shows that the Γ-operation is the best possible single edge replacement method by proving a series of theorems about edge disjoint MPGs, the proofs of which can be found in [12].

Theorem 3.3.1 *If two MPGs have edge difference one, then they can be an arbitrarily large number of α-operations apart.*

Theorem 3.3.2 *If two MPGs have edge difference one, then they are only one Γ-operation apart.*

Theorem 3.3.3 *If two MPGs have edge difference two, then they are only two or three Γ-operations apart.*

Theorem 3.3.4 *If two MPGs have edge difference three, then they can be an arbitrarily large number of Γ-operations apart.*

A number of improvement frameworks have been developed in order to implement these improvement procedures, ranging from simple iterative improvement [3, 13, 31, 34], to more complex meta-heuristic frameworks such as Simulated Annealing [22, 62] and Tabu Search [43, 44, 45], as implemented by Boswell [13]. Foulds *et al.* [31] perform a computational experiment between Deltahedron, Wheel Expansion, and Greedy (which recall, requires planarity testing). The authors also include Deltahedron with greedy α and β improvement operations. Both the SELECT and AVARICE constructions were used for Deltahedron, with and without improvements, and the Wheel Expansion heuristic. The computational results show that Greedy and Deltahedron with improvements perform the best overall, with the Wheel Expansion Heuristic performing poorly as it was dominated by all three methods in most cases. Foulds *et al.* noticed that this was a little surprising, as Wheel Expansion is a generalisation of Deltahedron. Al-Hakim [3] showed the superior performance of the Γ-operation, over the α-operation, on a set of 24 test problems. Boswell [13] undertook a full experiment comparing the Γ-Operation under Tabu Search, Simulated Annealing, and Iterative Improvement frameworks. The results show that the Γ-operation under the Tabu Search framework, is the most effective improvement procedure for obtaining an improved solution to an initial MPG, providing far superior solutions to iterative improvement, and slightly superior solutions to Simulated Annealing, but with far less running time required.

Coda:

In this chapter we have described the various methods in the literature for generating a highly weighted MPG. The most widely used methods are those which bypass the need for planarity testing by exploiting the structure of the MPG. Furthermore computational experiments conducted by various authors have shown that regardless of the initial construction of the MPG, the most powerful tool to constructing a highly weighted MPG, is the implementation of improvement procedures, the most powerful of which is the Γ-operation, using a Tabu Search framework to direct the improvement process. We will return to the role of the MPG later, and in particular its lack of correspondence with the most practical layout.

In the next chapter, we examine the second phase of the GTFLP, the development of the layout from the MPG and area requirements. We examine the current methods in the literature, and where appropriate develop these algorithms to facilitate implementation. We will see that there exists a simple procedure for Deltahedron MPGs, and we describe and develop a variety of more theoretical methods for arbitrarily constructed MPGs. We examine the greater flexibility of layout procedures which do not enforce an overlying grid, and finally show that characterisations of some MPGs are possible so that the layout is developed easily, utilising the Deltahedron type of layout approach, by exploiting structure inherent in these MPGs.

Chapter 4

Obtaining the Block Plan

In this chapter we examine the construction of the dimensioned orthogonal geometric dual, or layout, for any specified MPG with area requirements. We discuss in detail previous methods from the literature, providing enhancements to these methods where appropriate to facilitate implementation. Previously, little had been done to implement the more theoretical methods that have been presented. The modifications, either in the implementation, or in rounding out the theoretical background, are in some instances to the point where the original algorithm is no longer recognisable, with only the motivational argument remaining. This effectively leads to some new methods for designing a layout, and indeed we will also see the motivation for, and implementation of, some rather different methods. Finally we present a characterisation of MPGs with special structural properties which enable a dimensionalisable layout to be easily constructed by exploiting the MPG structure.

4.1 Deltahedron

The first method that we will discuss is the Deltahedron layout method. As could probably be surmised, this method uses the special structure of the Deltahedron MPG from Section 3.2.1 to create the layout. The principle behind this approach was first shown by Giffin, Foulds and Cameron [41] for Deltahedron MPGs with at most two distance classes, plus the exterior facility, and was fully developed by Giffin, Watson and Foulds [42] for arbitrary Deltahedron MPGs. The description of the Deltahedron Layout Algorithm which follows is based on that of the later.

4.1.1 The Deltahedron Layout Algorithm

In Section 3.2.1 we showed that the Deltahedron heuristic proceeds by inserting one vertex and three edges at a time into a face of the current MPG, starting with the Tetrahedron as the initial triangulation. The layout is constructed in essentially the same manner by considering an initial template representing the Tetrahedron, and then placing facilities in the layout in the same order in which they were inserted into the MPG. Throughout this description, we assume that the exterior facility, labelled 1, is in the initial Tetrahedron, which is a requirement for the creation of the layout by this method. We will see in Section 4.1.3 that this requirement can always be satisfied by a rearranging of the insertion order used to generate the MPG, which forces the exterior to be in the initial Tetrahedron without changing the final MPG.

The initial tetrahedron may be constructed as shown in Figure 4.1. The wall intersections of the layout labelled J are the 3-joints, whereas those labelled j denote 2-joints. This labelling can be easily updated to reflect subsequent placements of facilities into the layout.

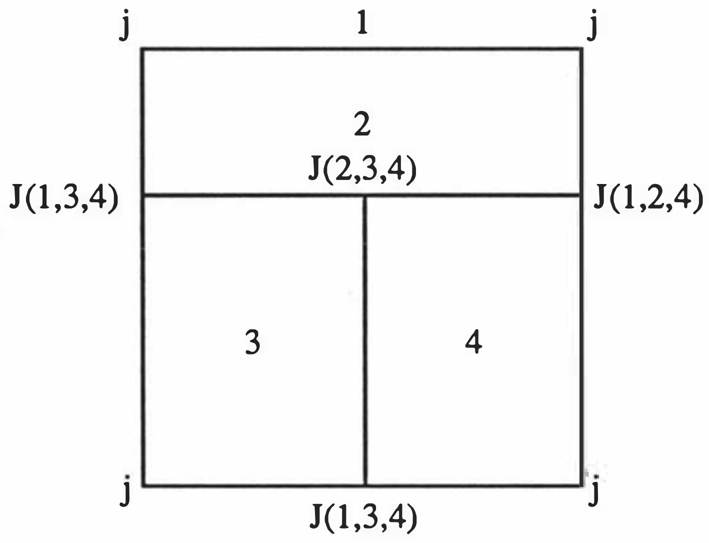


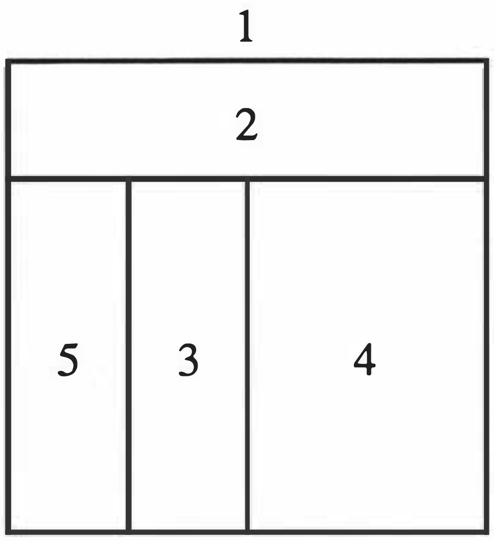
Figure 4.1: The Initial Deltahedron Layout

If a face (f_1, f_2, f_3) is deleted from the MPG by inserting a vertex f in this face, then in the layout f must be placed adjacent to the facilities represented in the MPG by f_1, f_2 and f_3. In order to maximise the resultant regularity of these four facilities, we require a suitable placement mechanism, which maintains as much of the current structure of the layout as possible. In order to attain this, we place f *inside* one of f_1, f_2, or f_3 bordering the other two facilities. The dimensioning of the layout must also be addressed, and the placement of facilities in the layout must ensure that we can dimension the layout once the layout is completed, by ensuring that no faultlines are formed during the placement process. Figure 4.2 shows the four possibilities for the insertion of a facility 5 into the initial layout, corresponding to vertex 5 being placed in each of the four faces of the initial MPG. We describe such placements of a facility to be *at* a particular 3-joint.

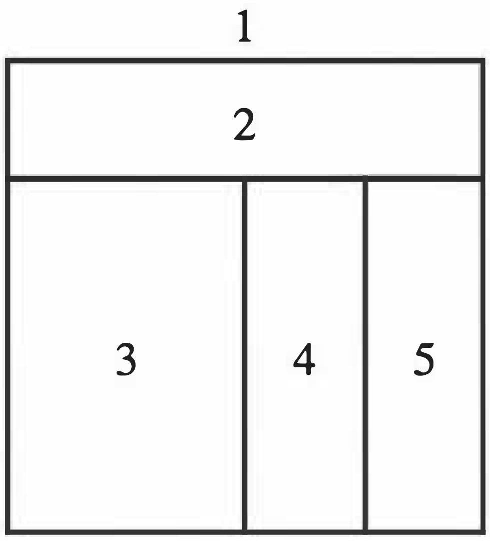
In Figure 4.2(a) facility 5 is being placed at 3-joint $(1,2,3)$, *i.e.* adjacent to facilities 1, 2, and 3. In order to maintain the rectangular layout perimeter the placement host must be either facility 2 or 3. If facility 2 were to be chosen, then the layout would take the form of Figure 4.3, with all facilities maintaining their rectangularity, and the adjacencies $(1,5)$, $(2,5)$ and $(3,5)$ would all be assured, regardless of the facility areas. However if $a_4 a_5 > a_2 a_3$, then adjacency $(2,3)$ would not be satisfied. Therefore we do not allow placements of the form of Figure 4.3.

Some 3-joint labels require updating to reflect the placement of facility 5. In the MPG, face $(1,2,3)$ is removed, being replaced by faces $(1,2,5)$, $(1,3,5)$, and $(2,3,5)$, which is parallelled in the layout, with the 3-joint $(1,2,5)$ replacing $(1,2,3)$, and two new 3-joints $(1,3,5)$, and $(2,3,5)$ being created. The placement of facility 5 in Figures 4.2(b) and 4.2(c) are similar to that of Figure 4.2(a), however the placement of facility 5 at the 3-joint $(2,3,4)$ is different, since facility 5 cannot be adjacent to the exterior. This cannot be accommodated, while maintaining rectangularity of each facility, and either facility 3 or 4 must become L-shaped as shown in Figure 4.2(d).

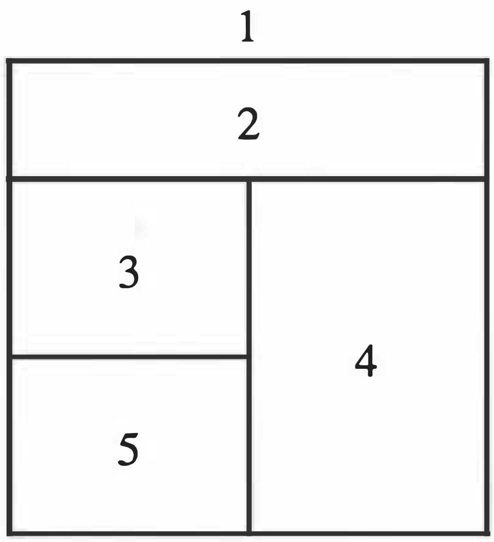
Two basic *placement operations* are easily identifiable, as those of Figures 4.2(a) and 4.2(d); each having an obvious variation if the placement host is L-shaped rather than rectangular, and implicit variants determined by reflections and rotations of these operations. Describing fully these two operations will enable us to see the choice of placement hosts, and the use of *placement directions.* We call the two operations PO1, the general form of which is given in Figure 4.4, and is akin to the operation of Figure 4.2(a), while the second operation of Figure 4.5 is called PO2, and is akin to Figure 4.2(d). Figure 4.4 shows facility f_2 being placed in f_1 at 3-joint J. For simplicity, the fact that f_1 is chosen as the placement host at J is indicated by an arrow emanating from J into f_1, called a *placement direction.*



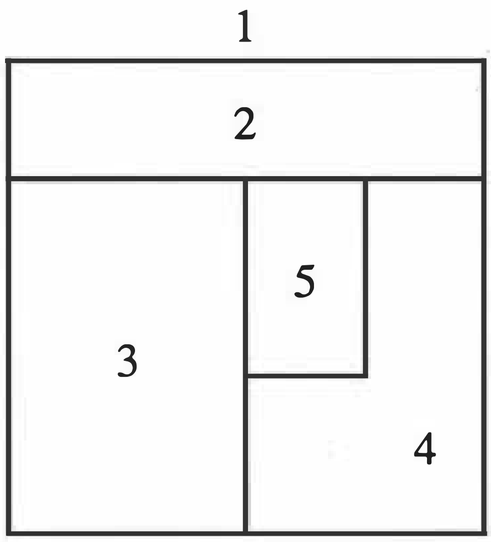
(a) 5 at (1,2,3)



(b) 5 at (1,2,4)



(c) 5 at (1,3,4)



(d) 5 at (2,3,4)

Figure 4.2: Four options for placing facility 5 within initial layout

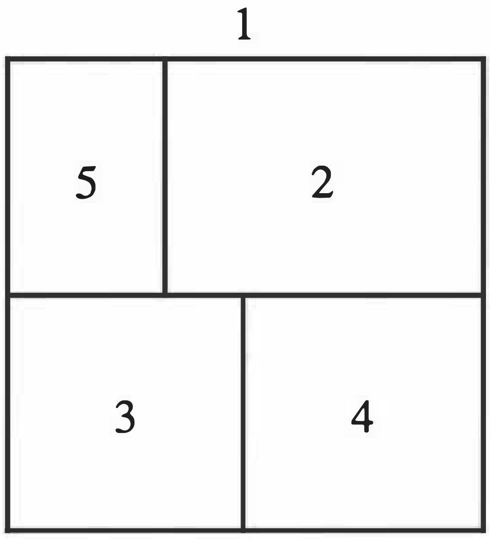


Figure 4.3: A forbidden placement of facility 5 at (1, 2, 3)

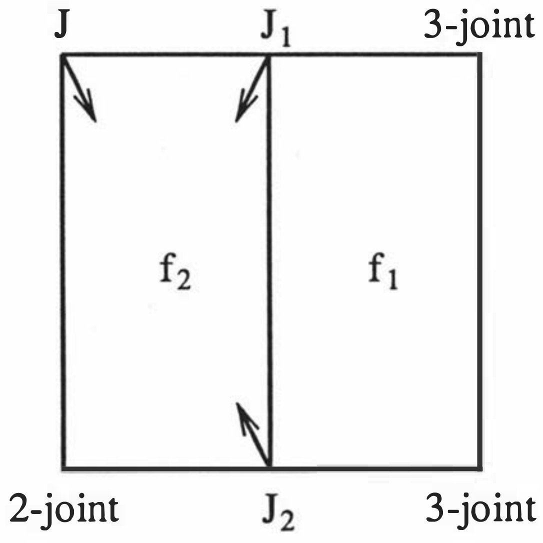


Figure 4.4: Deltahedron Placement Operation PO1

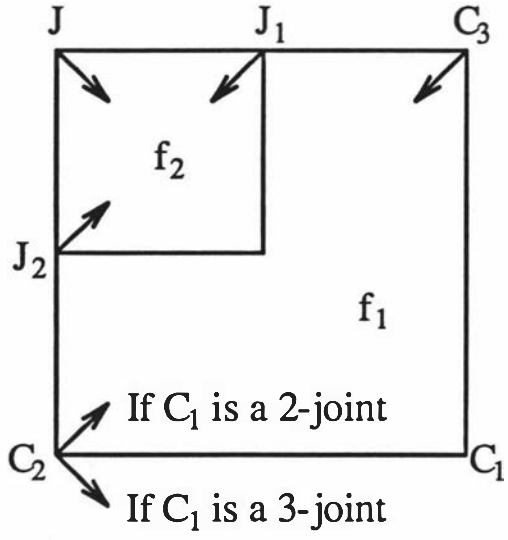


Figure 4.5: Deltahedron Placement Operation PO2

New 3-joints J_1 and J_2 are created by the placement of f_2. In the undimensioned layout, J_1 and J_2 are created by bisecting the wall containing the adjacent joints. The placement direction at J_2 is as shown, so that any further facility placed at J_2 will enable f_1 and f_2 to maintain rectangularity, whereas placing within f_1 would make f_1 L-shaped. Having determined the placement direction for J_2, the direction for J_1 must follow on the same *side* of the wall, in order that a faultline is not created. This ensures the maintenance of the adjacency between f_1 and f_2 irrespective of subsequent placements at either J_1 or J_2. When applying PO1 to an L-shaped placement host, the 3-joint J_1 bisects the wall connecting J and the 3-joint adjacent to J, with J_2 equidistant from the 2-joint, whilst maintaining orthogonality. Using the L-shaped placement host for PO1, the placement directions at J_1 and J_2 are defined similarly as for the rectangular version of PO1, whereas those for the other existing 3-joints of f_1 have already been defined by some previous application of PO2, which we now consider.

The general form of the placement operation PO2 is shown in Figure 4.5 for the case of a rectangular placement host, with the obvious variation for the L-shaped placement host. The placement is of facility f_2 in f_1 at J. The two cases where C_1 is a 2- or 3-joint may be considered together as only the placement direction at C_2 is affected in each instance. Note that C_2 must be a 3-joint, otherwise we would be able to apply PO1. The new 3-joints, are again labelled J_1 and J_2, with the placement directions at these 3-joints motivated by the desire to maintain facility shape regularity. Had either of these placement directions been directed into f_1, f_1 would prematurely lose its L-shape. It is evident from their descriptions that applications of PO1 and PO2 will not result in prescribed adjacencies being lost.

Note that following the insertion of the first facility into the initial layout, all subsequent placement directions are then uniquely defined. Regularity of the layout is maintained, as Deltahedron layouts have a worst case room shape topologically equivalent to a T, which we prove in Theorem 4.1.1.

Theorem 4.1.1 *The operations PO1 and PO2 together with the initial layout configuration for the Tetrahedron will always generate at worst T-shaped facilities*

Proof: Certainly this is true for all layouts of four or five facilities, as shown by Figures 4.1 and 4.2. Let us then assume that we have a current layout with at least five facilities, so all placement directions are now uniquely prescribed. The insertion of the entering facility is made totally within the appropriate placement host, therefore only the placement host can have its shape worsened. Note that the application of PO1 from Figure 4.4 does not worsen the shape of f_1, and therefore only PO2 can. Similarly, if we perform PO2 when f_1 is a rectangle, then f_1 becomes L-shaped. The only way in which we may create a facility shape worse than a rectangle or an L is to perform PO2 on an L-shaped placement host. Consider then PO2, with f_1 L-shaped; referring to Figure 4.5, the only way in which the shape of f_1 can be worsened is via placement at C_3, when C_1 is a 3-joint, as if C_1 is a 2-joint, we are performing PO1. If we perform this placement at C_3, then f_1, will become T-shaped; but, by construction, no placement directions now lie within f_1, hence f_1 can never again be a placement host, consequently never worsening its shape beyond a T. □

We must now consider the final phase of the Deltahedron Layout Algorithm: dimensioning. Deltahedron layouts by construction are dimensionalisable. This enables the dimensioning to be applied as a post-construction phase. Indeed, the layout itself is constructed without recourse to the area specifications, which as we will see in examining other methods is a most desirable property. The process whereby we enforce the area specifications on an undimensioned dimensionalisable layout is called *inflation*. By sequentially considering *blocks* of facilities created by the initial layout, or by applications of PO1, we are able to easily determine the

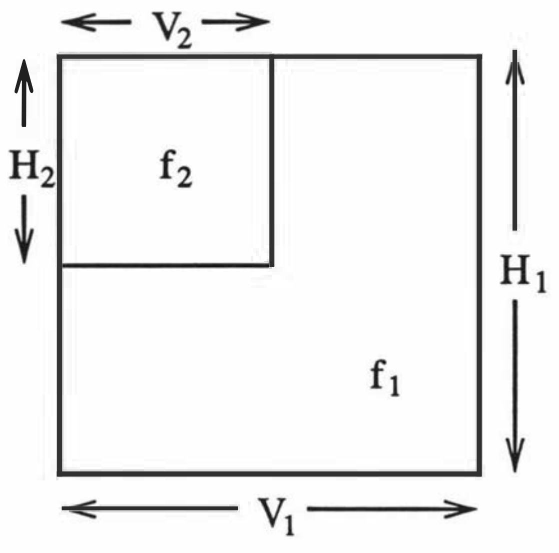


Figure 4.6: Dimensioning of L-shaped Facilities

dimensions of these blocks given only the dimensions of the layout perimeter. The only difficult component of this process is the dimensioning of T- and L-shaped facilities, and the blocks of facilities within them. Firstly let us consider Figure 4.6 which examines a facility f_1, and a facility or block of facilities f_2 placed within f_1. A standard way of dimensioning f_1 and f_2 is given by Equation 4.1.

$$V_2 = \sqrt{\frac{a_{f_2}}{a_{f_1} + a_{f_2}}} V_1 \qquad H_2 = \sqrt{\frac{a_{f_2}}{a_{f_1} + a_{f_2}}} H_1 \tag{4.1}$$

We treat the case for the T-shaped facility in much the same way, with reference to Figure 4.7. Facilities f_2 and f_3 must not be adjacent, let alone overlap, and this is easily accomplished via Equations 4.2 and 4.3. Note that H_2 and H_3 are the same value, hence f_1 will have 4 collinear corner points, and hence can be divided easily into two rectangles.

$$H_2 = \frac{1}{2}(1 + \frac{a_{f_2} + a_{f_3}}{a_{f_1} + a_{f_2} + a_{f_3}})H_1 \qquad V_2 = \frac{2a_{f_2}}{a_{f_1} + 2a_{f_2} + 2a_{f_3}} V_1 \tag{4.2}$$

$$H_3 = \frac{1}{2}(1 + \frac{a_{f_2} + a_{f_3}}{a_{f_1} + a_{f_2} + a_{f_3}})H_1 \qquad V_3 = \frac{2a_{f_3}}{a_{f_1} + 2a_{f_2} + 2a_{f_3}} V_1 \tag{4.3}$$

The full Deltahedron Layout Algorithm is given in Algorithm 4.1.

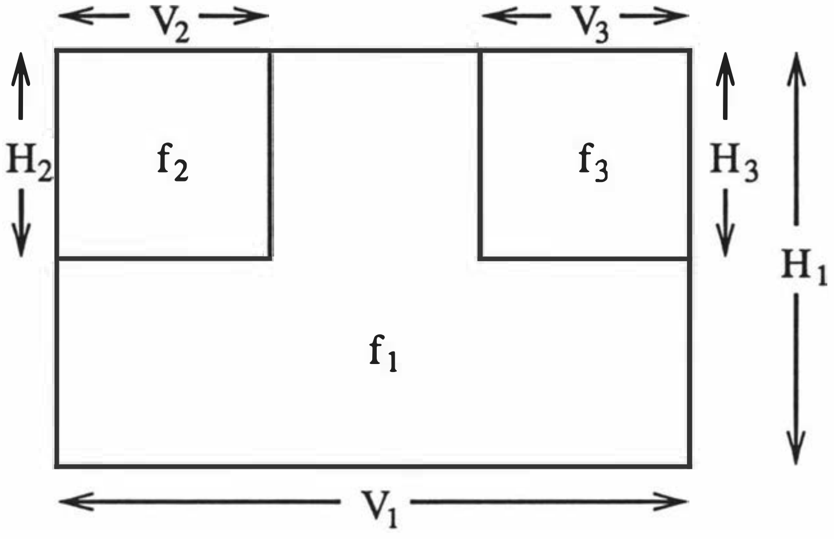


Figure 4.7: Dimensioning of T-shaped Facilities

Algorithm 4.1 *The Deltahedron Layout Algorithm*

```
Input: Deltahedron Insertion Order in form (vertex,triangle) of MPG, together with initial tetrahedron, exterior in tetrahedron; area specifications
Output: Dimensioned Layout dual to MPG
Create initial layout from tetrahedron
for each (vertex,triangle) of Deltahedron Insertion Order do
    $f_2$ $\leftarrow$ new facility to be placed
    J $\leftarrow$ 3-joint
    $f_1$ $\leftarrow$ placement host indicated by placement direction at J
    if there is a 2-joint $J'$ adjacent to $J$ then
        apply PO1 at J
        add placement directions for $f_2$
    end
    else
        apply PO2 at J
        add placement directions for $f_2$
    end
end
Dimension layout via inflation

end
```

4.1.2 An Illustrative Example

We now proceed to illustrate the Deltahedron Layout Algorithm using the MPG of Figure 4.8, and area specifications of Table 4.1. The initial tetrahedron has vertices 1,2,3 and 4, while the insertion order is given in Table 4.2. Initially we arbitrarily set the placement directions at joints $(2,3,4)$, and $(1,3,4)$ into facility 4, however following the insertion of facility 5 these must now be directed into facility 3; this is the only occasion where such a redirection is required. The final dimensioned layout is given in Figure 4.9, where the arrows indicate the placement directions of any further placements which could occur. Notice that facility 3 (which is T-shaped) has no placement directions directed into it, and hence will never become worse than a T-shape. The dimensioning is first examined by considering the ratio $a_2{:}a_3+a_4+a_5+a_6+a_7+a_8+a_9+a_{10}$, followed by $a_3+a_6+a_7+a_8{:}a_4+a_5+a_9+a_{10}$, and $a_4{:}a_5+a_9+a_{10}$. The dimensioning of facility 6 is completed by the ratio $a_6{:}a_3+a_7+a_8$, while facility 9 is dimensioned via $a_9{:}a_5+a_{10}$. Facilities 7, 8 and 10 are dimensioned via Equations 4.1 - 4.3.

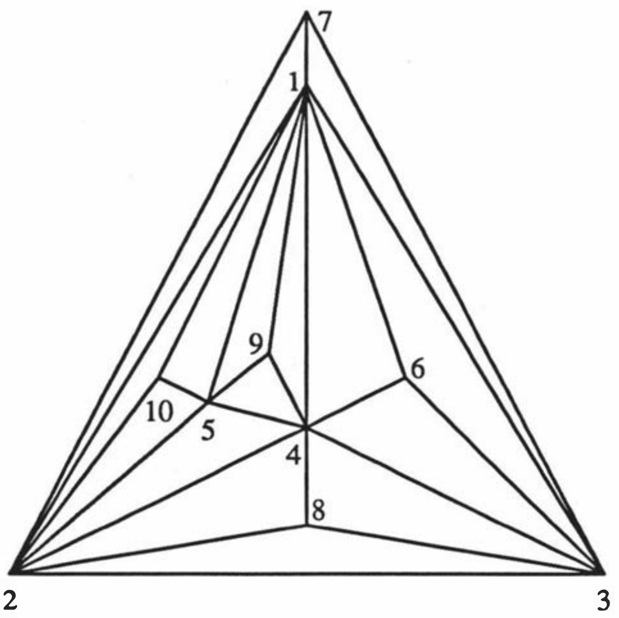


Figure 4.8: Deltahedron Illustrative Example

Facility	1	2	3	4	5	6	7	8	9	10
Area	-	15	12	12	10	19	11	18	21	10

Table 4.1: Area specifications for the Deltahedron Illustrative Example

Facility	3-joint	Placement Operation
5	(1,2,4)	PO1
6	(1,3,4)	PO1
7	(1,2,3)	PO2
8	(2,3,4)	PO2
9	(1,4,5)	PO1
10	(1,2,5)	PO2

Table 4.2: Placement Operations for the Deltahedron Illustrative Example

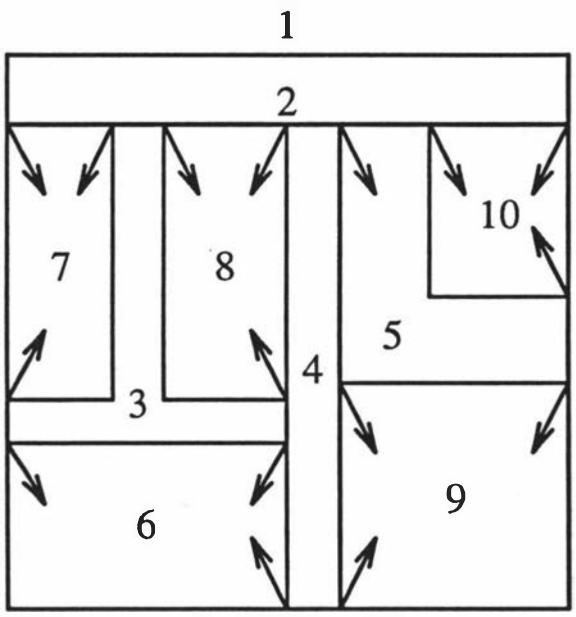


Figure 4.9: The Deltahedron Layout for the Illustrative Example

4.1.3 Obtaining an Initial Tetrahedron containing the Exterior

It has been assumed that in order for the Deltahedron Layout Algorithm to generate a feasible layout, the exterior vertex must be one of the vertices in the initial tetrahedron to ensure that the exterior facility is to be on the outside of the final layout [31]. We present two algorithms to show that we can transform an arbitrary Deltahedron-generated MPG into one, where the exterior vertex is in the initial Tetrahedron. In order to describe the first algorithm, define two graphs G_I, the given arbitrary Deltahedron MPG, and G_O, the graph generated from the algorithm. We will show that, regardless of the insertion order used to create G_I, the algorithm will produce the exterior vertex in the Tetrahedron of G_O, and $G_I = G_O$. We call this algorithm the Exterior to Tetrahedron Algorithm 1 (ETA1), which is shown in Algorithm 4.2. The motivation behind this algorithm, is that there exists a Tetrahedron in the MPG which contains the exterior; by first finding this Tetrahedron, and then subsequently building up the insertion order from it, we regenerate our original MPG.

Algorithm 4.2 *Exterior to Tetrahedron Algorithm 1*

```
    Input: $G_I$
    Output: $K_4$; Revised Insertion Order in form (vertex,triangle)
    $V_O \leftarrow \emptyset$
    $E_O \leftarrow \emptyset$
    $T_O \leftarrow \emptyset$
    Determine $a, b, c \in V_I$, such that $\{(a,e),(b,e),(c,e),(a,b),(a,c),(b,c)\} \subseteq E_I$
    $V_O \leftarrow \{a,b,c,e\}$
    $E_O \leftarrow \{(a,b),(a,c),(a,e),(b,c),(b,e),(c,e)\}$
    $T_O \leftarrow \{(a,b,c),(a,b,e),(a,c,e),(b,c,e)\}$
    $K_4 \leftarrow (a,b,c,e)$
    if ($|V_O| < n$) then
        Determine $x \notin V_O$ and $p,q,r \in V_O$ such that $\{(p,x),(q,x),(r,x)\} \subseteq E_I$
        and $(p,q,r) \in T_O$
        Insertion:$(x,(p,q,r))$
        $V_O \leftarrow V_O \cup \{x\}$
        $E_O \leftarrow E_O \cup \{(p,x),(q,x),(r,x)\}$
        $T_O \leftarrow T_O \cup \{(p,q,x),(p,r,x),(q,r,x)\} \setminus \{(p,q,r)\}$
    end

end
```

In order to prove that the algorithm works, we must first prove Lemmas 4.1.1 and 4.1.2.

Lemma 4.1.1 *For every vertex* $x \in V_I$ $\exists$ $\{p,q,r\} \subseteq V_I$ *such that* $\{(p,q),(p,r),(q,r),(p,x),(q,x),(r,x)\} \subseteq E_I$

Proof: During the Deltahedron insertion process, we insert a vertex x into a face (p,q,r) that is already in the adjacency graph. The insertion removes the face (p,q,r) and creates three more - (p,q,x), (p,r,x), and (q,r,x). The insertion of vertex x into face (p,q,r), means that the edges (p,x), (q,x), and (r,x) are all placed in E_I. While additional insertions may delete faces from T_I, no edges are ever removed from E_I. Consider the special case of the initial Tetrahedron: let $K_4 = (i,j,k,l)$, then the edges $(i,j),(i,k),(i,l),(j,k),(j,l)$, and (k,l) are all added to E_I. □

Lemma 4.1.2 *For every face* $(p,q,r) \in T_I$, $\exists\ x \in V_I$ *such that* $\{(p,q),(p,r),(q,r),(p,x),(q,x),(r,x)\} \subseteq E_I$

Proof: Since every vertex is inserted into a face, there must exist a vertex x, such that *w.l.o.g.* vertex q, say, was inserted in face (p,r,x), which would create the face $(p,q,r) \in T_I$, and the edges $\{(p,q),(p,r),(q,r),(p,x),(q,x),(r,x)\} \in E_I$. □

Theorem 4.1.2 *ETA1 will always generate an insertion order containing the exterior face in the initial tetrahedron, given any Deltahedron generated MPG.*

Proof: Having determined which vertex will constitute the exterior facility, we know by Lemma 4.1.1 that the three vertices a, b, and c must exist. If there is no vertex x adjacent to a triangle $(p,q,r) \in T_O$, then $T_O \subseteq T_I$. If $T_O = T_I$, then $|V_O| = n$, and we have finished. Therefore let us assume that $T_O \subset T_I$ and that there is no candidate x to enter V_O which satisfies the condition that $\{p,q,r\} \in V_O$, such that $\{(p,x),(q,x),(r,x)\} \in E_I$, and $\{(p,q,r)\} \in T_O$. Now the graph consisting of the faces in T_O is always a Deltahedron generated MPG. This is because we start with an initial Tetrahedron, consisting of (a,b,c,e) and we add vertices one at a time into a face. Since $T_O \subset T_I$, by Lemma 4.1.2, we know that there must exist a vertex x which will enter one of the faces in T_O. Therefore we have a contradiction, and x is a candidate to enter. □

Theorem 4.1.3 *The worst case time complexity of ETA1 is* $O(n^2)$

Proof: ETA1 is most efficient when, during the generation of the initial MPG, we store the face that each vertex is inserted into (the vertices in the initial Tetrahedron can each have the other 3 vertices in the initial Tetrahedron as their face). In general this is all that is stored, as the adjacency graph is largely redundant. Having determined the exterior facility, and the three vertices adjacent to it, we can order the remaining $n-4$ vertices by their facility number. These steps take O(n) time.

Consider now the partial graph of k vertices. In the worst case, there will be only one vertex which can enter the partial graph, and this vertex will be the last one considered. Therefore, this will take $n-k$ steps. Therefore the total time will be given by Equation 4.4.

$$O(n) + O(\sum_{k=1}^{n-4}(n-k)) \equiv O(n^2) \tag{4.4}$$

□

We now describe a second more elegant, method for producing a revised Deltahedron insertion order, in which the exterior facility is in the initial Tetrahedron. This algorithm, called ETA2, sequentially deletes vertices of degree 3 until we are left with a tetrahedron which contains the exterior facility. ETA2 is described in Algorithm 4.3, again calling the given MPG G_I.

Algorithm 4.3 *Exterior to Tetrahedron Algorithm 2*

```
    Input: $G_I$
    Output: $K_4$; Revised Insertion Order in form (vertex,triangle)
    while ($|V_I| > 4$) do
        if $d_x = 3$ and $x \neq$ exterior facility and $x$ adjacent to $p, q, r$ then
            Insertion:($x$,($p, q, r$))
            $V_I \leftarrow V_I \backslash \{x\}$
            $E_I \leftarrow E_I \backslash \{(p, x), (q, x), (r, x)\}$
            $T_I \leftarrow T_I \backslash \{(p, q, x), (p, r, x), (q, r, x)\} \cup \{(p, q, r)\}$
        end
    end
    Reverse insertion order

end
```

In order to prove that ETA2 also works we require Lemmas 4.1.3 and 4.1.4.

Lemma 4.1.3 *Insertion of a vertex into a Deltahedron MPG on $n \geq 5$ vertices will not decrease the number of vertices of degree 3.*

Proof: Consider a Deltahedron MPG on k vertices, which has i vertices of degree 3. The insertion of a vertex into a face, will only increase the degree of the vertices of degree 3 if they are part of that face. Therefore we can only decrease the number of vertices of degree 3 if there are at least two vertices of degree 3 in the face. But in a MPG we cannot have two adjacent vertices of degree 3 and therefore we cannot decrease the number of vertices of degree 3. An entering vertex can be placed in a face with no vertex of degree 3, and therefore the subsequent Deltahedron MPG on $k+1$ vertices has either i or $i+1$ vertices of degree 3. □

Lemma 4.1.4 *There are at least two vertices of degree 3 in a Deltahedron MPG on $n \geq 5$ vertices.*

Proof: The initial Tetrahedron K_4 has all vertices of degree 3. Now consider the first Deltahedron insertion: a vertex will be placed in a face of this initial Tetrahedron, and will have degree 3. Also the other vertex in the Tetrahedron which is not in the face, will still have degree 3. So a Deltahedron MPG on 5 vertices has two vertices of degree 3, therefore for $n > 5$ we must have at least 2 vertices of degree 3, by Lemma 4.1.3. □

Theorem 4.1.4 *ETA2 will always generate an insertion order containing the exterior face in the initial tetrahedron, given any Deltahedron generated MPG.*

Proof: By Lemma 4.1.4 we know that there are at least 2 vertices of degree 3 initially, and after each deletion. Therefore we can always delete a vertex of degree 3 which is not the exterior, and hence the exterior must be one of the vertices in the final K_4. □

Theorem 4.1.5 *The worst case time complexity of ETA2 is $O(n^2)$*

Proof: Given a list of the vertex degrees we can simply delete the vertex with minimum degree which is not the exterior vertex at each stage, taking O(n) time. However we must update the degrees of the vertices that made up the face that the vertex was deleted from which will also take $O(n)$ time. We must repeat this process $n - 4$ times hence the complexity is given by Equation 4.5.

$$\sum_{i=1}^{n-4}(O(n) + O(n)) \equiv O(n^2) \tag{4.5}$$

□

Obviously ETA2 appears less cumbersome than ETA1, and in practise, when building up an adjacency graph by Deltahedron, as soon as the exterior facility is placed in the MPG, the feasible Deltahedron insertion order can be found; this saves us some work rather than waiting until all vertices are in the MPG before finding the feasible insertion order. ETA2 is conceptually more simple, and easier to implement. The two methods could be thought of as starting at opposite ends, where ETA1 finds the initial Tetrahedron from the arbitrary MPG, and then finds vertices to add, ETA2 finds the initial Tetrahedron as the final step. Therefore we have overcome the problem outlined initially by Foulds *et al.* [31], so that in constructing the Deltahedron MPG, the exterior vertex need mot be treated any differently than the other vertices of the MPG.

4.1.4 The Extended Deltahedron Layout Algorithm

We conclude our discussion of the Deltahedron Layout Algorithm, with an extension of the layout procedure motivated from the extension of Leung [78] to the Deltahedron MPG Algorithm. We discussed in Section 3.2.2 how Leung provided a second operation for the Deltahedron MPG Algorithm to obtain the Extended Deltahedron MPG Algorithm, whereby we insert three mutually adjacent vertices into a single face of the currently constructed MPG, each having two adjacencies with the vertices of the face. This extra operation can be helpful in constructing highly weighted MPGs, however the corresponding layout cannot be directly constructed via the Deltahedron Layout Algorithm given in Section 4.1.1. In this section we will show that we need define two further placement operations PO3, and PO4, which are analogous to PO1 and PO2 respectively, in order to complete the Deltahedron Layout Algorithm Extension. We will discuss the extended method of determining the insertion order, and examine the ramifications upon the worst case facility shape.

Firstly let us determine how we might obtain the insertion order from a given arbitrary Extended Deltahedron MPG, also including the rationale of ETA2, in order to complete both processes simultaneously. Previously we would delete vertices of degree three (which were not the exterior) to obtain our insertion order. In this case, we do exactly the same, except we also delete structures of three mutually adjacent vertices, none of which is the exterior, each of degree four. This second structure is that proposed by Leung. This is done until either we have the Tetrahedron, or the Regular Octahedron, remaining. Note that Leung initialised with only the Tetrahedron, but this could easily be extended to include the Regular Octahedron starting configuration, as outlined in Section 3.2.2. We see here, that there now exist two initial structures, the first is the Tetrahedron, for which we have already developed an initial layout; the Regular Octahedron provides the initial layout configurations of Figure 4.10, with the initial placement directions, again indicated by arrows.

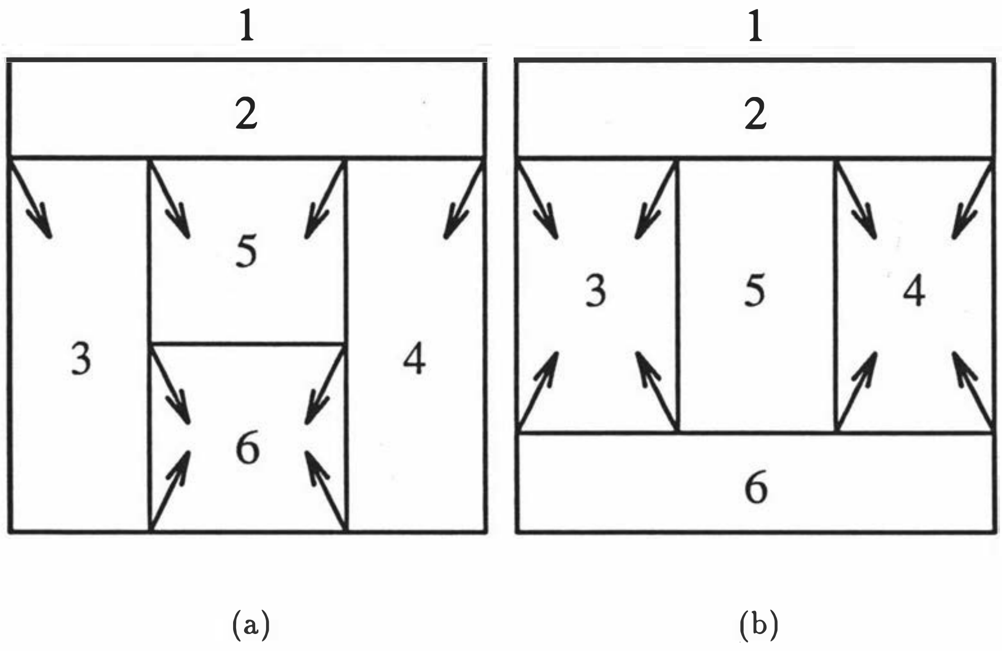


Figure 4.10: Initial layouts for the Regular Octahedron

Note that in Figure 4.10(a), the placement directions on the wall between facilities 5 and 6 could be directed into facility 5 or 6, as long as both placement directions are directed into the same facility, similarly for Figure 4.10(b) for the placement directions between facilities 3 and 5, and facilities 4 and 5. Which initial configuration and set of placement directions to use would be a matter of preference, and indeed all possibilities could be examined.

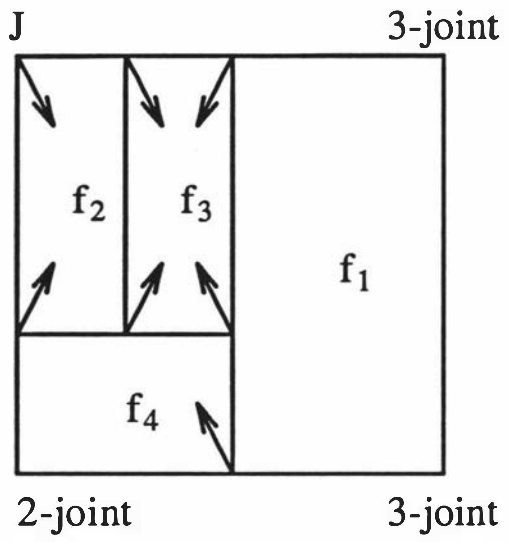


Figure 4.11: Extended Deltahedron Placement Operation PO3

We now introduce placement operations PO3 and PO4. The form of PO3 is shown in Figure 4.11, and is analogous to PO1, where we do not worsen the shape of the placement host, and we place at J which is adjacent to a 2-joint. Again the placement directions between f_2 and f_3 could be on either side of the wall. Placement operation PO4 is shown in Figure 4.12, and we see here that the placement directions for C_1 and C_2 are linked, in that they must both be on the same side of the wall. This operation is akin to PO2, where there is no 2-joint adjacent to J.

Previously, any facility in the Deltahedron Layout Algorithm would become at worst T-shaped, the Extended Deltahedron Layout Algorithm cannot maintain this guarantee, relaxing the worst case room shape to that of an X; the worst possible for dimensionalisable layouts. Therefore we now allow X, S and Y shaped facilities under this construction. Obviously the dimensionalisability of Extended Deltahedron Layouts, is retained by the consistent use of placement directions.

The final issue which we need to discuss here, is the dimensioning of any S-, Y- or X-shaped facilities. With reference to Figure 4.13, we obtain the dimensions provided in Equations 4.6 - 4.8. The values for H_i can be derived as a function of a_{f_i}, and H_1 only, if desired. Furthermore Equations 4.6 - 4.8 provide the most general form of the dimensioning of irregular facilities. Dimensioning for a T-shaped facility for example, could be provided by setting $a_{f_4} = 0$ and $a_{f_5} = 0$. This

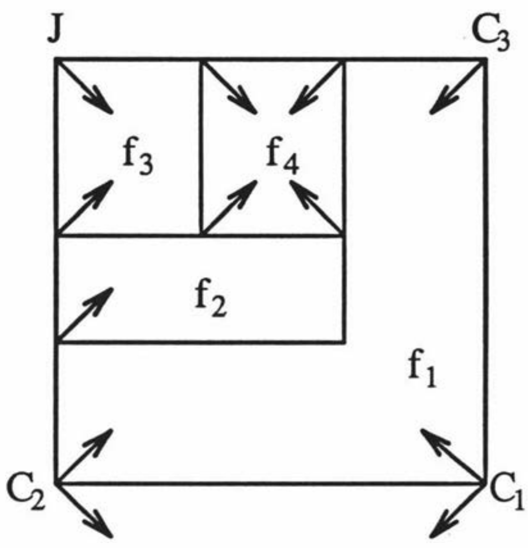


Figure 4.12: Extended Deltahedron Placement Operation PO4

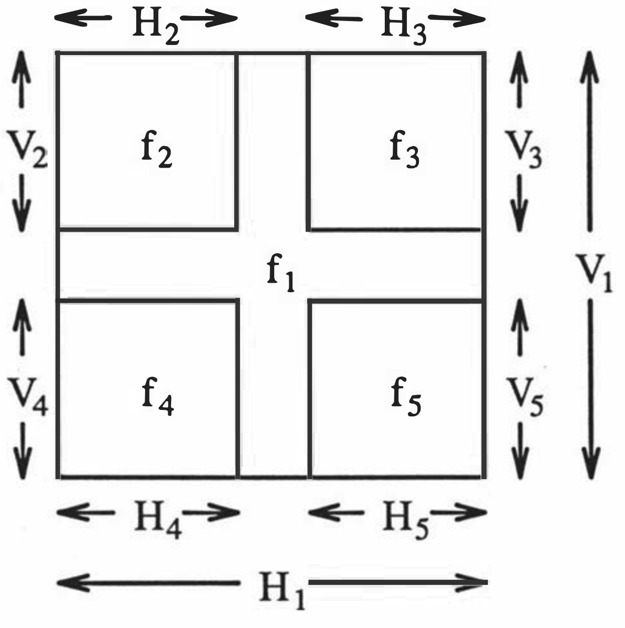


Figure 4.13: Dimensioning of X-shaped facilities

will not produce the same dimensioning of Equations 4.2 and 4.3, but is still a valid dimensioning.

$$V_2, V_3 = \frac{1}{2}\left(\frac{0.5a_{f_1} + 2a_{f_2} + 2a_{f_3}}{a_{f_1} + a_{f_2} + a_{f_3} + a_{f_4} + a_{f_5}}\right)V_1 \tag{4.6}$$

$$V_4, V_5 = \frac{1}{2}\left(\frac{0.5a_{f_1} + 2a_{f_4} + 2a_{f_5}}{a_{f_1} + a_{f_2} + a_{f_3} + a_{f_4} + a_{f_5}}\right)V_1 \tag{4.7}$$

$$H_i = \frac{a_{f_i}}{V_i} \quad i = 2, 3, 4, 5 \tag{4.8}$$

4.2 Grid Approaches

The first integrated attempt at the development of an algorithm which would dualise any given MPG was proposed by Hassan and Hogg [52]. Their method used the theme of the classical ALDEP and SHAPE algorithms, within a graph theoretic framework. They outlined three phases for constructing a layout: the order of selecting facilities to enter the layout, the placement of the entering facility, and finally the construction of facility shapes, adhering to specified area and regularity requirements.

We now discuss the method proposed by Hassan and Hogg in more detail. The procedure considers facilities one at a time for both construction and placement within the layout. A rectangular grid is constructed of unit squares having total area equal to the sum of the areas of all facilities. The first facility is placed in the northwest corner, and extends downwards. Subsequent facilities are placed adjacent to the previously placed facility, and at least one other placed facility, starting from where the previous facility ended. When the layout boundary is reached the direction of expansion is reversed, following the classical ALDEP approach.

The entering facility is determined by establishing a set G_i for all unplaced facilities i; G_i is the set of all placed facilities that have adjacency in the MPG with the facility i. If $|G_i| \geq 2$, denote two of its elements as g_1 and g_2, which direct the placement of facility i in the layout, with one of g_1 or g_2 being the last facility that was placed. There may be more than one i which satisfies these requirements; the tie-breaking rule used by Hassan and Hogg chooses the facility to enter as the one with the greatest number of adjacencies in the current partial layout; if the tie remains unsolved the candidate facility with the largest area is then chosen.

Upon designating the entering facility, a set of candidate squares must be developed, within which the entering facility can be placed in order to maintain feasibility of the adjacencies in the MPG in the current layout, without precluding subsequent facility placements. These candidate squares are defined by determining starting(S) and ending(E), rows(R) and columns(C) for the candidate squares. $SR = R(g_1)+\lambda$, where $R(g_1)$ is the last boundary row of g_1 reached in the direction of expansion, and $\lambda = 1$ if the direction of expansion is downwards, and -1 otherwise. $ER = R(g_2)-\lambda\alpha(g_2)$, where $\alpha(g_2)+1$ is the number of remaining adjacencies to be satisfied for g_2. $SC = min_{g\in G_i}C_2(g)+1$, where $C_2(g)$ is the rightmost column of facility g. Finally $EC = SC+\delta c$, where δc is the number of columns determined proportional to the area of the entering facility and the number of candidate rows obtained. From these candidate squares a set of size $|a_i|$ is selected by considering the minimum cost of assigning $|a_i|$ of these candidate squares to define facility i. Following this assignment, p and q which are the 3-joints at g_1 and g_2 are denoted, assigning one as q^* to ensure vertical direction of expansion of the layout, as the next facility to enter will have q^* as one of its 3-joints. The authors provide an example of the algorithm working for a seven facility problem proposed by Moore [88]. We will return to the theoretical motivation of this approach in Section 4.4.

Al-Hakim [4] provided the first cracks in the method proposed by Hassan and Hogg by producing a counter example. We provide another example here, to demonstrate the method of Hassan and Hogg, and to show the flaws in it. Consider the six facility problem of Figure 4.14, with area specifications given by Table 4.3. Table 4.4 give the process for inserting the first three facilities, while Figure 4.15 provides the layout after these first three placements. At this point we see the flaw of this approach; facility 5 cannot obtain its adjacency with facility 2 from this partial layout. Therefore the approach of Hassan and Hogg fails to dualise the MPG in this case.

Al-Hakim demonstrates by a series of examples how the method will fail to place all facilities in the presence of so-called nested facilities, which are essentially facilities that lie within a separating triangle. Notice in the example of Figure 4.14 facilities 5 and 6 laid within the separating triangle (2, 3, 4). By identifying sets of nested facilities Al-Hakim presents a remedy to this problem, by assuming blocks of nested facilities are treated as a single facility initially, and by then considering each block of nested facilities in turn, and laying out the facilities in each block

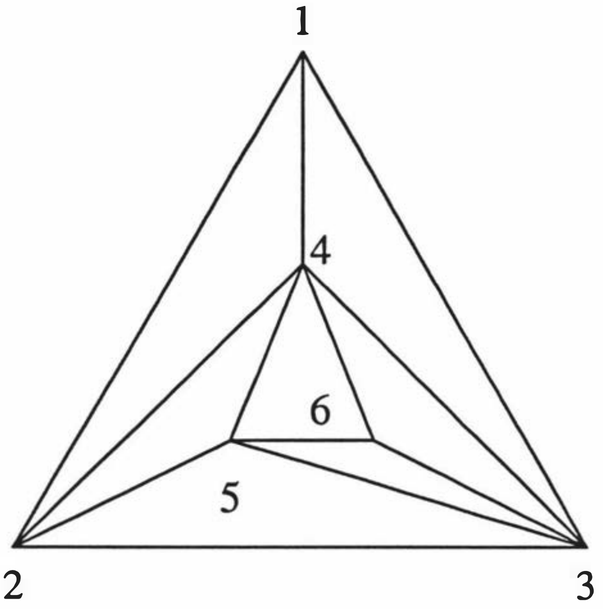


Figure 4.14: Illustrative Example of the method of Hassan and Hogg

Facility	1	2	3	4	5	6
Area	-	25	23	18	19	25

Table 4.3: Area Specifications for the Hassan and Hogg Illustrative Example

i	G_i	λ	g_1	g_2	p	q	q^*
2	1	1	1	1	a	b	-
3	1,2	-1	1	2	c	d	b
4	1,2,3	-1	1	3	f	e	d

Table 4.4: Insertion Processes for the Hassan and Hogg Illustrative Example

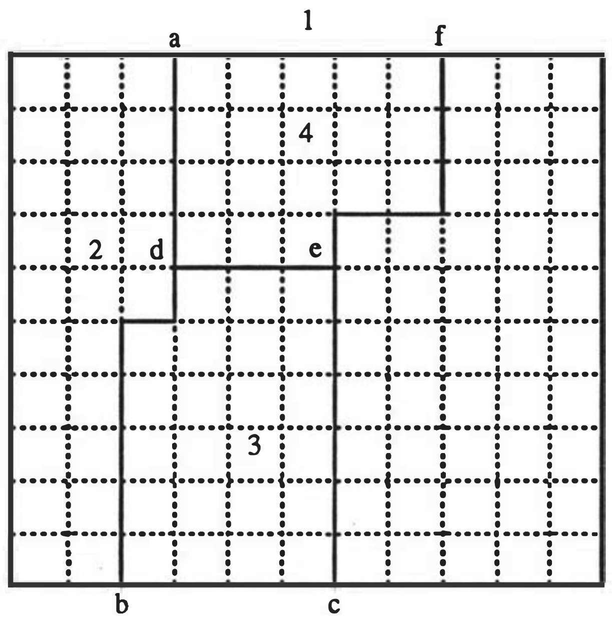


Figure 4.15: Partial Hassan and Hogg Solution

by the same method, as was employed during the initial layout. Hassan [50] defended the original paper in the light of Al-Hakim's comments in [4], using the argument that Al-Hakim violated the two assumptions of the original paper [52], namely that the umbrella effect described in [41] was not present, and that the facility areas were *compatible*. The existence of the umbrella effect is easily determined, but the compatible area requirement is not well defined in the original paper and although Hassan [50] provides examples of compatible areas, there is still no guarantee of existence of the layout; indeed, the first counterexample proposed by Al-Hakim consisted of only five facilities, which didn't violate these assumptions. Hassan [50] eventually acknowledged that the method is best suited to uncomplicated situations, showing further that the method is very sensitive to complicated substructures within the MPG. The dimensions of the layout perimeter can play a large part in the applicability of the method, as can even small deviations from the assumptions in the original paper. Hassan shows examples that will only work under manual implementation, as they cannot work under the layout process described in the original paper.

Overcoming the difficulties inherent in the counterexamples proposed by Al-Hakim turns out to be very simple. The basic problem with the original method is that, in the presence of separating triangles, there can exist more than one vertex adjacent to two already placed facilities, one of which is the last facility placed. By choosing the facility to enter which actually *shares a face* with the two facilities to which it is adjacent, we can remove this problem. In Section 4.4, we will develop this further, as well as producing a method based on Hassan and Hogg's original motivation, but without the clumsiness exhibited by both the facility selection and placement phases. Al-Hakim's counterexamples show the flaws in the selection phase, as making an alternative choice of facility to enter the layout at crucial points can determine whether or not the layout can to be completed. In Section 4.4 we will show that these crucial decisions can be easily made algorithmically, by considering the nested facilities discussed by Al-Hakim. The placement phase in the original method is also awkward. By having an inflexible grid of unit squares it makes it very difficult to assign small facilities, as there is little flexibility allowed. Furthermore, the designation of the squares for facility placement is made by considering the *cost* of assigning a facility to a set of squares, which is part of the classical, rather than Graph Theoretic model philosophy. The authors give no indication of what action might be taken if the set of squares assigned to a facility has *holes* in it or is disconnected. For example, in Figure 4.15, facility 2 could be assigned any 25 (if all costs are the same) of the 30 squares in the first 3 columns of the grid. If all three squares of row 3 say were not assigned facility 2 would become disconnected. Although it could be assumed that these assignments would be disallowed, there is no comment made by the authors that this could even occur. The Hassan and Hogg method is fundamentally therefore a classical approach, which attempts to use a given MPG as a guide to creating the layout but with no guarantee of generating a layout preserving the adjacencies specified by the given MPG.

We made an attempt to improve the approach of Hassan and Hogg; this new approach proved theoretically correct, but was hindered by an exploding complexity under implementation. The method relied upon making the correct choice of the entering facility (as outlined above), so that a feasible layout would be constructed; however, the main thrust of the placement routine was to allow grid squares to be split in half, by row or by column, in order to accommodate tricky placements. Therefore, in placing a facility, the set of candidate squares was redefined as all unassigned squares, except those which bordered facilities the entering facility was not adjacent to. From this set we assigned squares to ensure the facility was as compact as possible. As a result, there became no possibility of not acquiring all

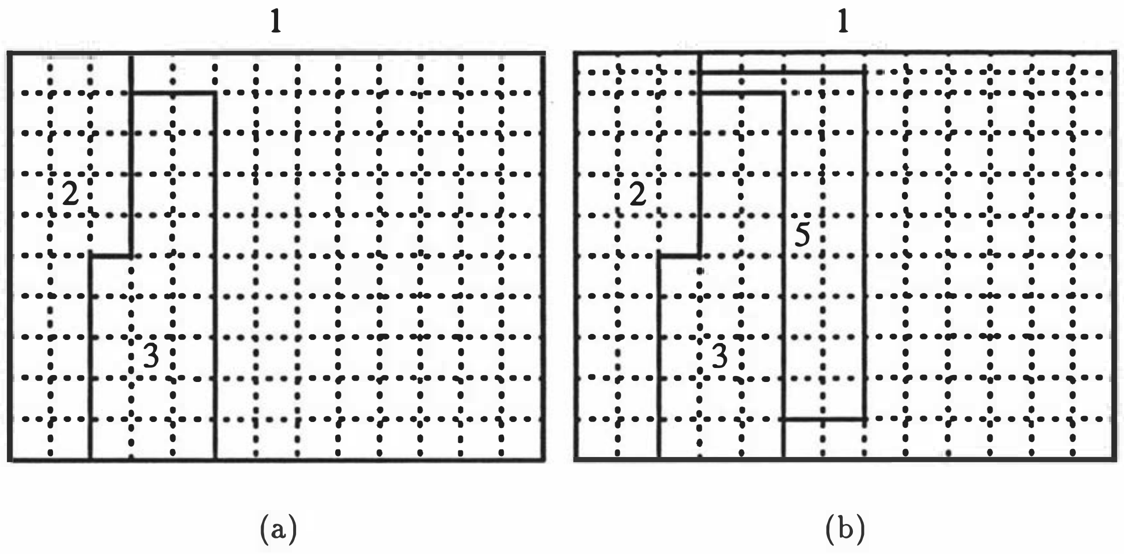


Figure 4.16: Splitting the first row to accommodate facility 2 adjacencies

the adjacencies of the MPG, as if infeasibility was imminent, a row or column of the grid could have each of its squares split in half in order to allow more flexibility for subsequent placements. This posed no problem for already placed facilities, as the assignment for these facilities remained the same. This process is shown by Figure 4.16, based on the example of Figure 4.14 and Table 4.3, where we are in danger of not being able to complete the layout, as only one square adjacent to facility 2 remains, with two adjacencies yet to be met. This is overcome by splitting row 1, as in Figure 4.16(b), allowing room for facilities 4 and 5. Further note that in order to place facilities 4 and 6 we will need to split the final row, to meet the adjacencies with facility 3.

This method is theoretically valid, as we can split grid squares *ad nauseum*, however a consequence of allowing grid splitting is an increased number of grid squares. This led to severe implementation problems when it came to examining squares for potential assignment, as it took an inordinate amount of time to search through a very much enlarged grid set. This was the first attempt at developing a method which would circumvent the problems exhibited by Hassan and Hogg's approach, but we will see in Section 4.4 that it is superseded by a more relaxed version of this type approach, which does not require the restrictive grid structure. The one benefit of an unsplit grid structure is a guaranteed minimum facility width, which cannot easily be incorporated within more generic approaches. We will see in later chapters that, while we cannot guarantee a minimum facility width under more flexible approaches, we can in general develop acceptable solutions which overcome this issue.

4.3 The Contraction Algorithm

In this section we present the first algorithm which was guaranteed to generate a layout to an arbitrary MPG. Rinsma, Giffin and Robinson [96] presented what we term the Contraction Algorithm, for determining a facility layout. This method is important theoretically, not only as the existence of the layout in the arbitrary case is proved, but also as it forms a building block via which we can develop other layout algorithms, by using the motivational ideas and, especially, the placement routine – the Orthogonal Division Algorithm, which we will discuss later.

The Contraction Algorithm sequentially reduces the given layout to a more manageable structure via a set of contraction operations, lays out this reduced problem, and then reverses the contraction process to place the remaining facilities. We will discuss this method in two parts; the first examines the contraction process, the second the placement algorithm.

4.3.1 The Contraction Process

Rinsma *et al.* reduced the given MPG to a standard form, in which every vertex, except the exterior, became an element of distance class D_1. This was achieved by performing a sequence of so-called *contraction operations.* Successive contractions involving the vertices in the innermost distance class of the partially transformed MPG provided the mechanism whereby the standard form could be obtained. Rinsma *et al.* define the contraction operation in the following manner: for a given edge $e = (i,j)$ in G, let the set of vertices adjacent to i and j be $A(i)$ and $A(j)$, respectively. Consider the operation on e whereby vertex j is removed, and all edges previously incident upon j are now made incident upon i, with all consequent loops and multiple edges coalesced. This operation is called the *contraction of j to i*, and the graph obtained from this operation is denoted by $G\backslash ij$. The set of vertices adjacent to the new vertex i in $G\backslash ij$ is $A(i) \cup A(j) - \{i,j\}$. Rinsma *et al.* further defined the notion of *weighted contraction*, where a_i is the vertex weight of i, and we augment a_i by a_j. In the layout this would correspond to the merging of the areas of facilities i and j. This process is shown in Algorithm 4.4, where $S(i)$ is the set of vertices directly contracted into vertex i.

Algorithm 4.4 *The Contraction Operation($j \rightarrow i$)*

```
    Input: $G$
    Output: $G\backslash ij$
    $a_i \leftarrow a_i + a_j$
    $S(i) \leftarrow S(i) \cup j$
    $V_G \leftarrow V_G \backslash j$
    for $k = 1$ to $n$ do
        if $((j,k) \in E_G$ and $(i,k) \notin E_G)$ then
            $E_G \leftarrow E_G \cup (i,k)$
        end
        $E_G \leftarrow E_G \backslash (j,k)$
    end

end
```

The contraction process works by considering blocks, which are maximal outerplanar subgraphs of the MPG, of the innermost distance class. The blocks are contracted to a single vertex, by sequentially contracting the vertices of degree two in the maximal outerplanar subgraph of the block of vertices in the innermost distance class (and their edges), to one of the adjacent vertices in the subgraph. This is achieved using a technique from Rinsma [94], where a labelling is forced upon the subgraph, by sequentially labelling vertices adjacent to two other labelled vertices, starting with an initial labelling of an arbitrarily chosen triangle of the subgraph. If we call this subgraph H, then, by successively applying $H\backslash ij$ (where i is the vertex adjacent to j of highest label, initialising with j as the highest labelled vertex), a tree will be obtained which can be successively contracted to a single vertex by contracting pendant vertices of the tree. Once the distance class, D_k say, has been contracted to a single vertex, that vertex is contracted into a vertex of D_{k-1}. The contraction process continues until the resulting transformed graph has all vertices in D_1 (and the exterior vertex in D_0). Rinsma *et al.* state, but do not prove, that no vertex is contracted into more than twice.

Algorithm 4.5 *The Contraction Algorithm*

```
    Input: MPG $G$, and facility area specifications
    Output: Transformed MPG on $m$ vertices of which $m-1$ are in $D_1$
    for $i = max$ to 2 do
        for each maximal outerplanar subgraph $H$ of $D_i$ do
            perform Contraction Operation($j \rightarrow k$), where $j$ is the highest labelled vertex of $H$, and $k$ the highest labelled vertex adjacent to $j$ until $H$ is a single vertex following the labelling of Rinsma [94]
        end
        for each tree subgraph $R$ of $D_i$ do
            perform Contraction Operation($j \rightarrow k$) until $R$ is a single vertex by contracting pendant vertex $j$ of $R$ to adjacent vertex $k$ of $R$
        end
        for each vertex $j$ of $D_i$ do
            perform Contraction Operation($j \rightarrow k$), where $(j,k) \in E_G$, $k \in D_{i-1}$, and $|S(k)| < 2$
        end
    end
end
```

4.3.2 The Orthogonal Division Algorithm

The second phase of the Contraction Algorithm focuses on the construction of the layout, firstly by constructing an initial layout using techniques of either the Deltahedron Layout Algorithm or Rinsma [94] which is described in Section 4.7.1, as in the original paper. We will prove in Section 4.7.1 that all MPGs with $n-1$ facilities in D_1 are Deltahedron-generateable.

Following the construction of this initial layout, a *reverse contraction* procedure is performed, in order to generate a layout which is dual to the original MPG. This is done by considering the reverse order in which facilities were contracted during the contraction process. This forms the basis of the Orthogonal Division Algorithm (ODA), which is conceptually simple but very hard to implement.

The ODA considers a vertex w which was contracted into a vertex x. Facility w is to be *placed* within facility x, so that w attains, and x maintains, their respective adjacencies and, following the nomenclature of the Deltahedron Layout Algorithm, x is the placement host of w. The authors define $N(x)$ to be the *neighbours* of vertex x, *i.e.* the facilities adjacent to x, arranged in a *cyclic ordering* around x. The neighbours of x are assumed to be $\{N_1, \ldots, N_t\}$, and the neighbours of w, $\{N_1, \ldots, N_s\}$, where t is the cardinality of $N(x)$, and similarly for s and $N(w)$.

Consider now the facility x in the currently constructed layout. Let α and β be two points, which are not 2- or 3-joints on the common wall between x and N_1, and x and N_s respectively. The portion of wall of x between the two points α and β, which is common successively to N_1, N_2, up to N_s, is denoted by $b(x; \alpha, \beta)$. Let L be a rectilinear path contained wholly within x, which connects α and β. The line L is determined as having a minimum number of corners, and being a minimum distance δ from any wall of x. L divides facility x into two rectanguloids, one labelled q, which has area a_q, and neighbours $\{N_1, \ldots, N_s\}$, which are facility w's adjacencies, and the other, p, having area a_p, and neighbours $\{N_{s+1}, \ldots, N_t, N_1\}$, being facility x's adjacencies. This is more clearly shown by Figure 4.17. If $a_q = a_w$, then $a_p = a_x$, and we can relabel the two rectanguloids such that p is now facility x, and q is now facility w. Note that this is very unlikely to occur. Otherwise $a_q > a_w$ or $a_q < a_w$, and we must perturb the line L until q has area a_w and p has area $a_x - a_w$.

If $a_w < a_q$ then we define a very small facility labelled $q_1(\varepsilon)$, determined by $b(x; \alpha, \beta)$, and a second rectilinear line $L_q(\varepsilon)$, a distance ε away from the wall joining α and β inside q. If h and v are the minimum distances between any two horizontal and vertical walls of x respectively, then define $\varepsilon = \frac{1}{3}\min(h, v, \delta)$. The remainder of q is labelled $q_2(\varepsilon)$. This is shown in Figure 4.18.

If $a_w = a_{q_1(\varepsilon)}$, then relabel $q_1(\varepsilon)$ as w. If $a_w < a_{q_1(\varepsilon)}$, then perform an iterative bisection search on ε to obtain facility w. Otherwise, successively add rectanguloids from $q_2(\varepsilon)$ to $q_1(\varepsilon)$, until $a_{q_1(\varepsilon)} = a_w$. We now describe how this rectanguloid transfer can be achieved. We consider sets of consecutive walls of q_2, $\{c, d, e, f, g\}$ as shown in Figure 4.19, with at least one of the corner points at the intersections of $\{d, e\}$, $\{e, f\}$, or $\{f, g\}$ being a corner of the line $L_q(\varepsilon)$. In Figure 4.19, $||d|| > ||f||$, however $||d||$ can equal $||f||$, forcing the lines c and g to be collinear, or even, $||d|| < ||f||$. We will discuss the case given in Figure 4.19, where $||d|| > ||f||$, but the other two cases can be developed similarly. Extend g by a line j to a point d_j on d, which divides d into two segments d^*, and d'. This divides $q_2(\varepsilon)$ into

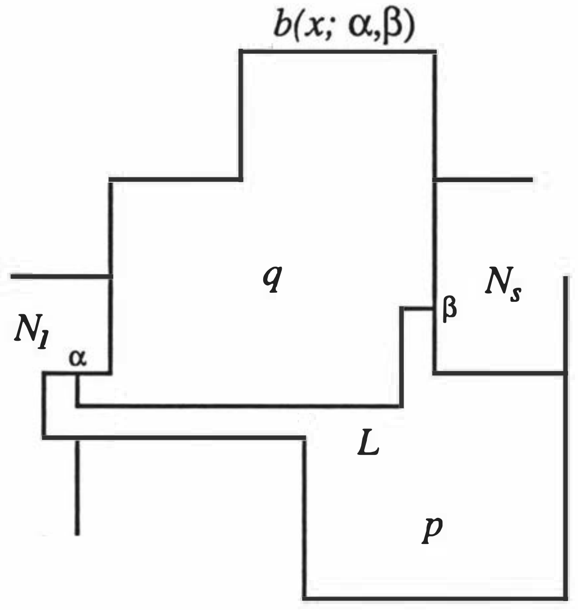


Figure 4.17: Initial Division of Facility x in the ODA

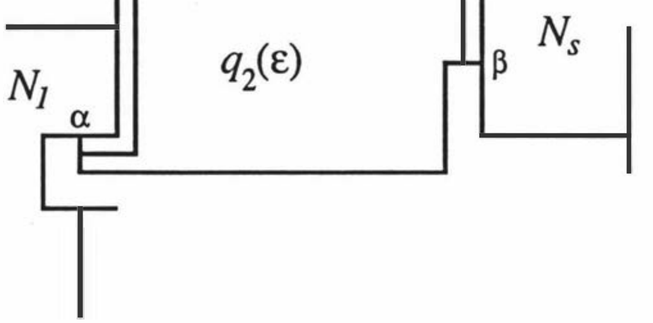


Figure 4.18: Division of Rectanguloid q

rectanguloid $q_2'(\varepsilon)$, and a rectangle q_3, which has walls j, d', e and f, as shown in Figure 4.19.

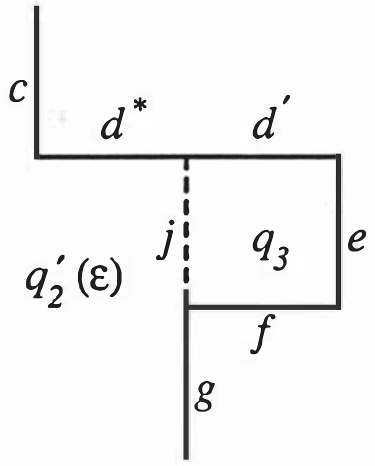


Figure 4.19: Identification of a Rectangle to Transfer

Now if $a_w \geq a_{q_1(\varepsilon)} + a_{q_3}$, augment $q_1(\varepsilon)$ by q_3, and update the wall description of $q_1(\varepsilon)$ and $q_2(\varepsilon)$. If $a_w < a_{q_1(\varepsilon)} + a_{q_3}$, then only a portion of q_3 of area $a_w - a_{q_1(\varepsilon)}$ is required. This is easily achieved by dividing d' into two sections, d_1' and d_2' (where $||d_2'|| = \frac{a_w - a_{q_1(\varepsilon)}}{||e||}$) Creating a line j' parallel to j, and orthogonal to the point at which d_1' and d_2' meet completes the division. The rectangle defined by the walls d_2', e, f_2, and j' is labelled q_4, which is added to $q_1(\varepsilon)$. This is shown by Figure 4.20.

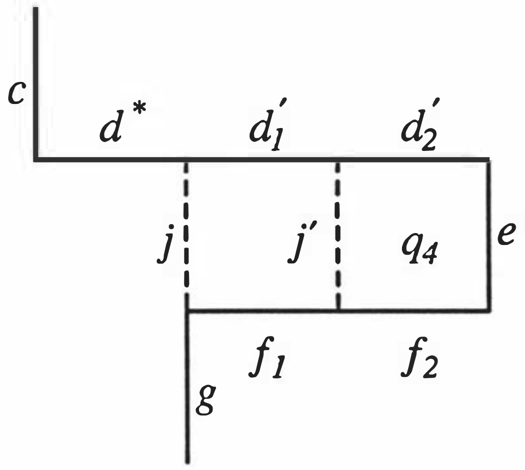


Figure 4.20: Transferring only a part of the identified rectangle

From this point Rinsma *et al.* go on to describe the case when $a_w > a_q$. It is, however, rather clumsily devised, and involves performing similar steps as for the case when $a_w < a_q$, but with p and q, and x and w interchanged throughout, then decreasing a_p by adding rectanguloids to q which are incident upon L. It makes more sense to simply reverse the entire process described above, by constructing a rectilinear line $L_p(\varepsilon)$ from α to β a distance ε from the line $b(w; \alpha, \beta)$, to create $p_1(\varepsilon)$ and then adding rectanguloids in the same manner as also described above. We will discuss further in Section 4.3.5 the changes that were made in the implementation of the ODA in order to simplify the rather complex description of Rinsma *et al.* Algorithm 4.6 describes the complete ODA process for placing facility w within facility x, where Y is the set of walls of facility x and we swap the labels of q and p, and x and w, if $a_q > a_w$.

Algorithm 4.6 *The Orthogonal Division Algorithm*$(x \to (x, w))$

```
Input: $A(w)$, $A(x)$, the rectanguloid facility $x$
Output: Facilities $x$ and $w$
determine $\alpha$ and $\beta$, hence determining $b(x;\alpha,\beta)$, and $b(w;\alpha,\beta)$
determine the line $L$ a distance $\delta$ from any wall with a minimum number of
corners, label the two rectanguloids $p$ and $q$, as in Figure 4.17
if $(a_q = a_w)$ then
    $w \leftarrow q$
    $x \leftarrow p$
end
else
    if $(a_w > a_q)$ then
        $q \leftrightarrow p$
        $x \leftrightarrow w$
    end
    determine $h$ and $v$
    $\varepsilon \leftarrow \frac{1}{3}min(h, v, \delta)$
    determine the line $L_q(\varepsilon)$ to give rectanguloids $q_1(\varepsilon)$ and $q_2(\varepsilon)$, as in Fig-
    ure 4.18
    if $(a_{q_1(\varepsilon)} = a_w)$ then
        $w \leftarrow q_1(\varepsilon)$
        $x \leftarrow q_2(\varepsilon) + p$
    end
    if $(a_{q_1(\varepsilon)} > a_w)$ then
```

```
        perform iterative bisection search on $\varepsilon$, until $a_{q_1(\varepsilon)} = a_w$
        $w \leftarrow q_1(\varepsilon)$
        $x \leftarrow q_2(\varepsilon) + p$
    end
    while $(aq_1(\varepsilon) < a_w)$ do
        determine set of walls $c, d, e, f, g$ as defined in Figure 4.19, with
        $||d|| \geq ||f||$
        determine the line $j$, and the resulting dissection of $d$, into $d^*$ and
        $d'$
        $q_3 \leftarrow$ the rectangle enclosed by $j$,$d'$,$e$, and $f$
        $q_2'(\varepsilon) \leftarrow q_2(\varepsilon) - q_3$
        if $(a_w > a_{q_1(\varepsilon)} + a_{q_3})$ then
            if $(||d|| = ||f||)$ then
                $g \leftarrow c + j + g$
                $Y \leftarrow Y - \{e, f\}$
            end
            else
                $g \leftarrow g + j$
                $d \leftarrow d^*$
                $Y \leftarrow Y - \{c, d, e, f\}$
            end
            $w \leftarrow q_1(\varepsilon) + q_3$
            $x \leftarrow q_2'(\varepsilon) + p$
        end
        else
            $\rho \leftarrow \frac{a_w - a_{q_1(\varepsilon)}}{||e||}$
            construct a wall $j'$ a distance $\rho$ from, and parallel to $e$
            partition $d'$ into $d_1'$ and $d_2'$, with $d_2'$ incident on $e$
            partition $f$ into $f_1$ and $f_2$, with $f_2$ incident on $e$
            $q_4 \leftarrow$ the rectangle enclosed by $j'$,$d_2'$,$e$, and $f_2$
            if $(||d|| = ||f||)$ then
                $d \leftarrow d_1'$
            end
            else
```

```
                $d \leftarrow d^* + d_1'$
            **end**
            $f \leftarrow f_1$
            $Y \leftarrow Y \cup \{j'\} - \{e\}$
            $w \leftarrow q_1(\varepsilon) + q_4$
            $x \leftarrow p + q_2'(\varepsilon) + q_3 - q_4$
        **end**
    **end**
  **end**

**end**
```

An example of rectangle addition is given in Figure 4.21, where the rectanguloids were added in numerical order to $q_1(\varepsilon)$. Note that the addition of rectangle 3 has the two lines c and e collinear. Further notice that all the rectangles all have the edges c, d, e, f, g adjacent on $L_q(\varepsilon)$.

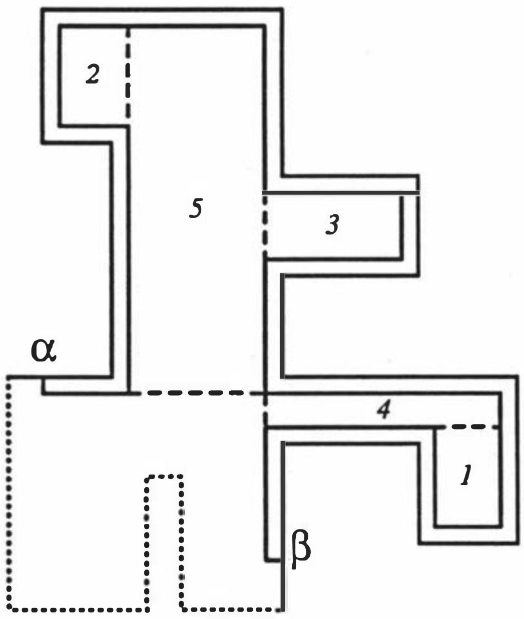


Figure 4.21: A division of x via the Orthogonal Division Algorithm

4.3.3 An Illustrative Example

We now proceed to provide a full example of the Contraction Algorithm by considering the MPG of Figure 4.22 and area specifications given in Table 4.5. Firstly we determine the distance class sets $D_1 = \{2,3,4\}$, $D_2 = \{5,6,7,8\}$, and $D_3 = \{9,10\}$. The contraction of the tree of the vertices of D_3 involves contracting facility 10 to facility 9, to obtain the single vertex, and then contracting vertex 9 to vertex 5, resulting in the MPG shown in Figure 4.23. Note that vertex 9 could have been contracted to any of vertices 5,6,7 or 8.

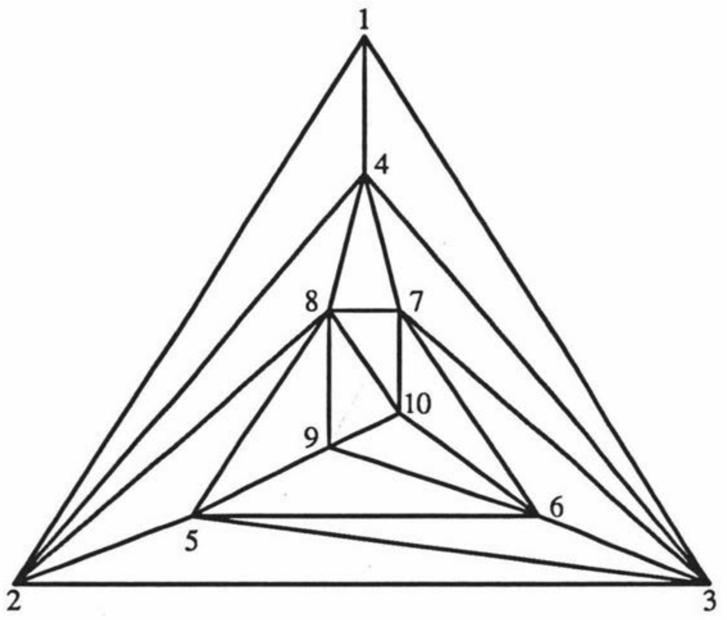


Figure 4.22: Example Problem to illustrate the Contraction Algorithm

Facility	1	2	3	4	5	6	7	8	9	10
Area	-	20	15	15	10	20	10	20	25	10

Table 4.5: Area specifications for the Contraction Algorithm Illustrative Example

Following this we contract vertices 6 and 8, since they are of degree 2 in the maximal outerplanar component consisting of the vertices in D_2, into vertex 7, which is then contracted into vertex 5 to obtain the single vertex in D_2. Vertex 5 is then contracted into vertex 4 of D_1, providing the MPG with all vertices, except the exterior, in D_1, being the tetrahedron. The layout for this transformed MPG is easily constructed by the Deltahedron Layout Algorithm, to give the layout of Figure 4.24. Note that facility 4 has area equal to that of itself plus any facilities which were contracted, directly or indirectly, into vertex 4.

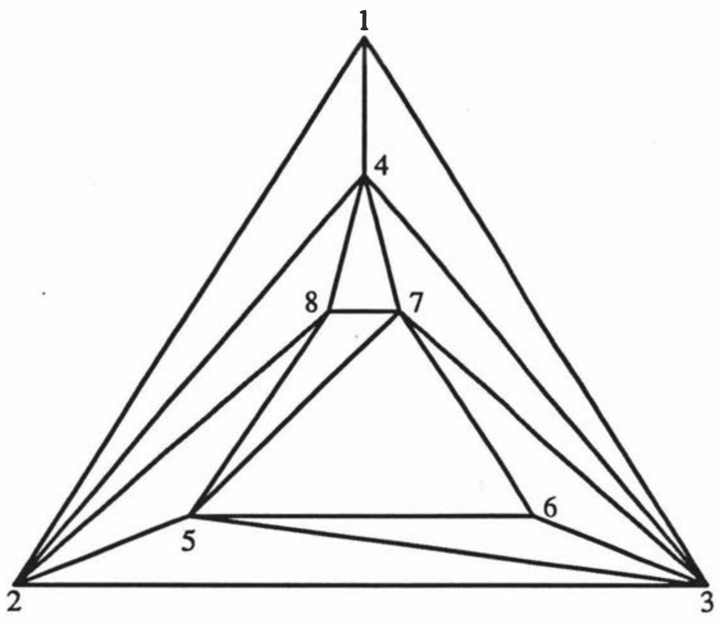


Figure 4.23: Transformed MPG after the contraction of D_3

Figure 4.24: Layout of the transformed MPG

The first placement is that of facility 5 into facility 4, followed by the placements of facility 7 into facility 5, and also facility 6 into facility 5, to provide the partially constructed layout of Figure 4.25. Note that the two different layouts in Figure 4.25 correspond to different values used in the ODA for α and β when

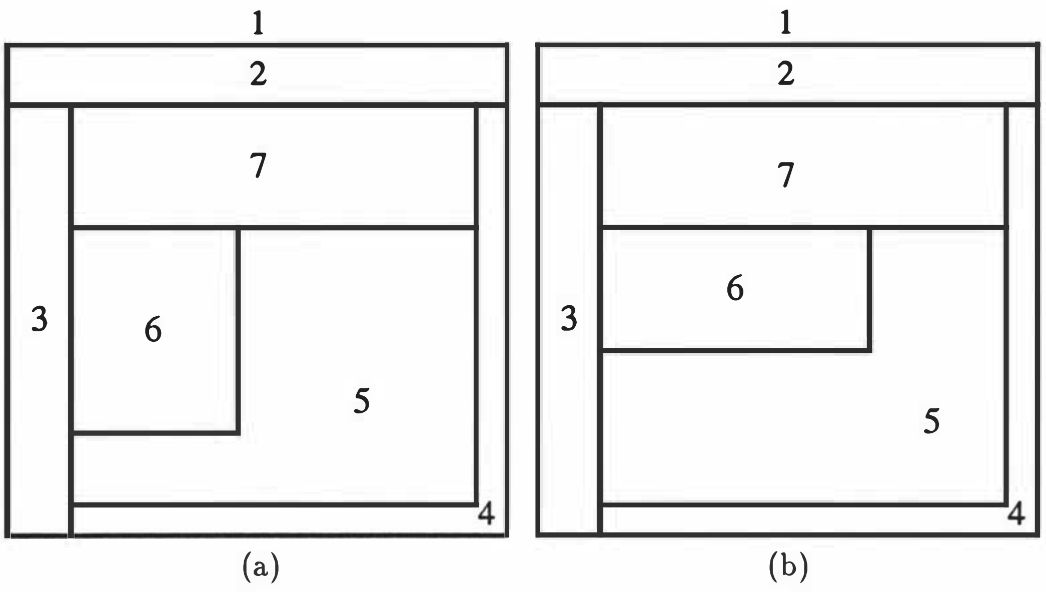


Figure 4.25: Partial construction of the layout via reverse contraction of facilities 5, 6 and 7, with different α and β values for the placement of facility 6

placing facility 6. Continuing from Figure 4.25(a), the remaining three facilities are placed in the reverse order to which they were contracted, to obtain the layout shown in Figure 4.26.

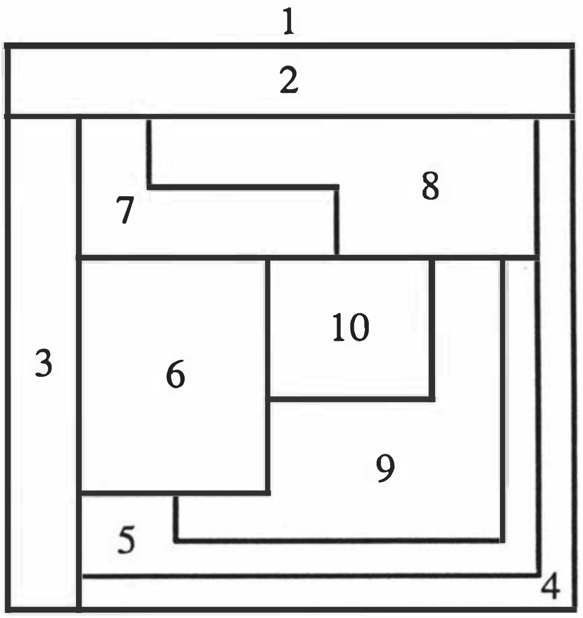


Figure 4.26: Final Layout of the example problem

Note that the placement of facility 6 impacts quite severely on the final layout, as facility 8 cannot be placed as a rectangle, and further, the size of facility 9 when it is placed forces facility 5 to become quite irregular. Contraction of facility 6 into facility 7 would have alleviated the problem of irregularity in this layout and we will see, in Chapter 9, ways in which we might choose more carefully the contraction ordering, in order to produce more regular layouts.

4.3.4 A Pathological Counter-Example

In this section we examine a counter-example to the original Contraction Algorithm as provided by Rinsma *et al.* The MPG for this problem is given in Figure 4.27,

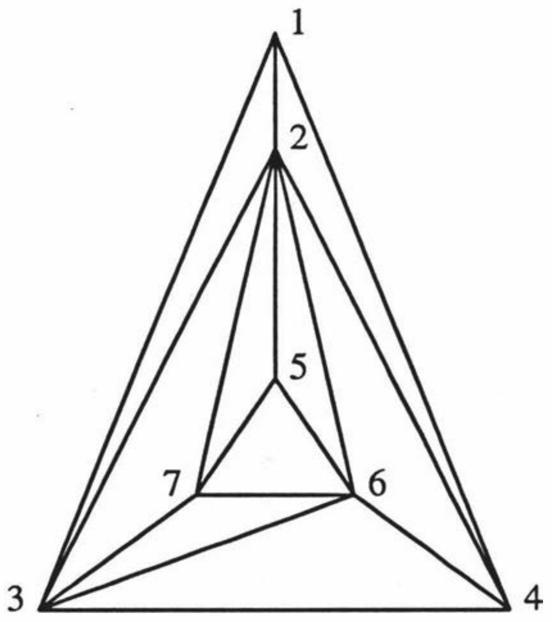


Figure 4.27: MPG to exhibit pathological problem

where facility 1 is the exterior, and the distance class sets are $D_0 = \{1\}$, $D_1 = \{2, 3, 4\}$ and $D_2 = \{5, 6, 7\}$. Now we consider the contraction of those vertices in D_2. They form a maximal outerplanar component, and so we enforce the labelling as given by Rinsma *et al.* Then the precursor of 7 is 6, and under the contraction process, we are required to contract vertex 7 into vertex 6. This contraction gives the MPG shown in Figure 4.28.

In order to maintain maximal planarity, we are required to add in the edge $(3, 5)$. Vertex 5 does not form part of the contraction operation, and therefore

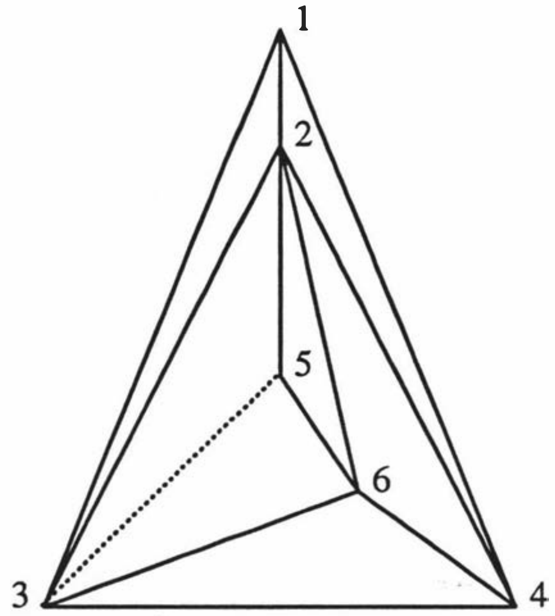


Figure 4.28: MPG after the contraction of facility 7 into facility 6

should not have an additional adjacency from this operation. The ramifications of allowing the edge $(3, 5)$, is that when using the ODA we will not be able to place facility 7 totally within facility 6 as the ODA requires. We will need to delete the adjacency $(3, 5)$, which will require part of facility 7 being placed partly in either or both of facilities 3 and 5.

The method proposed by Rinsma *et al.* requires only a minor correction to overcome this difficulty. Rather than enforce the labelling outlined by Rinsma *et al.* onto the maximal outerplanar components, we simply contract subsequent vertices of the maximal outerplanar component, by choosing the next vertex to be contracted as a vertex of degree 2 with other vertices in the same distance class, such that the vertex which is contracted into, the vertex being contracted and a vertex from the next outer distance class do not create a separating triangle; this is proved in Theorem 4.3.1. The contraction processes for trees and single vertices remain valid, however. This may be seen more easily in the example above, where the separating triangle $(2, 6, 7)$ existed, and when we tried to contract vertex 7 into vertex 6, we were required to coalesce $(2, 7)$ with $(2, 6)$, yet 5 *lay between* them.

Theorem 4.3.1 *Contraction of a vertex of a maximal outerplanar component is valid if and only if the vertex has degree 2 within its distance class, and the vertex* *which will be contracted into, the vertex being contracted and a vertex from the next outer distance class do not create a separating triangle.*

Proof: Rinsma [94] has shown that there will always exist at least two vertices of degree 2 in a maximal outerplanar component. There is no problem with separating triangles between three vertices of the same distance class, since there must be at least one vertex *inside* this separating triangle, of distance class one more than the three vertices that make up the separating triangle, and hence we would not have completed contracting the previous distance class.

Let us assume that there is no candidate vertex which we can contract. Because we know there must be vertices of degree 2, they all must form separating triangles with their adjacent vertices. Consider any one of these vertices of degree 2. If it forms part of a separating triangle with a vertex it is adjacent to in the same distance class, and one in the next distance class below, then the vertices in the separating triangle must all be in the same distance class as the vertex of degree 2. But, since the vertex is of degree 2, there can only be one vertex in the separating triangle. We therefore have a contradiction, as we can contract this vertex in the separating triangle into either of the vertices in the same distance class which comprise the other vertices of the separating triangle; no separating triangle exists with this vertex, as it is adjacent to only one vertex of the distance class below, and the two faces comprising this new vertex to be contracted, the vertex in the distance class below, and respectively, the other two vertices it is adjacent to, exist. □

This correction ensures multiple edges can be coalesced without violating maximal planarity, and hence without needing to add in arbitrary edges.

4.3.5 Notes on the implementation of the ODA

In this final section on the Contraction Algorithm, we examine the description of the ODA given in Section 4.3.2 and Rinsma *et al.* [96] from an implementation perspective. The purpose of this section is to examine the motivation behind the ODA, and to use it to simplify this rather complicated mechanism both theoretically and practically. Some of the issues here were initially raised in Section 4.3.2, and we will return to these throughout the course of this section.

The ODA as described by Rinsma *et al.* attempted to find a *minimal* facility to represent the placement of the entering facility, in the sense that would satisfy all the current adjacencies of the entering facility. Having obtained this minimal facility, which is the rectanguloid labelled $q_1(\varepsilon)$ in the ODA, it was then possible to ignore the adjacency structure of the MPG, concentrating instead on satisfying the area requirement of the entering facility, by transferring rectanguloids from the placement host to the entering facility. The concept of the L-line however, is redundant, as we can create minimal facilities for both the placement host and the entering facility, leaving the rectanguloid surrounded by these two minimal facilities, which we will call the *redistribution* region, to be assigned to these two minimal facilities according to their respective area requirements. This leads to a re-assigning of the placement host. This modification is significant if the area of the entering facility is greater than the area of the rectanguloid q in the ODA, as Rinsma *et al.* proceeded to implement a further, more complicated, routine to deal with this. However, by creating a minimal facility for the placement host, we are able to remove the necessity for this entire extra routine.

Using the notation of Section 4.3.2, where facility x is the placement host and facility w is the entering facility, we create two minimal facilities labelled $q_x(\varepsilon)$, and $q_w(\varepsilon)$ ($q_w(\varepsilon)$ is the same as $q_1(\varepsilon)$ in the original version), leaving rectanguloid p to represent the redistribution region. This scenario is shown by Figure 4.29, where again the neighbours N_1 and N_s are adjacent to both facilities x and w.

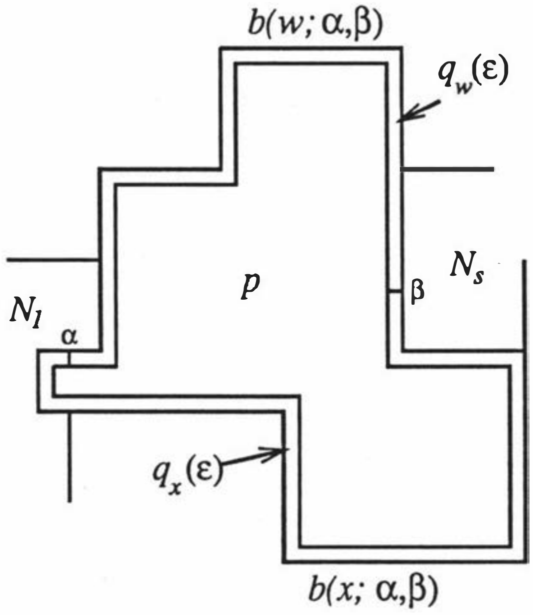


Figure 4.29: Dissection of facility x using the modified ODA approach

From this point, all that is required is the correct dissection of p, so that facilities x and w have the correct area. This is achieved by implementing essentially the same approach of Rinsma *et al.*, but instead of choosing a rectanguloid to transfer from the placement host to the entering facility, we now choose a rectanguloid from the redistribution region to join either the entering facility or placement host. Having obtained a rectanguloid from the redistribution region, we can transfer part or all of this rectanguloid to one of the minimal facilities. The minimal facility this rectanguloid is assigned to must share a common wall with the rectanguloid to maintain contiguity. This process continues until either of the facilities attains its correct area, resulting in the remainder of the redistribution region being assigned to the other facility. This is always possible, as both minimal facilities always border the redistribution region. If a facility can not accommodate an entire rectanguloid, the rectanguloid is split in the same way as for the original ODA. Furthermore, this method extends the original ODA in aspects such as the ability to perform an iterative search on the value of ε, dependent on both $q_w(\varepsilon)$ and $q_x(\varepsilon)$, as opposed to just $q_1(\varepsilon)$ for the original ODA.

This approach may sound as complicated as that proposed by Rinsma *et al.*, but it significantly reduces the complexity of the algorithm, by requiring only one procedure to assign rectanguloids. Furthermore, implementation on a computer is made simpler by the generation of this one mechanism. The modification uses the same underlying motivation of the original ODA, by first ensuring all adjacencies are satisfied, and then satisfying the area requirements. By also generating a minimal facility for the placement host, however, we are able to more fully employ the motivational strategy.

4.4 SIMPLE

This section provides a correct implementation of the ideas and motivation of Hassan and Hogg [52], Al-Hakim [4], and Irvine and Rinsma [60]. The initial paper by Hassan and Hogg, has been shown to not work in all cases by Al-Hakim, and both Al-Hakim, and Irvine and Rinsma have attempted to correct the inherent problems exhibited by the initial algorithm outlined by Hassan and Hogg, whilst maintaining the intuitive motivation behind the algorithm.

4.4.1 The SIMPLE Algorithm

In this Section we describe an algorithm for generating an insertion order in the MPG which will determine the processing order for placing the facilities into the layout during the ODA. The approach of Hassan and Hogg was described in Section 4.2, but we will briefly reiterate their approach here. Hassan and Hogg generated the insertion order by selecting a vertex adjacent to the vertex corresponding to the previous facility that was placed, and at least one other vertex corresponding to a placed facility. Unfortunately this insertion approach fails to generate an insertion order in the presence of a separating triangle in the MPG, as shown by Al-Hakim [4]. Facilities were subsequently placed in the layout by assigning each facility to an integral number of grid squares, which the layout perimeter enclosed.

To help with this discussion, define the empty face of a graph X, denoted by ∂X, which is simply a face of the planar graph dual to a current partial layout, which is not a face of the MPG. ∂X is not necessarily a face of length three. We refer to the sets of vertices and edges of ∂X as $V_{\partial X}$, and $E_{\partial X}$, respectively.

The new algorithm for generating an insertion order we call Sequential Insertion Method for Planning Layouts Effectively (SIMPLE). It simply chooses a facility to be placed in the layout which is adjacent to at least two already-placed facilities, and such that no non-existing faces are created in the MPG (an exception is ∂G_p, which will always be a face which is not in T_G). This process continues until the last facility is placed. Initialisation comprises the choice of a vertex adjacent to the exterior vertex. The output will be a layout with the adjacency graph G as its dual, as is justified below. We notice here that this is the necessary addition to Hassan and Hogg's insertion approach to overcome the presence of separating triangles. The SIMPLE insertion order is generated by Algorithm 4.7.

Algorithm 4.7 *Generation of the SIMPLE Placement Order*

```
**Input:** *MPG G*
**Output:** *SIMPLE Insertion Order*
$V_{G_p} \leftarrow \emptyset$
$E_{G_p} \leftarrow \emptyset$
$T_{G_p} \leftarrow \emptyset$
$V_{G_p} \leftarrow \{e\}$
*Determine the first facility to be placed in the layout (call it $x$) as one which is*
```

```
    *adjacent to the exterior facility; since there is always more than one candidate,*
    *we usually choose the one of maximum degree*
    $V_{G_p} \leftarrow V_{G_p} \cup \{x\}$
    $E_{G_p} \leftarrow \{(e,x)\}$
    while $(|V_{G_p}| < n)$ do
        *Select $t \in V_G \backslash V_{G_p}$, which creates faces $(t, y_i, y_{i+1}) \in T_G$, $i = 1, \dots, k-1$,*
        *such that $\{y_1 \dots, y_k\}$ form a chain on $\partial G_p$, or, if $p = n-1$, select $x \notin V_{G_p}$*
        *(Any tie breaking rule may be used).*
        $V_{G_p} \leftarrow V_{G_p} \cup \{t\}$
        for $v = 1$ to $n$ do
            if $(v \in V_{G_p}$ and $(t,v) \in E_G)$ then
                $E_{G_p} \leftarrow E_{G_p} \cup \{(t,v)\}$
            end
        end
        for $v = 1$ to $n$ do
            for $w = 1$ to $n$ do
                if $(\{v,w\} \subseteq V_{G_p}$ and $(t,v,w) \in T_G\})$ then
                    $T_{G_p} \leftarrow T_{G_p} \cup \{(t,v,w)\}$
                end
            end
        end
    end

end
```

4.4.2 Proof of the Correctness of SIMPLE

We now proceed to show that the SIMPLE insertion order of Algorithm 4.7 is guaranteed to produce an insertion order for arbitrary MPGs. In order to prove that the SIMPLE procedure will work, we need to show that there always exists a facility which can be placed, and if there is not, then the layout must be complete.

Theorem 4.4.1 *When choosing the next facility to be placed in the layout, SIMPLE will always provide at least one corresponding vertex in $V_G \backslash V_{G_p}$ from which to choose, or else all facilities have been placed in the layout.*

Proof: Let us assume that $|V_{G_p}| < n$, and that there is no candidate $x \in V_G \backslash V_{G_p}$ which can be inserted into G_p.

By maximal planarity, there exists a vertex x which creates a face $(x, y, z) \in T_G$, where (y, z) is an edge on ∂G_p. Now by definition x is not a candidate to enter, hence the insertion of x would create a face $(x, a, b) \notin T_G$, where (a, b) is also an edge on ∂G_p. Note that the chain formed by the vertices on ∂G_p adjacent to x includes a, b, y and z. Now as $(x, a, b) \notin T_G$, there must exist $c \in V_G \backslash V_{G_p}$ which creates a face $(a, b, c) \in T_G$, but as x is also adjacent to a and b, by maximal planarity, c must be a candidate to enter. Therefore we have a contradiction. Therefore either there is a candidate to be inserted, or else $G = G_p$. □

4.4.3 Application of the SIMPLE Insertion Order to the ODA

In this section we describe the construction of the layout using the SIMPLE insertion order, by using a modification of the ODA. The placement order generated by SIMPLE can be easily incorporated into the ODA framework. We begin by placing the first facility in the placement order across the *top* of the predefined layout perimeter, leaving the remaining area as the *empty space*. We then successively add the corresponding facilities according to the vertex insertion order, where the entering facility (EF) is always considered as having been *contracted* into the empty space. Therefore, the EF is w, while the empty space is x in the description of the ODA in Section 4.3.2. To complete the placement process for each facility, the empty space is adjusted so as to no longer include the region used by the facility just placed. Explicitly placing the final facility is unnecessary, as it simply comprises the empty space remaining after the placement of all other facilities.

4.4.4 Properties of ∂G_p

Throughout the SIMPLE procedure, we allow the non-existing face ∂G_p to be present until the final vertex is placed. This face could be thought of as a pseudovertex which corresponds to the empty space in the layout. Figure 4.30 depicts this more clearly; the shaded empty space is represented in the MPG by the pseudovertex PS, and the dotted lines represent adjacencies with the facilities which are adjacent to the empty space, *i.e.* those in $V_{\partial G_p}$. Note that the empty space is

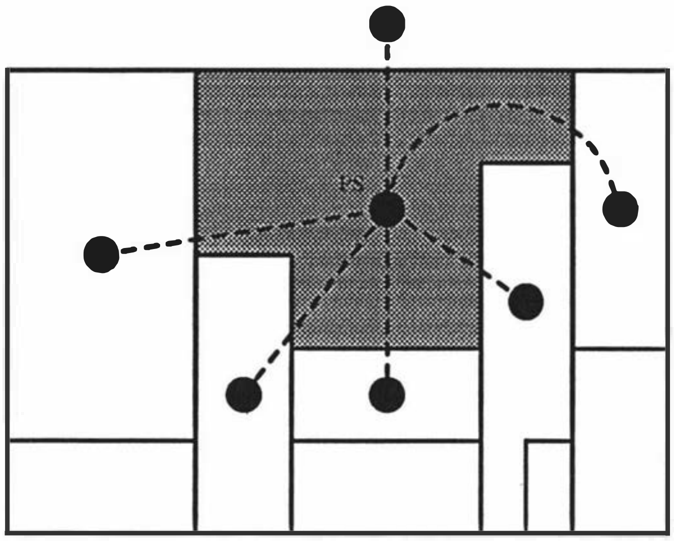


Figure 4.30: An example showing the empty space, and its relationship to the facilities in ∂G_p

not required to be adjacent to the exterior.

We have proven that SIMPLE will always admit an insertion order, but now we must prove that EF can be placed in the partial layout, as shown by Theorem 4.4.2.

Theorem 4.4.2 *Using the SIMPLE insertion order and the ODA modification, EF can be placed in the partial layout.*

Proof: Firstly, due to the criterion that only the nonexisting face ∂G_p cannot exist, there can never be more than one empty space. This means that we will never encounter infeasibility when inserting the vertex corresponding to EF into G_p with respect to the area of the facility, since the area of the empty space is equal to the sum of the areas of all facilities not yet placed. Therefore we need only be concerned with whether or not EF will meet all of its adjacencies with those facilities which have already been placed.

Let us consider all 3-joints which include the empty space; call them *pending* 3-joints. EF will have adjacencies with at least 2 facilities in V_{G_p}, and all facilities adjacent to EF will appear consecutively in cyc_{EF} (this is easily seen by the criterion for choosing EF). Since G is maximal planar, all pending 3-joints which are adjacent to EF will be able to be *covered* (*i.e.* become true 3-joints rather than pending 3-joints) without covering any which are not adjacent to EF. Therefore, we can draw the line $L_q(\varepsilon)$ of the ODA from the first placed vertex in cyc_{EF} to the last one to create the rectanguloid $q_{EF}(\varepsilon)$, and this will enclose all other facilities which are already placed and adjacent to EF. Therefore it is always possible to place EF into the current partial layout. □

4.4.5 An Illustrative Example

In this Section we illustrate SIMPLE by considering the 10 facility problem shown in Figure 4.31 and Table 4.6, where facility 1 is the exterior. A SIMPLE insertion order could be 2,7,4,9,3,8,6,5,10. Variations are possible depending on the choice of the first vertex, and at subsequent points where there exists more than one candidate vertex to insert.

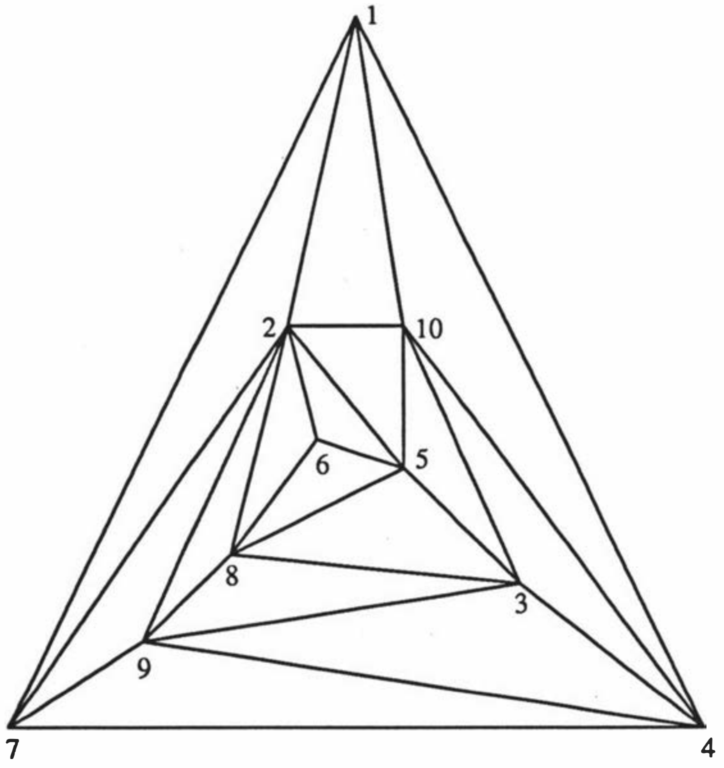


Figure 4.31: An MPG on 10 vertices (exterior = facility 1)

Facility	1	2	3	4	5	6	7	8	9	10
Area	-	30	25	10	40	35	15	20	10	30

Table 4.6: Areas Specifications for the SIMPLE Illustrative Example

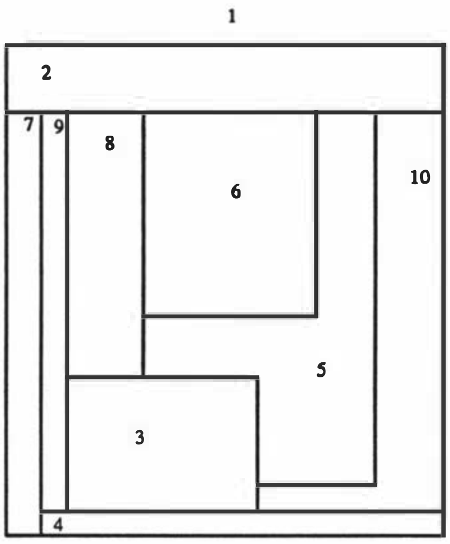


Figure 4.32: The final layout for the example problem using SIMPLE

The layout generated by the ODA using SIMPLE is shown in Figure 4.32. For simplicity the choices of α and β were taken to be simply half way along the common wall between the relevant facility and the empty space. Following the placement of a facility, the values of α and β could be implicitly changed by searching the newly-placed facility for redundant concave corners. This process and some related ideas and issues will be discussed fully in Section 5.1.

4.5 The Vertex Splitting Algorithm

In this section we describe a new method for obtaining a layout to a given arbitrary MPG. This method is motivated by a desire to transform the MPG into one in which the layout can easily be constructed, in much the same way as the Contraction Algorithm does. This method however is based on vertex splitting or expansion, as opposed to the contraction ideas used by Rinsma *et al.*

4.5.1 MPGs with Concentric Distance Classes

Consider the set of MPGs in which each vertex in D_i for $1 \leq i \leq max - 1$ is adjacent to exactly two other vertices in D_i. We call this the set of *concentric* MPGs. Consider the subgraph P of G where P includes all vertices except the exterior facility, and includes all edges (i, j) for which $\{i, j\} \subseteq D_k$, $k = 1, \ldots, max$.

Then an embedding of the subgraph P is equivalent to an embedding of concentric circles, with the vertices of D_1 in the outer circle, and those of D_{max} in the innermost circle.

MPGs of this type are amenable to layout construction, by considering the innermost distance class first. This distance class, D_{max}, may be laid out using either the Deltahedron method [42], as all maximal outerplanar MPGs are Deltahedron when the exterior is added, (which we prove in Section 4.7.1), or by the approaches of Rinsma [94]. Further, we are guaranteed that each facility in D_{max} will be at worst L-shaped. The facilities in D_{max} are laid out within a square perimeter with sides of length $\sqrt{\sum_{i \in D_{max}} a_i}$. Working now from distance class D_{max-1} to D_1, we sequentially lay out each distance class. This is done by first determining a perimeter or *frame* within which to lay out the facilities in the current distance class. The frame is dimensioned by determining on which *sides* of the previous frame facilities to be placed must be in order to satisfy their adjacency requirements. This is equivalent to determining within which region(s) each facility will lie, as indicated in Figure 4.33.

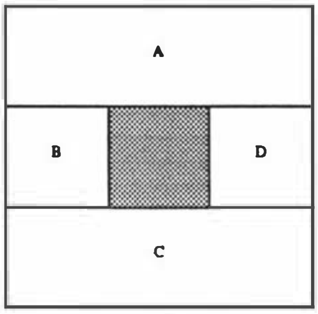


Figure 4.33: The regions of D_i, where the shaded region represents the layout of all facilities in distance classes greater than i

Facilities which have adjacencies in more than one region have their area split evenly among these regions. Facilities which are adjacent to only one facility situated at a corner of the previous frame are placed in one region only. Dimensioning of the regions is then carried out with respect to the previous distance class; using the length and width of the previous frame, the dimensions of the four regions are well defined.

Once the frame has been determined, facilities are dimensioned using an iterative process which assures that each facility to be placed fills the space enclosed by lines joining the new frame and the previous one, each line containing at most three rectilinear segments.

In order to perform the iterative scheme, we first find two facilities, termed the min-facility and the max-facility (which are akin to the rectanguloids labelled $q_1(\varepsilon)$ and $q_2(\varepsilon)$ in the ODA), by identifying a facility, termed the *reference facility*, which is the *last* facility in the previous distance class to which the entering facility is adjacent. Since facilities are laid out sequentially, by considering the concentric cycle of the facilities, an orientation is implied, so this reference facility is the last facility the entering one will encounter as it is placed, *i.e.* the next facility to be placed will also be adjacent to this reference facility. We determine a minimum adjacency parameter (ε), which scales the total possible physical adjacency between the entering facility and its reference facility to its desired minimum physical adjacency. A straight wall is placed to the frame of the distance class from the frame of the previous distance class so that this minimum adjacency requirement is obtained. This minimum adjacency parameter is a desirable minimum adjacency, however it may need to be reduced in order to guarantee existence of the layout. The polygon between this line and the last line of the previous facility is the min-facility. The max-facility is determined similarly by considering a maximum adjacency parameter, normally $(1-\varepsilon)$, between these two facilities. This is better represented by Figure 4.34, where the min-facility is the rectanguloid defined by $(1,2,3,4)$, while the max-facility is the rectanguloid defined by $(1,2,5,6)$. The reference facility being denoted by r.

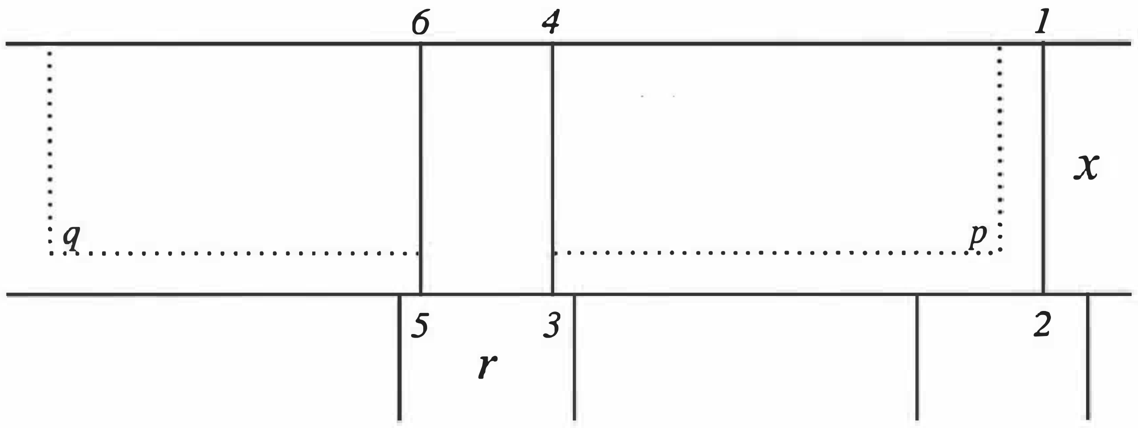


Figure 4.34: Facility placement using the ODA variant derived for the VSA

The iterative search between the min- and max-facilities is based upon a midpoint search. The average of every pair of corresponding coordinates representing the corners of each facility is calculated, and if the new facility defined by the region enclosed by these average coordinates has area less (greater) than a_i, it is set to be the new min(max)-facility. This continues until $|a_{min-facility} - a_{max-facility}| < \delta$, where δ is small. In some circumstances coordinates may need to be duplicated in order that both the min- and max-facilities have the same number of coordinates; this is always possible in order that we can then have distinct pairs of corresponding coordinates for each of the min- and max-facilities. This occurs if one of the min- or max-facilities has a straight line between frames and the other does not.

If the min-facility has area greater than that of the entering facility, a line is created from the previous frame to the frame of the current distance class consisting of three rectilinear segments, by following the outline created by the previous frame, and the previous facility placed, maintaining a width α (α small) from this outline to obtain a new min-facility, with the old min-facility becoming the max-facility (note that $\alpha = \varepsilon$ can be used). A similar procedure applies if the max-facility has area less than that of the entering facility. This is shown by the dotted lines of Figure 4.34, where the previous facility placed was x, and the new min-facility is bounded by the line p, and similarly for the new max-facility which is bounded by q.

If the min-facility thus created still has area greater than a_i, we halve ε and re-lay the previous facility. This process is perhaps the least desirable scenario, as we may be required to make ε very small in order to maintain that each line between the frames has at most three rectilinear segments, and we may have to re-lay more than one previously laid facility in order to lay out all the facilities in the region. This phenomenon may lead to arbitrarily small wall lengths between adjacent facilities, however successful completion of the layout is guaranteed. Note that the area of the enlarged max-facility is never smaller than the area of the entering facility if α is small enough (A special case exists if the entering facility is the last facility to be placed in a region; obviously it is placed by placing a line between the frames at the end of the region as defined in Figure 4.33).

The placement of the first facility for each distance class is a special case of the iterative midpoint procedure, where we consider an artificial facility whose boundary starts at the edge of a region, and has area equal to the sum of the areas of all facilities in that region which will not be placed until all facilities in the other three regions have been placed. In other words, if the first facility to be placed lies in the *middle* of a particular region, we will be dividing this region into two parts. Whichever orientation we use to lay out the facilities, one of these parts will contain the final facilities to be placed in this distance class; we must therefore ensure that both of these halves of the region are of the correct area to accommodate this. This artificial facility is then laid out using the above procedure. The initial line defining the edge of the region is then deleted. The procedure for laying out a facility i is described in Algorithm 4.8.

Algorithm 4.8 *Concentric Layout* (i)

```
Input: Partial Layout, with entering facility $i$
Output: Partial Layout containing $i$
if $i$ is last facility to be placed in region then
    Place facility $i$ using special case
end
else
    Determine the last facility in $D_{k+1}$ that $i$ is adjacent to; call it $j$
    Determine the min-facility and the max-facility, and their respective areas.
    while ($a_{min\text{-}facility} > a_i$) do
        Determine reduced min-facility and its corresponding area
        $max\text{-}facility \leftarrow min\text{-}facility$
        if ($a_{min\text{-}facility} > a_i$) then
            $\varepsilon \leftarrow \varepsilon/2$
            if ($\varepsilon$ too small and previous facility laid was $m$) then
                Concentric Layout ($m$)
            end
        end
    end
    while ($a_{max\text{-}facility} < a_i$) do
        Determine enlarged max-facility and its corresponding area
        $min\text{-}facility \leftarrow max\text{-}facility$
        if ($a_{max\text{-}facility} < a_i$) then
```

```
                $\varepsilon \leftarrow \varepsilon/2$
            end
        end
        Do iterative midpoint search between the min- and the max-facility
    end


end
```

4.5.2 Transforming an Arbitrary MPG to one with Concentric Distance Classes

The layout method proposed for MPGs with concentric distance classes can be applied to an arbitrary MPG by transforming that MPG into one whose distance classes are concentric. This allows an arbitrary MPG to be successfully dualised automatically, using the procedure described in Section 4.5.1.

The transformation of an arbitrary MPG into one with concentric distance classes uses an expansion technique, which augments the vertex set. We term the new vertices *pseudo vertices*, as they do not correspond to facilities, but are produced as a result of splitting the facility corresponding to a vertex of the MPG into two parts of equal area. Obviously, the two halves of any split facility must be adjacent in the final layout. This is the basis for the Vertex Splitting Algorithm (VSA).

In order to obtain concentric distance classes, we must ensure that each vertex in D_m is adjacent to exactly two other vertices in D_m for $1 \leq m \leq max-1$. In order to achieve this we employ two operations. The first is the deletion of a shortcut edge, *i.e.* an edge (i,j) such that $\{i,j\} \subset D_m$ and (i,j) is not an edge of any face whose vertices are i, j and a vertex from D_{m-1}. An example of the shortcut edge (i,j) defined above, where the curved dotted line represents the circuit through the other vertices in D_m, is given in Figure 4.35(a).

To delete a shortcut edge (i,j), $\{i,j\} \subset D_m$, where $a_i \geq a_j$, and $\{(i,j,k),(i,j,l)\} \subset T$ we use Algorithm 4.9, where we add in a vertex i' which deletes the adjacency between i and j. Application of Algorithm 4.9 results in the transformation of the MPG shown in Figure 4.35(b).

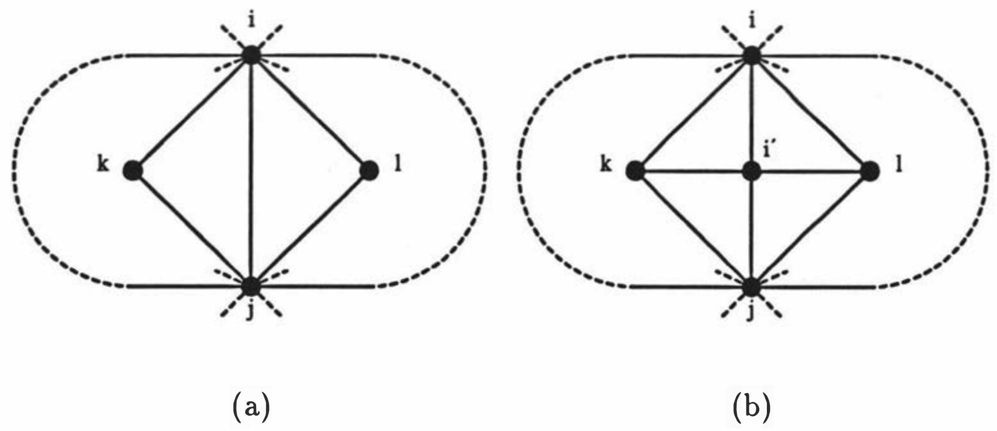


Figure 4.35: Deletion of a shortcut edge (i,j)

Algorithm 4.9 *Shortcut Eliminate* (i,j)

```
    $V \leftarrow V \cup \{i'\}$
    $E \leftarrow E \setminus \{(i,j)\}$
    $E \leftarrow E \cup \{(i,i'),(j,i'),(i',k),(i',l)\}$
    $T \leftarrow T \setminus \{(i,j,k),(i,j,l)\}$
    $T \leftarrow T \cup \{(i',i,k),(i',i,l),(i',j,k),(i',j,l)\}$
    $a_{i'} \leftarrow a_i/2$
    $a_i \leftarrow a_i/2$
    $D_{m+1} \leftarrow D_{m+1} \cup \{i'\}$
    $parent[i'] \leftarrow i$
end
```

The second operation required is the expansion of a cut vertex. A cut vertex is a vertex which, if deleted from the subgraph of the vertices in D_m and any edges that exist in the MPG between these vertices, would divide the subgraph into more more than one component. (Obviously if a distance class contains only one or two vertices, it contains no cut vertices). An example of a cut vertex $i \in D_m$, where the curved dotted line again represents the circuit around the other vertices in D_m, is shown in Figure 4.36(a).

Suppose we which to delete a cut vertex $i \in D_m$, with $\{p,q,r,s\} \subset D_m$ and p,q,r, and s all adjacent to i, such that p and r are in the same component if i is deleted, and q and s are in the same component if i is deleted. Note that if a component consists of a single vertex, then $p = r$ and/or $q = s$. Also let $\{a^1, \ldots, a^y\} \subset D_{m-1}$ and $\{b^1, \ldots, b^z\} \subset D_{m-1}$, be vertices adjacent to i, as shown in Figure 4.36. This procedure is described by Algorithm 4.10. Algorithm 4.10 results in the transformation of the MPG as shown in Figure 4.36(b).

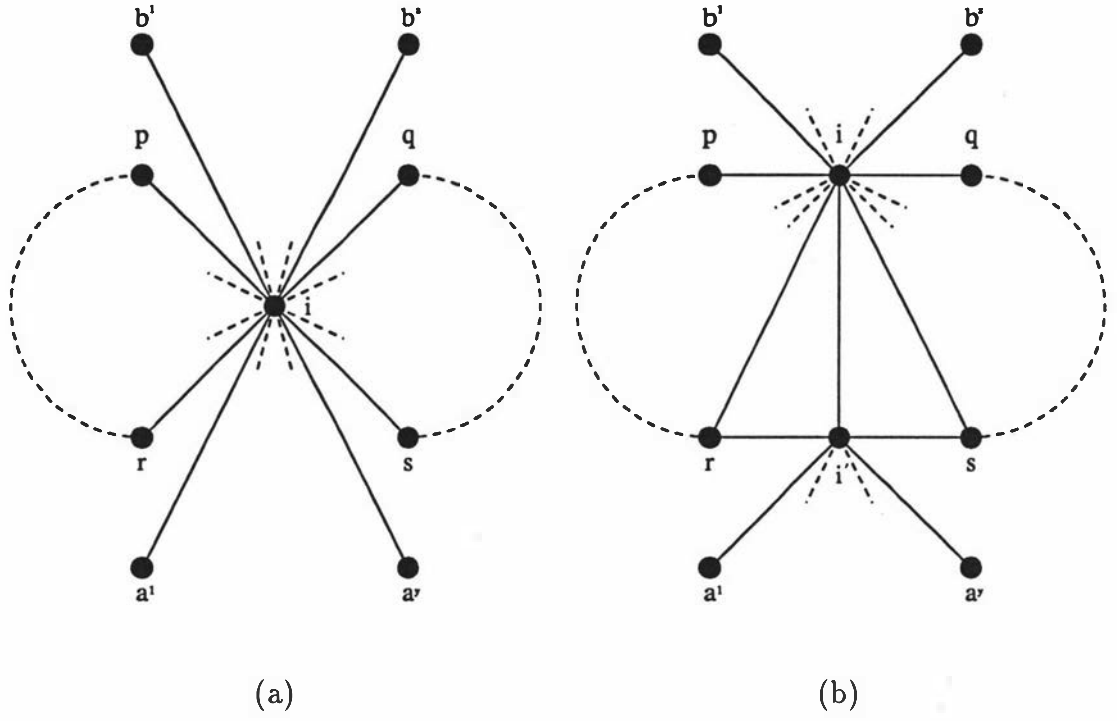


Figure 4.36: Expansion of a cut vertex i

Algorithm 4.10 *Cut Vertex Expand* (i)

```
    $V \leftarrow V \cup \{i'\}$
    $E \leftarrow E \setminus \{(i,a^1),\ldots,(i,a^y)\}$
    $E \leftarrow E \cup \{(i,i'),(r,i'),(s,i'),(i',a^1),\ldots,(i',a^y)\}$
    $T \leftarrow T \setminus \{(i,r,a^1),(i,s,a^y),(i,a^i,a^{i+1}) : i = 1 \ldots y-1\}$
    $T \leftarrow T \cup \{(i,i',r),(i,i',s),(i',r,a^1),(i',s,a^y),(i,a^i,a^{i+1}) : i = 1 \ldots y-1\}$
    $D_m \leftarrow D_m \cup \{i'\}$
    $a_{i'} \leftarrow a_i/2$
    $a_i \leftarrow a_i/2$
    $parent[i'] \leftarrow i$

end
```

Theorem 4.5.1 *The two operations Cut Vertex Expand and Shortcut Eliminate are sufficient to transform an arbitrary MPG into one with concentric distance classes.*

Proof: Firstly note that D_1 cannot contain any cut vertices, as the exterior is the only facility in D_0, and it cannot have two edges adjacent to a facility in D_1. The deletion of all shortcuts in D_1 adds pseudo vertices to D_2, and so D_1 is now of the form where all of the vertices in D_1 have degree two with all other vertices in D_1, *i.e.* D_1 is now concentric.

Consider D_i, which has inherited pseudo vertices from the deletion of D_{i-1}'s shortcuts. If we delete any cut vertices that exist in D_i, then each new facility becomes a part of D_i, and a shortcut edge is added. Hence all that remains is to delete the shortcuts from D_i; the pseudo facilities created from the elimination of these shortcuts are placed in D_{i+1}, and hence D_i is now in the required form. We do this for $2 \leq i \leq max - 1$. The pseudo vertices that D_{max} inherits from D_{max-1} do not create a problem, as we simply eliminate all cut vertices, and we are assured of having a maximal outerplanar graph for D_{max}, since we do not delete the shortcut edges from D_{max}. The transformed MPG now contains concentric distance classes. □

As each pseudo facility is created, we record which facility it was split from, *i.e.* when the MPG has been transformed into an MPG with concentric distance classes there is a tree-like structure, which records relationships between the original facility called the ancestor, and any pseudo vertex (called a child), which came from it. This means that the complete method for generating the layout of an arbitrary MPG for the Vertex Splitting Algorithm (VSA) is given in Algorithm 4.11, where G_{max} is simply the subgraph of the vertices in D_{max} together with the edges between vertices in D_{max} from G.

Algorithm 4.11 *The Vertex Splitting Algorithm*

```
Input: MPG G, area specifications
Output: Layout dual to G
for $i = 1$ to $n - 1$ do
    for $j = i + 1$ to $n$ do
        if (($i, j$) is a shortcut in $D_1$) then
```

```
                        Shortcut Eliminate $(i, j)$
                    end
                end
            end
            for $k = 2$ to $max - 1$ do
                for $i = 1$ to $n$ do
                    while ($i$ is a cut-vertex in $D_k$) do
                        Cut Vertex Expand $(i)$
                    end
                end    for $i = 1$ to $n - 1$ do
                    for $j = i + 1$ to $n$ do
                        if $((i, j)$ is a shortcut in $D_k)$ then
                            Shortcut Eliminate $(i, j)$
                        end
                    end
                end
            end
            for $i = 1$ to $n$ do
                while ($i$ is a cut-vertex in $D_{max}$) do
                    Cut Vertex Expand $(i)$
                end
            end
            $length_{max} \leftarrow \sqrt{\sum_{i \in D_{max}} a_i}$
            $width_{max} \leftarrow \sqrt{\sum_{i \in D_{max}} a_i}$
            Deltahedron Layout Algorithm $(G_{max})$
            Set $\varepsilon$
            for $k = max - 1$ down to 1 do
                Dimension regions for $D_k$
                while (all facilities in $D_k$ not laid out) do
                    for $i = 1$ to $n$ do
                        if ($i$ next to be laid) then
                            Concentric Layout $(i)$
                        end
                    end
```

```
        end
    end
    Combine child and ancestor facilities together

end
```

4.5.3 An Illustrative Example

In this Section we illustrate the VSA, by considering the 11 facility problem shown in Figure 4.37 and Table 4.7. The distance classes are as follows: $D_0 = \{1\}$, $D_1 = \{2,3,4\}$, $D_2 = \{5,6,7,8,9\}$ and $D_3 = \{10,11\}$.

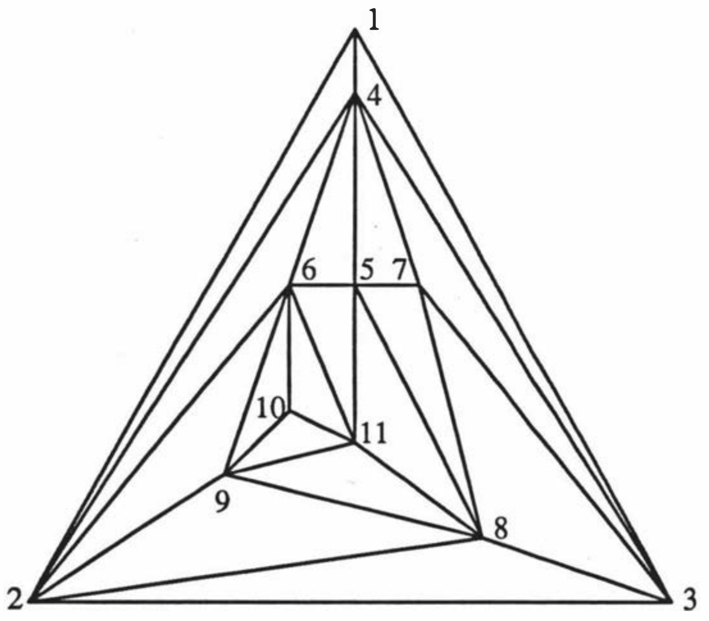


Figure 4.37: VSA Illustrative Example

Facility	1	2	3	4	5	6	7	8	9	10	11
Area	-	20	25	30	30	10	15	20	10	15	20

Table 4.7: Areas specifications for the VSA Illustrative Example

There are no shortcut edges in D_1, nor are there any cut vertices which exist in D_2. However we need to eliminate the shortcut edge $(5,8)$ in D_2. We do this by adding a vertex 12, which is a child facility of 5. Following the split of vertex 5, vertex 11 is then a cut vertex, which is deleted by introducing facility 13, which is a child of 11. Following the steps of the algorithm, facility 5 will have area 15, as will facility 12, and the areas of 11 and 13 will now be 10. The resulting transformed MPG is shown in Figure 4.38; it now has concentric distance classes, with facility 5 from the original MPG now represented by vertices 5 and 12, and facility 11 now represented by vertices 11 and 13.

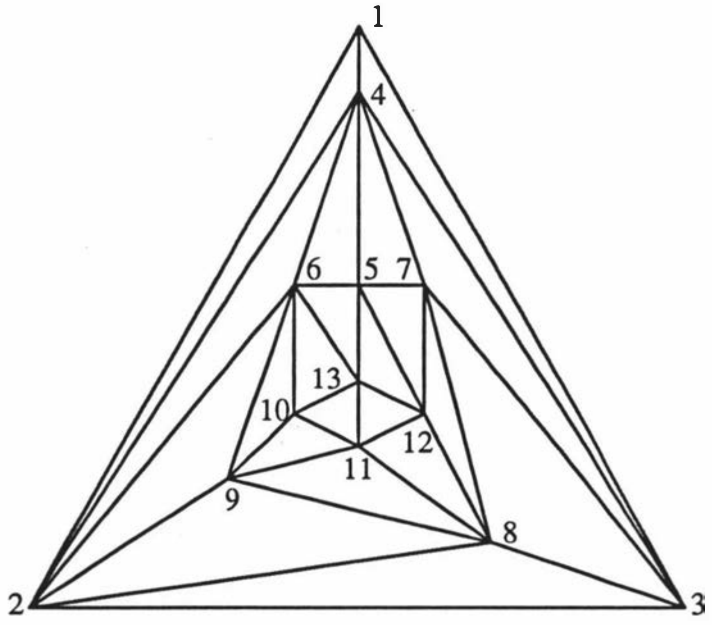


Figure 4.38: The transformed MPG with concentric distance classes

We now turn our attention to the layout phase. D_3 is the maximum distance class, so we lay out this distance class by considering the subgraph of the vertices in D_3 and the edges between vertices in this distance class. We also consider an exterior, e', which is adjacent to all of these facilities. A Deltahedron insertion order could be 12 in (e',11,13), with initial tetrahedron (e',10,11,13).

The layout of the vertices in D_3 is shown in Figure 4.39 within the innermost bold frame, where the length and width for the D_3 frame is $\sqrt{50}$.

The next step is to determine the dimensions of the regions for D_2. We require facility 5 to be in regions A and B, facilities 7,8 and 9 to be in region C, and facility 6 to be in region D.

Therefore the area of Region A must be $7\frac{1}{2}$, as must Region B; Region C must have area 45 and Region D must have area 10. The dimensions of each region are found easily by considering the dimensions of the frame for D_3, and proceeding accordingly. Placing the facilities in D_2 is easily performed: firstly facility 5, followed by 7,8 and 9, and lastly 6. The layout of the vertices contained in D_2 is shown in the second frame of Figure 4.39. D_1 is laid out similarly, by requiring facility 4 to be in Regions A and B, facilities 2 and 3 to be in region C, with facilities 2 and 4 in Region D. The areas of each of these regions will be 10 for Regions A and B, 35 for Region C, and 20 for Region D. This process is also seen in Figure 4.39, where we place facility 3 followed by facility 2 and lastly facility 4. The final dimensioned layout is presented in Figure 4.40, where facilities 5 and 12 have been combined to give facility 5, and facilities 11 and 13 have been combined to give facility 11 from the original MPG.

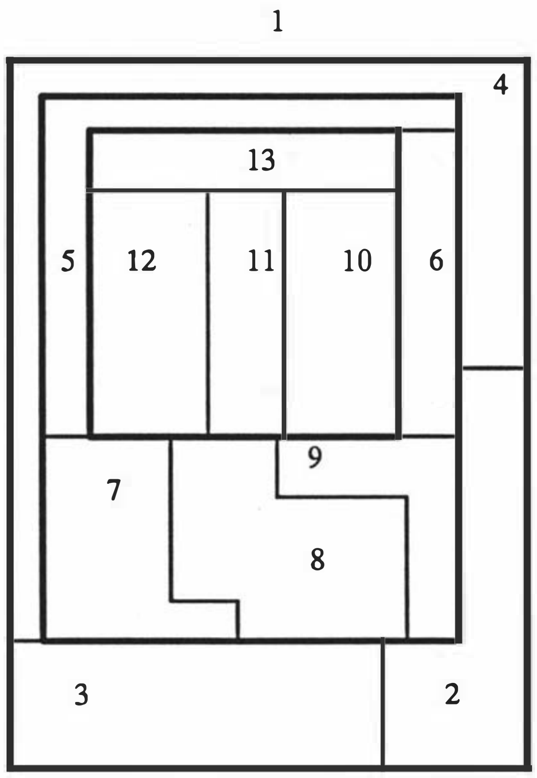


Figure 4.39: The layout before combination of the ancestor and child facilities

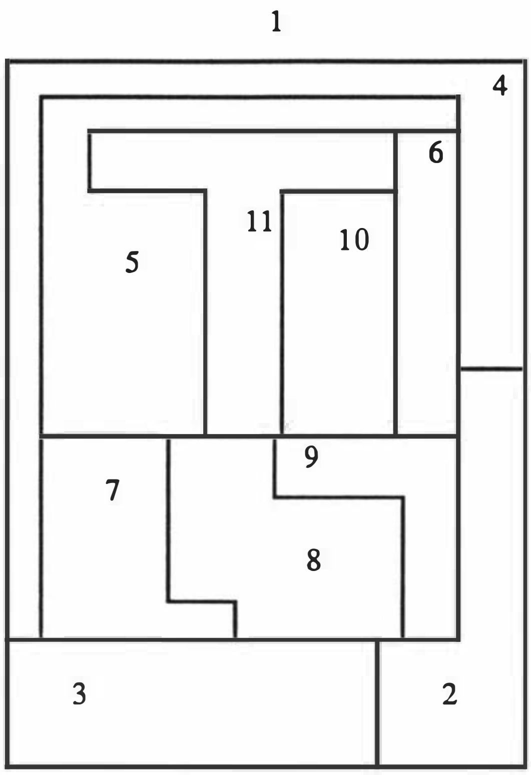


Figure 4.40: The final layout of the example MPG using the vertex splitting algorithm

4.5.4 Upper Bound on the Number of Pseudo Vertices Required

When using the VSA, an obvious question that arises is, how much splitting might we need? or, is there a limit to the amount of splitting we require? In this Section this question is answered by providing an upper bound on the number of pseudo vertices required in Theorem 4.5.2.

Theorem 4.5.2 *Let the number of pseudo vertices required for an MPG be PV. If a given MPG has distance class cardinalities of* $\{n_1, n_2, \ldots, n_{max}\}$ *then:*

$$n + PV \leq 9 + n_1(2^{max} - 1) + \sum_{i=2}^{max} n_i \sum_{j=i}^{max} 2^{j-i+1} + 8(max - 2^{max})$$

Proof: In a distance class i comprising n_i vertices there can be at most $n_i - 3$ shortcuts, corresponding to the maximal outerplanar subgraph consisting of the vertices in D_i, and any edges in the MPG that exist between these vertices. Pseudo vertices which eliminate cut vertices in D_i belong to D_{i+1}. For every cut vertex, we insert a vertex to eliminate that cut. A cut vertex whose deletion will create p components will require the insertion of at most $p-1$ cut vertices. But for every component there exists at least one vertex which is not a cut vertex. Therefore we will require at most $n_i - 2$ pseudo vertices to eliminate the cut vertices in D_i.

The pseudo vertices used to eliminate cut vertices in D_i are also in the set D_i. Therefore let us consider the new cardinality of D_i, call it n_i^+. Then

$$\begin{aligned} n_i^+ &= n_i + (n_{i-1}^+ - 3) + (n_i + (n_{i-1}^+ - 3) - 2) \\ &= 2n_i + 2n_{i-1}^+ - 8 \end{aligned}$$

We want to find a form for n_i^+ which is independent of n_{i-1}^+, given the boundary conditions $n_0 = n_0^+ = 1$ and $n_1^+ = n_1$.

It is easy to verify that, for $2 \leq i \leq max$

$$n_i^+ = 2^{i-1}n_1 + \sum_{j=2}^{i} 2^{i-j+1}n_j - 8(2^{i-1} - 1)$$

Then the complete number of vertices in the MPG must be

$$\begin{aligned} \sum_{i=0}^{max} n_i^+ &= n_0^+ + n_1^+ + \sum_{i=2}^{max} n_i^+ \\ &= 1 + n_1 + \sum_{i=2}^{max}(2^{i-1}n_1 + \sum_{j=2}^{i} 2^{i-j+1}n_j - 8(2^{i-1} - 1)) \\ &= 1 + n_1 + \sum_{i=2}^{max} 2^{i-1}n_1 + \sum_{i=2}^{max}\sum_{j=2}^{i} 2^{i-j+1}n_j - \sum_{i=2}^{max} 8(2^{i-1} - 1) \\ &= 9 + n_1(2^{max} - 1) + \sum_{i=2}^{max} n_i \sum_{j=i}^{max} 2^{j-i+1} - 2^{max+3} + 8max \\ & \qquad \text{(using } \sum_{i=0}^{n} 2^i = 2^{n+1} - 1 \text{ and swapping the order of summation)} \\ &= 9 + n_1(2^{max} - 1) + \sum_{i=2}^{max} n_i \sum_{j=i}^{max} 2^{j-i+1} + 8(max - 2^{max}) \end{aligned}$$

□

Therefore, given an arbitrary MPG, we can calculate *a priori* the maximum number of pseudo vertices that will be required to transform this MPG. Hopefully on average it will not require the amount specified by the upper bound, however MPGs with a large number of disjoint distance classes may come close to or even attain the upper bound. For the example of the Section 4.5.3, the upper bound on the number of vertices was 24, where we required only 13. The upper bound will be achieved, for example, on an MPG with 6 vertices, where $|D_1| = 4$ and the vertex in D_2 has degree 3; in this case the bound will be 7.

4.5.5 Layout Enhancements

Using the VSA can cause an overconstraining of adjacencies between facilities: if a vertex and one of its child vertices (or two child vertices with the same ancestor) are both adjacent to the same vertex in the transformed MPG, then only one of these adjacencies needs to be met in the layout before the corresponding facilities are combined with their ancestor facilities (in order to satisfy the adjacencies in the original MPG). Therefore we can relax some adjacencies of this type in the transformed MPG, without compromising the original MPG's adjacencies. This relaxation comes in the form of allowing 4-joints.

Consider the portion of an MPG with concentric distance classes in Figure 4.41, where i, j, l, and m are in distance class D_t and i', k, n and p are in distance class D_{t+1}.

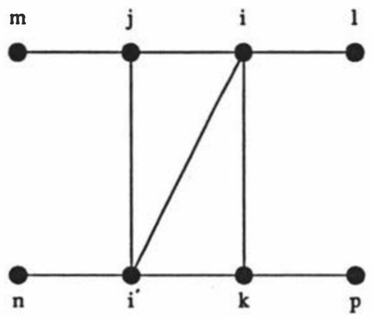


Figure 4.41: Portion of an MPG with concentric distance classes

Note that the relationship between i and i' can be distant (as long as they have the same ancestor), and they cannot be in the same distance class (otherwise a shortcut exists). An exception to this is in D_{max}; due to the special maximal outerplanarity property of the subgraph of the vertices in D_{max} and any edges between these vertices, we do not allow 4-joints within D_{max}.

Note that we cannot remove the edge (i, i'), since the ancestor facility may then become disconnected in the layout. We cannot remove edges (i, j) or (i', k) or our distance classes will be such that the 4-joint created will have a facility from D_{t+1}

and D_{t-1} meeting at a point; this is infeasible in an orthogonal layout of an MPG, as i and j would meet at a point, since they must be *opposite* each other in the 4-joint.

For the deletion of edge (i',j) we have 2 cases. Either; edge (m,i') exists, in which case i and m would be required to meet at a point, which is infeasible. (The exception to this is if i' is on a corner of a frame, in which case we can allow the 4-joint; however this occurrence can be determined only during the layout phase, not prior to it); or, edge (j,n) exists, in which case we can allow deletion of (i',j) to create a 4-joint. The deletion of edge (i,k) is similar to the deletion of edge (i',j), in which case, if edge (l,k) exists, we can allow the 4-joint. Otherwise, if edge (i,p) exists we cannot, even if k is at a corner.

The changes to the layout method required to allow these 4-joints are minor. Rather than create a min-facility and max-facility, we simply create a facility, with a coordinate meeting at the 4-joint and call it the min(max) facility depending on whether or not it has area greater(less) than required. The max(min) facility is constructed in the same manner as for the general case, and then the iterative scheme is applied. It may seem that in most cases having to create an L-shaped facility in order to create the 4-joint does not gain much; however, if we have to create the facility as an L-shape without the 4-joint, then we are increasing the degree of enforced irregularity in the block plan, and if the facility could have been created without the need for an L shape (without the 4-joint), we do not increase the degree of irregularity.

4.6 *Quasi* Graph Theoretic Techniques

In this section we describe four layout algorithms which do not fit exactly into the Graph Theoretic model that we are using, but are included for completeness. The first three methods are presented previously in the literature, while the fourth, the Tiling Algorithm, is a new, more classical, method akin to CORELAP, but with a more structured design to ensure higher objective function values, by guaranteeing $3n-6$ adjacencies, and hence an underlying MPG to the layout.

4.6.1 The Spiral Algorithm

Goetschalckx [46] presents an interactive two-stage heuristic for generating the layout, called the Spiral Algorithm. The first phase is the construction of the planar adjacency graph, which is specially constructed so that no vertex has degree greater than six (hence coining the term hexagonal adjacency graphs). The construction of the hexagonal adjacency graph is undertaken by considering three operations. The first is a *unary* tuple, and consists of the total closeness rating from CORELAP; the *binary* tuple considers two facilities, one of which is in the currently constructed graph, and the other yet to be inserted, which have a high relationship value, akin to the ALDEP methodology; the *ternary* tuple considers three vertices, one of which is not in the current graph, in the same way which the first TESSA case does. These tuples are adjusted to incorporate the adjacencies of the facilities with the exterior. Maximality of the adjacency graph is not assured under this process.

The second phase of this approach is the development of the layout, using the specially constructed hexagonal adjacency graph. The maximum degree of the vertices ensures the umbrella effect does not impinge upon the layout, and the layout is constructed with all rectangular shaped facilities. The layout is able to be constructed by considering a stratified approach. The adjacency graph is amenable to construction by assigning levels to the vertices of the adjacency graph. Each row of the graph becomes a row of facilities in the layout, providing layouts which look like brick walls. The author however does not appear to notice the subsequent dimensioning of the layout may involve the loss of adjacencies from the adjacency graph, due to the creation of fault lines between levels imposed on the adjacency graph. This dimensioning problem does not therefore guarantee the existence of a dimensioned layout using the Spiral Algorithm which is a dual to the given hexagonal adjacency graph.

4.6.2 Matching Based Layout Algorithm

Montreuil, Ratliff and Goetschalckx [85] presented a somewhat different approach to the GTFLP, by considering a so-called matching property. Montreuil *et al.* determined that in an adjacency graph, not only should adjacencies be specified, but also the length of wall that these adjacencies should have. In order to determine the length of these adjacencies, a b-matching approach was used; this is essentially a minimum cost flow network problem, with a flow of size k between vertices i

and j specifying that facilities i and j should have k segments in common, where segments are sides of grid squares imposed upon the layout. The resulting solution does not necessarily generate a planar graph however, and hence represents an upper bound on the value of the solution able to be obtained under this objective. The method uses a grid to lay out the facilities, but the authors do not elaborate further - it appears to be a jigsaw type of approach, where the facility shapes are determined *a priori*, and a b-matching solution is attempted. If this proves infeasible the solution to the b-matching is relaxed, by considering violation of the planarity property. Under this approach the eventual feasible adjacency graph is not necessarily maximal.

4.6.3 The Spanning Tree Algorithm

Rinsma [95] provides an approach which constructs the layout using an underlying tree structure. This approach uses a stratified approach as well, by assigning a root to the tree, and constructing the layout in a top-down fashion. The algorithm is simply described by considering a tree T. A vertex X is chosen as the root of the tree, and is placed as the top facility. The remainder of the layout is divided into k rectangles each of area equal to the sum of the areas of the vertices in each of the k subtrees of $T - X$, where k is the degree of X in T. The root of each subtree is the vertex in the subtree which is adjacent to X in T. This root is laid out as the top facility within its rectangle, and the process continues until all facilities have been placed, guaranteeing the construction of a rectangular layout. The process is formally described in Algorithm 4.12.

Algorithm 4.12 *The Spanning Tree Layout Algorithm*

```
Input: Tree $T$, and area requirements on facilities
Output: Layout dual to $T$
$R \leftarrow \{e\}$
$P_e \leftarrow BuildingPerimeter$
$a_e \leftarrow 0$
while ($R \neq \emptyset$) do
    Choose $v \in R$  Place $v$ as the top facility of area $a_v$ in rectangle $P_v$; label
    remainder of $P_v$ as $P'_v$
    $V_T \leftarrow V_T \backslash \{v\}$
```

```
        $R \leftarrow R\backslash\{v\}$
        for $i = 1$ to $n$ do
            if $(\{(i,v)\} \in E_T)$ then
                $E_T \leftarrow E_T\backslash\{(i,v)\}$
                $R \leftarrow R \cup \{i\}$
                Create rectangle of area equal to sum of areas of vertices in
                the tree rooted at i, adjacent to facility v, within $P'_v$, label the
                rectangle $P_i$
            end
        end
    end

end
```

The appearance of faultlines created in dividing the layout into rectangles for the subtrees poses no problem to the dimensioning of the layout, as only the horizontal adjacencies (*i.e.* those determined by a horizontal wall) of the layout are specified within the tree. Therefore there is no preconceived requirement for the adjacencies along these faultlines, and hence the adjacency benefit of the tree provides only a lower bound on the total benefit of the layout.

To illustrate the Spanning Tree Algorithm consider Figure 4.42, where we see a given specified tree, and Table 4.8 which provides its area specifications. The constructed layout is shown in Figure 4.43, where we see that the adjacencies of the tree are indeed horizontal, and every vertical adjacency can only increase the total benefit derived from the layout. In the layout of Figure 4.43, vertex 5 was chosen as the root of the tree, and the layout was completed from there.

Facility	1	2	3	4	5	6	7	8	9	10	11
Area	15	10	25	20	15	30	15	20	20	10	20

Table 4.8: Areas Specifications for Spanning Tree Illustrative Example

There are a few points which should be noted while we are discussing the Spanning Tree approach. The first is the role of the exterior facility, which was essentially ignored by Rinsma. In order to maintain consistency of the Spanning Tree approach with the other methods discussed, the exterior is considered to be a

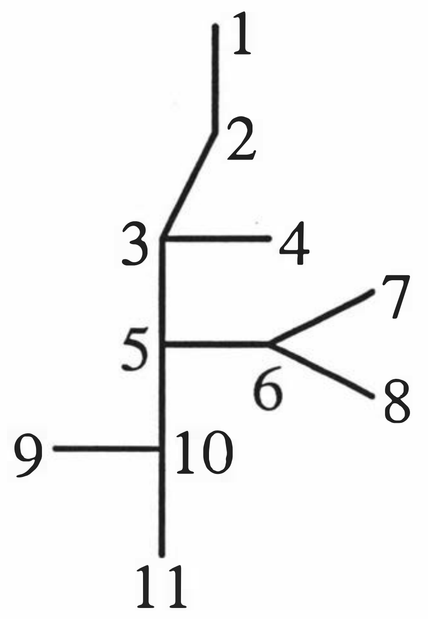


Figure 4.42: Spanning Tree Illustrative Example

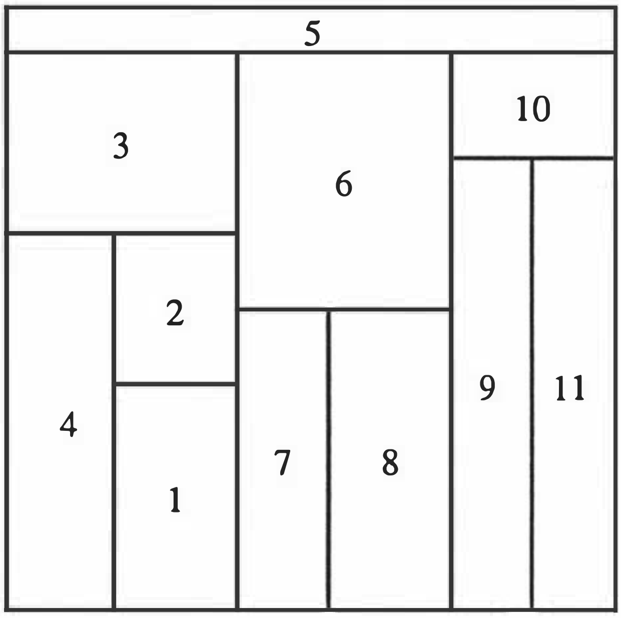


Figure 4.43: Spanning Tree Layout for the Illustrative Example

part of any tree which is constructed; further it is assigned zero area, and is always chosen to be the root of the tree. Furthermore, in order to bring the Spanning Tree approach into line with the Graph Theoretic model, it is assumed that there are $3n - 6$ adjacencies within the layout. This is sometimes hindered by the creation of a 4-joint, on a vertical faultline, which generally only occurs if facility areas are common multiples of one another. In instances where a 4-joint occurs, a slight perturbation is applied to the 4-joint, so that one of the adjacencies across the 4-joint can be realised. This perturbation requires the loss of the rectangularity of the layout, however the facilities in general become only L-shaped at worst.

The actual construction of the Spanning Tree uses Kruskal's Maximal Spanning Tree Algorithm [67], to ensure that the $n - 1$ adjacencies we are guaranteed are of largest possible weight. We attempt to increase the overall value of the layout by using a simple swapping technique between blocks of facilities within the layout. By considering rotations of the subtrees, we can produce different overall adjacency values in the layout. For example, in Figure 4.43, facilities 7 and 8, or blocks $\{1,2,3,4\}$ and $\{9,10,11\}$, could be swapped, whilst still maintaining the vertical adjacencies specified in the tree. This process could be used within a Tabu Search framework for instance in order to systematise the process of attaining vertical adjacencies.

4.6.4 The Tiling Algorithm

In this section we introduce a new classical type of approach to developing the layout. This method, which we term the Tiling Algorithm, uses ideas similar to those of CORELAP, but ensures the underlying adjacency graph is maximal planar.

Hassan and Hogg [53] develop a related method, which attempts to minimise the total transportation cost, but it does not adequately allow enough generality to be applicable within a true Graph Theoretic framework. Using a grid structure, Hassan and Hogg place initially two or three mutually adjacent facilities, with minimum transportation cost in the layout. Subsequent facilities are added by considering the transportation cost between facilities on a *side* of the layout, and a possible entering facility. The chosen facility of minimum cost, is placed in an integral number of grid squares, with its dimensions determined iteratively, so that each facility can be placed as a rectangle.

The method described by Hassan and Hogg goes some way towards developing a classical method which could be compared to the layout methods especially designed for the GTFLP, however it has several drawbacks as we will outline now. Firstly in Section 4.2, we outlined the problems involved in using a grid structure for facility placement, and this problem is even more pronounced in this method, since each facility must be a rectangle. Therefore a facility i may not attain its correct specified area if for example, a_i is a prime number. Furthermore, Hassan and Hogg place minimal consideration upon the adjacencies with the exterior. The only emphasis on the exterior is to divide the facilities into two groups, one containing facilities having movement cost with the exterior, and the other containing facilities with no movement cost with the exterior, and then placing the second group in the layout first. This rationale appears a little coarse, and does not consider the consequences of a placed facility having material flow with the exterior being cut off from the exterior.

We now develop the Tiling Algorithm, which builds upon CORELAP and the ideas of Hassan and Hogg. Firstly we formalise the concept of a *side* of the layout, by defining that facilities having adjacency with the exterior in a current partial layout to be on the side(s) of the layout where that adjacency is satisfied. This means that a corner facility will be on at least two sides. The sides are labelled t, b, l, and r, corresponding to top, bottom, left and right respectively. The set of facilities adjacent to each side is denoted by T, B, L, and R respectively, and for any partial layout having m facilities adjacent to the exterior, $|T| + |B| + |L| + |R| = m + 4$, since the facility at each corner is counted twice.

The Tiling Algorithm builds upon the idea of placing a facility on only one side of the layout as used by Hassan and Hogg, but the placement routine is taken further to allow a more versatile approach. Firstly we abolish the grid structure, in order to allow facility areas to be satisfied exactly and, as we will see, to produce a dimensionalisable layout. This is all achieved by placing facilities along an entire side of the layout. The basic idea of the Tiling Algorithm is to greedily choose a facility to be placed in the layout, and to then place that facility along an entire side so that, in the subsequent dimensioning phase, no adjacencies are lost. Dimensioning is carried out using the Inflation process as described in Section 4.1.1. Initialisation of the Tiling Algorithm involves choosing three facilities whose mutual adjacencies with each other and the exterior is maximised. This initialisation in the MPG is effectively the greedy initial Tetrahedron from the Deltahedron Algorithm. The placement of these three facilities is also the same as that for the initial Deltahedron layout, except in this instance we do not place subsequent facilities inside these initial three facilities, but around the outside instead.

Subsequent facilities are placed in the layout by considering the available sides on which facilities can be placed so that maximal planarity of the underlying adjacency graph is preserved. Obviously the *exposed* side the last facility was placed on is not available, as this would lead to the entering facility only being adjacent to the exterior and the previously placed facility, hence violating maximal planarity of the partial layout. For each facility which is a candidate to enter the layout, we determine the increase in benefit of the total adjacency of the layout on each side of the layout. The facility/side pairing which gives the greatest increase to the total benefit is chosen as the entering facility and placement side respectively.

Two objective measures were used to find the facility(i) and side(J) pairings (i, J). These are shown in Equations 4.9 and 4.10, corresponding to the best overall increase in the adjacency graph, and the best average adjacency value of the entering adjacencies, respectively, where the following sets are defined: S is the set of sides which are candidates for placement sides; $J^* = B \cup T \cup L \cup R \backslash J$; and the current graph is labelled G_p, containing p vertices including the exterior vertex e.

$$(i, J) = \arg\max_{\substack{i \notin V_{G_p} \\ J \in S}} \left(w_{ie} + \sum_{k \in J} w_{ik} - \sum_{\substack{k \in J \\ k \notin J^*}} w_{ke} \right) \tag{4.9}$$

$$(i, J) = \arg\max_{\substack{i \notin V_{G_p} \\ J \in S}} \left(\frac{w_{ie}}{|J|} + \sum_{k \in J} \frac{w_{ik}}{|J|} - \sum_{\substack{k \in J \\ k \notin J^*}} \frac{w_{ke}}{|J|} \right) \tag{4.10}$$

Furthermore we can extend the Tiling Algorithm to allow a greater variety of facility shapes, with few changes. This is undertaken by considering how many rectangles each facility shape can be broken down into, *e.g.* L-shaped facilities can be made up of two rectangles, and similarly for T-shaped facilities. Therefore if we determine that L and T shaped facilities are to be permitted, we allow facilities to be placed *twice*, with the only restriction being that the second time the facility is placed, it is placed adjacent to its first placement position. This allows for a final merging of the two placements to give the required facility shapes. Since the resulting layout is dimensionalisable, there is no problem in satisfying the area of any facility which is placed more than once. We can also allow imposition of user-defined constraints with this approach, so that facilities which are required to be rectangular are placed only once, while Material Handling Systems, for example, could be placed multiple times. The modifications to the objective values for choosing the facility/side pairs (i, J) are shown in Equations 4.11 and 4.12, corresponding to Equations 4.9 and 4.10 respectively, where I_i is the set of facilities adjacent to i in the current layout. Therefore when considering a facility i, if $p_i = 0$, we use Equation 4.9 or 4.10, while if $0 < p_i < i^{max}$, we use Equation 4.11 or 4.12, where i^{max} is the maximum number of times i can be placed, and the number of times i has been placed is denoted by p_i.

$$(i,J) = \arg\max_{\substack{J\in S\\ i\in J}} \left(\sum_{\substack{k\in J\\ k\notin I_i}} w_{ik} - \sum_{\substack{k\neq i\\ k\in J\\ k\notin J^*}} w_{ke} \right) \tag{4.11}$$

$$(i,J) = \arg\max_{\substack{J\in S\\ i\in J}} \left(\sum_{\substack{k\in J\\ k\notin I_i}} \frac{w_{ik}}{|J|} - \sum_{\substack{k\neq i\\ k\in J\\ k\notin J^*}} \frac{w_{ke}}{|J|} \right) \tag{4.12}$$

We are now in a position to give the full description of the Tiling Algorithm which is shown in Algorithm 4.13, for the case using Equations 4.9 and 4.11, and where J_1 and J_2 are the sides adjacent to side J.

Algorithm 4.13 *The Tiling Algorithm*

```
Input: Adjacency matrix of benefits, $i^{max}$ values, and area specifications for each facility
Output: Dimensioned Layout
$best \leftarrow -1$
$V_p \leftarrow \emptyset$
for $i, j, k = 1$ to $n$ do
    if $(best < w_{ie} + w_{je} + w_{ke} + w_{ij} + w_{ik} + w_{jk})$ then
        $best \leftarrow w_{ie} + w_{je} + w_{ke} + w_{ij} + w_{ik} + w_{jk}$
        $V_p \leftarrow \{e, i, j, k\}$
    end
```

```
end
$adjacency\ benefit \leftarrow best$
Place vertices of $V_p$ via initial Deltahedron layout
for For each element $i$ of $V_p$ do
    $I_i \leftarrow V_p \backslash i$
end
$S \leftarrow L, R, B$
while (Improvement in Total Adjacency Value or All Facilities Not Placed)
do
    for $i = 1$ to $n$ do
        for $J = B, R, L, T$ do
            $B(i, J) \leftarrow -1$
            if ($p_i = 0$ and $J \in S$) then
                $B(i, J) \leftarrow w_{ie} + \sum_{k \in J} w_{ik} + \sum_{\substack{k \in J \\ k \notin J^*}} w_{ek}$
            end
            if ($0 < p_i < i^{max}$ and $i \in J$ and $J \in S$) then
                $B(i, J) \leftarrow \sum_{\substack{k \in J \\ k \notin I}} w_{ik} - \sum_{\substack{k \neq i \\ k \in J \\ k \notin J^*}} w_{ke}$
            end
        end
    end
    $B(i^*, J^*) \leftarrow \max_{i,J} B(i, J)$
    $V_{G_p} \leftarrow V_{G_p} \cup i^*$
    $adjacency\ benefit \leftarrow adjacency\ benefit + (i^*, J^*)$
    Place facility $i^*$ along side $J^*$
    for $k = 1$ to $n$ do
        if ($k \in J$ and $k \notin J^*$) then
            $I_k \leftarrow I_k \cup i^* \backslash e$
        end
        if ($k \in J$ and $k \in J^*$) then
            $I_k \leftarrow I_k \cup i^*$
        end
    end
    $I_{i^*} \leftarrow e \cup J^*$
    $p_{i^*} \leftarrow p_{i^*} + 1$
```

```
        $S \leftarrow S \cup \{J_1, J_2\} \backslash J$
        $J \leftarrow i^*$
        $J_1 \leftarrow J_1 \cup i^*$
        $J_2 \leftarrow J_2 \cup i^*$

    **end**
    *Dimension layout via inflation*

**end**
```

Consider now the following example problem. The benefit matrix is given in Table 4.9, and area requirements in Table 4.10, and we assume that each facility

	2	3	4	5	6	7	8	9	10
1	409	0	0	0	179	0	444	353	229
2		0	0	0	0	0	0	437	190
3			0	283	52	0	208	204	0
4				0	20	0	239	0	0
5					138	249	410	391	0
6						0	0	124	0
7							0	0	283
8								325	0
9									0

Table 4.9: Adjacency matrix to illustrate the Tiling Algorithm

Facility	1	2	3	4	5	6	7	8	9	10
Area	-	15	15	10	20	10	25	20	20	10

Table 4.10: Area specifications for the Tiling Algorithm Illustrative Example

can be placed at most twice. The initial layout comprises facilities 2, 8 and 9, along with the exterior facility 1. Facility 5 is then placed on the bottom side, note that at this point only the left and right sides are available for placement. Subsequent placements are the placement of facility 3 on the right side, followed by facilities 6 and 10 on the bottom and top respectively. Facility 7 is then placed on the right,

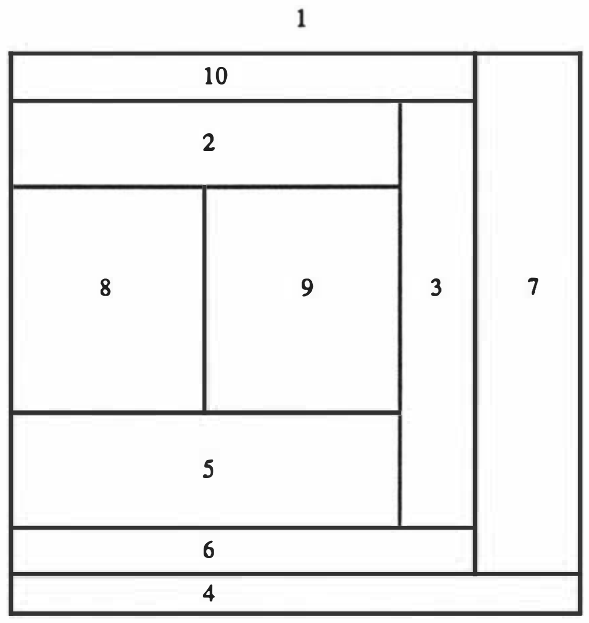


Figure 4.44: Tiling Algorithm Layout for the Illustrative Example

and lastly facility 4 is placed at the bottom. The corresponding layout to this problem is shown in Figure 4.44. Notice that we cannot now improve the total benefit even though we are able to place facilities more than once. The only extra adjacencies which could be currently obtained are placing one of facilities 2,4,5,6,8, or 10 along the left side, but this would result in at least two of the adjacencies $(1,2)$, $(1,6)$ and $(1,8)$ being removed. For the purpose of illustration, assume that placing facility 8 say along the left side would increase the total benefit. The layout after this second placement of facility 8 is given in Figure 4.45, where facility 8's area was split evenly between its two placements.

Finally let us consider improvement techniques which could be applied to Tiling Algorithm layouts. Two processes are immediately available, called facility swapping and facility transformation. The facility swap operation simply considers two facilities i and j which we attempt to swap; note that there exists no problem regarding facility shapes here (as long as the p_i values permit), as the layout is always maintained as a dimensionalisable layout. The potential benefit of swapping two facilities i and j is denoted by $P_s(i,j)$, and is given by Equation 4.13. If there exists $P_s(i,j) > 0$, then the swapping facilities i and j will improve the adjacency value of the layout.

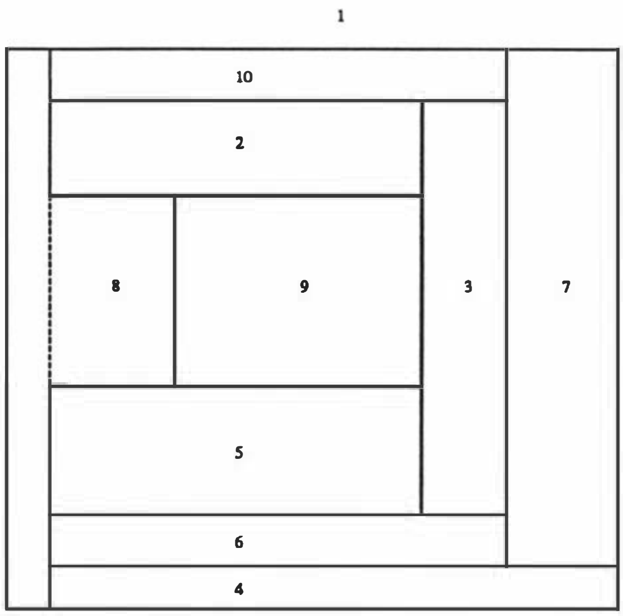


Figure 4.45: Example of placing a facility twice into a Tiling Algorithm Layout

$$P_s(i,j) = \sum_{k\in I_i}(w_{kj} - w_{ki}) + \sum_{k\in I_j}(w_{ki} - w_{kj}) \tag{4.13}$$

The second operation, facility transformation, involves transferring one rectangle from a facility to another facility. The facility the rectangle is taken from must have been placed more than once, the reassigning of the rectangle must not disconnect the facility the rectangle is taken from, and the facility the rectangle is being reassigned to must have been placed fewer than its allowed maximum number of placings. The potential benefit of transforming a rectangle r from facility i to facility j is denoted by $P_t(i_r, j)$, and the benefit derived in making this transformation is given by Equation 4.14, where I_{i_r} is the set of facilities adjacent to rectangle i_r.

$$P_t(i_r,j) = \sum_{k\in I_{i_r}\setminus(I_j\cap I_{i_r})} w_{kj} - \sum_{k\in I_{i_r}\setminus(I_i\cap I_{i_r})} w_{ki} \tag{4.14}$$

Two further operations are available, involving the deletion of, or addition of, rectangles to facilities, and are denoted by $P_d(i_r, J)$ (where we consider deleting a rectangle r of i from side J with benefit given by Equation 4.15) and $P_a(i, J)$ (where we consider adding another rectangle to i along side J, with benefit given by Equation 4.16).

$$P_d(i_r, J) = \sum_{k \in I_{i_r} \setminus (I_{i_r} \cap (J_1 \cup J_2))} w_{ek} - \sum_{k \in I_{i_r} \setminus (I_i \cap I_{i_r})} w_{ki} \tag{4.15}$$

$$P_a(i, J) = \sum_{k \in J \setminus (I_i \cap J)} w_{ki} - \sum_{k \in J \setminus (J \cap (J_1 \cup J_2))} w_{ek} \tag{4.16}$$

These two latter operations ensure that we are able to create and destroy rectangles to avoid having a fixed number of possible rectangles in any layout. Rectangle addition and deletion may only be carried out as long as the dimensionalisability of the layout is retained, and for that reason are conducted on the outskirts of the layout, with rectangles being only being deleted if they are the only element in B, R, L, or T; similarly rectangles must be added in the same way as in the original algorithm - along an entire side of the layout, whilst maintaining maximal planarity. The four operations could be used within a Tabu Search framework for instance, in order that we *move through* a variety of layouts.

4.7 Special Classes of MPGs

In this section we examine some MPGs which allow the simple generation of dimensionalisable layouts with guaranteed worst-case facility shape. We are motivated by an attempt to characterise as many classes of MPGs as possible which, due to their structure, allow dimensionalisable layouts to be constructed easily. Structural characterisation is important, as each MPG can represent an infinite number of different layout problems, given different area conditions, thereby we can effectively solve a vast array of layout problems (although each set is probably of measure zero within the entire set of GTFLP instances), satisfying any of the conditions characterised.

4.7.1 All Vertices Adjacent to the Exterior Vertex

Structural Condition: $|D_1| = n - 1$. This first class has been used extensively throughout this thesis, especially as starting points for the ODA and the VSA. We now proceed to show that all MPGs of this type are Deltahedron-generatable and further, when using the Deltahedron Layout Algorithm, generate layouts with a worst case facility shape of L. Consider a MPG on n vertices in which there exists a vertex i adjacent to every other vertex, *i.e.* $d_i = n - 1$. Let us call this vertex of degree $n - 1$ the *centre* vertex.

Theorem 4.7.1 *If an MPG contains a center vertex, then the MPG is Deltahedron generatable.*

Proof: By definition, the wheel corresponding to the centre vertex must have the form of Figure 4.46.

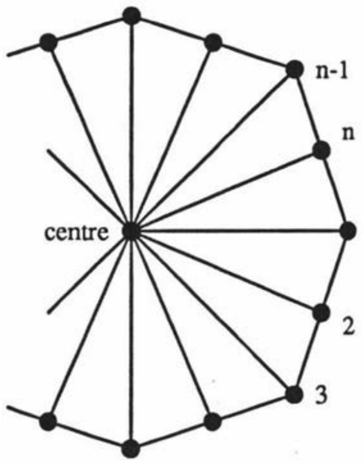


Figure 4.46: Wheel for a centre vertex

Consider the faces in the MPG which do not involve the centre vertex. Since all vertices are in the partial graph of the centre vertex's wheel, the other faces will be placed via case (b) of the TESSA heuristic, whereby a face is created involving three vertices and two edges of the partial graph. The insertion of a face in this instance is shown in Figure 4.47.

One of the vertices in the face added has now become interior, and has degree three. Once a vertex becomes interior, it cannot be a part of any more faces, and hence cannot increase its degree. This vertex of degree three is then an equivalent Deltahedron insertion, *i.e.* we know that if the final MPG is indeed Deltahedron, then we will be able to delete this vertex of degree three in order to find the Deltahedron insertion order, via ETA2. Therefore this vertex may be deleted. Hence, we now have a partial MPG on $n - 1$ vertices. By repeating this process of adding faces via TESSA, and deleting interior vertices of degree three, we will end up with the Tetrahedron as our final MPG, the starting point for Deltahedron, with the insertion order being the reverse order in which the vertices were deleted.

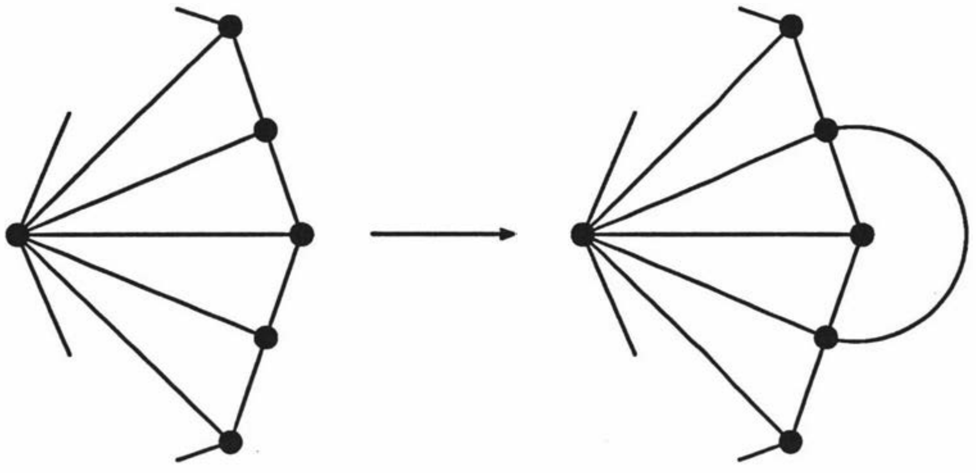


Figure 4.47: Placement of a face not adjacent to the centre vertex

This can be further described as follows: all MPGs have $2n-4$ faces; the centre vertex and its wheel account for $n-1$ of these faces, leaving $n-4$ faces to be added by TESSA. TESSA creates an MPG by adding $2n-5$ faces, and the remaining face is the final infinite face. Now each face added creates a vertex of degree three, which can then be deleted. Therefore we can delete $n-4$ vertices from the MPG, which leaves four vertices, the centre vertex plus the vertices in the final infinite face, from which to create the final MPG, and only one MPG on four vertices exists. □

Corollary 4.7.1 *If $|D_1| = n-1$, then the MPG is Deltahedron-generatable.*

Proof: If $|D_1| = n-1$, then the exterior is a centre vertex, the result follows from Theorem 4.7.1 □

Theorem 4.7.2 *If, in an MPG the exterior facility is a centre vertex, then there exists a layout with every facility at worst L shaped.*

Proof: Consider the initial tetrahedron layout. Each 3-joint containing the exterior can place a facility using PO1, therefore the initial placement retains the rectangularity of the layout. Consider the Deltahedron placement directions; the two facilities on the bottom left and right corners will have two placement directions directed into them from the 3-joints containing the exterior. Placement at these corner facilities are PO1 operations, and therefore do not violate the rectangularity of the corner facility. Therefore the facility shape can only be worsened by placement within a facility which is not a corner facility. We are only concerned with the 3-joints involving the exterior, of which there are exactly two for each facility. If we can show that each rectangular facility not having a 2-joint, has exactly one placement direction into it, and each L-shaped facility has none, we have completed the proof, as rectangular facilities with a 2-joint will perform similarly to the corner facilities. Therefore, consider the left side of the layout, having k rectangular facilities bordering this side, labelled $f_1, f_2, \ldots, f_k$, where f_1 is the top facility, and f_k the bottom left corner facility. This chain of facilities provides the 3-joints $(f_1, f_2, e), (f_2, f_3, e), \ldots, (f_{k-1}, f_k, e)$. Now, if each of the placement directions at these 3-joints is directed into the second facility of each 3-joint, and considering similar analysis for the bottom and right sides, each facility has at most one placement direction, except the bottom left, and bottom right corners (which we have considered already). Consider some L-shaped facility; the reflex corner of this facility is a 2-joint of some rectangular facility. Thus, the placement direction for this L-shaped facility is directed into this rectangular facility, which therefore has two placements directions directed into it; however, this second one is adjacent to a 2-joint, and therefore only one of the placement directions can worsen the shape of this facility. □

Rinsma [94] also provided a mechanism for producing layouts of this type, although Rinsma's method was defined in terms of Maximal Outerplanar Graphs (which are equivalent to this class by adding a vertex representing the exterior adjacent to every other vertex). Rinsma's method and the Deltahedron Layout Algorithm perform essentially the same steps, the only difference being in the choice of initial Tetrahedron, where any face not including the exterior can be chosen to complete the Tetrahedron, together with the exterior. Deltahedron uses the ETA1 or ETA2 methods to choose this face, while the choice using the Rinsma approach is the first face encountered when the faces of the maximal outerplanar graph are ordered lexicographically.

4.7.2 One Vertex is Not Adjacent to the Exterior

Structural Condition: $|D_2| = 1, \Rightarrow |D_1| = n - 2$. We will discuss this class of MPGs in two sections. The first is a simple sub-case, where we impose the extra condition that $d_x = n - 2, x \in D_2$. The layout is formed by choosing two facilities to be placed in regions A and C of the VSA, respectively. Unless $n = 5$, they should be non-adjacent facilities. The other facilities are placed by considering the cyclic ordering of the facility in D_2. This is best illustrated by Figure 4.48, where x is the vertex in D_2, and i and j are chosen to be placed in regions A and C.

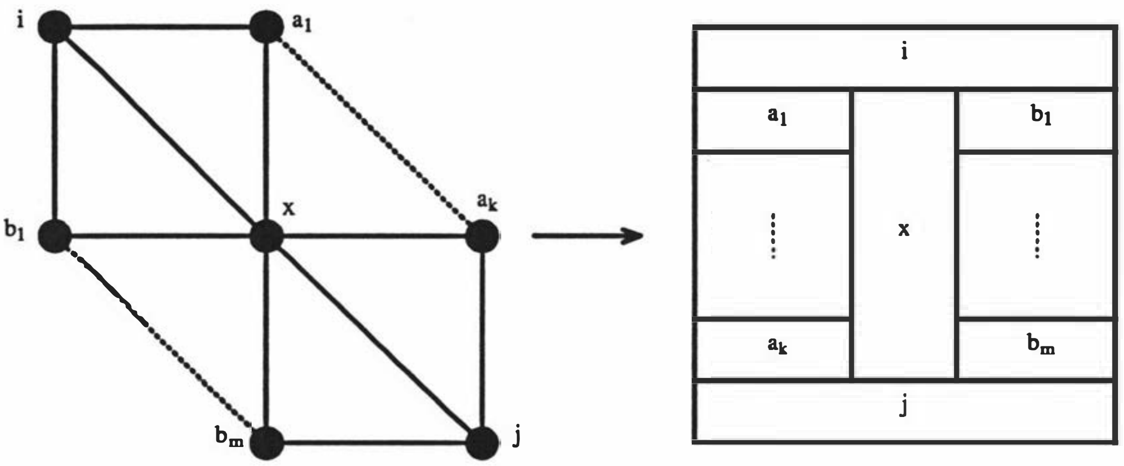


Figure 4.48: The Layout of a MPG having conditions $|D_1| = n - 2, |D_2| = 1$, and $d_x = n - 2, x \in D_2$

The layout generated in this case will have a worst case room shape of a rectangle, as long as the facilities chosen to be placed in regions A and C are non-adjacent; otherwise the worst case room shape becomes L. Note that for $n = 5$ we cannot generate a layout with the facilities in regions A and C non-adjacent.

Having considered this special case, we are now in a position to discuss the complete set of MPGs of this type. We must consider the D_1 facilities in two sets: those which are adjacent to the facility in D_2, and those which are not. The first step is to lay out the set of D_1 facilities adjacent to the D_2 facility, as we would the special case discussed previously. The remaining facilities to be laid out will all be Deltahedron type PO1 and PO2 insertions. Note that if the two facilities to be placed in regions A and C are adjacent, one must be placed as an L. This does not increase our worst case room shape as this facility will only have one placement direction within it, which is adjacent to a two joint, and hence will remain an L. Therefore we can lay out this class of MPGs with a worst case room shape of T. Also note that this set is not a subset of Deltahedron MPGs; consider the Regular Octahedron.

4.7.3 Two Mutually Adjacent Vertices Not Adjacent to the Exterior

Structural Condition: $D_2 = \{x, y\}, (x, y) \in E, \Rightarrow |D_1| = n - 3$. This class of MPGs is very similar to the the class of Section 4.7.2. The difference is that we lay out the facilities in D_2 as a block, with a horizontal wall separating them, and the facilities in D_1 to be placed in regions A and C are chosen as the facilities which are adjacent to both facilities in D_2. The remainder of the class is laid out in the same way as for the class of Section 4.7.2. MPGs in this class have layouts with at worst T room shape, and are not Deltahedron-generatable. The layout of an MPG of this type is shown in Figure 4.49, where x and y are in D_2 and i and j are chosen to be in regions A and C, as they are the two vertices adjacent to x and y.

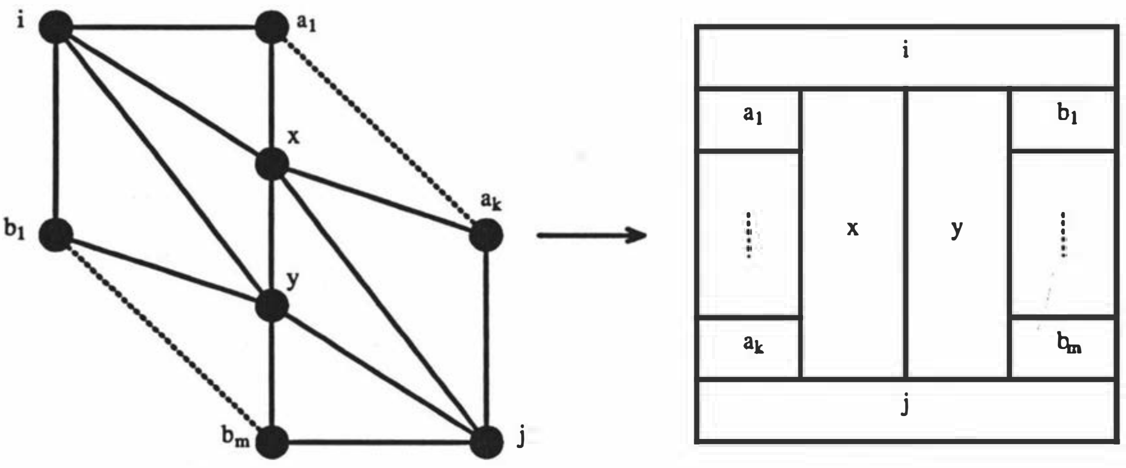


Figure 4.49: The Layout of a MPG having conditions $|D_1| = n - 3$, and $D_2 = \{x, y\}, (x, y) \in E$

4.7.4 m Distinct Vertices Not Adjacent to the Exterior

Structural Condition: $|D_2| = m, \forall\{x, y\} \in D_2, (x, y) \notin E, \Rightarrow |D_1| = n - m - 1$. The ability to be able to create dimensionalisable layouts from this class of MPGs is extremely fortuitous, since included in this class are all MPGs which exist after the contraction of groups of D_2 vertices to a single D_2 vertex when using the Contraction Algorithm. The layout process for this class is similar to that used in Section 4.7.2. We choose a vertex in D_2, and create the partial layout consisting of it and all facilities in D_1 adjacent it, as in Section 4.7.2. Now for each subsequent

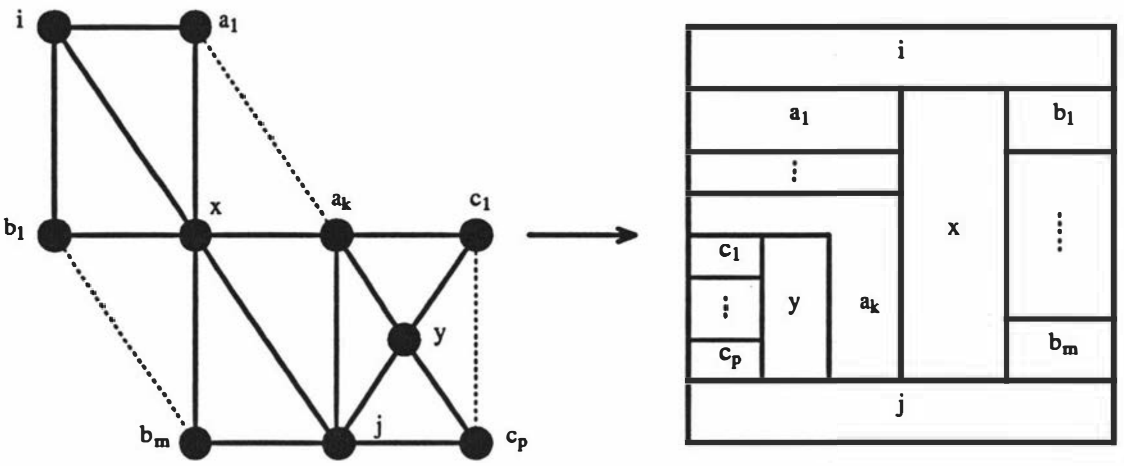


Figure 4.50: The Layout of an MPG having structural condition $|D_2| = m, \forall\{x,y\} \in D_2, (x,y) \notin E$

vertex in D_2 which is adjacent to facilities in the partial layout (each facility is either adjacent to two facilities in the partial layout, since if it were adjacent to more, it must be adjacent to another vertex in D_2, or it is adjacent to none), we can place the facility in D_2 at the 3-joint defined by the exterior and the two facilities it is adjacent to which are already placed, using Deltahedron placement directions. This allows the illegal adjacency between the exterior and the vertex in D_2, but this is rectified when we place the, as yet unplaced, facilities in D_1 that the newly placed facility from D_2 is adjacent to. These facilities are placed as a block which extends from the 3-joint defined by the exterior, one of the facilities in D_1 the newly placed facility in D_2 is adjacent to, and the newly placed facility, to the 3-joint consisting of the exterior, the newly placed facility, and the other facility it is adjacent to. This procedure is shown better by Figure 4.50, where we first lay out facility x and all the facilities in D_1 adjacent to it, followed by vertex y, and then the block of c facilities.

Once all of the facilities in D_2 are placed along with the facilities in D_1 which are adjacent to them, we can place any remaining facilities in D_1 using the Deltahedron insertion operations. MPGs of this type have a worst case room shape of T.

4.7.5 Small Problems

Structural Condition: $n \leq 8$. The final class we will consider explicitly gives a guaranteed worst case of T and is essentially a trivial class. The complete set of these MPGs and their corresponding dimensionalisable layouts are shown in Appendix A. The value of n for which dimensionalisable layouts can no longer be guaranteed is not yet known. The 12 facility problem the Regular Dodecahedron is one which currently defies the generation of a useful dimensionalisable layout.

4.7.6 Others

While there exist other possible sets of MPGs which allow dimensionalisable layouts with worst case room shape, they are not as general as the classes described in Sections 4.7.1 - 4.7.5. Examples would include some special cases of MPGs with concentric distance classes, and obviously the MPGs created using the Deltahedron and Regular Octahedron structures. Unfortunately it does not appear possible to generate useful characterisations for the complete set of MPGs - while further characterisations may be found, it is unlikely that the full set of MPGs will be able to be characterised in this way. Each new class found however does effectively solve an infinite number of problems as we have stated, due to arbitrary area conditions.

Coda

In this chapter we have described the available Graph Theoretic techniques for developing a layout. We have described, and enhanced, one method from the literature, the Contraction Algorithm, which will find a layout for any given problem instance. We have produced a correct implementation of another approach for this problem, SIMPLE, and further, have developed a completely new and different approach to finding a layout for any given problem instance, the Vertex Splitting Algorithm. Within the context of these three methods we have explored the Deltahedron Layout Algorithm which will provide a layout to a Deltahedron MPG with a worst case guarantee equivalent to a T, as well as characterising various special classes of MPG structures which allow dimensionalisable layouts to be simply constructed. Also in this chapter we have discussed the relative merits of grid layout approaches and the more generic *continuous* approaches which use the ODA or a variant. Finally we have described two other methods, the Spanning Tree method, which uses a tree rather than an MPG to structure the layout, and the Tiling Algorithm, which is more in line with more classical approaches, but guarantees to provide the $3n - 6$ adjacencies of Graph Theoretic techniques.

In the next chapter we introduce some local improvement procedures, which can be applied to a layout in an attempt to increase the efficiency and regularity of the layout, and in Chapter 6 we will perform an extensive computational experiment on these layout approaches.

Chapter 5

Layout Improvement Procedures

In this chapter we identify two fundamental procedures which enable the quantifiable improvement of a layout with respect to at least one of the regularity measures described in Section 2.5. These procedures will work on both dimensionalisable and undimensionalisable layouts. Obtaining procedures which can work on all layouts are difficult, due to the possibility of foregoing adjacencies, or area requirements, because of the presence of faultlines in undimensionalisable layouts.

5.1 Rectilinear Segment Reduction

The first improvement procedure which we introduce here takes two adjacent facilities which share at least 3 adjacent rectilinear segments, and attempts to eliminate two corners common to both facilities. This procedure is called Rectilinear Segment Reduction (RSR). A simple example of which is shown in Figure 5.1. There

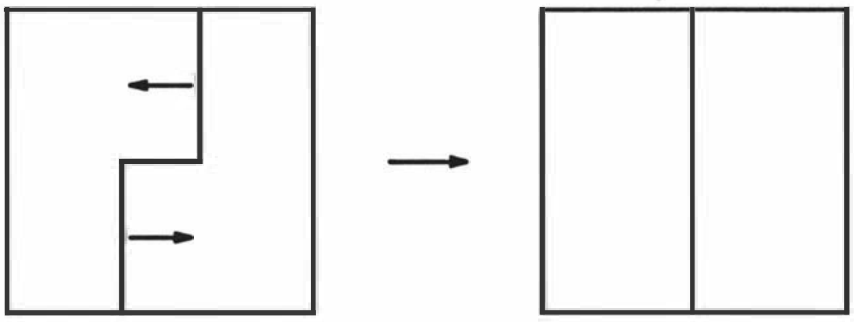


Figure 5.1: Rectilinear Segment Reduction - Simple Example

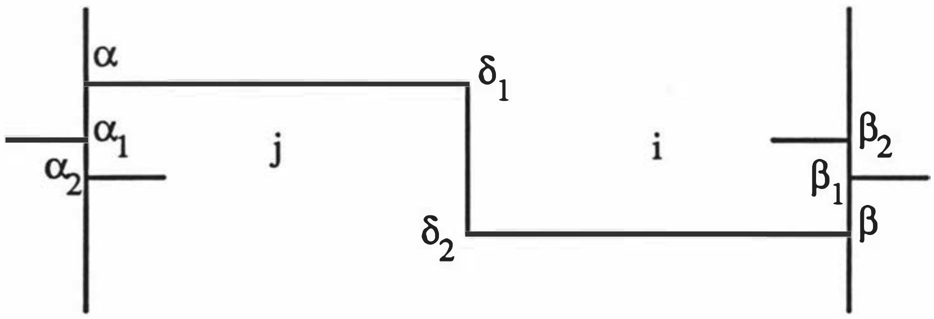


Figure 5.2: Rectilinear Segment Reduction - Conditions of Use

are two things which we must be aware of when performing RSR. The first is retaining the correct areas for each of the facilities, hence eliminating the need for re-dimensioning, and thus retaining the improvement at a local level. The second is that in changing the segments no adjacencies should be violated.

Let us assume that there exist two facilities, i and j, which share three adjacent rectilinear segments as shown in Figure 5.2, where each corner k is defined by its coordinate (x_k, y_k). We wish to find new α and β coordinates. With reference to Figure 5.2, consider two lengths, $y_1 = |y_\alpha - y_{\alpha(new)}|$ and $y_2 = |y_\beta - y_{\beta(new)}|$. Also let $x_1 = |x_\alpha - x_{\delta_1}|$, $x_2 = |x_{\delta_2} - x_\beta|$. Regardless of how much we change the segments, we are constrained by Equation 5.1.

$$y_1 x_1 = y_2 x_2 \tag{5.1}$$

Now either the 3-joint α_1 or the 3-joint α_2 will constrain the RSR on the left, and the 3-joint β_1 or the 3-joint β_2 will constrain the RSR on the right, or we will be able to successfully perform the reduction. The 3-joints which constrain the RSR are called α^*, and β^*, respectively. Let ε be a minimum specified adjacency wall length, providing Equations 5.2 and 5.3.

$$\begin{aligned} y_1 &= min(|y_\alpha - y_{\alpha^*}| - \varepsilon, \frac{x_2(|y_\beta - y_{\beta^*}| - \varepsilon)}{x_1}, \frac{x_2(|y_\alpha - y_\beta|)}{x_1 + x_2}) & (5.2) \\ y_2 &= min(|y_\beta - y_{\beta^*}| - \varepsilon, \frac{x_1(|y_\alpha - y_{\alpha^*}| - \varepsilon)}{x_2}, \frac{x_1(|y_\alpha - y_\beta|)}{x_1 + x_2}) & (5.3) \end{aligned}$$

Choosing y_1 and y_2 via Equations 5.2 and 5.3 ensures that Equation 5.1 is satisfied. Furthermore, if y_1 satisfies the first minimisation term, y_2 will satisfy its second, and vice-versa. In the first instance we are constrained by the 3-joint α^*, and in the second by β^*. If both satisfy their third term, then the reduction will be successful.

If the reduction is successful, then the number of corners of each facility will be reduced by two. This will naturally lead to an increase in regularity with respect to the number of corners; further, reducing the perimeter of the facility, will also enhance the perimeter ratio values. We cannot guarantee an increase in the bounding polygon measures, but they will not decrease. The usable space polygon ratios may either increase or decrease.

RSR is useful mainly for methods which employ the ODA or a variant of it, encompassing SIMPLE, the VSA and the Contraction Algorithm. RSR may be incorporated into the VSA by deleting three rectilinear segments formed during the recoupling of the facilities in the layout, while in SIMPLE, and the Contraction Algorithm, RSR essentially provides the α and β values used in the ODA, which is preferable to attempting to input them by hand or fixing them at a set value for all facilities.

Note that we can reduce quite complicated structures by using this one simple operation, as two facilities sharing k adjacent rectilinear segments can have their shape improved using repeated RSRs. Also note that the RSR may not be successful in attaining a single straight line segment, but it may be beneficial in making y_1 and y_2 as long as possible in order to reduce the perimeter ratio values, by reducing $||\delta_1 - \delta_2||$ (the length of the middle line segment).

Figure 5.3 shows an example of a layout constructed using the Contraction Algorithm, before and after the use of RSR, complete with the respective regularity values in Table 5.1.

If we are constrained by β_2 or α_2, as opposed to α_1 and β_1 in Figure 5.2, we may have sets of nested facilities of the form of Figure 5.4. By iteratively changing these segments as much as possible, we can eliminate such nestings, providing the constraining 3-joints allow this, even though to begin with there may seem initially to be no possible reductions.

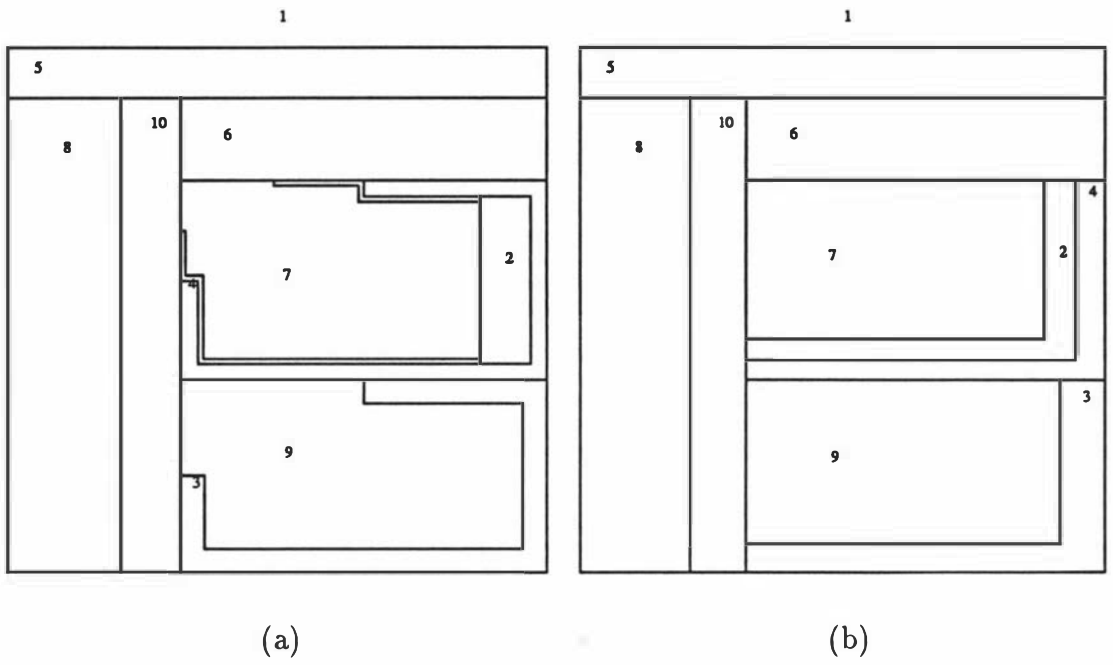


Figure 5.3: The Layouts of a 10 Facility Problem (a) Before and (b) After performing RSR

Measure	Average Values		% Increase
	Before RSR	After RSR	
Enclosed Rectangle	0.8255	0.8568	3.8%
Enclosed Golden Rectangle	0.3385	0.3631	7.3%
Enclosed Square	0.2151	0.2244	4.3%
Bounding Rectangle	0.7143	0.7367	3.1%
Bounding Golden Rectangle	0.3643	0.3755	3.1%
Bounding Square	0.2252	0.2321	3.1%
Perimeter	0.6049	0.6629	9.6%
Number of Corners	8.2222	4.6667	-43.2%

Table 5.1: Changes in Selected Regularity Values for Layouts in Figure 5.3

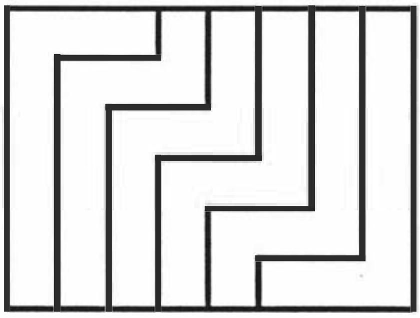


Figure 5.4: An example where there are no immediate RSR reductions, but iterative reduction may be successful

5.2 Linear Transformation

This improvement procedure is a global improvement, yet still requires minimal effort. We consider a Linear Transformation (LT) of the layout. Fundamental components of this linear transformation are that area (as the determinant of the transformation matrix is 1) and shape are preserved, allowing us to never need be concerned about redimensioning, or about losing adjacencies.

Consider every 2- and 3-joint within a layout with coordinates (x_i, y_i). Let (x_0, y_0) denote the top left coordinate of any layout *i.e.* $x_0 \leq x_i$, and $y_0 \leq y_i$, $\forall i$. Then we can perform a linear transformation of the layout as shown in Equation 5.4, where a is a scale parameter.

$$[x_i^{new}, y_i^{new}] = [(x_i - x_0), (y_i - y_0)] \begin{bmatrix} a & 0 \\ 0 & \frac{1}{a} \end{bmatrix} + [x_0, y_0], \quad a > 0 \qquad (5.4)$$

The value of a corresponds to the factor by which we *stretch* the layout. The determination of the scale parameter is the key to the success of this improvement. Firstly note that the enclosed rectangle, bounding rectangle and number of corners regularity values are unchanged by this transformation. We can optimize the value of a with respect to one of the other regularity measures only. It is difficult to predict the behaviour of the other enclosing and bounding polygon measures, but the perimeter ratio follows a smooth curve, leading up to a point where the perimeter is minimised for a particular shape. Therefore we have chosen the perimeter ratio with which to optimize a.

As an example of the effect of a on a facility, consider Figure 5.5. We see that

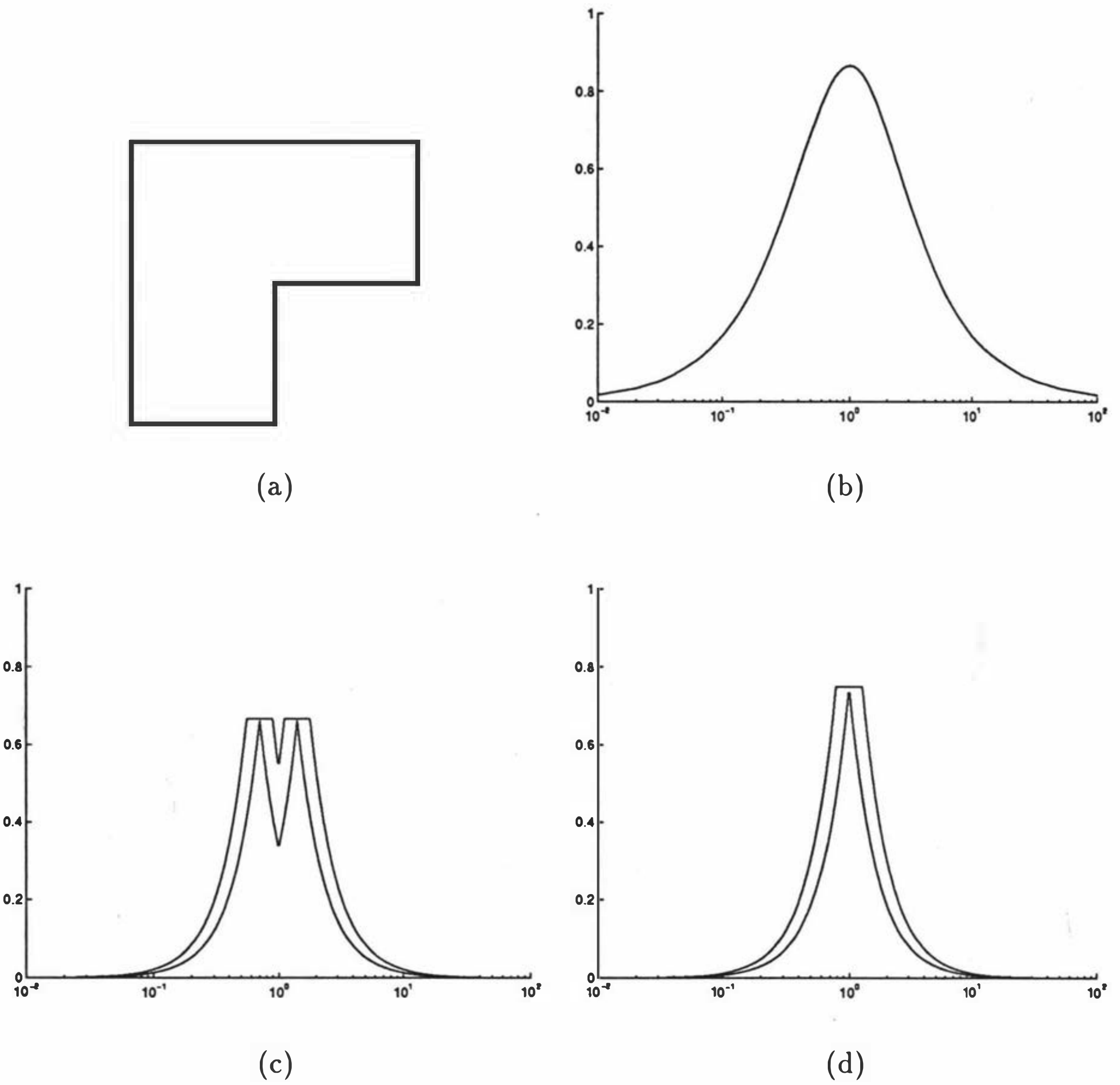


Figure 5.5: An L-shaped facility (a) and graphs of the regularity measures as a varies; (b) perimeter ratio; (c) bounding polygon ratios; (d) enclosed polygon ratios

the perimeter ratio is a smooth robust curve; the golden rectangle polygon ratios *bound* the square polygon ratio. Note that the enclosed polygon measures have two local optima corresponding to the two values of a where each of the parts of the L become squares, or golden rectangles, respectively.

Note that we do not consider maximising the average perimeter ratio, as this may lead to very elongated layouts, which are impractical. For example, consider the layout in Figure 5.6, where there are $n+1$ facilities all of area 1, and the top facility has degree n.

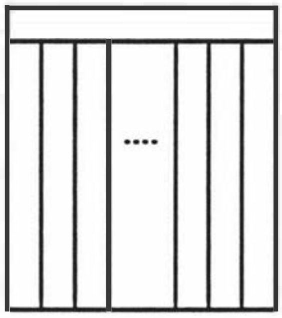


Figure 5.6: An example where maximising the average perimeter ratio will result in an impractical layout

If we maximize the average perimeter ratio, the optimal value of a, will be where all facilities except the top facility will be squares, having perimeter ratios of 1. This means that the average perimeter ratio for the layout of Figure 5.6 is given by Equation 5.5.

$$\frac{1}{n+1}\sum_{i=1}^{n+1}\frac{4\sqrt{a_i}}{p_i} = \frac{1}{n+1}(n + \frac{4}{2n+\frac{1}{2n}}) \tag{5.5}$$

Now as $n \to \infty$, in Equation 5.5 the average perimeter ratio $\to 1$, yet the perimeter ratio of the top facility $\to 0$, leading to an impractical layout where the top facility cannot be used. Therefore when using the linear transformation, we seek to maximise the minimum perimeter ratio. This ensures that the layouts are attempting to maximize usability. This example also highlights the necessity for considering minimum and maximum statistics, in tandem with average statistics, in general.

Consider the example we saw in Figure 5.3. When we perform the linear transformation, the changes in the regularity values given in Table 5.2.

Measure	Average Values		% Increase
	Before LT	After LT	
Enclosed Golden Rectangle	0.3631	0.3837	5.7%
Enclosed Square	0.2244	0.2371	5.7%
Bounding Golden Rectangle	0.3755	0.4098	9.1%
Bounding Square	0.2321	0.2537	9.3%
Perimeter	0.6629	0.6689	1.0%
Measure	Minimum Values		% Increase
	Before LT	After LT	
Enclosed Golden Rectangle	0.1043	0.1185	13.6%
Enclosed Square	0.0645	0.0733	13.6%
Bounding Golden Rectangle	0.1553	0.1753	12.9%
Bounding Square	0.0960	0.1091	13.7%
Perimeter	0.4013	0.4079	1.6%

Table 5.2: Changes in Selected Regularity Values for the Layout in Figure 5.3(b) after application of the LT

Coda:

In this chapter, we have introduced two methods for improving the regularity values of a layout. We have been able to implement these improvement procedures locally, and hence retain the duality with the MPG and area specifications. These improvement procedures can prove very effective, as we shall see in Section 6.4. The next chapter, in fact, more fully examines the methods described in Chapter 4, via an extensive computational experiment which considers a variety of different attributes of the layouts.

Chapter 6

Computer Implementation

In this chapter we provide the main computational study of the thesis. As far as we are aware, this is the first-in depth analysis that has been done on layouts in the GTFLP. We will examine the usefulness of each of the respective methods, comparing layouts generated by each method on a set of test problems, and investigate the trade off between attempting to maximize regularity and adjacency concurrently. In tables of this chapter and Chapter 7, we employ a number of abbreviations; those not previously described are found for reference in Table 6.1. Further the VSA, the Contraction Algorithm, and SIMPLE commonly have bracketed abbreviations relating to improvement procedures, and/or initialisation.

CA	The Contraction Algorithm
TA(γ)	The Tiling Algorithm with worst case facility shape γ
ST	The Spanning Tree Algorithm
R	The initialisation of Rinsma
D	The Deltahedron initialisation
NI	Methods used with no improvements
I	Methods used with improvements

Table 6.1: Commonly Used Abbreviations

6.1 Generation of Test Problems

The set of test problems generated consisted of 45 problems on each of n=10, 15, 20, 25, 30, 40, and 50. Test problems between 10 and 30 facilities were used to analyse the layouts, while those test problems of size 40 and 50 were used at times in order to gain more understanding of the asymptotic affects of some parameters. It was felt that generating layouts of size at most size 30 was large enough to gain insight into what we might expect in the general case. Within each set of 45 benefit matrices for each problem size, 15 have 50% non zero elements, 15 have 75% non zero elements, and the remaining 15 have 100% non zero elements on average, and all problem sets are generated so that the average edge weight was 112.5. This ensured that any averaging of adjacency benefits was unbiased, and furthermore, given a problem instance, we could determine the expected value of the total adjacency benefit. Adjacency benefits were randomly chosen from a uniform distribution, with range dependent on the percentage of non-zeros required - for example, problems with 75% non-zeros had their benefits determined from the uniform distribution with range $(-99, 300)$, with all negative benefits generated set to zero, in order to generate our required percentage of non zeros. Further, using the assumption that any negative entries in the adjacency matrix could be eliminated by adding a large enough constant to each entry in the matrix [39], it was not considered worthwhile to allow negative entries in the benefit matrix. Within each set of 15, we have 5 problems all of area 40, 5 with areas in the range 30-50, and 5 with areas in the range 10-70. This ensured that layout methods were well tested on a variety of different area specifications, so as to eliminate any advantage a method may have had on a certain set of area requirements. Areas also were randomly chosen from a uniform distribution with ranges equating to those specified above.

Following the generation of the benefit matrices, we were required to generate highly weighted MPGs, in order to firstly create layouts from the MPGs, and secondly to be able to use the adjacency benefits of the MPGs when comparing with the regularity values. To generate the MPGs, we implemented TESSA as a construction phase, and followed that with a Tabu search routine using the Γ-operation. The Tabu Search routine was set to terminate when no improvement had been found in the last $100n$ iterations. It was assumed that TESSA, being able to create any MPG on n vertices, would give unbiased initial starting points, whereas using Deltahedron as the initial construction may have affected the final MPG. Following the results of [13], it was tacitly assumed that this method of generating the MPGs would generate highly weighted MPGs. Table 6.2 gives the results of the generation. All tests and generations of test problems were carried out on a Sparc Station ELC.

	Averages			
n	Upper Bound	MPG Value	% of Upper Bound	Time(s)
10	4582.24	4258.02	93%	5.05
15	8676.84	7463.27	86%	26.67
20	13732.16	11550.47	84%	91.63
25	18432.64	15267.22	83%	243.20
30	23077.96	18791.44	81%	589.65

Table 6.2: Generation of arbitrary MPGs

6.2 Existence

Our first experiment deals with the ability of the given layout methods to successfully dualise a given MPG. The results of this are seen in Table 6.3.

Method	n=10	n=15	n=20	n=25	n=30
VSA(D)	100%	100%	100%	100%	100%
VSA(DI)	100%	100%	100%	100%	100%
VSA(R)	100%	100%	100%	100%	100%
VSA(RI)	100%	100%	100%	100%	100%
CA(D)	100%	100%	100%	100%	100%
CA(DI)	100%	100%	100%	100%	100%
CA(R)	100%	100%	100%	100%	100%
CA(RI)	100%	100%	100%	100%	100%
SIMPLE	100%	100%	0%	0%	0%
SIMPLE(I)	100%	100%	27%	2%	0%
Deltahedron	44%	2%	0%	0%	0%

Table 6.3: Existence of Solutions to Test Problems

We see that the VSA, and the Contraction Algorithm are able to successfully dualise an arbitrary MPG with or without improvements for $n \leq 30$, regardless of whether the Deltahedron or Rinsma starting points are used. In fact further experiments with $n = 40$ and $n = 50$, showed that these methods could successfully dualise MPGs with $n \leq 50$. Unfortunately due to computer limitations, we were unable to proceed higher. However, it seems reasonable to hypothesise, that $n = 50$ represents the top of the range that the unimproved versions could be expected to handle. Regarding the improved versions, it would be difficult to estimate the highest value of n for which they could successfully dualise. The VSA will be limited eventually by the sheer size of the number of facilities that it will create during the splitting phase, while the Contraction Algorithm will eventually be constrained by the number of corners generated during creation of later facilities using the ODA.

SIMPLE showed disappointing results, failing to make it past $n = 15$, for the unimproved version, and only managing one problem out of the 45 for $n = 25$. SIMPLE is, in fact, constrained by the ODA, in that once a facility becomes irregular in the layout, shapes of facilities subsequently adjacent to it become worse. This is the same phenomenon which would be exhibited by the Contraction Algorithm at the top of its facility size range. The performance of Deltahedron was not surprising since, as n increases, the number of Deltahedron MPGs on n vertices which exist relative to the total number of MPGs which exist on n vertices tends to zero.

In the next two sections we concentrate our thoughts on the VSA and the Contraction Algorithm. Since these two methods are guaranteed to dualise all of our test problems, and beyond, it seems worthwhile to give them more thought.

6.3 Starting Points

One of the first interesting questions is whether the two starting points that exist for constructing the layouts by these methods, *i.e.* Deltahedron and Rinsma, perform similarly. Tables 6.4 - 6.8 provide data on how the methods perform using the different starting methods. Comparison is made on four aspects: time, which allows comparison of the speeds of the respective methods; number of corners and bounding rectangle regularity measures, which give an indication of the shape irregularity that exists; and the perimeter ratio measure, which provides an estimate of the usable space available. At each step we are examining our hypothesis H_O against the alternative H_A, shown in Equation 6.1, where Ave_R is the average value of the measure we are considering using the Rinsma starting point, and similarly for Ave_D and the Deltahedron starting point. The percentage increase, *i.e.* the relative increase in the measure incurred between the Deltahedron and Rinsma initialisation routines, is evaluated using Equation 6.2.

$$H_O : Ave_R = Ave_D;\ H_A : Ave_R > Ave_D \text{ or } Ave_R < Ave_D \tag{6.1}$$

$$\%Inc = 100 * (\frac{Ave_R - Ave_D}{Ave_D})\% \tag{6.2}$$

We examine these hypotheses and, for each test, either cannot reject the null hypothesis H_O, or accept the alternative hypothesis at the highest significance level possible; 90%, 95% or 99%.

Examining the tables, we see that for $n = 10$ (Table 6.4) VSA using Deltahedron takes a significant amount of extra time to complete than VSA using Rinsma, in both the improved and unimproved cases, yet the Contraction Algorithm shows no significant difference between the starting points timewise. Furthermore, VSA using Rinsma is significantly better than VSA using Deltahedron according to the two shape regularity measures, while the perimeter ratio has essentially constant values between the two starting points. Note that the Contraction Algorithm appears to be unaffected by the starting points, showing only a slight significant difference in the number of corners for the Contraction Algorithm with improvements. The results are consistent with expectations; the difference between the two starting points should be minimal, as they perform essentially the same steps. We see essentially the same patterns for $n = 15$ (Table 6.5). Table 6.6 exhibits the results for $n = 20$, and we now start to see some changes in the performance of the starting methods. There is suddenly no significant difference in the times of either method, and furthermore the regularity values appear to be coming into a more similar line. There is only one difference of 99% significance, being that the Contraction Algorithm with improvements averages more corners per facility using the Rinsma initialisation. Examining Table 6.7 for $n = 25$, we see that there are no significant differences at the 99% level, and it appears that the starting points are essentially equivalent. Table 6.8 appears to show the start of an interesting pattern, however. The times for the Rinsma starting point using the VSA having eclipsed those for Deltahedron. However, although the differences are not significant, the

Measure	Method	Ave_D	Ave_R	$Ave_R - Ave_D$	% Inc
	VSA(NI)	0.28	0.24	-0.04	-14.88
Times(s)				$Ave_D > Ave_R$ (99%)	
	VSA(I)	0.46	0.38	-0.08	-16.64
				$Ave_D > Ave_R$ (99%)	
	CA(NI)	0.76	0.77	0.00	0.64
				Cannot Reject H_0	
	CA(I)	0.48	0.50	0.02	4.21
				Cannot Reject H_0	
	VSA(NI)	6.97	6.01	-0.96	-13.81
Ave C_i				$Ave_D > Ave_R$ (99%)	
	VSA(I)	6.61	5.70	-0.90	-13.68
				$Ave_D > Ave_R$ (99%)	
	CA(NI)	11.15	11.19	0.03	0.31
				Cannot Reject H_0	
	CA(I)	6.28	6.65	0.37	5.90
				$Ave_R > Ave_D$ (90%)	
	VSA(NI)	0.6813	0.6997	0.0184	2.70
Ave $4\sqrt{a_i}/P_i$				Cannot Reject H_0	
	VSA(I)	0.7147	0.7130	-0.0017	-0.24
				Cannot Reject H_0	
	CA(NI)	0.6024	0.6041	0.0016	0.27
				Cannot Reject H_0	
	CA(I)	0.6895	0.6794	-0.0100	-1.46
				Cannot Reject H_0	
	VSA(NI)	0.6558	0.7135	0.0577	8.79
Ave $a_i/\bar{R}_i$				$Ave_R > Ave_D$ (99%)	
	VSA(I)	0.6895	0.7308	0.0413	6.00
				$Ave_R > Ave_D$ (95%)	
	CA(NI)	0.6427	0.6331	-0.0096	-1.49
				Cannot Reject H_0	
	CA(I)	0.7238	0.7006	-0.0232	-3.21
				Cannot Reject H_0	

Table 6.4: Performance of Starting Points for $n = 10$

Measure	Method	Ave_D	Ave_R	$Ave_R - Ave_D$	% Inc
Times(s)	VSA(NI)	1.04	0.76	-0.28	-26.96
				$Ave_D > Ave_R$ (95%)	
	VSA(I)	1.51	1.09	-0.41	-27.36
				$Ave_D > Ave_R$ (95%)	
	CA(NI)	3.46	3.42	-0.05	-1.30
				Cannot Reject H_0	
	CA(I)	1.46	1.41	-0.05	-3.36
				Cannot Reject H_0	
Ave C_i	VSA(NI)	8.93	8.06	-0.87	-9.70
				$Ave_D > Ave_R$ (95%)	
	VSA(I)	7.73	7.00	-0.73	-9.49
				$Ave_D > Ave_R$ (99%)	
	CA(NI)	16.55	16.45	-0.10	-0.58
				Cannot Reject H_0	
	CA(I)	9.22	9.37	0.15	1.58
				Cannot Reject H_0	
Ave $4\sqrt{a_i}/P_i$	VSA(NI)	0.5738	0.5915	0.0177	3.09
				Cannot Reject H_0	
	VSA(I)	0.6169	0.6178	0.0008	0.13
				Cannot Reject H_0	
	CA(NI)	0.4475	0.4610	0.0136	3.03
				Cannot Reject H_0	
	CA(I)	0.5621	0.5709	0.0087	1.55
				Cannot Reject H_0	
Ave $a_i/\bar{R}_i$	VSA(NI)	0.5430	0.5889	0.0460	8.46
				$Ave_R > Ave_D$ (99%)	
	VSA(I)	0.5751	0.6165	0.0414	7.19
				$Ave_R > Ave_D$ (99%)	
	CA(NI)	0.4596	0.4581	-0.0015	-0.32
				Cannot Reject H_0	
	CA(I)	0.5679	0.5659	-0.0020	-0.36
				Cannot Reject H_0	

Table 6.5: Performance of Starting Points for $n = 15$

Measure	Method	Ave_D	Ave_R	$Ave_R - Ave_D$	% Inc
Times(s)	VSA(NI)	3.10	3.32	0.22	7.12
				Cannot Reject H_0	
	VSA(I)	4.12	4.08	-0.04	-0.98
				Cannot Reject H_0	
	CA(NI)	7.45	7.42	-0.03	-0.46
				Cannot Reject H_0	
	CA(I)	2.63	2.78	0.15	5.69
				Cannot Reject H_0	
Ave C_i	VSA(NI)	10.55	9.93	-0.62	-5.83
				Cannot Reject H_0	
	VSA(I)	8.51	8.15	-0.36	-4.21
				Cannot Reject H_0	
	CA(NI)	19.84	20.02	0.18	0.91
				Cannot Reject H_0	
	CA(I)	10.02	10.90	0.88	8.80
				$Ave_R > Ave_D$ (99%)	
Ave $4\sqrt{a_i}/P_i$	VSA(NI)	0.5343	0.5269	-0.0073	-1.37
				Cannot Reject H_0	
	VSA(I)	0.5886	0.5635	-0.0250	-4.25
				$Ave_D > Ave_R$ (95%)	
	CA(NI)	0.3644	0.3733	0.0089	2.44
				Cannot Reject H_0	
	CA(I)	0.5284	0.5184	-0.0099	-1.88
				Cannot Reject H_0	
Ave $a_i/\bar{R}_i$	VSA(NI)	0.4820	0.4892	0.0072	1.49
				Cannot Reject H_0	
	VSA(I)	0.5286	0.5368	0.0082	1.54
				Cannot Reject H_0	
	CA(NI)	0.3503	0.3497	-0.0005	-0.15
				Cannot Reject H_0	
	CA(I)	0.5142	0.4917	-0.0225	-4.37
				$Ave_D > Ave_R$ (90%)	

Table 6.6: Performance of Starting Points for $n = 20$

Measure	Method	Ave_D	Ave_R	$Ave_R - Ave_D$	% Inc
Times(s)	VSA(NI)	7.10	7.29	0.19	2.69
				Cannot Reject H_0	
	VSA(I)	9.75	8.38	-1.37	-14.04
				Cannot Reject H_0	
	CA(NI)	12.28	11.81	-0.47	-3.82
				Cannot Reject H_0	
	CA(I)	5.16	5.34	0.18	3.47
				Cannot Reject H_0	
Ave C_i	VSA(NI)	11.86	11.37	-0.49	-4.11
				Cannot Reject H_0	
	VSA(I)	9.34	9.24	-0.11	-1.13
				Cannot Reject H_0	
	CA(NI)	21.38	21.25	-0.13	-0.62
				Cannot Reject H_0	
	CA(I)	11.57	12.38	0.81	6.99
				$Ave_R > Ave_D$ (90%)	
Ave $4\sqrt{a_i}/P_i$	VSA(NI)	0.4822	0.4723	-0.0099	-2.05
				Cannot Reject H_0	
	VSA(I)	0.5394	0.5170	-0.0223	-4.14
				$Ave_D > Ave_R$ (95%)	
	CA(NI)	0.3341	0.3419	0.0078	2.34
				Cannot Reject H_0	
	CA(I)	0.4621	0.4532	-0.0090	-1.94
				Cannot Reject H_0	
Ave $a_i/\bar{R}_i$	VSA(NI)	0.4289	0.4386	0.0097	2.26
				Cannot Reject H_0	
	VSA(I)	0.4779	0.4790	0.0011	0.24
				Cannot Reject H_0	
	CA(NI)	0.3255	0.3185	-0.0070	-2.14
				Cannot Reject H_0	
	CA(I)	0.4463	0.4258	-0.0205	-4.60
				Cannot Reject H_0	

Table 6.7: Performance of Starting Points for $n = 25$

Measure	Method	Ave_D	Ave_R	$Ave_R - Ave_D$	% Inc
Times(s)	VSA(NI)	17.26	22.79	5.52	32.00
				$Ave_R > Ave_D$ (90%)	
	VSA(I)	23.62	25.73	2.10	8.90
				Cannot Reject H_0	
	CA(NI)	26.09	24.76	-1.33	-5.11
				Cannot Reject H_0	
	CA(I)	10.37	9.38	-1.00	-9.61
				Cannot Reject H_0	
Ave C_i	VSA(NI)	13.70	13.30	-0.40	-2.91
				Cannot Reject H_0	
	VSA(I)	10.41	10.18	-0.23	-2.24
				Cannot Reject H_0	
	CA(NI)	25.49	24.92	-0.57	-2.24
				Cannot Reject H_0	
	CA(I)	14.27	14.13	-0.14	-1.00
				Cannot Reject H_0	
Ave $4\sqrt{a_i}/P_i$	VSA(NI)	0.4394	0.4204	-0.0190	-4.32
				$Ave_D > Ave_R$ (95%)	
	VSA(I)	0.4928	0.4645	-0.0283	-5.74
				$Ave_D > Ave_R$ (99%)	
	CA(NI)	0.2654	0.2720	0.0066	2.48
				Cannot Reject H_0	
	CA(I)	0.4130	0.4126	-0.0005	-0.11
				Cannot Reject H_0	
Ave $a_i/\bar{R}_i$	VSA(NI)	0.3958	0.3893	-0.0064	-1.63
				Cannot Reject H_0	
	VSA(I)	0.4382	0.4326	-0.0056	-1.27
				Cannot Reject H_0	
	CA(NI)	0.2362	0.2377	0.0015	0.65
				Cannot Reject H_0	
	CA(I)	0.3810	0.3760	-0.0050	-1.31
				Cannot Reject H_0	

Table 6.8: Performance of Starting Points for $n = 30$

average number of corners for each facility is less using the Rinsma starting point, yet the perimeter ratios appear to indicate that VSA using Deltahedron will produce better perimeter ratio values. Figure 6.1 shows some of these trends more effectively.

These results appear to show that it is worthwhile to consider each starting point in order to obtain a layout to a given MPG, as there it appears that either could perform better on any given problem.

6.4 Improvement vs Non-Improvement Experiment

In this section we explore how well the improvements introduced in Chapter 5 actually perform in practise. We again present our null hypothesis H_O, and alternative hypothesis H_A, shown in Equation 6.3, where Ave_{NI} is the average value for the methods without improvements (non-improved), and similarly for Ave_I, and the methods with improvements.

$$H_O : Ave_{NI} = Ave_I;\ H_A : Ave_I > Ave_{NI} \text{ or } Ave_I < Ave_{NI} \tag{6.3}$$

Tables 6.9 - 6.13 exhibit the results for this experiment, where we again measure the results against the four measures used previously, but now the percentage increase column entries represent the relative increase in measure incurred between not having and having improvements, as evaluated using Equation 6.4.

$$\%Inc = 100 * (\frac{Ave_I - Ave_{NI}}{Ave_{NI}})\% \tag{6.4}$$

Once again the results confirm expectations, obviously the values will improve, otherwise the improvements wouldn't be necessary. However with reference to the tables, and also to Figure 6.2, there are a number of more interesting observations that can be made. Firstly with respect to the times that the improved versions required, it is worthwhile observing that as n increased for the VSA, the proportion of time spent on the improvements decreases, until for $n = 30$, the increase in time

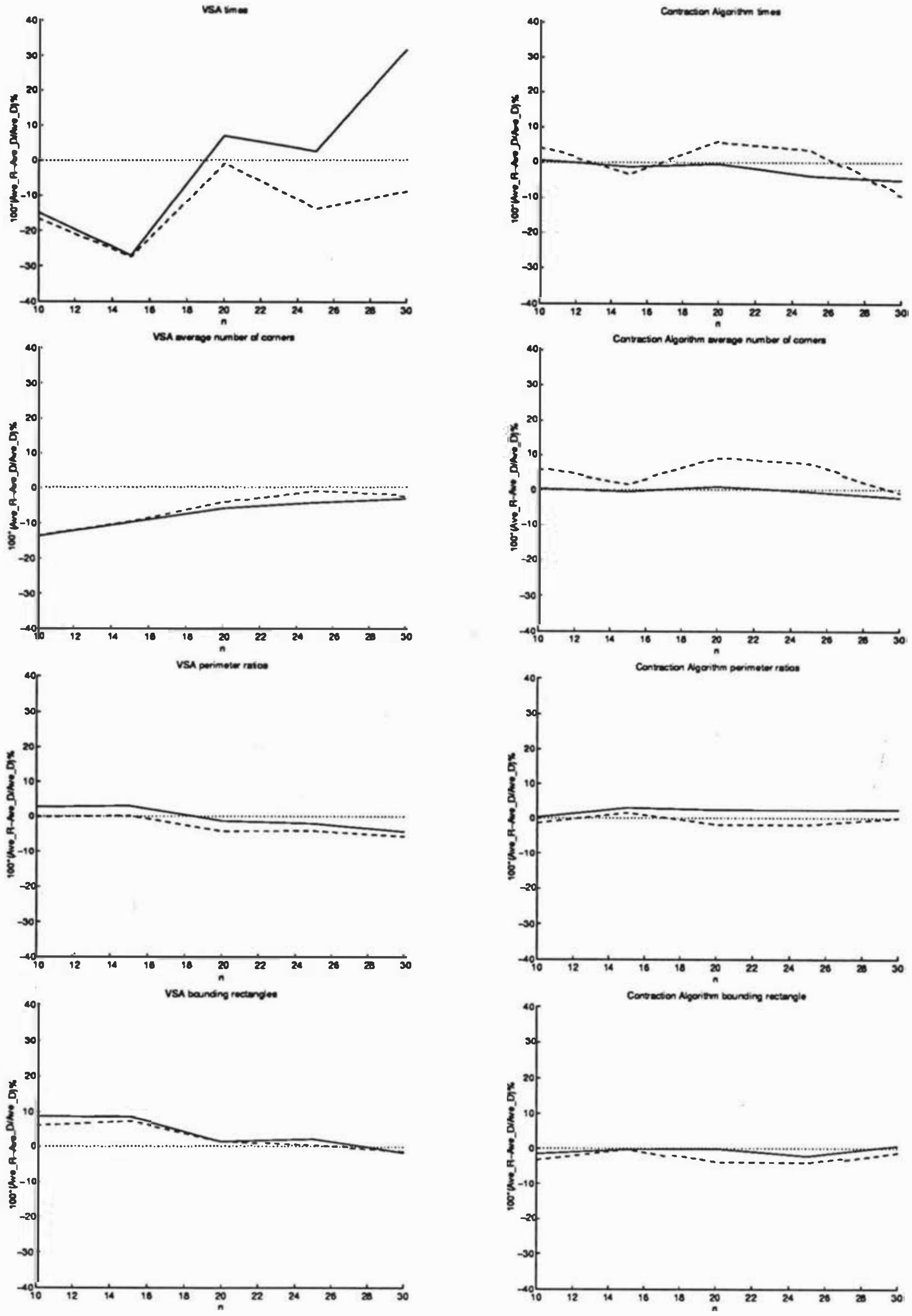


Figure 6.1: Graphs of the percentage difference between Rinsma and Deltahedron starting points for the four measures used; dashed line is the unimproved versions, and the solid line is the improved versions

for the improvements was not significant. This observation pales in comparison, however, to the fact that the Contraction Algorithm was about 37% faster *with* improvements than without for $n = 10$, and for larger n, even better at around 60%. This means that the running times were reduced by more than half the time over the unimproved version. Although this result is initially startling, consideration of the reduced numbers of corners produced for each facility meant that, during the running of the ODA, there were fewer corners to search through at various stages in the implementation, thereby significantly reducing the complexity of facility placement in the layout.

The performance of the regularity measures was also as expected. The VSA performance showed steady improvement over all three measures and, at $n = 30$, the average number of corners had decreased by more than 20%, while the other two measures had both increased by more than 10%. By $n = 15$, all three measures had changed significantly. The Contraction Algorithm made tremendous advances with the regularity measures, with all three having significant changes at the 99% level, throughout the range of n. Furthermore the number of corners was steady at a reduction of more than 40%, while the other two measures increased from around 12% for $n = 10$, to the giddy heights of around 55% increase for $n = 30$.

These results conclusively support the effectiveness of the two improvement procedures of Chapter 5, in both efficiency of implementation and in the increase in regularity. One important note involves contrasting results between the VSA, and the Contraction Algorithm. The VSA performs both improvements at the completion of the layout, while the Contraction Algorithm implements RSR as each facility is placed in the layout, with the global transformation coming at the completion of the layout. The effect of performing RSR as each facility is placed in the Contraction Algorithm is the reason for the better performance of the improvements, as each RSR on a facility in a partially constructed Contraction Algorithm layout impinges not only on the regularity of that facility, but on subsequent facility placements in the layout. VSA, on the other hand, is constrained by its specialised version of ODA, wherein each facility is placed in the layout, with the minimum number of corners possible, and it is not until we recombine the facilities that we can identify RSRs to eradicate redundant corners. In this case, the RSR will only affect the two facilities directly involved in the RSR, and hence the impact of each RSR in a VSA layout is not as great.

Measure	Method	Ave_{NI}	Ave_I	$Ave_I - Ave_{NI}$	% Inc
Times(s)	VSA(D)	0.28	0.46	0.18	65.33
				$Ave_I > Ave_{NI}$ (99%)	
	VSA(R)	0.24	0.38	0.15	61.91
				$Ave_I > Ave_{NI}$ (99%)	
	CA(D)	0.76	0.48	-0.28	-37.00
				$Ave_{NI} > Ave_I$ (99%)	
	CA(R)	0.77	0.50	-0.27	-34.76
				$Ave_{NI} > Ave_I$ (99%)	
Ave C_i	VSA(D)	6.97	6.61	-0.37	-5.24
				$Ave_{NI} > Ave_I$ (90%)	
	VSA(R)	6.01	5.70	-0.31	-5.09
				$Ave_{NI} > Ave_I$ (90%)	
	CA(D)	11.15	6.28	-4.87	-43.67
				$Ave_{NI} > Ave_I$ (99%)	
	CA(R)	11.19	6.65	-4.53	-40.53
				$Ave_{NI} > Ave_I$ (99%)	
Ave $4\sqrt{a_i}/P_i$	VSA(D)	0.6813	0.7147	0.0334	4.90
				$Ave_I > Ave_{NI}$ (95%)	
	VSA(R)	0.6997	0.7130	0.0133	1.90
				Cannot Reject H_0	
	CA(D)	0.6024	0.6895	0.0870	14.45
				$Ave_I > Ave_{NI}$ (99%)	
	CA(R)	0.6041	0.6794	0.0754	12.48
				$Ave_I > Ave_{NI}$ (99%)	
Ave $a_i/\bar{R}_i$	VSA(D)	0.6558	0.6895	0.0337	5.13
				$Ave_I > Ave_{NI}$ (95%)	
	VSA(R)	0.7135	0.7308	0.0173	2.43
				Cannot Reject H_0	
	CA(D)	0.6427	0.7238	0.0810	12.61
				$Ave_I > Ave_{NI}$ (99%)	
	CA(R)	0.6331	0.7006	0.0674	10.65
				$Ave_I > Ave_{NI}$ (99%)	

Table 6.9: Performance of Improvements for $n = 10$

Measure	Method	Ave_{NI}	Ave_I	$Ave_I - Ave_{NI}$	% Inc
Times(s)	VSA(D)	1.04	1.51	0.46 $Ave_I > Ave_{NI}$ (95%)	44.41
	VSA(R)	0.76	1.09	0.33 $Ave_I > Ave_{NI}$ (99%)	43.61
	CA(D)	3.46	1.46	-2.00 $Ave_{NI} > Ave_I$ (99%)	-57.80
	CA(R)	3.42	1.41	-2.00 $Ave_{NI} > Ave_I$ (99%)	-58.68
Ave C_i	VSA(D)	8.93	7.73	-1.20 $Ave_{NI} > Ave_I$ (99%)	-13.44
	VSA(R)	8.06	7.00	-1.07 $Ave_{NI} > Ave_I$ (99%)	-13.23
	CA(D)	16.55	9.22	-7.33 $Ave_{NI} > Ave_I$ (99%)	-44.27
	CA(R)	16.45	9.37	-7.09 $Ave_{NI} > Ave_I$ (99%)	-43.06
Ave $4\sqrt{a_i}/P_i$	VSA(D)	0.5738	0.6169	0.0432 $Ave_I > Ave_{NI}$ (99%)	7.52
	VSA(R)	0.5915	0.6178	0.0262 $Ave_I > Ave_{NI}$ (95%)	4.43
	CA(D)	0.4475	0.5621	0.1147 $Ave_I > Ave_{NI}$ (99%)	25.63
	CA(R)	0.4610	0.5709	0.1098 $Ave_I > Ave_{NI}$ (99%)	23.82
Ave $a_i/\bar{R}_i$	VSA(D)	0.5430	0.5751	0.0322 $Ave_I > Ave_{NI}$ (95%)	5.93
	VSA(R)	0.5889	0.6165	0.0276 $Ave_I > Ave_{NI}$ (90%)	4.68
	CA(D)	0.4596	0.5679	0.1083 $Ave_I > Ave_{NI}$ (99%)	23.57
	CA(R)	0.4581	0.5659	0.1078 $Ave_I > Ave_{NI}$ (99%)	23.52

Table 6.10: Performance of Improvements for $n = 15$

Measure	Method	Ave_{NI}	Ave_I	$Ave_I - Ave_{NI}$	% Inc
Times(s)	VSA(D)	3.10	4.12	1.02 $Ave_I > Ave_{NI}$ (90%)	32.85
	VSA(R)	3.32	4.08	0.76 Cannot Reject H_0	22.80
	CA(D)	7.45	2.63	-4.82 $Ave_{NI} > Ave_I$ (99%)	-64.68
	CA(R)	7.42	2.78	-4.64 $Ave_{NI} > Ave_I$ (99%)	-62.50
Ave C_i	VSA(D)	10.55	8.51	-2.04 $Ave_{NI} > Ave_I$ (99%)	-19.38
	VSA(R)	9.93	8.15	-1.79 $Ave_{NI} > Ave_I$ (99%)	-17.99
	CA(D)	19.84	10.02	-9.82 $Ave_{NI} > Ave_I$ (99%)	-49.50
	CA(R)	20.02	10.90	-9.12 $Ave_{NI} > Ave_I$ (99%)	-45.55
Ave $4\sqrt{a_i}/P_i$	VSA(D)	0.5343	0.5886	0.0543 $Ave_I > Ave_{NI}$ (99%)	10.16
	VSA(R)	0.5269	0.5635	0.0366 $Ave_I > Ave_{NI}$ (99%)	6.94
	CA(D)	0.3644	0.5284	0.1639 $Ave_I > Ave_{NI}$ (99%)	44.99
	CA(R)	0.3733	0.5184	0.1451 $Ave_I > Ave_{NI}$ (99%)	38.86
Ave $a_i/\bar{R}_i$	VSA(D)	0.4820	0.5286	0.0466 $Ave_I > Ave_{NI}$ (99%)	9.67
	VSA(R)	0.4892	0.5368	0.0476 $Ave_I > Ave_{NI}$ (99%)	9.72
	CA(D)	0.3503	0.5142	0.1639 $Ave_I > Ave_{NI}$ (99%)	46.80
	CA(R)	0.3497	0.4917	0.1420 $Ave_I > Ave_{NI}$ (99%)	40.60

Table 6.11: Performance of Improvements for $n = 20$

Measure	Method	Ave_{NI}	Ave_I	$Ave_I - Ave_{NI}$	% Inc
Times(s)	VSA(D)	7.10	9.75	2.65	37.27
				$Ave_I > Ave_{NI}$ (95%)	
	VSA(R)	7.29	8.38	1.09	14.91
				Cannot Reject H_0	
	CA(D)	12.28	5.16	-7.12	-57.98
				$Ave_{NI} > Ave_I$ (99%)	
	CA(R)	11.81	5.34	-6.47	-54.79
				$Ave_{NI} > Ave_I$ (99%)	
Ave C_i	VSA(D)	11.86	9.34	-2.51	-21.21
				$Ave_{NI} > Ave_I$ (99%)	
	VSA(R)	11.37	9.24	-2.13	-18.77
				$Ave_{NI} > Ave_I$ (99%)	
	CA(D)	21.38	11.57	-9.81	-45.87
				$Ave_{NI} > Ave_I$ (99%)	
	CA(R)	21.25	12.38	-8.86	-41.72
				$Ave_{NI} > Ave_I$ (99%)	
Ave $4\sqrt{a_i}/P_i$	VSA(D)	0.4822	0.5394	0.0572	11.86
				$Ave_I > Ave_{NI}$ (99%)	
	VSA(R)	0.4723	0.5170	0.0448	9.48
				$Ave_I > Ave_{NI}$ (99%)	
	CA(D)	0.3341	0.4621	0.1280	38.32
				$Ave_I > Ave_{NI}$ (99%)	
	CA(R)	0.3419	0.4532	0.1112	32.53
				$Ave_I > Ave_{NI}$ (99%)	
Ave $a_i/\bar{R}_i$	VSA(D)	0.4289	0.4779	0.0490	11.41
				$Ave_I > Ave_{NI}$ (99%)	
	VSA(R)	0.4386	0.4790	0.0404	9.21
				$Ave_I > Ave_{NI}$ (99%)	
	CA(D)	0.3255	0.4463	0.1208	37.12
				$Ave_I > Ave_{NI}$ (99%)	
	CA(R)	0.3185	0.4258	0.1073	33.68
				$Ave_I > Ave_{NI}$ (99%)	

Table 6.12: Performance of Improvements for $n = 25$

Measure	Method	Ave_{NI}	Ave_I	$Ave_I - Ave_{NI}$	% Inc
Times(s)	VSA(D)	17.26	23.62	6.36	36.85
				Cannot Reject H_0	
	VSA(R)	22.79	25.73	2.94	12.90
				Cannot Reject H_0	
	CA(D)	26.09	10.37	-15.72	-60.25
				$Ave_{NI} > Ave_I$ (99%)	
	CA(R)	24.76	9.38	-15.39	-62.14
				$Ave_{NI} > Ave_I$ (99%)	
Ave C_i	VSA(D)	13.70	10.41	-3.29	-24.03
				$Ave_{NI} > Ave_I$ (99%)	
	VSA(R)	13.30	10.18	-3.13	-23.50
				$Ave_{NI} > Ave_I$ (99%)	
	CA(D)	25.49	14.27	-11.22	-44.02
				$Ave_{NI} > Ave_I$ (99%)	
	CA(R)	24.92	14.13	-10.79	-43.31
				$Ave_{NI} > Ave_I$ (99%)	
Ave $4\sqrt{a_i}/P_i$	VSA(D)	0.4394	0.4928	0.0534	12.15
				$Ave_I > Ave_{NI}$ (99%)	
	VSA(R)	0.4204	0.4645	0.0441	10.48
				$Ave_I > Ave_{NI}$ (99%)	
	CA(D)	0.2654	0.4130	0.1476	55.62
				$Ave_I > Ave_{NI}$ (99%)	
	CA(R)	0.2720	0.4126	0.1406	51.69
				$Ave_I > Ave_{NI}$ (99%)	
Ave $a_i/\bar{R}_i$	VSA(D)	0.3958	0.4382	0.0425	10.73
				$Ave_I > Ave_{NI}$ (99%)	
	VSA(R)	0.3893	0.4326	0.0433	11.13
				$Ave_I > Ave_{NI}$ (99%)	
	CA(D)	0.2362	0.3810	0.1448	61.32
				$Ave_I > Ave_{NI}$ (99%)	
	CA(R)	0.2377	0.3760	0.1383	58.18
				$Ave_I > Ave_{NI}$ (99%)	

Table 6.13: Performance of Improvements for $n = 30$

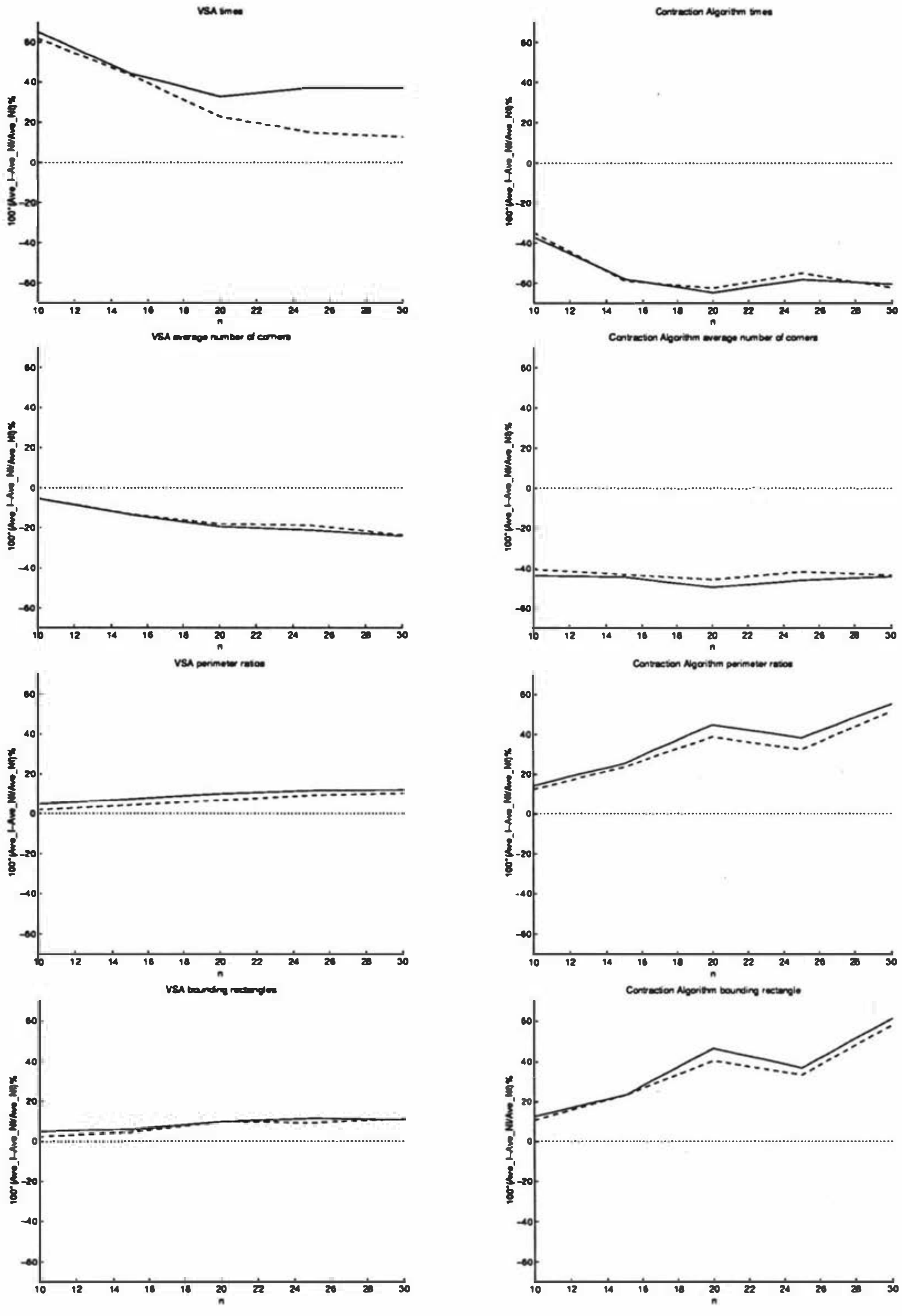


Figure 6.2: Graphs of the percentage difference between Improved and Non Improved versions for the four measures used; dashed line is Rinsma initialisation and the solid line is the Deltahedron initialisation

From this point onwards we will only consider the improved versions of the VSA, and the Contraction Algorithm. It appears non-sensical to carry around the excess baggage of the unimproved versions when we know they will always be out-performed by the improved versions in both regularity and proportionally time.

6.5 Comparison of Layout Methods for Arbitrary MPGs

In this section we examine the performance of the methods which act on an arbitrary MPG. We omit Deltahedron from this discussion, for the reasons encapsulated in Table 6.3. Nor do we include SIMPLE, as for $n \geq 20$, it fails to complete any of the given problems, and furthermore it will in general be outperformed by the Contraction Algorithm, which generates a more relaxed insertion order, in that there are many facilities to act as placement hosts, rather than the single one (the empty space) for SIMPLE. Therefore this experiment now becomes a showdown between the VSA, and the Contraction Algorithm, the only two methods which can successfully dualise all the given test problems.

This experiment attempts to determine which of the two methods is better on average. We used the methods with the Deltahedron initialisation only, for two reasons: firstly the regularity is not significantly affected by the initialisation, and secondly, comparing the VSA and the Contraction Algorithm against a particular initialisation is well nigh irrelevant, as the starting point impinges differently on each method.

Again we consider our null and alternative hypotheses as given by Equation 6.5, where Ave_{VSA} represents the average values for the VSA, and similarly for Ave_{CA}, and the Contraction Algorithm.

$$H_O : Ave_{VSA} = Ave_{CA};\ H_A : Ave_{VSA} > Ave_{CA} \text{ or } Ave_{VSA} < Ave_{CA} \tag{6.5}$$

Results are summarised in Tables 6.14 - 6.18 and Figure 6.3, where we note that the y-scale on the time graph is different to the scale on the other three. From these Tables we conclude that the VSA takes more than twice the time of the Contraction Algorithm for $n = 30$, but this is offset by the superiority of the VSA in all of the

Measure	Ave_{CA}	Ave_{VSA}	$Ave_{VSA} - Ave_{CA}$	% Inc
Times(s)	0.48	0.46	-0.02 Cannot Reject H_0	-4.91
Ave C_i	6.28	6.61	0.33 $Ave_{VSA} > Ave_{CA}$ (90%)	5.19
Ave $4\sqrt{a_i}/P_i$	0.6895	0.7147	0.0252 $Ave_{VSA} > Ave_{CA}$ (95%)	3.65
Ave $a_i/\bar{R}_i$	0.7238	0.6895	-0.0343 $Ave_{CA} > Ave_{VSA}$ (90%)	-4.74

Table 6.14: Performance of VSA vs Contraction Algorithm for $n = 10$

Measure	Ave_{CA}	Ave_{VSA}	$Ave_{VSA} - Ave_{CA}$	% Inc
Times(s)	1.46	1.51	0.05 Cannot Reject H_0	3.09
Ave C_i	9.22	7.73	-1.49 $Ave_{CA} > Ave_{VSA}$ (99%)	-16.18
Ave $4\sqrt{a_i}/P_i$	0.5621	0.6169	0.0548 $Ave_{VSA} > Ave_{CA}$ (99%)	9.75
Ave $a_i/\bar{R}_i$	0.5679	0.5751	0.0072 Cannot Reject H_0	1.27

Table 6.15: Performance of VSA vs Contraction Algorithm for $n = 15$

regularity measures by a significant margin. The real times for the VSA are around 20 seconds for $n = 30$, while the times for the Contraction Algorithm are around 10 seconds. Consider now the time to construct and improve the initial MPG. We saw in Table 6.2 that this time was around 10 minutes, which overwhelms the 10 second discrepancy between the times for the two layout methods. For small n the two methods are comparable; moreover, as we will see in Sections 6.6 and 7.3, the poor average performance of the Contraction Algorithm compared to the VSA, does not discount it from consideration, as it can perform significantly better than VSA on a given problem.

Measure	Ave_{CA}	Ave_{VSA}	$Ave_{VSA} - Ave_{CA}$	% Inc
Times(s)	2.63	4.12	1.48 $Ave_{VSA} > Ave_{CA}$ (99%)	56.34
Ave C_i	10.02	8.51	-1.51 $Ave_{CA} > Ave_{VSA}$ (99%)	-15.11
Ave $4\sqrt{a_i}/P_i$	0.5284	0.5886	0.0602 $Ave_{VSA} > Ave_{CA}$ (99%)	11.39
Ave $a_i/\bar{R}_i$	0.5142	0.5286	0.0144 Cannot Reject H_0	2.81

Table 6.16: Performance of VSA vs Contraction Algorithm for $n = 20$

Measure	Ave_{CA}	Ave_{VSA}	$Ave_{VSA} - Ave_{CA}$	% Inc
Times(s)	5.16	9.75	4.59 $Ave_{VSA} > Ave_{CA}$ (99%)	89.00
Ave C_i	11.57	9.34	-2.23 $Ave_{CA} > Ave_{VSA}$ (99%)	-19.30
Ave $4\sqrt{a_i}/P_i$	0.4621	0.5394	0.0772 $Ave_{VSA} > Ave_{CA}$ (99%)	16.71
Ave $a_i/\bar{R}_i$	0.4463	0.4779	0.0316 $Ave_{VSA} > Ave_{CA}$ (95%)	7.07

Table 6.17: Performance of VSA vs Contraction Algorithm for $n = 25$

Measure	Ave_{CA}	Ave_{VSA}	$Ave_{VSA} - Ave_{CA}$	% Inc
Times(s)	10.37	23.62	13.25 $Ave_{VSA} > Ave_{CA}$ (99%)	127.74
Ave C_i	14.27	10.41	-3.86 $Ave_{CA} > Ave_{VSA}$ (99%)	-27.05
Ave $4\sqrt{a_i}/P_i$	0.4130	0.4928	0.0797 $Ave_{VSA} > Ave_{CA}$ (99%)	19.30
Ave $a_i/\bar{R}_i$	0.3810	0.4382	0.0572 $Ave_{VSA} > Ave_{CA}$ (99%)	15.01

Table 6.18: Performance of VSA vs Contraction Algorithm for $n = 30$

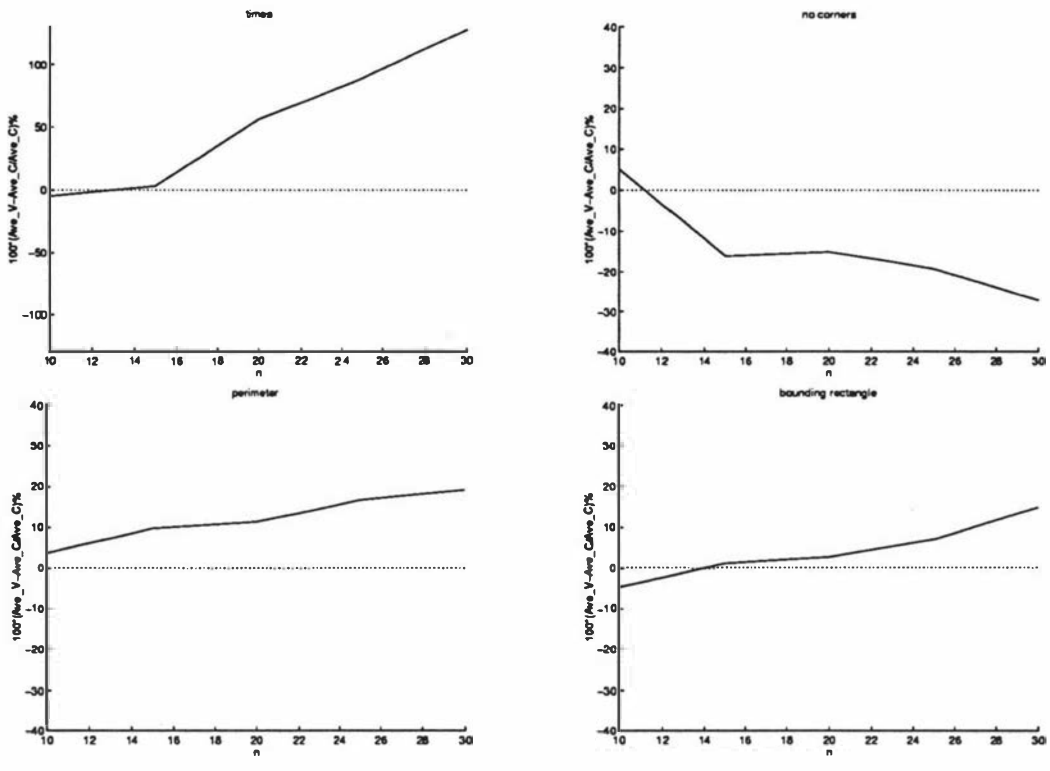


Figure 6.3: Graphs of the percentage difference between the VSA and Contraction Algorithm for the four measures used

6.6 Regularity vs Adjacency Experiment

This section forms the culmination of the work presented so far. Hence we attempt to answer the regularity versus adjacency tradeoff, with respect to carefully constructed objectives which attempt to fairly weight both the adjacency and the regularity values. We must be careful here in that we are comparing values on the range $[0, 1]$, the regularity values, against values dependent on the size of n, the adjacency values. Previous work by Rosenblatt [97], Dutta and Sahu [25] and Shang [102], who presented additive multiple objective functions, is somewhat misguided by the fact that weights must be found in order to treat both regularity and adjacency fairly.

In order to find a sensible objective function we need to consider what we can assume about our regularity and/or adjacency values. The conclusion reached is that if we assume regularity values give a measure of how effective a particular adjacency is, then we can introduce a multiplicative objective function. This means that a regularity value of 0.7 say, would mean that the layout is 70% effective, therefore the benefit derived as the value of the layout is in fact only 70% of what we initially estimated. Initially it seemed that we may be multiplying two incompatible commodities, but careful consideration seemed to indicate that this multiplicative approach was perhaps intuitive, and more realistic than attempting to use an additive model.

An advantage of this model is that it requires only one scalar or weight, λ, in contrast to the additive model, which requires at least two. We introduce this scalar into the objective in order to allow the impact of the regularity measure to be varied. For convenience, we introduce a generic measure r^*, as shown in Equation 6.6.

$$r^* = (1 - \lambda) + \lambda r \qquad 0 \leq \lambda \leq 1 \tag{6.6}$$

This allows us to adjust the importance we place on the regularity, using the linear combination between the best regularity measure we could attain, *i.e.* 1, against the worst *i.e.* r. r can represent any of the regularity measures outlined in Section 2.5, dependent upon the situation and the type of layout required.

We have developed two objective functions using this multiplicative model. The first considers the entire layout, while the second considers each adjacency individually; we consider these in turn. The first objective function utilises the motivation that the average regularity value gives a true weighting to the effectiveness of the total adjacency score. This objective function is shown in Equations 6.7 and 6.8.

$$\begin{aligned}\text{surrogate score} &= [(1-\lambda) + \frac{\lambda}{n}\sum_i r_i]\sum_i\sum_j a_{ij}x_{ij} && (6.7)\\ &= \sum_i\sum_j a_{ij}x_{ij} - \lambda[\sum_i[1-\frac{1}{n}r_i]\sum_j a_{ij}x_{ij}] && (6.8)\end{aligned}$$

The second objective function considered requires the further assumption that an adjacency a_{ij} has impact upon the average of r_i and r_j, or alternatively on the minimum of r_i and r_j. We implement this with the averaging assumption. The objective function then is given in Equations 6.9 and 6.10.

$$\begin{aligned}\text{surrogate score} &= \sum_i\sum_j[(1-\lambda) + \lambda\frac{r_i+r_j}{2}]a_{ij}x_{ij} && (6.9)\\ &= \sum_i\sum_j a_{ij}x_{ij} - \lambda\sum_i\sum_j[1-\frac{r_i+r_j}{2}]a_{ij}x_{ij} && (6.10)\end{aligned}$$

Equations 6.8 and 6.10 both have the form of Equation 6.11, a linear equation in λ which is of the form required to compare two layouts generated from the same benefit matrix.

$$\text{surrogate score} = c_1 - c_2\lambda \qquad (6.11)$$

Let us consider two layouts A and B, with corresponding surrogate scores $c_1^A - c_2^A\lambda$, and $c_1^B - c_2^B\lambda$ respectively. The question then, is under what conditions is A a better layout than B? Table 6.19 exhibits the conditions on the constants c_1 and c_2 for each layout, that are required to answer this question.

The comparison of two layouts is easily conducted using Table 6.19. The comparison of more than two layouts is also easily performed by repeated application of Table 6.19; this can be shown by induction. We can easily compare two layouts,

	$c_1^A < c_1^B$	$c_1^A = c_1^B$	$c_1^A > c_1^B$
$c_2^A < c_2^B$	$A : \frac{c_1^B - c_1^A}{c_2^B - c_2^A} \leq \lambda \leq 1$ $B : 0 \leq \lambda \leq \frac{c_1^B - c_1^A}{c_2^B - c_2^A}$	$A : 0 \leq \lambda \leq 1$ $B : \lambda = 0$	$A : 0 \leq \lambda \leq 1$
$c_2^A = c_2^B$	$B : 0 \leq \lambda \leq 1$	$A : 0 \leq \lambda \leq 1$ $B : 0 \leq \lambda \leq 1$	$A : 0 \leq \lambda \leq 1$
$c_2^A > c_2^B$	$B : 0 \leq \lambda \leq 1$	$A : \lambda = 0$ $B : 0 \leq \lambda \leq 1$	$A : 0 \leq \lambda \leq \frac{c_1^A - c_1^B}{c_2^A - c_2^B}$ $B : \frac{c_1^A - c_1^B}{c_2^A - c_2^B} \leq \lambda \leq 1$

Table 6.19: Comparison table for two layouts A and B: preferred ranges for λ

so consider k layouts which have already been compared, and their respective best ranges found. Then we need only compare a new layout on the ranges for each layout already examined. *i.e.* if a layout is the best on some range $[a, b]$, then comparison of this layout with a new one will only admit one of these two as the best on $[a, b]$, as we know that the other $k-1$ are inferior on $[a, b]$ (since the layout is already the best on that range). The process of comparing the layouts that exist in a problem instance is then easily achieved by calculating the respective values for c_1 and c_2, and then repeatedly applying Table 6.19.

As this experiment represents an opportunity to contribute a major development in the area of GTFLP, we examine carefully the impact of our chosen objective by not only examining the average performance of the methods (as in Section 6.6.1) but also the worst performance (as in Section 6.6.2) and also on a problem-by-problem basis (as in Section 6.6.3).

In this experiment, we compare the methods which work on an arbitrary MPG: the VSA and the Contraction Algorithm (considering both starting layouts) and SIMPLE, all using improvements. We call these Class A Algorithms and compare them with the methods which require the benefit matrix: Deltahedron, the Tiling Algorithm, and the Spanning Tree Algorithm. Note that this last set of methods can be divided further, by considering the worst-case facility shape of T for Deltahedron, and the Tiling Algorithm(T) (calling these Class T Algorithms) against the worst-case facility shape of I for the Tiling Algorithm(I) and Spanning Tree (calling these Class I Algorithms). Note that while we have highlighted the case where Spanning Tree may have L-shaped facilities in Section 4.6.3, we assume that this is rare enough not to impact too much on the grouping of Spanning Tree with the worst-case facility shape of I methods. It turns out that this assumption is valid, as we will see in Sections 6.6.1 - 6.6.3.

The hypothesis that we theorised before the undertaking of this experiment was that the Class A methods which work on arbitrary MPGs would be the best for small λ (due to the highly weighted MPG value and irregular layouts), while the Class I methods would be better for large λ (due to the higher regularity values but lighter weighted MPG), while the Class T methods would exist in some sort of middle ground.

6.6.1 Average Performance

Tables 6.20 - 6.24 exhibit the data for comparing the average regularity values for the problem instances. We recognise at the outset that any conclusions drawn here are based on taking averages of averages, and that while the conclusions reached in this section are valid, there is a large amount of information lost, hence leading to the increased analysis provided by the worst-case, and problem-by-problem performances of Sections 6.6.2 and 6.6.3 respectively. The tables herein all utilise Equation 6.8, as we are dealing with the averages of the facility values. Furthermore the tables are broken into two parts. The first contains the raw averages, for each of the measures, while the second part, provides the intervals for λ within which that particular method is the best. In all tables Deltahedron, refers to Deltahedron which has had an MPG generated especially (as there were insufficient arbitrary MPGs which were Deltahedron to make implications). Note however, that for $n = 10$, when 44% of the arbitrary MPGs were Deltahedron, the regularity values were almost identical to those for which a specially constructed Deltahedron MPG was generated.

Table 6.20 (for $n = 10$) provides us with some interesting points for discussion. Firstly note that the Tiling Algorithm(T), was able to generate MPGs of weight on average larger than the MPGs generated by TESSA and Tabu Search. While this is a little surprising, we have already stated in Section 6.1 that the implementation of Tabu Search was not a major component in the study, hence possibly allowing for an inferior implementation to that of [13] say, and secondly, due to the small size of the problem instances, it is likely that we will get some unusual behaviour not exhibited by larger problem instances. Further, recall from Section 4.7.5 that it is possible that for $n = 10$ layouts can be guaranteed to have worst-case facility shape of T. The Tiling Algorithm(T) would be able to generate a large number of these, providing another reason for its impressive performance.

The times for the different methods were largely variable. The Tiling Algorithm(T) required nearly nine times the computational effort compared to the Class A methods, including the time to generate the MPG, providing another possible reason for the impressive performance of the Tiling Algorithm(T) in generating higher weighted MPGs. However note that all of the Tiling Algorithm(T)'s regularity scores were inferior to those of the Class A methods, even SIMPLE (except on the number of corners), leading to the Tiling Algorithm(T), only being preferred if

Measure	VSA(D)	VSA(R)	CA(D)	CA(R)	SIMPLE	Deltahedron	TA(T)	TA(I)	ST
Benefit	4258.02	4258.02	4258.02	4258.02	4258.02	4186.42	4304.02	4110.58	3619.22
$4\sqrt{a_i}/P_i$	0.71	0.71	0.69	0.68	0.65	0.72	0.64	0.77	0.83
$a_i/\bar{R}_i$	0.69	0.73	0.72	0.70	0.72	0.83	0.64	1.00	1.00
$a_i/\bar{G}_i$	0.52	0.48	0.48	0.48	0.41	0.43	0.37	0.46	0.57
$a_i/\bar{S}_i$	0.36	0.34	0.34	0.34	0.29	0.29	0.27	0.31	0.39
C_i	6.61	5.70	6.28	6.65	8.34	4.56	5.20	4.00	4.03
Time(s)	5.05+0.46	5.05+0.38	5.05+0.48	5.05+0.51	5.05+0.85	0.22	42.55	4.47	0.30
$4\sqrt{a_i}/P_i$	[0.13,0.53]	-	-	-	-	-	[0.00,0.13]	[0.53,1.00]	-
$a_i/\bar{R}_i$	-	[0.12,0.13]	-	-	-	-	[0.00,0.12]	[0.13,1.00]	-
$a_i/\bar{G}_i$	[0.07,1.00]	-	-	-	-	-	[0.00,0.07]	-	-
$a_i/\bar{S}_i$	[0.11,1.00]	-	-	-	-	-	[0.00,0.11]	-	-
C_i	-	-	-	-	-	-	[0.00,0.13]	[0.13,1.00]	-

Table 6.20: Regularity vs Adjacency for $n = 10$

Measure	VSA(D)	VSA(R)	CA(D)	CA(R)	SIMPLE	Deltahedron	TA(T)	TA(I)	ST
Benefit	7463.27	7463.27	7463.27	7463.27	7463.27	7441.69	7518.36	7266.96	6100.78
$4\sqrt{a_i}/P_i$	0.62	0.62	0.56	0.57	0.47	0.67	0.55	0.69	0.79
$a_i/\bar{R}_i$	0.58	0.62	0.57	0.57	0.56	0.79	0.60	1.00	1.00
$a_i/\bar{G}_i$	0.40	0.38	0.36	0.38	0.24	0.39	0.27	0.35	0.52
$a_i/\bar{S}_i$	0.28	0.27	0.25	0.27	0.16	0.26	0.19	0.23	0.36
C_i	7.73	7.00	9.22	9.37	14.75	4.65	5.26	4.00	4.06
Time(s)	26.67+1.51	26.67+1.09	26.67+1.46	26.67+1.41	26.67+9.11	0.31	156.60	15.52	0.91
$4\sqrt{a_i}/P_i$	-	-	-	-	-	[0.08,0.72]	[0.00,0.08]	[0.72,1.00]	-
$a_i/\bar{R}_i$	-	-	-	-	-	[0.05,0.11]	[0.00,0.05]	[0.11,1.00]	-
$a_i/\bar{G}_i$	[0.05,0.88]	-	-	-	-	-	[0.00,0.05]	-	[0.88,1.00]
$a_i/\bar{S}_i$	[0.08,0.92]	-	-	-	-	-	[0.00,0.08]	-	[0.92,1.00]
C_i	-	-	-	-	-	[0.07,0.12]	[0.00,0.07]	[0.12,1.00]	-

Table 6.21: Regularity vs Adjacency for $n = 15$

Measure	VSA(D)	VSA(R)	CA(D)	CA(R)	SIMPLE	Deltahedron	TA(T)	TA(I)	ST
Benefit	11550.47	11550.47	11550.47	11550.47	-	11506.27	11367.22	11099.82	9046.56
$4\sqrt{a_i}/P_i$	0.59	0.56	0.53	0.52	-	0.61	0.48	0.63	0.77
$a_i/\bar{R}_i$	0.53	0.54	0.51	0.49	-	0.74	0.53	1.00	1.00
$a_i/\bar{G}_i$	0.38	0.34	0.34	0.34	-	0.33	0.22	0.28	0.50
$a_i/\bar{S}_i$	0.27	0.24	0.24	0.24	-	0.23	0.16	0.18	0.34
C_i	8.51	8.15	10.02	10.90	-	4.76	5.36	4.00	4.03
Time(s)	91.63+4.07	91.63+4.08	91.63+2.63	91.63+2.78	-	0.44	269.11	42.63	2.66
$4\sqrt{a_i}/P_i$	[0.00,0.16]	-	-	-	-	[0.16,1.00]	-	-	-
$a_i/\bar{R}_i$	-	[0.00,0.02]	-	-	-	[0.02,0.14]	-	[0.14,1.00]	-
$a_i/\bar{G}_i$	[0.00,0.94]	-	-	-	-	-	-	-	[0.94,1.00]
$a_i/\bar{S}_i$	[0.00,1.00]	-	-	-	-	-	-	-	-
C_i	-	[0.00,0.01]	-	-	-	[0.01,0.15]	-	[0.15,1.00]	-

Table 6.22: Regularity vs Adjacency for $n = 20$

Measure	VSA(D)	VSA(R)	CA(D)	CA(R)	SIMPLE	Deltahedron	TA(T)	TA(I)	ST
Benefit	15267.22	15267.22	15267.22	15267.22	-	15373.38	14963.78	14678.07	11683.40
$4\sqrt{a_i}/P_i$	0.54	0.52	0.46	0.45	-	0.63	0.45	0.58	0.75
$a_i/\bar{R}_i$	0.48	0.48	0.45	0.43	-	0.75	0.55	1.00	1.00
$a_i/\bar{S}_i$	0.33	0.31	0.27	0.28	-	0.36	0.19	0.24	0.48
$a_i/\bar{G}_i$	0.24	0.22	0.19	0.20	-	0.25	0.13	0.16	0.33
C_i	9.34	9.24	11.57	12.38	-	4.78	5.24	4.00	4.03
Time(s)	243.20+9.75	243.20+8.38	243.20+5.16	243.20+5.30	-	0.54	541.12	87.10	6.64
$4\sqrt{a_i}/P_i$	-	-	-	-	-	[0.00,1.00]	-	-	-
$a_i/\bar{R}_i$	-	-	-	-	-	[0.00,0.18]	-	[0.18,1.00]	-
$a_i/\bar{G}_i$	-	-	-	-	-	[0.00,1.00]	-	-	[1.00,1.00]
$a_i/\bar{S}_i$	-	-	-	-	-	[0.00,1.00]	-	-	-
C_i	-	-	-	-	-	[0.00,0.19]	-	[0.19,1.00]	-

Table 6.23: Regularity vs Adjacency for $n = 25$

Measure	VSA(D)	VSA(R)	CA(D)	CA(R)	SIMPLE	Deltahedron	TA(T)	TA(I)	ST
Benefit	18791.44	18791.44	18791.44	18791.44	-	19157.67	18572.24	18295.87	14309.18
$4\sqrt{a_i}/P_i$	0.49	0.46	0.41	0.41	-	0.60	0.41	0.54	0.74
$a_i/\bar{R}_i$	0.44	0.43	0.38	0.38	-	0.74	0.52	1.00	1.00
$a_i/\bar{G}_i$	0.29	0.26	0.25	0.25	-	0.33	0.16	0.21	0.47
$a_i/\bar{S}_i$	0.20	0.18	0.18	0.18	-	0.23	0.11	0.13	0.32
C_i	10.41	10.18	14.27	14.13	-	4.80	5.35	4.00	4.02
Time(s)	589.65+23.62	589.65+25.73	589.65+10.37	589.65+9.26	-	0.66	766.38	151.76	18.52
$4\sqrt{a_i}/P_i$	-	-	-	-	-	[0.00,1.00]	-	-	-
$a_i/\bar{R}_i$	-	-	-	-	-	[0.00,0.17]	-	[0.17,1.00]	-
$a_i/\bar{G}_i$	-	-	-	-	-	[0.00,0.93]	-	-	[0.93,1.00]
$a_i/\bar{S}_i$	-	-	-	-	-	[0.00,0.94]	-	-	[0.94,1.00]
C_i	-	-	-	-	-	[0.00,0.19]	-	[0.19,1.00]	-

Table 6.24: Regularity vs Adjacency for $n = 30$

λ is around 10% or less. The VSA with the Deltahedron initialisation surprisingly makes up a healthy chunk of the remaining usable area-based regularity measures, while the Tiling Algorithm(I) is preferred for the shape-based measures for λ above 10%. As we have seen in the previous experiments, the Class A methods all perform with about the same effectiveness, with SIMPLE not too far away (although clearly inferior, especially with respect to the number of corners). Spanning Tree was hindered by its inability to generate highly-weighted layouts, generating layouts of total weight around 15% less than TESSA with Tabu Search generated MPGs. Deltahedron appears to be essentially in the middle of the methods of the Class A and the Tiling Algorithm(I), with respect to benefit, and the shape based measures, but was slightly inferior to the Tiling Algorithm(I) and/or the Tiling Algorithm(T).

Table 6.21 (for n = 15) shows some significant differences to those exhibited in Table 6.20. The Tiling Algorithm(T) still generates the highest weighted MPGs, requiring more time than the Class A methods. However the highest λ value for dominance of the Tiling Algorithm(T) has dropped to around 7%. Deltahedron appears briefly among the shape-based measures. As with $n = 10$, the Tiling Algorithm(I) performs the best for λ above 11% on the shape-based measures, while the usable space measures see VSA with the Deltahedron initialisation between approximately 7% and 90% for the bounding square and bounding golden rectangle, followed by Spanning Tree above 90%, while Deltahedron is best for λ between 8% and 72% followed by the Tiling Algorithm(I) for the perimeter ratio. Note that Deltahedron has MPG values only slightly inferior to those of the arbitrary MPGs, superior regularity values for those which are shape-based, and values on a par with those which are usable space based. While Spanning Tree has a highly inferior weight, the superiority of its bounding square and golden rectangle lead to its inclusion for large λ.

Table 6.22 (for n = 20) shows yet more changes in the performance of the methods over the λ range. The Tiling Algorithm(T) no longer appears, overtaken for small λ by the Class A methods, which now have the highest MPG values. The shape-based measures are still dominated by the Tiling Algorithm(I), with Deltahedron appearing among these measures for small λ. The perimeter ratio is also dominated by Deltahedron, while VSA with Deltahedron is superior on the bounding square and golden rectangle measures. Note that SIMPLE no longer appears, as it was able to successfully dualise only 27% of the MPGs. The shape-based measures appear to be levelling out to what we originally expected, with the Class I methods having the best values followed by the Class T methods, and lastly the Class A methods. The usable area measures do not yet follow expectations. The Tiling Algorithm(T) appears to be generating some elongated facilities, as its poor performance on the usable area measures shows.

Tables 6.23 and 6.24 (for $n = 25$ and $n = 30$ respectively) show an impressive performance by Deltahedron, which completely dominates the usable area based measures, while the Tiling Algorithm(I) dominates the shape-based measures. The Class A methods do not appear due to the superiority of Deltahedron in both benefit and all regularity values, while taking only a fraction of the time of the Class A methods to complete the layout construction. For $n = 30$, Deltahedron is surpassed at the top end of the λ scale for the bounding square and golden rectangle values by Spanning Tree, which is not so affected by elongated facilities as n increases.

It appears therefore that the average performance of the methods studied is dominated for small n by the ability to generate a highly weighted MPG, while keeping irregularity to a minimum (which can usually be accomplished for small n), while for large n, the performance is dominated by a method which can generate highly weighted MPGs, whilst guaranteeing a good worst-case facility shape (to ensure the regularity values do not degrade unnecessarily). The results obtained were a little surprising but, even allowing for the ability of TESSA and Tabu Search to generate the highest weighted MPG, Deltahedron would soon surpass the Class A methods due to its superior regularity values and ability to generate highly-weighted MPGs.

6.6.2 Worst Performance

This section is designed to be a reinforcement of the conclusions reached in Section 6.6.1. We discussed in Chapter 2.5 that there were two different objectives which we could consider when evaluating any layout, being to maximize the average facility regularity, or to maximize the minimum facility regularity. This section examines the second of these objectives, by providing in Tables 6.25 - 6.29 the average regularity values of the worst facility in each layout. While we again take the averages of these values, there is far less information lost in this instance as we are examining only one facility in each layout, rather than all facilities. Of interest in this section are not only the relationships and patterns within Tables 6.25 - 6.29, but also the relationships to Tables 6.20 - 6.24 for each respective value of n. Note that we omit the time values, as they do not alter.

Table 6.25 (for $n = 10$) shows a consistency in the performance of the Tiling Algorithm(T). Naturally, since the benefit values of the MPGs do not change, the Tiling Algorithm(T) will be the best method for $\lambda = 0$, but the point at which it is superseded by another method is approximately the same as for the average case. Again the Class A methods dominate the bounding square and rectangle measures, although different Class A methods appear compared to those in Table 6.20. The Tiling Algorithm(I) parallels its behaviour from the average case, albeit with a slightly lower value of λ at the lower end. The main differences between Tables 6.20 and 6.25 are the best methods for the perimeter ratio. Deltahedron appears for the worst facility case for λ between 37% and 48%, which we stated was close to happening in the average case. However Spanning Tree overtakes the Tiling Algorithm(I), for λ above 50%, while a Class A method no longer appears. This change between the average and worst cases is not surprising, as the average case gave very similar perimeter ratios for each problem instance, while there was more variance in the worst case values. As for the data in the first part of Table 6.25, notice the large average number of corners in the worst facility of a SIMPLE layout, compared to the other Class A methods. Also it is hardly surprising that the Class I methods dominate the shape-based measures, having far superior values, especially for the bounding rectangle compared to the Class T and A methods, whereas the usable area measures show a higher degree of similarity over all the classes.

Table 6.26 (for $n = 15$) closely follows the patterns of its companion Table 6.21. The Tiling Algorithm(T) has ranges almost identical in both cases for the shape-based measures, while for the usable area measures, λ increases to around 20% compared to 6% in the average case. The Class A methods again take over from the Tiling Algorithm(T) for the bounding square and golden rectangle, though this time it is via the Contraction Algorithm with the Rinsma initialisation. Deltahedron does not appear in the worst case, but the patterns for the Class A methods are essentially preserved. Again note the astronomical value of the worst case number of corners for SIMPLE.

The shape measures for Tables 6.22 and 6.27 (for $n = 20$) are very similar, even

Measure	VSA(D)	VSA(R)	CA(D)	CA(R)	SIMPLE	Deltahedron	TA(T)	TA(I)	ST
Benefit	4258.02	4258.02	4258.02	4258.02	4258.02	4186.42	4304.02	4110.58	3619.22
$4\sqrt{a_i}/P_i$	0.31	0.38	0.32	0.30	0.28	0.43	0.34	0.60	0.67
$a_i/\bar{R}_i$	0.15	0.20	0.22	0.22	0.16	0.21	0.13	1.00	1.00
$a_i/\bar{G}_i$	0.15	0.17	0.19	0.19	0.15	0.16	0.12	0.18	0.26
$a_i/\bar{S}_i$	0.13	0.14	0.15	0.15	0.11	0.12	0.10	0.11	0.16
C_i	11.51	9.51	11.64	12.44	20.27	6.36	7.51	4.00	4.13
$4\sqrt{a_i}/P_i$	-	-	-	-	-	[0.37,0.48]	[0.00,0.17]	[0.17,0.37]	[0.48,1.00]
$a_i/\bar{R}_i$	-	-	-	-	-	-	[0.00,0.05]	[0.05,1.00]	-
$a_i/\bar{G}_i$	-	-	[0.14,0.86]	-	-	-	[0.00,0.14]	-	[0.86,1.00]
$a_i/\bar{S}_i$	-	-	-	[0.18,1.00]	-	-	[0.00,0.18]	-	-
C_i	-	-	-	-	-	-	[0.00,0.06]	[0.06,1.00]	-

Table 6.25: Worst Case Regularity vs Adjacency for $n = 10$

Measure	VSA(D)	VSA(R)	CA(D)	CA(R)	SIMPLE	Deltahedron	TA(T)	TA(I)	ST
Benefit	7463.27	7463.27	7463.27	7463.27	7463.27	7441.69	7518.36	7266.96	6100.78
$4\sqrt{a_i}/P_i$	0.21	0.25	0.19	0.19	0.10	0.32	0.26	0.48	0.55
$a_i/\bar{R}_i$	0.08	0.09	0.10	0.10	0.08	0.11	0.07	1.00	1.00
$a_i/\bar{G}_i$	0.08	0.08	0.09	0.10	0.08	0.09	0.07	0.11	0.16
$a_i/\bar{S}_i$	0.07	0.07	0.08	0.08	0.06	0.07	0.06	0.07	0.10
C_i	15.51	13.16	20.89	21.60	47.07	6.71	7.82	4.00	4.36
$4\sqrt{a_i}/P_i$	-	-	-	-	-	-	[0.00,0.14]	[0.14,1.00]	-
$a_i/\bar{R}_i$	-	-	-	-	-	-	[0.00,0.04]	[0.04,1.00]	-
$a_i/\bar{G}_i$	-	-	-	[0.22,0.71]	-	-	[0.00,0.22]	[0.71,0.87]	[0.87,1.00]
$a_i/\bar{S}_i$	-	-	-	[0.25,1.00]	-	-	[0.00,0.25]	-	-
C_i	-	-	-	-	-	-	[0.00,0.05]	[0.05,1.00]	-

Table 6.26: Worst Case Regularity vs Adjacency for $n = 15$

Measure	VSA(D)	VSA(R)	CA(D)	CA(R)	SIMPLE	Deltahedron	TA(T)	TA(I)	ST
Benefit	11550.47	11550.47	11550.47	11550.47	-	11506.27	11367.22	11099.82	9046.56
$4\sqrt{a_i}/P_i$	0.18	0.20	0.15	0.14	-	0.26	0.22	0.41	0.47
$a_i/\bar{R}_i$	0.06	0.06	0.07	0.06	-	0.07	0.05	1.00	1.00
$a_i/\bar{G}_i$	0.06	0.06	0.06	0.06	-	0.06	0.05	0.07	0.11
$a_i/\bar{S}_i$e	0.05	0.05	0.06	0.06	-	0.05	0.04	0.05	0.07
C_i	18.18	17.07	24.13	26.09	-	6.58	7.87	4.00	4.22
$4\sqrt{a_i}/P_i$	-	[0.00,0.06]	-	-	-	[0.06,0.21]	-	[0.21,1.00]	-
$a_i/\bar{R}_i$	-	-	[0.00,0.04]	-	-	-	-	[0.04,1.00]	-
$a_i/\bar{G}_i$	-	-	[0.00,0.83]	-	-	-	-	[0.83,0.94]	[0.94,1.00]
$a_i/\bar{S}_i$	-	-	-	[0.00,1.00]	-	-	-	-	-
C_i	-	[0.00,0.01]	-	-	-	[0.01,0.06]	-	[0.06,1.00]	-

Table 6.27: Worst Case Regularity vs Adjacency for $n = 20$

Measure	VSA(D)	VSA(R)	CA(D)	CA(R)	SIMPLE	Deltahedron	TA(T)	TA(I)	ST
Benefit	15267.22	15267.22	15267.22	15267.22	-	15373.38	14963.78	14678.07	11683.40
$4\sqrt{a_i}/P_i$	0.15	0.15	0.13	0.12	-	0.24	0.19	0.36	0.41
$a_i/\bar{R}_i$	0.04	0.05	0.05	0.05	-	0.06	0.04	1.00	1.00
$a_i/\bar{G}_i$	0.04	0.05	0.05	0.05	-	0.05	0.04	0.06	0.08
$a_i/\bar{S}_i$	0.04	0.04	0.04	0.04	-	0.04	0.04	0.04	0.05
C_i	20.58	20.53	28.89	31.11	-	7.11	7.82	4.00	4.40
$4\sqrt{a_i}/P_i$	-	-	-	-	-	[0.00,0.29]	-	[0.29,1.00]	-
$a_i/\bar{R}_i$	-	-	-	-	-	[0.00,0.05]	-	[0.05,1.00]	-
$a_i/\bar{G}_i$	-	-	-	-	-	[0.00,0.90]	-	[0.90,0.98]	[0.98,1.00]
$a_i/\bar{S}_i$	-	-	-	[0.63,1.00]	-	[0.00,0.63]	-	-	-
C_i	-	-	-	-	-	[0.00,0.07]	-	[0.07,1.00]	-

Table 6.28: Worst Case Regularity vs Adjacency for $n = 25$

Measure	VSA(D)	VSA(R)	CA(D)	CA(R)	SIMPLE	Deltahedron	TA(T)	TA(I)	ST
Benefit	18791.44	18791.44	18791.44	18791.44	-	19157.67	18572.24	18295.87	14309.18
$4\sqrt{a_i}/P_i$	0.12	0.12	0.09	0.09	-	0.19	0.17	0.32	0.38
$a_i/\bar{R}_i$	0.03	0.03	0.03	0.03	-	0.04	0.03	1.00	1.00
$a_i/\bar{G}_i$	0.03	0.03	0.03	0.03	-	0.04	0.03	0.05	0.07
$a_i/\bar{S}_i$	0.03	0.03	0.03	0.03	-	0.03	0.03	0.03	0.04
C_i	26.53	26.71	35.78	36.31	-	7.20	8.00	4.00	4.22
$4\sqrt{a_i}/P_i$	-	-	-	-	-	[0.00,0.29]	-	[0.29,1.00]	-
$a_i/\bar{R}_i$	-	-	-	-	-	[0.00,0.05]	-	[0.05,1.00]	-
$a_i/\bar{G}_i$	-	-	-	-	-	[0.00,0.88]	-	[0.88,0.97]	[0.97,1.00]
$a_i/\bar{S}_i$	-	-	-	[0.92,1.00]	-	[0.00,0.92]	-	-	-
C_i	-	-	-	-	-	[0.00,0.07]	-	[0.07,1.00]	-

Table 6.29: Worst Case Regularity vs Adjacency for $n = 30$

though Deltahedron does not appear for the bounding rectangle. The bounding square and golden rectangle are also similar in the worst case to the average case, with a Class A method possessing the bulk of the λ values. The perimeter ratio again does not follow the pattern set out by the average case, with the Tiling Algorithm(I) taking over λ above 20%, where Deltahedron was preferred in the average case.

For $n = 25$ (Table 6.28) and $n = 30$ (Table 6.29), the main surprise is the appearance of a Class A method as the best method for the bounding square for large λ, as it might have been expected that the Class T and I methods to be superior for large λ, since they are based on regularity. However the other trends, with the exception of the perimeter ratio, exhibited by Tables 6.23 and 6.24, appear to be preserved, with Deltahedron making up the bulk of the usable area measures, and the Tiling Algorithm(I) the bulk of the shape-based measures.

The comparison of the worst case to the average case, produced very pleasing results. It would have been difficult to draw conclusions if the λ ranges had varied considerably between the two objective functions. However, with the exception of the perimeter ratio, the two objectives exhibited essentially the same results. It is worth noting at this point that there may not be a large difference in a decision maker's mind between $\lambda = 0.35$ and $\lambda = 0.45$ say, and therefore we should be careful at the interface of two or more methods giving essentially the same objective value *i.e.* the point at which two methods' surrogate score lines would cross, would be a *fuzzy* area rather than a single point in a subjective environment.

The performance of the perimeter ratio was rather variable. It appears that the difference between the worst and average case has a significant impact on the best method ranges in this case. Examining this more closely, we must remember what the perimeter ratio measures: the length of the perimeter compared to the length of the perimeter for a square of the same area. Consider the elongated facilities which tend to appear in the Class T and I methods. In the average case, the effect of these elongated facilities is significantly reduced, while in the worst case a long top facility for example could significantly reduce the worst perimeter ratio in a layout. Consider again the example of Figure 5.6, where in the average case the perimeter ratio tends to 1, while in the worst case the perimeter ratio tends to 0. This phenomenon is not so pronounced in the other regularity measures, leading to the comparable trends in the average and worst cases.

6.6.3 Problem-by-Problem Performance

The final stage in examining the comparison of the layout methods is to compare them on a problem-by-problem basis. Obviously, in real terms, we wish to forgo building a large set of different layouts by the same method. In general, due to the cost of building a layout, or indeed changing an existing layout, the costs can be enormous, therefore we would not simply pick the method which has been shown on average to be the best, we would choose the one which is best for that particular circumstance. The solutions for each problem instance on a number of regularity measures were calculated showing largely variable results. Clearly it is not helpful to show all the problems on all the different measures; however, Table 6.30 gives examples chosen to show as closely as possible the variable values that can occur. We do not consider all possible methods, but rather consider the classes which depend upon the worst case facility shape. All measures use Equation 6.8. By examining Table 6.30 we can see that, for the three problem instances chosen, we cannot with certainty choose one class of method over another arbitrarily. Rather, being able to generate the *best* layout for some problem instance requires careful analysis of all possible layouts which could be generated, and upon which regularity measures and objectives we choose to base our decisions. We will return to this point a number of times throughout the remainder of this thesis.

Measure	n	Problem	Class A	Class T	Class I
$4\sqrt{a_i}/P_i$	10	22	[0.13,0.91]	[0.00,0.13]	[0.91,1.00]
$a_i/\bar{S}_i$	10	22	[0.04,1.00]	[0.00,0.04]	-
C_i	10	22	-	[0.00,0.34]	[0.34,1.00]
$4\sqrt{a_i}/P_i$	20	14	[0.00,0.10]	[0.10,0.75]	[0.75,1.00]
$a_i/\bar{S}_i$	20	14	[0.00,0.73]	-	[0.73,1.00]
C_i	20	14	[0.00,0.00]	[0.00,0.31]	[0.31,1.00]
$4\sqrt{a_i}/P_i$	30	35	[0.00,0.22]	[0.22,0.93]	[0.93,1.00]
$a_i/\bar{S}_i$	30	35	[0.00,0.62]	[0.62,1.00]	-
C_i	30	35	[0.00,0.03]	[0.03,0.16]	[0.16,1.00]

Table 6.30: Sample of problem instances, and their associated best ranges

Coda

In this chapter we have undertaken an extensive computational examination of the methods pertaining to the GTFLP. We have seen in Section 6.3 the similarity of the two different starting points for the VSA and the Contraction Algorithm and also in Section 6.4 the impact of the improvements from Chapter 5. We have examined the performance of the Class A methods in Section 6.5, and concluded that the VSA has an edge over the Contraction Algorithm in terms of the regularity measures. In Section 6.6 we have examined extensively the relationship between regularity and adjacency, by introducing an intuitive surrogate score which provides a measure of a layout incorporating both regularity and adjacency. We conclude that while the Deltahedron method in general provides an all-purpose solution, it is wise to make decisions at the problem level, rather than making sweeping generalisations. We will examine this point further in Chapter 7, where we consider problem instances which are designed specifically to perform well on one method, and not on the others. The purpose of this is to show that all the methods which have so far been designed to produce layouts to the GTFLP have their place within this framework, and can aid the production of a good final layout.

Chapter 7

Biased Examples

In Section 6.6 we saw that we could not accredit any one method with being the *best* layout method *i.e.* the one which we would always turn to regardless of the situation. We saw that while Deltahedron was perhaps the most robust method, a problem by problem analysis of the layouts generated by each method was required. In this chapter we continue to reinforce this idea by carefully constructing an MPG for each layout algorithm which works very well on that particular method. The purpose of this is to further demonstrate that all the methods from Chapter 4 have their niche within the GTFLP framework.

By considering the motivation and algorithmic process of each of the layout algorithms, we attempted to provide an example where that respective method would produce a more practical layout than the other methods. It was not deemed necessary for the other methods to perform poorly on each example, but, in many instances this phenomenon occurred. We will discuss further the generation of each problem instance as we encounter them. As formal characterisation of these problems is difficult, we felt that it sufficed to provide only one example for each method, as providing a single problem instance where each method performs well is sufficient to show that the method may provide important information within a general setting, even if it does not produce the best layout.

In order to determine that a particular layout was *best*, we examined the layout generated by each method, and made comparisons on two fronts: objective and subjective. The objective analysis involved the examination of the regularity values for each layout, while the subjective analysis investigated whether each facility was usable and each adjacency was able to be realised. As we have discussed previously, we do not simply base our decisions on a single value, but consider the objective and subjective issues to ensure a practical layout is obtained. We provide the best layout in each instance, but for brevity and coherency we do not include all the layouts generated on each problem instance. Figures 7.1 - 7.6 provide the best layout generated for each of the given problems, while Tables 7.1 - 7.6 provide the regularity values and area specifications for each problem. The problems are not explicitly stated, however the MPG used for the Class A and Deltahedron methods can be obtained from the layouts, with the area specifications given in the respective table for each layout, while the benefit matrices for the Tiling and Spanning Tree Algorithms have zero benefit for all edges which do not exist in the MPG generated from their layouts, and benefit of one for every edge that does exist, again with the area conditions given in the respective tables.

7.1 Deltahedron

The generation of a problem instance which would perform well on Deltahedron was not difficult. We saw in Chapter 6 that Deltahedron layouts consistently provide better regularity values than the Class A methods; therefore a problem which would be biased towards the Deltahedron Algorithm, would obviously have an MPG which was Deltahedron-generateable. It was hypothesised that the presence of the umbrella effect would reduce the effectiveness of the layout, and so we attempted to generate the MPG with vertex degrees as uniform as possible. Initially the layout was constructed with all area specifications the same. Following this initial construction we perturbed the area requirements until we were satisfied that each facility was adequately practical. This process was conducted by choosing facilities which were elongated beyond what we considered acceptable, adding a percentage to their area specification, and reconstructing the layout. This was based on a subjective examining of the layout, until we were satisfied that no facility suffered from elongation. Once we were satisfied that the Deltahedron layout was practical both objectively and subjectively, we implemented the Class A algorithms on this problem instance. Generation of Tiling and Spanning Tree layouts were not possible as they cannot generate layouts from an MPG.

The Deltahedron example produced performed exceptionally on all but one of the regularity values. Table 7.1 shows the shape-based, and perimeter ratio, measures being far higher than all other layouts on both objectives. The one

Facility	1	2	3	4	5	6	7	8	9	10	11	12
Area	-	30	50	40	20	20	40	10	10	20	20	20

	Average Value			
	VSA	CA	SIMPLE	Deltahedron
C_i	8.0000	6.5455	13.0909	**4.5455**
$a_i/\bar{R}_i$	0.5040	0.7530	0.6461	**0.9091**
$a_i/\bar{G}_i$	0.4908	0.4728	0.2631	**0.5583**
$a_i/\bar{S}_i$	**0.3721**	0.3334	0.1728	0.3637
$4\sqrt{a_i}/P_i$	0.6461	0.6900	0.5399	**0.8283**
	Worst Value			
	VSA	CA	SIMPLE	Deltahedron
C_i	14.0000	16.0000	44.0000	**8.0000**
$a_i/\bar{R}_i$	0.1622	0.2432	0.1250	**0.3333**
$a_i/\bar{G}_i$	0.1622	0.2103	0.1250	**0.2725**
$a_i/\bar{S}_i$	0.1031	0.1300	0.0883	**0.1684**
$4\sqrt{a_i}/P_i$	0.3370	0.3530	0.1804	**0.5663**

Table 7.1: Area Specifications and Regularity Values for the Biased Deltahedron Example

blemish is for the average bounding square, which is slightly better on the VSA layout. This is not a concern for two reasons: firstly, there is less than 0.01 (or 2%) difference between their respective values, and secondly the worst case values for this measure show that the Deltahedron has a better value, implying that there is less variance in the Deltahedron values for the bounding rectangle than for the VSA. Examining the layout itself (Figure 7.1) in a subjective manner, the layout appears directly implementable. Facilities 8 and 9 are small enough relative to facility 7 that they could be made squarer without unduly affecting the adjacency between facilities 3 and 7; further this increases the average bounding square measure for this layout beyond the VSA measure. The VSA measure is unlikely to be able to be improved due to the non-dimensionalisable layout generated by the VSA. Facility 4 is perhaps the least practical of the facilities, suggesting the need for a closer examination of the adjacencies with 4 (if facility 4 was is a material handling system, or corridor, for instance, then it is likely to be efficient without further changes).

This problem instance provides a basis for an archetypical *family* of MPGs which produce practical layouts using the Deltahedron Algorithm. Obviously the

MPGs will be Deltahedron-generateable, but also, will not exhibit the umbrella phenomenon. To produce area specifications which do not result in elongation requires interaction with the layout, a point we will return to in Chapter 8 as we modify the area requirements to ensure squarer facilities. Problem instances which exhibit these characteristics are likely to be very amenable to generation of directly implementable layouts.

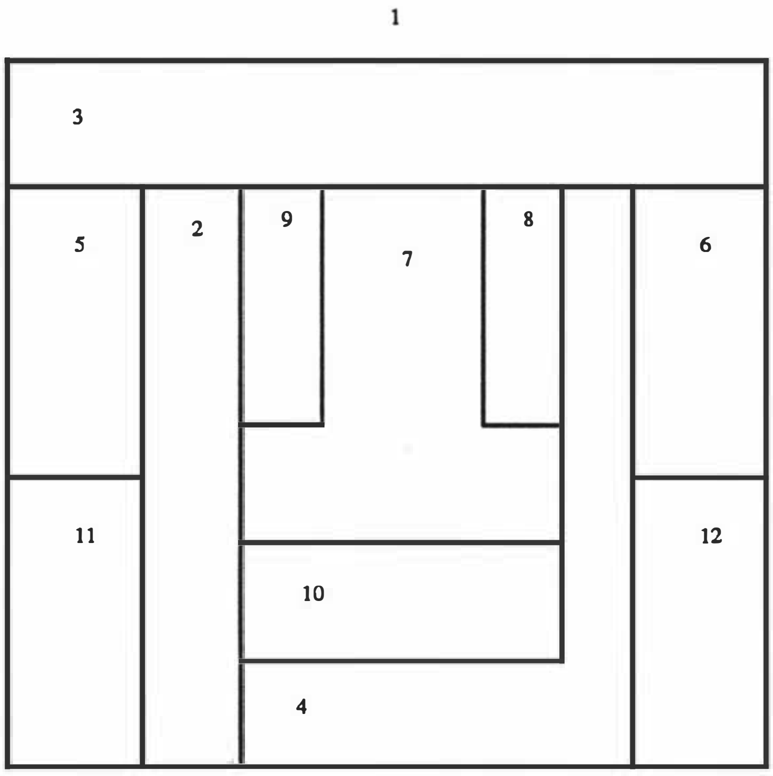


Figure 7.1: Biased Deltahedron Example

This family of problem instances is unlikely to allow generation of practical layouts using the other approaches we have discussed. The Tiling and Spanning Tree Algorithms cannot generate layouts from the MPG, while the VSA is likely to be hindered by the disjoint distance classes exhibited by many Deltahedron layouts (especially those not exhibiting the umbrella effect). The amount of vertex splitting required by the VSA for MPGs when there are disjoint distances classes, in general leads to impractical layouts. SIMPLE is also unlikely to perform well, as its placement mechanism places facilities *outside* previously placed facilities, in direct contrast to Deltahedron's placement operations. For this reason Deltahedron-generated MPGs are not in general amenable to the SIMPLE Algorithm. The

Contraction Algorithm is perhaps the most comparable to Deltahedron, as they both use the same mechanism, of placing facilities within other facilities. This correspondence is further exhibited by Table 7.1 where the Contraction Algorithm in general produced the layout which was *the best of the rest.* The Contraction Algorithm however is susceptible to losing the dimensionalisability of the layout, hence generating layouts less practical those of Deltahedron, especially as n increases.

7.2 The Vertex Splitting Algorithm

The VSA works best on MPGs which have concentric distance classes. The reason for this is that the MPG is then already in a form directly applicable to the layout routine of the VSA, without the requirement of splitting facilities. It was also desired that each facility could be placed as a rectangle. Two factors must be considered to ensure rectangular facilities: the sides each facility is placed on, and the area of each facility. Each corner facility of a distance class was required to have at least two adjacencies with the next outer distance class. This was easily achieved initially by ensuring that each facility in D_{max} had at least two adjacencies with the next outer distance class. A corresponding layout was generated with each facility of equal area. Adjustment was required to both the MPG and the area specifications to generate the final regular layout. Since VSA layouts in general do not produce dimensionalisable layouts, we were required to make adjacency swaps within the MPG to allow the walls between the frames to be placed as single straight lines, and/or to perturb the area specifications of some of the facilities. Having obtained rectangular facilities (still within a non-dimensionalisable layout), we were able to carefully perturb the area requirements to ensure elongation of facilities was not exhibited.

The VSA layout of Figure 7.2 is clearly the best layout that is constructed by the given methods in Table 7.2. Since the MPG was not Deltahedron-generateable, we could only generate layouts for this problem instance using the Class A algorithms. The worst case VSA values give the most conclusive evidence of this, as they are on a par with the average values of the other two methods. Furthermore, the difference between the VSA average and worst values is reasonably small, implying that all facilities in the layout have essentially the same degree of usefulness. Examination of the actual layout confirms this, with every facility being rectangular, and facility

Facility	1	2	3	4	5	6	7	8	9	10	11	12
Area	-	20	20	30	40	20	40	30	20	50	30	50

	Average Value			Worst Value		
	VSA	CA	SIMPLE	VSA	CA	SIMPLE
C_i	**4.0000**	6.5455	11.6364	**4.0000**	16.0000	38.0000
$a_i/\bar{R}_i$	**1.0000**	0.7588	0.6130	**1.0000**	0.1333	0.1091
$a_i/\bar{G}_i$	**0.5987**	0.3410	0.2439	**0.3298**	0.1333	0.1091
$a_i/\bar{S}_i$	**0.4035**	0.2318	0.1707	**0.2038**	0.1000	0.0900
$4\sqrt{a_i}/P_i$	**0.8682**	0.6626	0.4928	**0.7501**	0.2585	0.1348

Table 7.2: Area Specifications and Regularity Values for the Biased VSA Example

9 having the worst width to height ratio. The one blemish on this VSA layout would be the adjacency between facilities 2 and 5. Obviously with the arbitrariness in the area units used, it is difficult to know what exactly the current common length of wall is between 2 and 5, and further examination would be required to ensure the efficient use of this adjacency.

This problem instance exhibits characteristics of a set of problem instances where the VSA would produce very effective results. MPGs with concentric distance classes allow the direct implementation of the VSA, without the requirement of vertex splitting. If each corner facility of each distance class has degree at least two with the facilities in the next outer distance class, we can guarantee that each facility is placed on only one side of the layout when using the VSA. Concentric MPGs which have this property ensure VSA layouts with only rectangular, L- and S-shaped facilities. However we cannot be sure of which vertices of the MPG will correspond to corner vertices of each frame *a priori*.

The Contraction Algorithm does not perform well on these type of problems in general due to its contraction mechanism. Each distance class is connected, and therefore the entire distance class becomes a single vertex which is then contracted to the next outer distance class. When applying the reverse contraction process, all facilities in a distance class (and all facilities of the inner distance classes) will be placed (explicitly or implicitly) within the same placement host for that distance class. This leads to a high degree of irregularity in the placement hosts, and large variability in regularity between those facilities which are placement hosts, and those which are not. SIMPLE has the potential to mimic the behaviour of

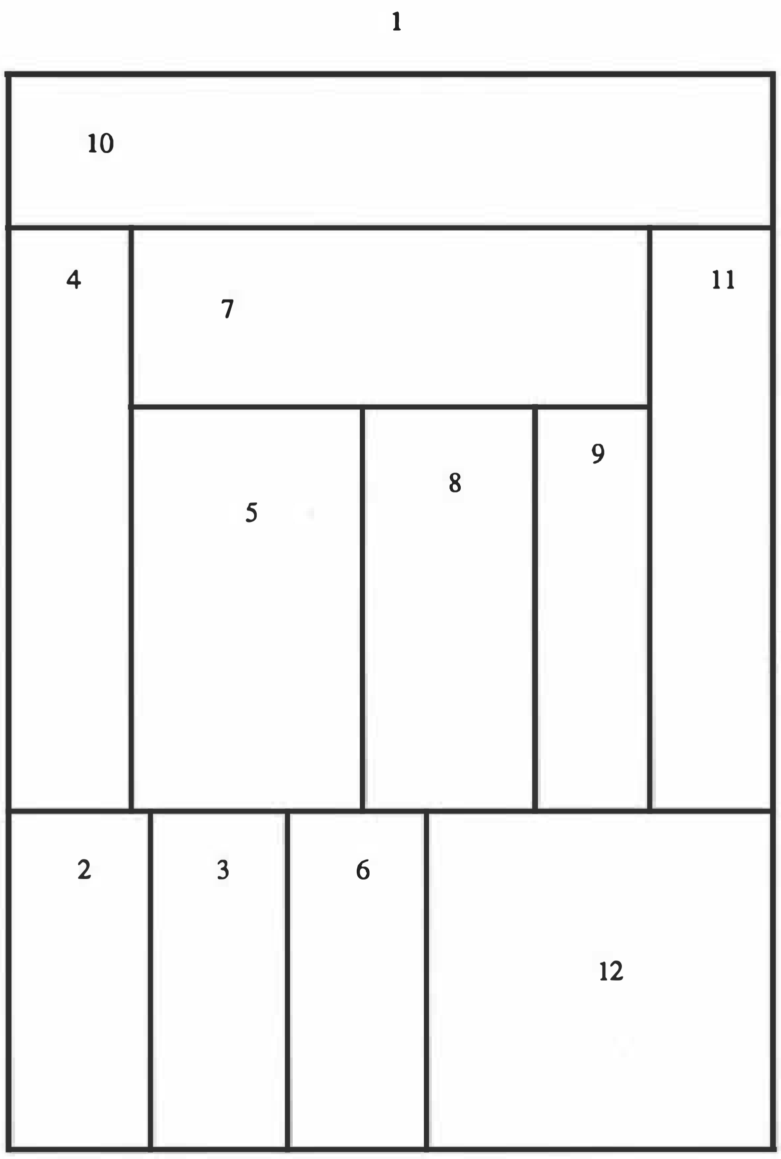


Figure 7.2: Biased Vertex Splitting Algorithm Example

the VSA when using concentric MPGs. If SIMPLE were to place an entire distance class (starting from D_1 then proceeding to the next inner distance class), it is possible that the SIMPLE and VSA layouts could be identical. However, this does not happen, as the VSA considers all facilities in a distance class simultaneously when the regions are dimensioned for placement of the next distance class. SIMPLE provides no such mechanism, placing each facility independently of subsequent placements. For this reason, even if SIMPLE were able to produce the *correct* placement order (which it very unlikely unless input manually), the regularity of facilities would become steadily worse. As we have seen SIMPLE is very susceptible to an irregular facility impinging on other facilities' regularity. Unless we are extremely fortuitous, the Deltahedron Algorithm will not generate a layout to problem instances of this type.

7.3 The Contraction Algorithm

Generating a problem instance which would produce a practical Contraction Algorithm layout, required careful analysis of the placement of facilities using the reverse contraction process. As we discussed in Section 7.1, the process is particularly amenable to disjoint distance classes. If each facility is a placement host explicitly to only one facility, and we are able to maintain regular facility shapes, then we are likely to produce practical Contraction Algorithm layouts. Facility areas were addressed in much the same way as for the VSA example, where we wanted to ensure that the area specifications did not impinge upon the ability to attain adjacencies without forcing irregularity.

The Contraction Algorithm layout of Figure 7.3 is perhaps the weakest of the examples produced in this chapter, as far as showing conclusive evidence of being superior to the other layouts on the same MPG. Table 7.3 shows that the average case values for the Contraction Algorithm layout are all clearly superior to the VSA and SIMPLE values, by significant margins. Disappointingly, however, the worst case values for the bounding golden rectangle and square are inferior to the corresponding VSA values. Examination of the layout shows that the four L-shaped facilities are the likely cause of this. Facility 3 especially appears to be more elongated than would be practical. Facilities 9 and 10 are unnecessarily

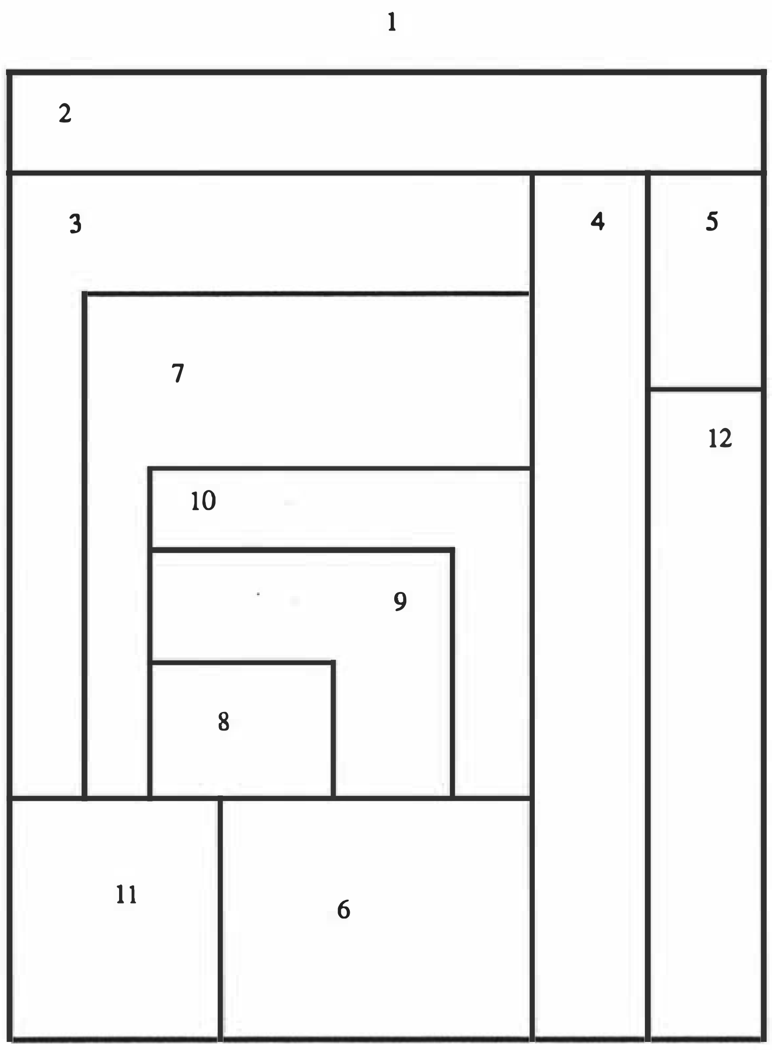


Figure 7.3: Biased Contraction Algorithm Example

L-shaped, and could both be made into rectangles by having facility 8 *cover* the 2-joint currently occupied by facility 10, and redimensioning the block of facilities 8,9 and 10. We formalise this process in Section 8.2.2. This modification (although not possible without manual intervention) will enhance the average regularity values of the Contraction Algorithm layout, however the worst case values for the bounding rectangle and square will remain the same. Perturbation of facility 2's area would remedy this, as it is that facility which currently produces the worst bounding square and golden rectangle values. These modifications would not change the VSA regularity values however. Facilities 3 and 7 would require further investigation into their use before a judgment on the usefulness of this layout could be made.

Facility	1	2	3	4	5	6	7	8	9	10	11	12
Area	-	30	40	40	10	30	40	10	20	20	20	30

	Average Value			Worst Value		
	VSA	CA	SIMPLE	VSA	CA	SIMPLE
C_i	6.9091	**4.7273**	14.7273	12.0000	**6.0000**	56.0000
$a_i/\bar{R}_i$	0.6241	**0.8017**	0.5789	0.2424	**0.3077**	0.1110
$a_i/\bar{G}_i$	0.4241	**0.5818**	0.3377	**0.2371**	0.2156	0.0771
$a_i/\bar{S}_i$	0.2703	**0.4443**	0.2477	**0.1465**	0.1332	0.0476
$4\sqrt{a_i}/P_i$	0.6686	**0.7813**	0.5132	0.3158	**0.5523**	0.1299

Table 7.3: Area Specifications and Regularity Values for Biased Contraction Algorithm Example

The Contraction Algorithm, as we have discussed, is particularly amenable to problem instances where the distance classes are disjoint, or more generally where each placement host can be monitored, to ensure that placements within it do not distort the placement host. The Contraction Algorithm will be dominated by the Deltahedron Algorithm in some instances, however for problem instances of this type which are not Deltahedron-generateable, the Contraction Algorithm is likely to produce the most efficient layouts. We will see in Chapter 9 that the Contraction Algorithm works best within an interactive environment, as the contraction process can be manipulated to produce practical Contraction Algorithm layouts.

The VSA and SIMPLE are not so amenable to MPGs of this type for the same reasons outlined for the Deltahedron Algorithm in Section 7.1. The disjointness of the distance classes for the VSA, and the placement of facilities on the *outside* for SIMPLE as opposed to the placement host idea used in the Contraction Algorithm and Deltahedron. Further the Tiling and Spanning Tree Algorithms again fail to generate layouts as they cannot work off a given MPG.

7.4 SIMPLE

Generating problem instances which generate practical SIMPLE layouts is difficult. As we saw in Chapter 6 SIMPLE is often dominated by either or both of the VSA and the Contraction Algorithm. In order to generate a example amenable to SIMPLE, we were required to identify the conditions under which the VSA and Contraction Algorithm exhibit irregularity. As we have discussed already, disjoint distance classes affect the practicality of VSA layouts, while the Contraction Algorithm can be hindered by the incorrect designation of placement hosts. In generating a problem instance amenable to SIMPLE, we attempted to exploit these two factors. Further we considered the placement order chosen by SIMPLE. By ensuring that each facility was placed along an entire side of the layout, we were able to generate an example which was not applicable to Deltahedron or the Tiling Algorithm, yet remained dimensionalisable. The example was thus constructed in a roundabout way, in which we constructed a layout (which could not be generated by Deltahedron or the Tiling Algorithm), from which we generated the MPG. The area conditions were considered last, whilst examining the presence of elongation in our initially constructed layout.

The SIMPLE layout is clearly more efficient than the layouts generated by the other two Class A methods for this problem - Table 7.4 shows that both the average and worst case values for SIMPLE are significantly better than the values for the other layouts. The layout itself, shown in Figure 7.4, has only facility 12 as non-rectangular, yet facility 12 is not necessarily inefficient. Probably the least usable facility would be facility 7, dependent of course on its function, which is rather more elongated than the other facilities. Furthermore all adjacencies are made by a significant sharing of common wall, with the adjacency between facilities 9 and 12 being the smallest. This example at first appears to be applicable to construction by the Tiling Algorithm; however there does not exist an initial Tetrahedron within this layout (even given the double placement of facility 12), hence construction of this layout by the Tiling Algorithm is infeasible.

In previous sections we have identified structures within the MPG which characterise the applicability of one method over the others. In this instance there appear to be no obvious structures which indicate that SIMPLE will perform well. An indicator of the success of SIMPLE can be seen if the placement of facilities can be made so as to maintain the empty space as a regular polygon. Facilities placed within the empty region become more irregular as the empty region becomes more irregular; however, this property cannot be identified within the MPG. We saw in Chapter 6 that SIMPLE was perhaps the least effective of the layout algorithms. This example does not discount this conclusion, however we maintain that in specific problem instances there may be information that can be gleaned from the

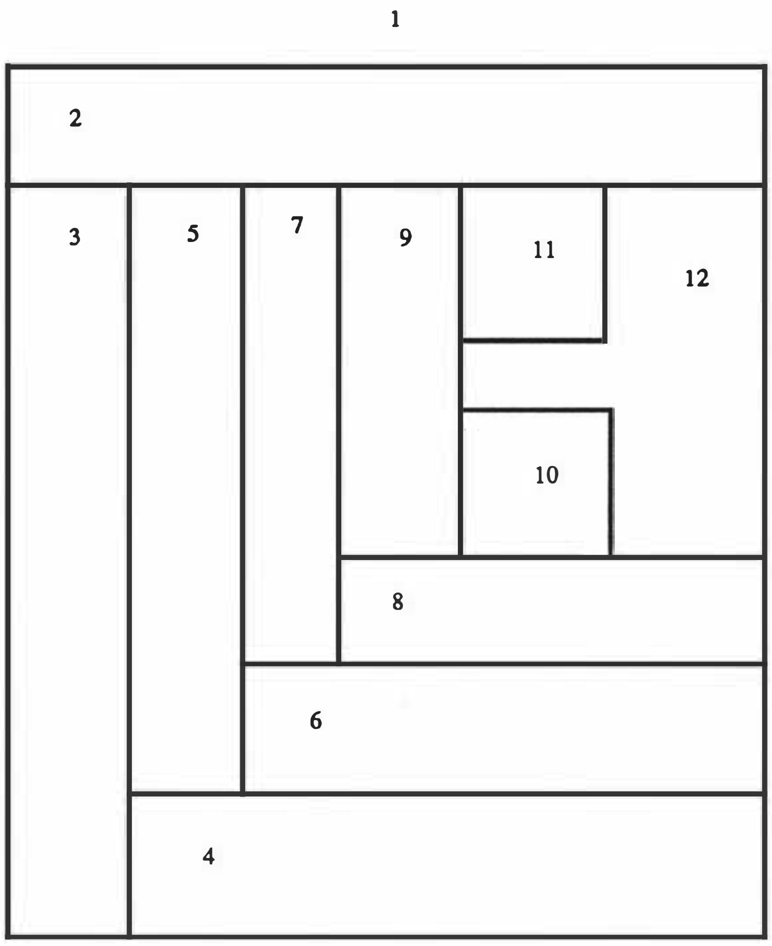


Figure 7.4: Biased SIMPLE Example

Facility	1	2	3	4	5	6	7	8	9	10	11	12
Area	-	40	40	40	30	30	20	20	20	10	10	30

	Average Value			Worst Value		
	VSA	CA	SIMPLE	VSA	CA	SIMPLE
C_i	5.6364	7.8182	**4.3636**	12.0000	18.0000	**8.0000**
$a_i/\bar{R}_i$	0.7867	0.6626	**0.9636**	0.1429	0.1111	**0.6000**
$a_i/\bar{G}_i$	0.4280	0.2911	**0.4945**	0.1429	0.1111	**0.2589**
$a_i/\bar{S}_i$	0.2725	0.2109	**0.3783**	0.1123	0.1057	**0.1600**
$4\sqrt{a_i}/P_i$	0.6997	0.5529	**0.8049**	0.3145	0.2762	**0.6897**

Table 7.4: Area Specifications and Regularity Values for Biased SIMPLE Example

SIMPLE layout, such as the one above, where the structure of the MPG could be identified as not being easily amenable to the VSA or the Contraction Algorithm, SIMPLE may produce more pleasing results.

7.5 The Tiling Algorithm

Generation of a problem instance which was amenable to the Tiling Algorithm was not difficult. As with Deltahedron, we were able to exploit the rigid nature of the placement process to produce a practical Tiling Algorithm layout. Tiling Algorithm layouts have a distinctive pattern to them. In most cases (especially those where all facilities are deemed to be rectangular) the layout appears to show an obvious tiled pattern (hence the name). Having produced a basic Tiling Algorithm layout, all that was required (again) was examination of the area conditions, and ensuring that no facility was unnaturally elongated. Having obtained a practical layout, the corresponding (non-Deltahedron) MPG was generated for application of the Class A algorithms.

The Tiling Algorithm generates a rectangular layout, shown in Figure 7.5. Table 7.5 provides the regularity values for this and the Class A layouts. All except the average bounding square value of Table 7.5 prove the superiority of the Tiling Algorithm layout over the others. The average bounding square value is better for the VSA, by approximately 3%. Note however that the worst case value for this regularity measure is significantly better for the Tiling Algorithm layout. The shape-based measures evaluate exceptionally the Tiling Algorithm layout, as all the facilities are rectangular. Examining the actual layout, we see that it is a realistic layout, with facility 2 the most elongated. This could be remedied by making facilities 2 and 3 a block, meaning that facility 12 would contain the bottom left corner of the layout. This however may lead to a reduction in the effectiveness of facility 12. The same argument could apply for facilities 4 and 10, and facilities 12 and 10.

Facility	1	2	3	4	5	6	7	8	9	10	11	12
Area	-	30	40	30	10	20	10	10	20	20	30	40

	Average Value			
	VSA	CA	SIMPLE	TA
C_i	6.5455	7.6364	8.0000	**4.0000**
$a_i/\bar{R}_i$	0.6140	0.6019	0.7907	**1.0000**
$a_i/\bar{G}_i$	0.4986	0.4168	0.4208	**0.5232**
$a_i/\bar{S}_i$	**0.3639**	0.2901	0.2744	0.3521
$4\sqrt{a_i}/P_i$	0.6901	0.6135	0.6831	**0.8256**
	Worst Value			
	VSA	CA	SIMPLE	TA
C_i	12.0000	14.0000	24.0000	**4.0000**
$a_i/\bar{R}_i$	0.1538	0.0973	0.2105	**1.0000**
$a_i/\bar{G}_i$	0.1538	0.0973	0.2105	**0.2409**
$a_i/\bar{S}_i$	0.1047	0.0973	0.1335	**0.1489**
$4\sqrt{a_i}/P_i$	0.3732	0.2288	0.3704	**0.6717**

Table 7.5: Area Specifications and Regularity Values for Biased Tiling Algorithm Example

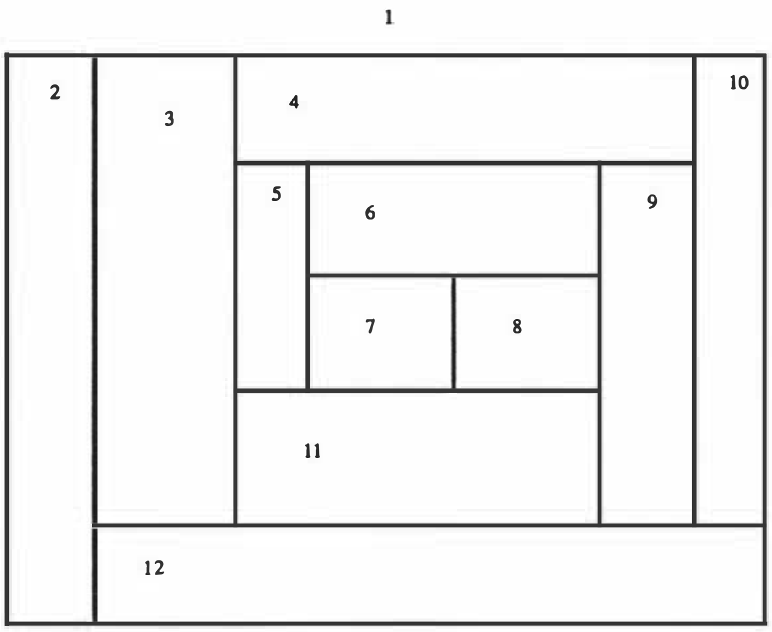


Figure 7.5: Biased Tiling Algorithm Example

The Tiling Algorithm does not generate its layout from the MPG, yet the MPG generated from a Tiling Algorithm layout can provide insights into the effectiveness of the Class A methods. Perhaps surprisingly the Class A methods, when used within an interactive framework, have the potential to mimic the Tiling Algorithm layout. The Tiling Algorithm cannot generate layouts with disjoint distance classes, implying amenability to the VSA, as there is a reduced amount of vertex splitting required. Reversal of the placement order for the Tiling Algorithm will generate the *correct* SIMPLE placement order to obtain the same layout, and the manual assignment of placement hosts for the Contraction Algorithm can ensure the same layout is generated in many instances. Note that although these methods have the potential to generate the same layout, this is in general only possible through manual intervention within the Class A algorithms. We will return to this point further in Chapter 9.

7.6 Spanning Tree

The final method that we have examined is the Spanning Tree method. Again generation of an example amenable to this particular algorithm is based on the generic structure exhibited in all Spanning Tree layouts. Since Spanning Tree layouts guarantee rectangular facilities (except the case where we force the expulsion of a 4-joint) we were able to simply concentrate on ensuring that the area specifications did not allow a 4-joint to be created within the layout, and that the facilities were not required to be elongated. As with the Tiling Algorithm we obtained the corresponding (non-Deltahedron) MPG for comparison with the Class A algorithms.

The layout for this example is shown in Figure 7.6. The regularity values of Table 7.6 give overwhelming evidence of the superior efficiency of this layout over the other methods on this problem. The layout itself subjectively *looks good*, with only facilities 6 and 7 perhaps requiring further examination as they are more elongated than the remainder of the layout. Furthermore all adjacencies are met with a significant amount of common wall length, ensuring no further examination of unusable adjacencies.

The Spanning Tree Algorithm does not work off an MPG, however we can glean information regarding the tree structure used. The Spanning Tree Algorithm performs badly when facilities near the root (the top facility) of the tree are also leaf nodes of the tree. Facilities of this type are in general elongated, and reduce

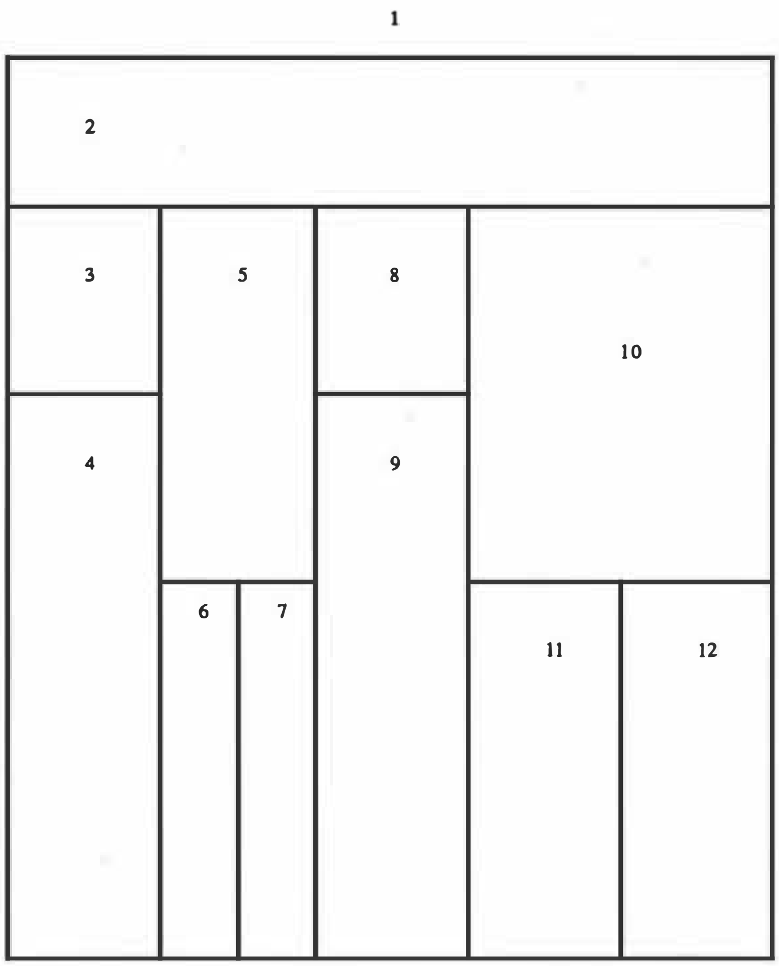


Figure 7.6: Biased Spanning Tree Example

Facility	1	2	3	4	5	6	7	8	9	10	11	12
Area	-	40	10	30	20	10	10	10	30	40	20	20

	Average Value			
	VSA	CA	SIMPLE	Spanning
C_i	7.0909	4.9091	9.2727	**4.0000**
$a_i/\bar{R}_i$	0.6576	0.8910	0.5812	**1.0000**
$a_i/\bar{G}_i$	0.4468	0.3904	0.2996	**0.6159**
$a_i/\bar{S}_i$	0.3060	0.2733	0.2187	**0.4303**
$4\sqrt{a_i}/P_i$	0.6926	0.6704	0.5429	**0.8690**
	Worst Value			
	VSA	CA	SIMPLE	Spanning
C_i	12.0000	8.0000	18.0000	**4.0000**
$a_i/\bar{R}_i$	0.3333	0.3333	0.0833	**1.0000**
$a_i/\bar{G}_i$	0.2430	0.1042	0.0833	**0.3236**
$a_i/\bar{S}_i$	0.1502	0.0644	0.0702	**0.2000**
$4\sqrt{a_i}/P_i$	0.5192	0.3203	0.2563	**0.7454**

Table 7.6: Area Specifications and Regularity Values for Biased Spanning Tree Example

the effectiveness of the layout. Binary Trees (where each vertex which is not a root (having degree 2) or leaf (having degree 1) node has degree 3) produce very practical results as paths from the root node to a leaf node pass through a number of vertices. In the layout this translates to vertical blocks of facilities, as opposed to a single elongated facility. Facility areas must be addressed for the Spanning Tree layouts, not only to ensure that no 4-joints exist, but also to ensure that the vertical adjacencies are adequately represented by the corresponding common wall length in the layout.

The underlying MPG of a Spanning Tree layout does not in general provide much information about the performance of the Class A methods. The Spanning Tree Algorithm is able to produce MPGs with disjoint and connected distance classes, and indeed it requires identification of the structures within the underlying MPG of a particular Spanning Tree layout before possible conclusions could be found for the expected performance of the Class A algorithms. The only method which is likely to perform badly in general is SIMPLE. The facilities are considered as blocks within the Spanning Tree layout; SIMPLE cannot mimic this process. In general the SIMPLE routine will not identify the obvious blocks that can be identified by the Spanning Tree Algorithm, and hence the SIMPLE layout becomes irregular. The distance classes are connected in the above example, implying that the VSA may be applicable. Indeed the worst case values indicate its usefulness over the Contraction Algorithm. However there are identifiable blocks within the layouts which the Contraction Algorithm has the ability to exploit. In contrast to the worst-case values, the average values identify a superiority of the Contraction Algorithm over the VSA. Both however are clearly inferior to the Spanning Tree regularity values.

Coda

In this chapter we have attempted to more closely examine the characteristics of the layout algorithms which make them amenable to *families* of MPGs having special characteristics or sub-structures. We cannot predict exactly the performance of the algorithms, however we have been able to identify some simple structures which direct us to a particular method. We have identified that the Deltahedron, Tiling Algorithm and Spanning Tree algorithms will in general perform the best on any problem to which they are applicable. The Class A algorithms required a deeper analysis. We concluded that MPGs with concentric distance classes were likely to be amenable to solution via the VSA, while the Contraction Algorithm was likely to perform well on problems with disjoint distance classes. We further identified that the area specifications can impact significantly on the resulting practicality of the layout, especially for methods where facilities could become elongated. We have identified that the process of area perturbation becomes important in remedying elongation, and that manual intervention in the determination of the placement order can be especially useful in the Contraction and SIMPLE Algorithms.

This chapter completes the more theoretical component of this thesis. Thus far we have examined the GTFLP algorithms for generating layouts. We have defined the concept of regularity, and have performed research into determining the characteristics of each method. A computational study has been undertaken by which we have seen that the solution process of the GTFLP is more complicated than simply generating the highest benefit layout possible. For this reason the GTFLP requires a closer examination of the problem at hand, rather than the existence of sweeping generalisations.

This chapter has introduced us to some of the more practical concepts of layout design, such as perturbation and interaction, leading us into the remainder of this thesis where we explore the interface between the theoretical model we have so far discussed, and some of the practical aspects which would be required in order to implement the GTFLP model as a basis for architectural or planning design. Chapter 8 examines the possibility of changing problem parameters such as area and adjacency at minimal cost in order to increase the flexibility and efficiency of the layout, and the conditions under which we might do this. Chapter 9 explores user interfaces, and identification of exploitable subgraphs within the MPG, culminating in the development of a generic framework incorporating these ideas.

Chapter 8

Perturbation of Problem Constraints

In this chapter we are interested in identifying possible changes that can be made to a layout problem's data, in order to obtain more realistic layouts, without unduly reducing the benefit that would be obtained from a more theoretical implementation applied to the original problem. We examine two possible ways in which we might perturb our problem: area and adjacency perturbations. Area perturbation encapsulates the process of changing the area specifications of facilities within a problem, and further identifies a relationship with a related topic: Floorplanning. Adjacency perturbation goes beyond simple adjacency swaps (which we will see are not simple when conducted in a layout), to an examination of one of the fundamental assumptions of the GTFLP: the validity of the maximal planarity assumption.

8.1 Area Perturbation

In this section we explore the possibility of changing given area specifications, with the aim of improving layout regularity. Firstly we examine the effect changing area requirements has on a layout; we then proceed to outline ways in which we can systematically perturb the area specifications of our problem. Subsequently we highlight connections between literature for the FLP and that for Floorplanning, and discuss how we might utilise ideas from this latter area, in order to apply them to the GTFLP.

8.1.1 Effect of Area on the Layout

In order to motivate the possible advantages of area perturbation, let us consider the impact area specifications have on a layout. Firstly consider a five facility layout which is undimensionalisable, as shown in Figure 8.1. Now the layout requires that the areas are compatible, *i.e.* $a_2a_4 > a_3a_5$, for this layout to dualise the MPG upon dimensionalisation. If this condition is not met, the layout cannot successfully correspond to the problem data in its current form.

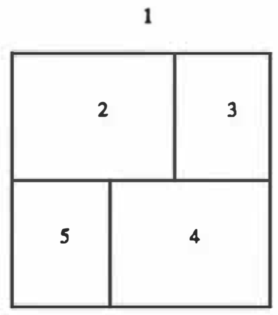


Figure 8.1: Example to show how area pertubation can create infeasibility

Now consider an undimensionalisable layout such as might be generated by the VSA. The area specifications dictate whether we can place each facility between two single rectilinear segments. Consider a situation which schematically resembles Figure 5.4. In this instance, there is likely to be an earlier placed facility whose area specification is either too small, or too large, compared to the other facilities being placed along that side of the layout. This phenomenon requires only one facility area to be incompatible, and it may impact on the regularity of the entire layout.

Finally consider a dimensionalisable layout, generated by Deltahedron or the Spanning Tree Algorithm, for instance. The layout can always be successfully dualised regardless of the area conditions, by definition of being dimensionalisable. The question here becomes: is the resulting layout practical? An obvious example is where the top facility has small area relative to the other facilities, it may become too elongated to be practical - recall Figure 5.6, where we showed that the average regularity would tend to one, while the regularity of the top facility tended to zero, and hence became impractical. This particular problem could have been remedied by allowing the top facility to increase in area to maintain a given width-to-length ratio for instance, although the remedy in general is not necessarily so simplistic.

The three examples described have highlighted three main areas where area specifications can severely impact upon the feasibility of a layout, ranging from possible infeasiblity in the first instance, through to unnatural elongation of dimensionalisable facilities in the last. The existence of these area incompatibilities in a large number of the problems generated, led to a more detailed examination of the importance of the area requirements. We will return to this point in Section 8.1.3, but we first examine the interaction between Floorplanning and the GTFLP in Section 8.1.2, and in particular the issue of area specifications, in order to lay a stronger foundation for Section 8.1.3.

8.1.2 Floorplanning and the Graph Theoretic Facility Layout Problem

Floorplanning remains a largely untapped area in GTFLP. Floorplanning techniques are applied to similar applications such as VLSI design, but there is one important difference: Floorplanning ideas and methodologies do not consider facility areas. Floorplanning literature appears to direct future research at the incorporation of facility areas, but thus far nothing applicable to the GTFLP has appeared. This difference between the two problems actually forms a vast chasm between the two areas, because of the significant impact the area specifications have on the GTFLP; hence Floorplanning is not directly applicable to the GTFLP, as the layouts generated by Floorplanning methodology are usually undimensionalisable, and imposing area conditions will in general lead to infeasibility of the layout. We have already seen examples of this in Section 8.1.1.

The exterior facility is not considered to be a vertex, but vertices u, l, r and b, called the *outer* vertices, corresponding to the top, left, right and bottom respectively, are added to form a 4-cycle. Edges are then added between vertices on the infinite face of the original graph and the new 4-cycle. This new graph is called an extended plane triangulated graph (EPTG). A vertex adjacent to two or three of these vertices u, l, r or b will be a corner facility in the layout. In this section the intent is to examine the literature on Floorplanning, in order to gain insight into how the Floorplanning methodology might be applicable to the GTFLP. Unless stated otherwise, the layouts discussed in this section are all assumed to be undimensioned.

Initial advances in the area of Floorplanning were concerned only with rectangular layouts. Leinwand and Lai [70, 75] introduced an $O(n^2)$ time algorithm for determining if a given plane triangulated graph (PTG) will admit a rectangular dual. The algorithm searches for separating triangles, and if none exist, then a rectangular dual will exist, via a perfect matching. Kozminski and Kinnen [64, 65] introduced a $O(n^2)$ algorithm for determining an undimensioned rectangular dual, if it exists, from a given planar graph. Essentially all that is required is to assign a horizontal or vertical orientation to each of the edges in the PTG. This is done by a recursive division of the ETPG, into subgraphs, until the orientations become trivial, based on an initial orientation of the edges between the corner vertices, and the outer vertices. Kozminski and Kinnen [66] presented a graph theoretic characterization of rectangular duals, giving necessary and sufficient conditions for their existence. They also developed a technique for enumerating all rectangular duals for a given PTG, by examining alternative orientations during the development of the duals. Bhasker and Sahni [9] improved upon the previous work by presenting a linear time algorithm to determine if a rectangular dual exists for a given planar graph. Bhasker and Sahni then extended this work in [10], describing a linear time algorithm for finding an undimensioned rectangular dual. Their method is based upon construction of a path digraph reflecting the *on top of* relations defined by the dual; traversing down the digraph from the *head* node, which is deemed to be *on top of* all facilities dictates the spatial location of each facility in the layout; this is a similar methodology to that of the Spanning Tree Algorithm, where the root node of the Spanning Tree Algorithm would be the head node in this case.

Sun and Sarrafzadeh [103] extended the work on rectangular duals to include L-shaped (or 1-concave rectilinear) modules. A test in $O(n^{\frac{3}{2}})$ time determines if a given planar graph will admit an L-shaped dual and, if it does, a construction is generated it in $O(n^2)$ time. This is done by finding the nesting of the complex (separating) triangles, and developing a rooted tree representing this nesting. A vertex from each complex triangle adjacent to the root is chosen, and if all the vertices chosen can be distinct, and not any of u, r, l or b, then a L-shaped dual will exist; with each chosen vertex being L-shaped in the resulting dual. The dual is constructed via a direct modification of Kozminski and Kinnen [65], by recursively dividing the graph into smaller ones satisfying the conditions of L-shaped duals (rather than those for rectangular duals originally in [65]), until the trivial case of five vertices is produced. The split parts are then merged in the same fashion of Kozminski and Kinnen [65].

Yeap and Sarrafzadeh [109] further extended this work to allow T, U, W, and Z shaped (or 2-concave rectilinear) modules, and proved that these facility shapes, together with the L and rectangular shapes, were sufficient to generate an undimensioned dual for any planar graph. The method relied upon the proof that complex triangles could be assigned to vertices, with the number of complex triangles assigned to any one vertex not exceeding two, hence the requirement of only 2-concave (or less) rectilinear modules. This assignment performs similar steps to those of the placement directions of the Deltahedron Layout Algorithm, to ensure that worst case facility shape is maintained. The authors also showed by the use of a counterexample that L and rectangular shaped modules were insufficient for use with arbitrary planar graphs. The construction of the dual is performed in linear time, and is a further modification of the *divide and conquer* type of strategy employed by both Kozminski and Kinnen, and Sun and Sarrafzadeh, where the graph is divided into smaller more manageable components.

The only work involving area in the Floorplanning literature was due to Yeap and Sarrafzadeh [108] who explored the dimensioning of rectangular duals, by allowing *dead area*, or empty space, within the layout, and allowing adjacencies to be relaxed from those specified in the PTG in order to complete the dimensioned layout.

This section has described the Floorplanning literature, and provides interesting results; especially Yeap and Sarrafzadeh's proof that at worst 2-concave rectilinear modules are required to produce undimensioned layouts for arbitrary planar graphs. In the next section we aim to ulitise this and other ideas from the Floorplanning literature, as we examine the possibility of perturbing facility areas, and the conditions under which this is advantageous.

8.1.3 Methods for Perturbation of Facility Areas

In this section we investigate models which allow *soft* area conditions *i.e.* conditions which we desire to satisfy, but which can be violated if necessary. Any violations are performed within a controlled environment, for instance within upper and lower bounds on the area of facilities, as we still desire to minimise the deviation from the original problem data. There are three models we wish to consider here. The first is a different Graph Theoretic method to those which have been discussed already, and is due to Roth, Hashimshony and Wachman [98]. In their model the area specifications are stated in terms of upper and lower bounds on the lengths of the walls in the x and y directions (although which is which, is not known initially), providing a range of possibilities for the dimensioning of the layout, with the overall objective to minimise the length of the building perimeter in each direction. The method considers only rectangular facilities, which can be overly restrictive but allows easier manipulation of the facility areas and shapes. The exterior is again represented by four vertices, labelled north, south, east and west, and the adjacencies with these vertices are given in the (not necessarily maximal) planar graph. The graph is split into two subgraphs representing adjacencies between facilities in the x (north-south) direction, and the other representing adjacencies between facilities in the y (east-west) direction. This is achieved by directing the original graph, and then dividing it using a colouring technique, adding edges to attain maximal planarity if required. The two graphs are called the Coloured Directed Subgraphs (CDSG). These CDSGs are then converted to Dimensions Subgraphs (DSG) where the vertices represent walls, and the edges the distances between these walls. The DSGs are converted to flow networks with a sink and source. The PERT technique [91] is then used to determine the minimum cost flow through these two networks in order to obtain the minimum perimeter dimensions in each direction. Those facilities on the critical path will be assigned their minimum wall length allowable in that direction. The realisation of the layout is obtained by examining combinations of the dimensions available to ensure there is no violation of adjacency requirements (which the authors state can be easily achieved), providing a set of alternative layouts which can be then evaluated using the regularity measures of Section 2.5, or via the subjectivity of an experienced layout planner. The flexibility of this approach lies in the availability of alternative decisions at four key points in the algorithm: the various ways of adding extra edges to the graph to enable the colouring process, the different ways of colouring the graph, the varying of the slacks for dimensions of facilities not on the critical path, and the ability to interchange the dimensions in each direction specified for each facility. Roth *et al.* state that the method can also be extended to include non-convex layout perimeters and non-rectangular facilities, however the authors do not describe how this might be achieved. Since we are examining area perturbations, it is interesting to note that the wall dimension bounds must be *flexible enough* to allow a feasible solution. Consider for example a problem which has a facility with lower bounds of 100 in both dimensions, and 10 other facilities have upper bounds of 5 in each direction. The method of Roth *et al.* would then fail to complete even one layout, as there would be no feasible solution to the flow network problems.

The second model that we wish to consider is developed for application to dimensionalisable layouts. We have seen already that methods such as Deltahedron and the Tiling Algorithm generate dimensionalisable layouts. However, as we have the seen in Figure 5.6 for example, dimensionalisable layouts are not always usable or efficient layouts. The concept of dimensionalisable layouts is useful primarily for a shape-oriented generation of layouts, with a specified worst case facility shape. In each dimensionalisable layout there will exist a set of *optimal* area specifications, where every facility is bounded by a square, leading to exceptional regularity values for each facility, or such that each facility's area allows the facility to function as intended. However this *optimal* set of area requirements may be very different from those specified in the original problem data, as the *optimal* specifications are dependent on the structure of the layout. Our goal then is to minimally vary the area specifications on the facilities, in order to make facilities *squarer*, *i.e.* to make elongated facilities come within a specified width-to-length ratio, or to maximise ulitisation. In order to implement this type of idea it is desirable to have a variety of different layouts available, (*e.g.* for Deltahedron, in principle, generate all possible layouts using a search tree), as having, if possible, a facility with large area as the top facility may reduce elongation exhibited in an alternative layout without actually changing the area conditions.

We can formulate this problem as a non-linear problem with a quadratic objective function and linear constraints. Firstly we show the general formulation for at worst T-shaped layouts in Formulation 3, and then, with the aid of an example, show a particular formulation (mainly to illustrate the *problem defined continuity constraints*). We will see that the formulation is adaptable to any specified worst-case facility shape of a dimensionalisable layout. The objective function is quadratic because area is determined by multiplying the width and the length of a facility. Needless to say, this non-linearity is the reason that finding minimal area perturbations to satisfy the given constraints is difficult.

Formulation 3 *Dimensionalisable Area Perturbation*

w_i,l_i	Width and length of the bounding rectangle of facility i
J_i,K_i	Sets of those facilities which define blocks of facilities contained within i
δ	Length to width ratio ($\delta \geq 1$)
β	Minimum wall length ($\beta > 0$)

$$\begin{array}{lllll} Min & \sum_{i=2}^{n} |a_i - (w_i l_i - \sum_{j \in J_i} w_j l_j - \sum_{k \in K_i} w_k l_k)| & & & \\ s.t. & \delta w_i - l_i & \geq & 0 & i = 2 \ldots n \\ & \delta l_i - w_i & \geq & 0 & i = 2 \ldots n \\ & \textit{Problem-defined continuity constraints} & & & \\ & l_i, w_i & \geq & \beta & \forall i \end{array}$$

Note firstly the definitions of w_i and l_i. w_i and l_i define the bounding rectangle of facility i. From this we can delete any blocks of facilities which are contained within i defined by the sets J_i and K_i. We assume that facility 1 is the exterior. The length to width ratio δ, may be facility dependent requiring the use of δ_i. For rectangular facilities $J = \emptyset$ and $K = \emptyset$, while L-shaped facilities will have $K = \emptyset$ only. Layouts which have at worst L or rectangular shaped facilities generate an obvious special form of the objective function in Formulation 3, while as we have seen, the most general dimensionalisable layout has a worst case facility shape of X; the formulation can be easily extended by allowing two new sets L_i and M_i, representing the other blocks of facilities which may exist within a facility. The *problem-defined continuity constraints* are better shown via the aid of the following example; they simply ensure that there are no *holes*, or overlapping facilities, in the layout.

Equations 8.1 - 8.21 depict the area perturbation problem as applied to the layout given in Figure 8.2. As was the case with δ, the β values can depend on i, and indeed the β values for Equations 8.9 and 8.15, which both relate to facility 5, could be different, depending on the problem.

$$Min \quad \sum_{i=2}^{n} |a_i - (w_i l_i - \sum_{j \in J_i} w_j l_j - \sum_{k \in K_i} w_k l_k)| \tag{8.1}$$

$$s.t. \quad \delta w_i - l_i \geq 0 \qquad i = 2 \ldots 10 \tag{8.2}$$

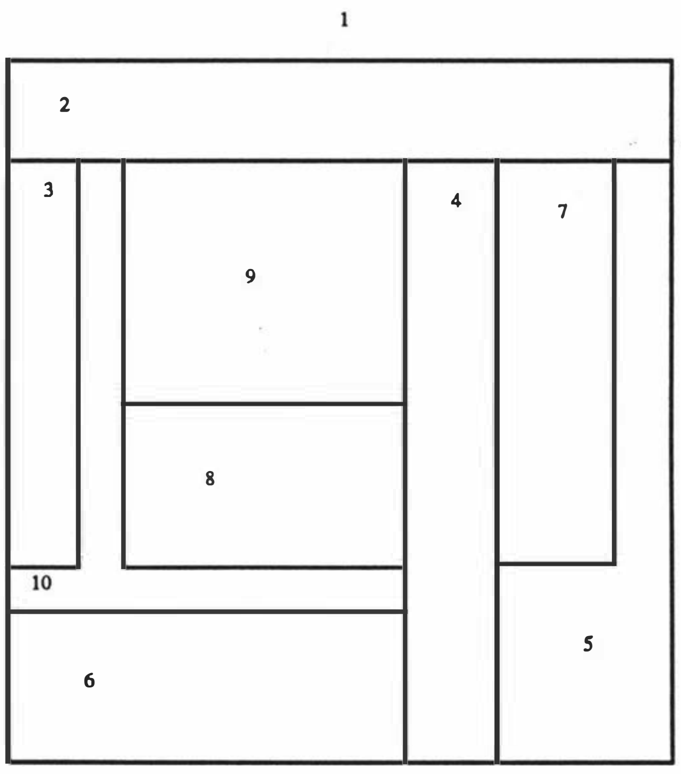


Figure 8.2: The dimensionalisable layout on which Equations 8.1 - 8.21 are based

$$\begin{aligned}
\delta l_i - w_i &\geq 0 & i = 2\ldots10 && (8.3)\\
w_1 &= w_2 &&& (8.4)\\
w_1 &= w_{10} + w_4 + w_5 &&& (8.5)\\
w_6 &= w_{10} &&& (8.6)\\
w_8 &= w_9 &&& (8.7)\\
w_{10} &\geq w_3 + w_9 + \beta &&& (8.8)\\
w_5 &\geq w_7 + \beta &&& (8.9)\\
l_1 &= l_2 + l_{10} + l_6 &&& (8.10)\\
l_4 &= l_6 + l_{10} &&& (8.11)\\
l_5 &= l_4 &&& (8.12)\\
l_{10} &\geq l_8 + l_9 + \beta &&& (8.13)\\
l_{10} &\geq l_3 + \beta &&& (8.14)\\
l_5 &\geq l_7 + \beta &&& (8.15)\\
J_5 &= \{7\} &&& (8.16)\\
J_{10} &= \{3\} &&& (8.17)\\
K_{10} &= \{8,9\} &&& (8.18)\\
J_i &= \emptyset & i = 2,3,4,6,7,8,9 && (8.19)\\
K_i &= \emptyset & i = 2,3,4,5,6,7,8,9 && (8.20)\\
l_i, w_i &\geq \beta & \forall i && (8.21)
\end{aligned}$$

The general formulation of Formulation 3 is not easily solvable. It requires the use of non-linear programming techniques which we do not desire to discuss here, however the interested reader is referred to Bunday [20]. The point is that we could solve this formulation as it stands if we desired, however we can linearise our objective function by incurring a small percentage error, between the resulting actual area and that stated in the objective function. The following work on the linearisation of the equation $a = lw$ is due initially to Lacksonen [69].

Consider the graph of Figure 8.3, where the dotted line is the line corresponding the equation $a = wl$, and the solid line represents a piecewise linear approximation of this equation. The segment endpoints are given by Equations 8.22 - 8.24.

$$\begin{aligned}
l^{max} &= w^{max} = \sqrt{\delta a} && (8.22)\\
l^{med} &= w^{med} = \sqrt{a} && (8.23)\\
l^{min} &= w^{min} = \sqrt{a/\delta} && (8.24)
\end{aligned}$$

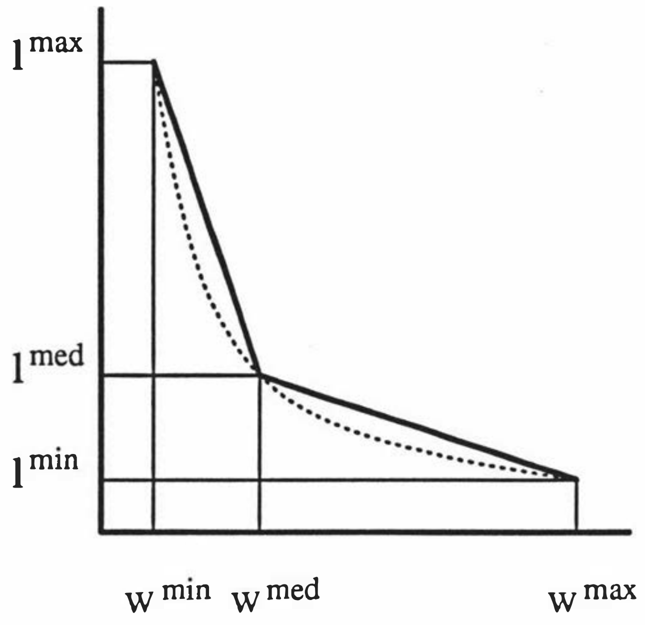


Figure 8.3: Graph of $a = wl$, and its piecewise approximation

Define two new variables y_1 and y_2, representing the length of w in the left and right segments respectively. The equations for the length and width are then defined by Equations 8.25 - 8.28.

$$l = l^{max} - \sqrt{\delta}y_1 - \frac{1}{\sqrt{\delta}}y_2 \quad (8.25)$$
$$w = w^{min} + y_1 + y_2 \quad (8.26)$$
$$y_1 \leq w^{med} - w^{min} \quad (8.27)$$
$$y_2 \leq w^{max} - w^{med} \quad (8.28)$$

Note that the slopes of the left and right hand segments, are $-\sqrt{\delta}$ and $\frac{-1}{\sqrt{\delta}}$ respectively, leading to the coefficients for y_1 and y_2 in Equation 8.25. This linearisation leads easily to a mixed integer formulation, by replacing the non-linear objective function with the piecewise linear approximation for the area, coupled with the standard techniques for prioritisation of the segments, and linearisation of absolute value functions.

Lackonsen states that if $\delta = 2$, the areas are between 0% and +3% of their true values, while a single segment linearisation of the curve $a = wl$ would result in errors from −9% to +12%. If further accuracy is required, a four segment linearisation will result in areas with at most 0.8% error. The cost of the small errors in the areas is offset by the ability of the integer program to be more amenable for solution than the non-linear problem. Furthermore we may impose a cost coefficient upon each of the terms in the objective function representing preferences for perturbations, or alternative costs on perturbing different facilities. This is useful for ranking which facility areas we prefer to perturb.

The third model examines possible area perturbations in undimensionalisable layouts, involving ideas from Floorplanning. We have discussed already that undimensionalised layouts can be created with at worst 2-concave rectilinear modules. Therefore the problem that remains is to impose (if possible) a dimensioning upon the layout which is as close as possible to the specified facility areas, without violating the adjacencies in the layout. Upon generation of a Floorplanning layout, we simply impose a grid over the layout, with an intersection of a vertical and horizontal grid line at every corner point in the layout. A non-linear problem can then be formulated, whose solution would generate the width and length of each row and column of the grid, thus obtaining the areas for each facility. We desire to satisfy a set of equations of the form of Equation 8.29, where the set I is the set of grid squares (j, k) comprising facility i.

$$\sum_{(j,k)\in I} w^j l^k = a_i \qquad i = 2 \ldots n \tag{8.29}$$

As an example, consider the layout of Figure 8.4, where the dotted lines represent the overlaid grid. For this layout the set of equations would be as given in Equations 8.30 - 8.40. The lower bounds of Equation 8.40 ensure that every grid is of at least a specified minimum length, and as with the second model we considered, could be dependent on i.

$$\begin{aligned}
l^a(w^a + w^b + w^c + w^d + w^e + w^f) &= a_2 && (8.30)\\
w^a(l^b + l^c + l^d + l^e + l^f) &= a_3 && (8.31)\\
l^b(w^c + w^d) + l^c(w^b + w^c + w^d + w^e) &= a_4 && (8.32)\\
l^b w^b &= a_5 && (8.33)\\
l^b w^e &= a_6 && (8.34)\\
l^c(w^b + w^c) &= a_7 && (8.35)\\
l^c(w^d + w^e) &= a_8 && (8.36)\\
l^e w^b &= a_9 && (8.37)\\
l^e(w^c + w^d + w^e) &= a_{10} && (8.38)
\end{aligned}$$

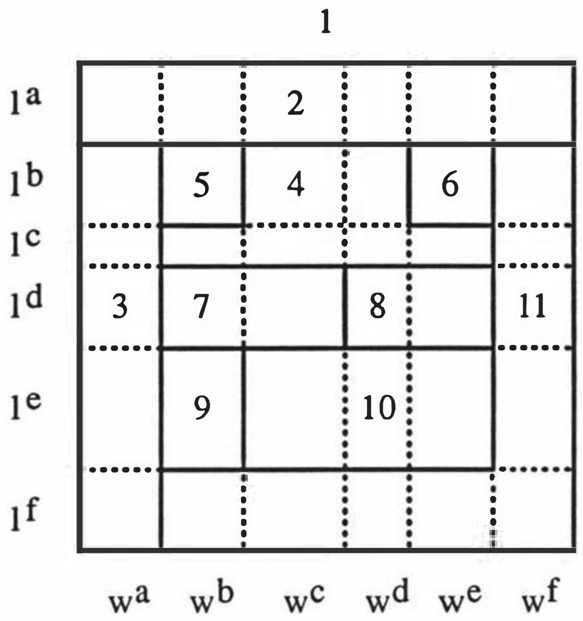


Figure 8.4: An undimensionalisable layout, with an overlaid grid

$$l^f(w^b + w^c + w^d + w^e + w^f) + w^f(l^b + l^c + l^d + l^e) = a_{11} \qquad (8.39)$$
$$l^i, w^i \geq \beta \qquad \forall i \qquad (8.40)$$

The problem with solving these systems of equations, other than the quadratic terms, is the inability to determine whether a feasible solution exists, as there is no non-linear equivalent to the rank function used to determine if linear systems have a solution. Therefore it is wise to consider a relaxation of the system, whereby we formulate a nonlinear problem, with the objective of minimising the difference between the required area, and the actual area. This system can be further approximated using the linearisation technique for the area as we discussed for the second model. The second and third models are essentially equivalent, as the dimensionability of the layouts in the second model implicitly impose a grid structure. However the formulation for the dimensionalisable layout is more elegant, and less restrictive, due to the ability of the dimensionalisable layouts to not lose their adjacencies.

The Floorplanning ideas are interwoven with the third model as we attempt to generate dimensionalisable layouts from the layouts generated using Floorplanning techniques. The formulations require either powerful techniques for solving non-linear problems or, alternatively, approximations of the formulation for ease of solution generation, via approximate linearisation of the nonlinear functions. The design of heuristics using these ideas may prove useful as the formulations, although theoretically correct, are very difficult to solve. Perhaps in conjunction with the use of powerful search techniques such as Tabu Search, reasonable solutions may be found quite quickly. A special case exists for rectangular layouts. The Floorplanning theory has developed necessary and sufficient conditions on the existence of rectangular layouts, and techniques for generating a corresponding layout. The results of Rinsma [94] can be extensively applied to undimensionalisable layouts, as Rinsma developed a theory for area conditions on rectangular layouts. By considering blocks of facilities, as we would for the Deltahedron method, we can develop a set of conditions for the areas which satisfy the adjacencies of the MPG. Unfortunately this does not easily extend to 1- or 2-concave rectilinear facility layouts.

In this section we have outlined three models which could be considered in perturbing facility areas. The first model was essentially added for completeness, although it may provide an alternative perspective to problem instances which we may consider. The second and third models are the two models which we would see as being most useful for dimensionalisable, and undimensionalisable layouts respectively. Although as we have discussed, the development of heuristics to solve these would prove most useful.

8.2 Adjacency Perturbation

We now turn our attention to a second way in which we might change our problem data. We saw in Section 3.3 that the Γ-operation could perturb an MPG by swapping an edge in the MPG for another edge not currently in the MPG. We want to now explore the likelihood of extending the simplicity of adjacency swaps in the MPG to the layout, for the purpose of making layouts more effective.

8.2.1 The Non-Triviality of Adjacency Perturbation

If possible it is desirable to be able to make changes in the MPG, which could have easy and direct impact on the layout. For example if we make one α-operation on an MPG, that change affects only four of the vertices in the MPG, independent of the other vertices in the MPG. However, the same change in a layout could have

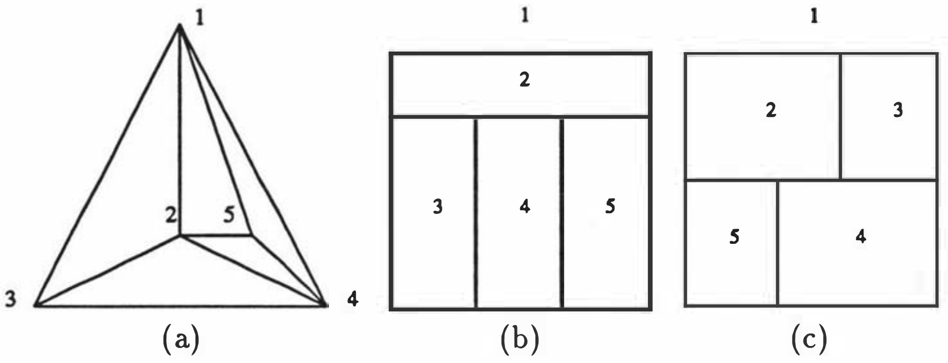


Figure 8.5: First example of the non-trivial process of adjacency perturbation

catastrophic affects on the ability of the layout to now reflect all adjacencies in the MPG. We will now show this with the aid of three examples.

Consider the 5-facility problem of Figure 8.5(b) and the Γ-operation $(1,2) \rightarrow (3,5)$. There is no easy way of implementing this adjacency swap without rearranging the entire layout, *i.e.* the local MPG Γ-operation is non-local in the layout.

Consider Figure 8.5(c) with area condition $a_2a_4 > a_3a_5$, and the Γ-operation $(2,4) \rightarrow (3,5)$. There is no easy way of implementing this adjacency swap without rearranging the entire layout. *i.e.* area specifications may induce non-trivial swaps in the layout.

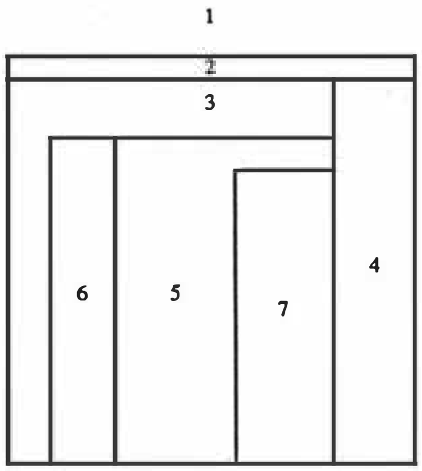


Figure 8.6: Second example of the non-trivial process of adjacency perturbation

The above two examples are somewhat contrived, so let us consider a slightly more realistic problem, shown in Figure 8.6. This example has $|D_1| = n - 1$, and is laid out using the method of Rinsma [94]. Consider the Γ-operation $(3,4) \rightarrow (2,5)$. Again, there is no easy way of implementing this adjacency swap without rearranging the entire layout. This swap would require a re-dimensioning of the block of facilities (3,5,6,7), and possibly the repositioning of facility 6. Hence we have seen that adjacency perturbation cannot be guaranteed to be local, if done arbitrarily. Therefore, while it appears that we cannot extend the simplicity of the adjacency swap in the MPG to the layout in a general setting, we can examine conditions under which we may make inroads into this problem, which we investigate next.

8.2.2 Methods for Perturbation of Adjacencies

In this section we attempt to develop some methodologies for adjacency perturbations within a layout. Firstly we must determine the conditions under which we would want to exchange an adjacency, and then we must determine how we would complete that exchange. When developing the MPG, we assign edges to the MPG based upon the benefit derived from having the two facilities incident on that edge adjacent. Now, in order for this benefit to be fully realised within the layout, the practicality of this adjacency must be assessed. Obviously if we require movement of large amounts of materials between two facilities, it is unreasonable to join them by a small door, leading to the benefit of having those two facilities adjacent being severely compromised. We define *practical adjacency* to be an adjacency specified in the MPG, which is fully realised, *i.e.* we attain the fullest benefit, within the layout. Practical adjacency then could be the requirement of a minimum common wall length between two facilities, or it could be that the adjacency does not require either of the facilities to become unnecessarily irregular because of it. The opposite to practical adjacency would be *theoretical adjacency*, where adjacencies exist in the MPG, but in the layout the benefit of having them adjacent is compromised. Therefore the objective is to ensure (as far as possible) that every adjacency is practical. This was in part addressed by the experiment of Section 6.6, where we constructed a multiplicative model in which it was assumed that the regularity measured proportionally how much of the adjacency benefit for the layout was realised. In this section we are attempting to extend these ideas, by examining each adjacency as independently as possible, in order to ensure regularity, whilst maintaining a larger proportion of each adjacency benefit. The problem then is to attempt to modify an existing layout by either making theoretical adjacencies become practical, or by swapping theoretical adjacencies for practical adjacencies without reducing the overall adjacency benefit excessively.

Welgama, Gibson and Al-Hakim [106] proposed a knowledge-based approach to this problem. The knowledge base uses a web grammar, which is a set of logical statements which are designed to mimic the way in which a decision planner might tackle the problem. Their method uses fundamental ideas from the classical CORE-LAP, and SIMPLE approaches. The layout is built by choosing a facility already in the layout which does not have all its adjacencies satisfied, termed the *central facility.* All facilities which have adjacency with this central facility which are not in the layout, are then sequentially placed, so as to form a chain around the central facility. The web grammar rules are used for the actual placing of each facility. Facilities are placed as dimensioned rectangles, firstly by considering the adjacency with the central facility, and then other adjacencies within the partial layout, with the objective of minimising the number of adjacencies which are not met, and also minimising the empty space within the layout. Once the layout is completed an adjustment phase is used to further reduce the empty areas. Initialisation is by choosing the exterior as the central facility. This method was designed to produce layouts with regularly shaped facilities. As we are examining the possibility of adjacency perturbations, it is not unreasonable to expect that the final layout does not completely represent the MPG, however the objective function (which minimises the *number* of adjacencies not met) is completely unrealistic. There would be a large discrepancy between the non satisfaction of two small weighted adjacencies compared to two large weighted adjacencies. The repercussions of this could be the resulting layout representing only a fraction of the adjacency benefit derived from the MPG. The authors do however partially redeem themselves by stating that an interactive feature can be enabled to produce alternative layouts, which would alleviate some of this problem. In order to determine the usefulness of procedures such as this, the eventual benefit of the layout, and its corresponding regularity must be weighed against what could have been achieved by techniques such as Deltahedron, the Tiling Algorithm, and the Spanning Tree Algorithm. The shape regularity of these methods is guaranteed, and the adjacencies are considered corresponding to their benefit at the time of construction of the MPG, rather than the approach employed by Welgama *et al.*, where the benefits are regarded as irrelevant. These methods can determine benchmark figures for the layout before attempting to remedy higher weighted, irregular layouts. Obviously, if these layouts with the guaranteed worst case facility shape perform within an acceptable decrease in benefit, then the problem is nullified.

Let us now consider ways in which we may modify an existing irregular layout, to hopefully make theoretical adjacencies become practical. We have seen in Section 8.2.1 that attempting any sort of change in adjacencies within the layout is non trivial. The strategy then must be to develop procedures which remain as local as possible. Figure 8.7 presents a fundamental operation from which complicated perturbations can be produced.

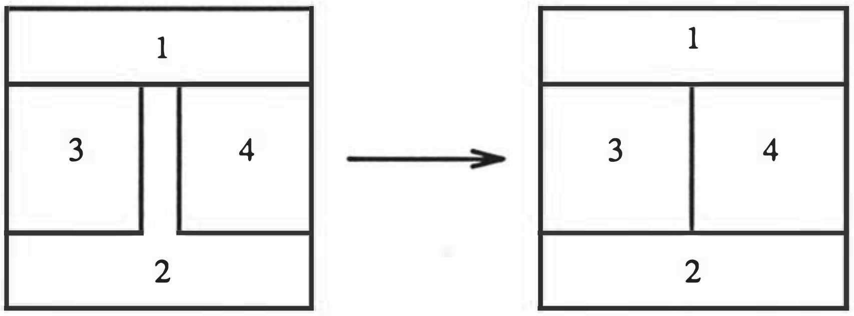


Figure 8.7: Fundamental Operation for Adjacency Perturbation: Γ-operation $(1,2) \rightarrow (3,4)$

The increase in benefit of making this swap will be $w_{34} - w_{12}$. If this cost is acceptable, then we allow the swap. The redimensioning poses a problem, however, so in general we make the assumption that the rectanguloid of facility 2 which was deleted was *small*. This assumption is valid, as if the area was not small, the adjacency between facilities 1 and 2 may well have been practical. Blind application of this operation is likely to lead to difficulties; if, for example, facilities 3 and 4 were actually blocks of facilities, then we would need to be careful of the impact of creating a fault line at the new common border of the blocks 3 and 4. In that case the cost of the swap would then be a more complicated function, as we may be making more than one adjacency swap at a time. If we are making k swaps at once say, then in the MPG, this corresponds to a cycle of length $k+3$ consisting of the facilities involved in the k swaps, where the edges within the cycle are removed, and replaced in a new formation.

The operation is also useful for getting rid of so-called *tentacles*. Methods which use the ODA are prone to having some facilities with a large number of corners, but with more than 90% of the area contained within one rectangle. The remaining area is used to create narrow segments which are formed to maintain the adjacencies from the MPG, hence the formation of tentacled facilities. The adjacencies maintained by these tentacles are, in most cases, destined to be theoretical adjacencies. Figure 8.8 presents a simple example of this, where we see facility 5 has two small tentacles. By deleting the horizontal tentacle first, we are able to redimension facilities 3 and 6 to form a block, while the deletion of the second tentacle allows facility 5 to become a rectangle. Note that the ability to redimension facilities 3 and 6 was fortunate, and more often than not, is not possible; if we had performed the adjacency swaps in the reverse order the redimensioning would not have been (locally) possible. The increase in benefit of these two swaps was $(w_{36} - w_{57}) + (w_{23} - w_{15})$.

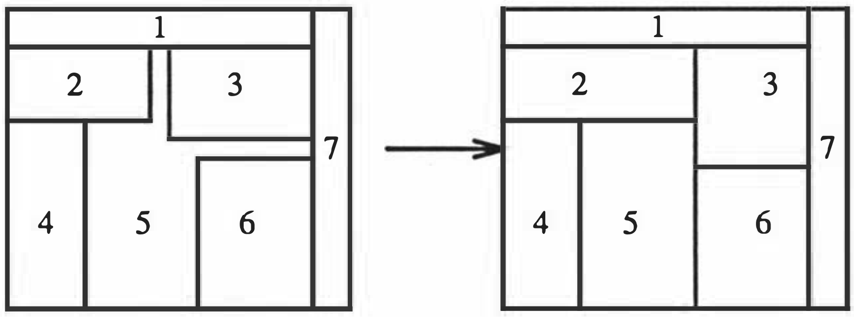


Figure 8.8: An example of the performance of multiple layout adjacency swaps

Figure 8.9 presents another example of the repeated application of the adjacency swap operation. Note that in this instance we have no choice about the order of performing the swaps. Firstly we swap $(2,5)$ for $(1,3)$, followed by swapping $(1,5)$ for $(3,4)$. The associated increase in benefit will be $(w_{13} - w_{25}) + (w_{34} - w_{15})$.

The examples of Figures 8.8 and 8.9 are simplistic examples. In general the performance of this operation is far more complicated, to the extent we would not recommend comprehensive employment of this procedure. The compounding errors produced in the facility areas are a constant concern, and it is recommended that on layouts generated using the ODA or a variant, the operation be performed manually only by a decision planner, as the implementation using a computer would be difficult and is unlikely to give satisfactory results. We reiterate that there is little sense in performing repeated applications of the operation which decrease the

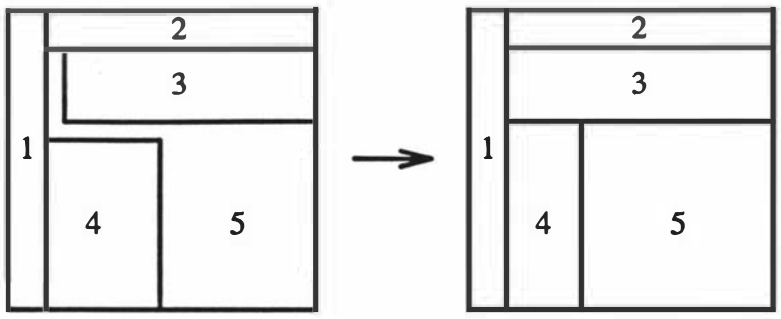


Figure 8.9: A second example of the performance of multiple layout adjacency swaps

adjacency benefit below that which could be obtained instead via a method with a guaranteed worst-case facility shape.

In spite of this however, we can make more progress with the operation of Figure 8.7 within dimensionalisable layouts. Methods such as the Tiling Algorithm and Deltahedron especially, generate layouts which are very amenable to the application of this operation. The expectation of success is dependent upon the ability to retain the dimensionalisability property within the layout. Furthermore, the continued retention of dimensionalisability nullifies the impact of errors creeping into the facility areas (as occurs in the general case), as we can redimension the layout after each swap.

In Figure 8.10 we present a family of five operations for application to dimensionalisable layouts, two based upon the operation of Figure 8.7, and the other three based upon alternative decisions which can exist in the application of placement operations. The motivation of these five operations came from a consideration of dimensionalisable layouts, and how they can be altered without violating dimensionalisability. For example, consider the Deltahedron method; operations (c)-(e) alone would provide the necessary operations for transforming one Deltahedron layout to any other on the same MPG *i.e.* without considering adjacency swaps. This is because operation (c) allows for the choosing of different placement directions, operation (d) allows for the choosing of a different top facility, and operation (e) allows the choosing of a different facility to *cover* a 2-joint. For general dimensionalisable layouts, considering all the operations (a)-(e) would allow a large amount of flexibility to the type of layout which could be created. Furthermore the facilities 1-4 in Figure 8.10 could be blocks of facilities, provided that at all times

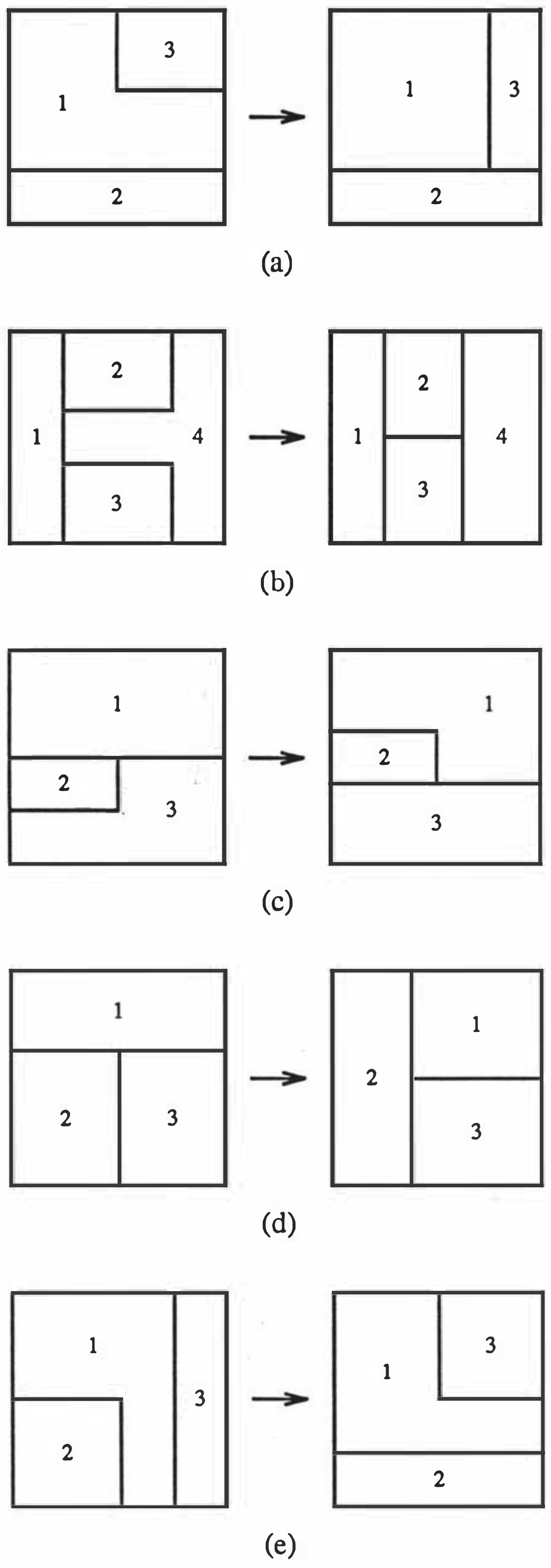


Figure 8.10: Family of Adjacency Swap Operations for Dimensionalisable Layouts

we retain the dimensionalisability property. Operations (a)-(e) are well suited to be embedded within a Tabu Search framework, using the multiplicative model used in Section 6.6, in order to ensure the best possible layout is created in terms of both regularity and adjacency benefit. Note that operations (a) and (b) are essentially the same, the difference being the L and T shapes respectively.

As an example, we provide a sequence of operations performed on a 10 facility layout in Figure 8.11. The first operation is of type (e), and does not actually swap any adjacencies. This is useful if we are attempting to maintain all the adjacencies, and improve regularity. Facility 3 now covers the lower left 2-joint, with facility 6 now contained within facility 4. Facility 4 remains T-shaped, with the adjacency with the exterior now on the bottom side, rather than the left, and facility 5 is redimensioned to avoid overlapping and to become squarer. The second operation performed is of type (a), where we perform the adjacency swap $(1,4) \rightarrow (3,6)$. The increase in benefit for this swap is $w_{36} - w_{14}$; facility 4 now becomes L-shaped. The third and fourth operations are both of type (d). The adjacency swap of the third operation is $(7,8) \rightarrow (2,9)$, while the swap for the fourth operation is $(1,7) \rightarrow (6,9)$. Note that these two operations could not be performed in the reverse order, as doing the swap $(1,7) \rightarrow (6,9)$ first would lose the dimensionalisability of the layout by creating a faultline between facilities 7 and 9. The last operation shown in Figure 8.11 is of type (c), where we again do not change the adjacency benefit of the MPG, but facility 5 now has facility 7 as its placement host, as opposed to 4 previously. There are other operations that we could continue to perform on the sixth layout such as a type (d) operation so that facility 3 becomes the top facility of the 3-joint $(2,3,4)$.

8.2.3 The Validity of the Maximal Planarity Assumption

The classical Graph Theoretic Facility Layout model considers that the adjacency graph will always be maximally planar. Indeed many methods require maximal planarity and, if the graph is not maximally planar, will arbitrarily add edges as necessary to attain maximal planarity. All of the methods we have studied thus far have required maximal planarity, except the Spanning Tree approach of Rinsma [94] which requires only a Maximal Spanning Tree. In the course of the work thus far we have also enforced maximal planarity on that approach if a 4-joint has been created in the layout. The GTFLP model assumes (for benefit matrices with non-negative

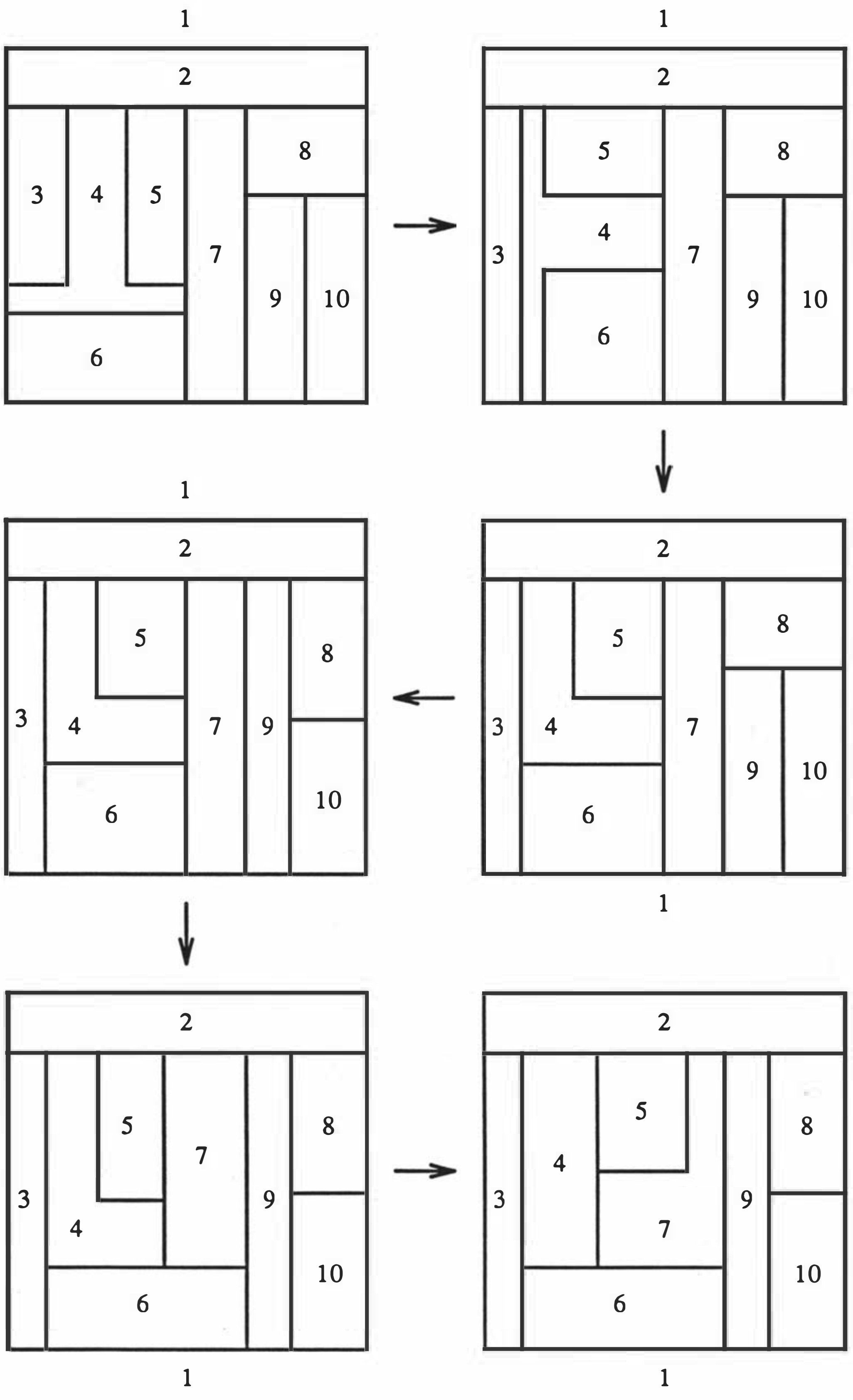


Figure 8.11: A sequence of operations applied to a dimensionalisable layout

entries) that the addition of edges to a less than maximal planar graph will not decrease the value of the adjacency graph. It therefore assumes that the preferred approach is to generate adjacency graphs with as many edges as possible, giving the largest possible adjacency benefit. Planarity is a requirement of the existence of a layout, hence we create maximally planar graphs. This assumption is certainly valid when determining the adjacency graph, however, as we have seen, the requirement of meeting all of the adjacencies in the layout can be very restrictive, and often overly optimistic. The predominant cause of irregularity characteristic of the layout methods is the maximal planarity assumption, although we have seen in Section 8.1 that the area requirements can influence this significantly as well.

Let us therefore examine possible ways in which we might relax maximal planarity. Firstly let us consider the repercussions of *removing*, as opposed to replacing, an edge in the MPG. Figure 8.12 shows the two cases that can occur, where either edge $(2,5)$ or edge $(3,4)$ has been removed. Either we will create a 4-joint in the layout, as the top layout of Figure 8.12 shows, or we will create a *hole*, represented by the shaded region of the bottom layout of Figure 8.12; both are valid representations of the non-maximal planar graph (NMPG).

Given a NMPG, in general a 4-joint would be uncommon, especially with facility areas which were largely variant. If it is infeasible to have holes in our layout, then in general we would allow that an adjacency will be created which is not in our NMPG. Doing this does not violate the conditions of the NMPG, as all adjacencies of the NMPG are still met, however other edges have been arbitrarily added during the construction of the layout, to ensure the layout does not form holes. This is the principle employed by the Spanning Tree method, where the vertical adjacencies between blocks in the layout are arbitrary. An advantage of this type of approach is to include only the greatest k say, edges in the adjacency graph, whilst maintaining planarity, and then during the layout construction, choose the remaining edges to ensure there are no holes.

If we are not concerned about having holes in our layout, we can utilise several ideas from Baybars [7]. Baybars used NMPGs to create layouts, and if it proved advantageous to create a hole in the layout, the hole was deemed to be a courtyard or circulation space. In fact, using this type of approach, the NMPG does not even need to be connected. A disconnected NMPG translates to a layout with a circulation space which contains at least one cycle. This can be useful when defining

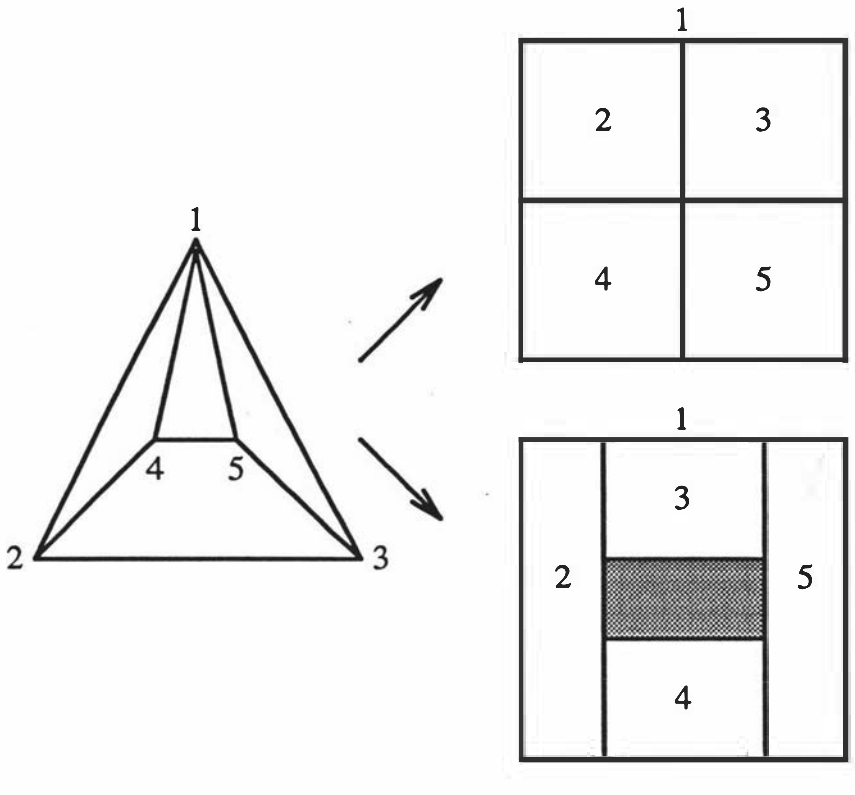


Figure 8.12: An example of a non maximal planar graph, and two possible layouts

and incorporating Material Handling Systems, and will be addressed further in Chapter 10.

A second assumption of the Graph Theoretic Facility Layout model, is that the optimally weighted maximal planar graph will create the best layout. We have seen in Chapter 6 that this assumption is unrealistic, yet all layouts we have considered (holes and 4-joints aside) have an MPG as their underlying adjacency structure. We have essentially answered this problem through this chapter, where we retain the maximal planarity of the layout as we make adjacency swaps within the layout. To reiterate, the motivation of making adjacency swaps was to increase the regularity of the layout, while the overall benefit was likely to decrease. Hence we were essentially nullifying the assumption that the optimally weighted MPG would create the best layout. Further, as we have discussed already, it is perhaps more important to include the best k edges (where k is say a third of the number of edges of an MPG). From here we can create practical layouts via the Spanning Tree Algorithm say, whilst still satisfying the required number of adjacencies to attain maximal planarity (if required). We will discuss further ways of addressing the problem of creating effective layouts in Chapter 9, where we present ways of ensuring the MPG will be amenable to creating practical layouts. These layouts however do not necessarily have an optimally weighted underlying adjacency graph; rather, they have large weight while guaranteeing practicality of the layout. This allows the adjacency graph to be created without recourse to the layout, yet still retains (approximate) information for the layout construction. We will address this and some more powerful techniques which seek a unification of the entire GTFLP process.

In conclusion, the maximal planarity assumption is an integral part of the GTFLP model. The removal of this assumption can be an intelligent decision, as the removing of adjacencies with small benefit allows the decision planner an increased amount of freedom. For instance, the Spanning Tree approach could be a first attempt at a layout, as it incorporates the best adjacencies whilst still allowing for an increase in the benefit of the resulting layout; the layout is then created with adjacencies not in the NMPG to ensure there are no holes in the layout.

Coda

This chapter has examined the GTFLP model, and the implications of relaxing integral assumptions within this model. We have identified the impact of both the area requirements and the specified adjacencies on the layout and its regularity. The importance of these ideas is best seen within an interactive framework with a decision maker, where the principles outlined in this chapter can be used to guide enhancements to the layout, using the problem data for the specific problem, the plethora of alternative layouts which can be generated for the same MPG (or NMPG), and area data.

In the next chapter we continue to examine the interface between the graph theoretic model and a decision maker, by considering two important ideas: decomposition and interaction. We will examine the usefulness of identifying substructures within the MPG, and the role of a decision maker on the entire layout process.

Chapter 9

Interaction and Decomposition

In this chapter we continue to examine the Graph Theoretic model in an attempt to provide more practical results. We saw in the previous chapter the concepts involved in the relaxation of problem constraints. In this chapter we endeavour to analyse two further mechanisms for generating practical layouts. The first method that we will explore is Decomposition, which attempts to divide the MPG into smaller, more manageable, subgraphs, each of which is an MPG in its own right, to facilitate the creation of the layout using a building block type of approach. The second approach is that which has been alluded to a number of times already: the interaction of the decision maker with the layout process.

9.1 Decomposition

In solving medium sized, or larger, layout problems, even as small as $n = 9$, there is so much happening within the process that it is virtually impossible to predict the implications of each decision, as shown by Lewis and Block [79]. Even predicting the performance of the perturbations we have already examined in Chapter 8 is difficult. To facilitate identification of key elements of the process, we attempt to divide our problem into smaller more manageable subproblems, which can be solved virtually independently of the other subproblems. Therefore the important concepts to grasp are: how might we perform the decomposition, how easy will subproblems be to manoeuvre, and how will the solution to the original problem be reconstructed?

9.1.1 Decomposition via Separating Triangles

Firstly let us consider how we might decompose an MPG. The most obvious and easily identifiable structure within an MPG is a separating triangle. We have seen that the presence of a separating triangle, or complex triangle from the Floorplanning literature, will enforce at least one of the three vertices of that separating triangle to generate an L-shaped facility in the corresponding layout. We have recognised that the existence of separating triangles can be a hindrance to the construction of the layout, however we will see that they can also prove useful.

Figure 9.1 shows a typical layout in which the separating triangle (a, b, c) exists in the corresponding MPG. We define a vertex to be on the *inside* of a separating triangle if all paths from that vertex to the exterior vertex must pass through one of the vertices of the separating triangle. Conversely a vertex on the *outside* of a separating triangle has all shortest paths to the exterior not passing through any vertex of the separating triangle. Analogous definitions for the facilities in the layout are shown by the shaded regions of Figure 9.1.

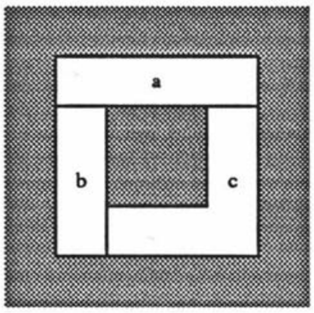


Figure 9.1: A typical layout consisting of the separating triangle (a, b, c)

We can see from Figure 9.1 that the inside region is disjoint from the outside region. We can therefore perform the layout process on the inside and outside regions independently, whilst maintaining an eye on consistency. In order to do this within the nesting of the subproblems, the vertices of each separating triangle are placed within two subproblems: the subproblem of the facilities outside the separating triangle, and the subproblem of the facilities inside the separating triangle. Note that we know that one of the facilities corresponding to a vertex of the separating triangle must be L-shaped, and this facility cannot be at the top of the 3-joint of the vertices on the separating triangle in the subproblem of the facilities outside the separating triangle. This is better seen with reference to Figure 9.2, where the separating triangle (3, 4, 5) appears in both subproblems. Note further that we could continue a decomposition using the separating triangles (2, 4, 5) and (6, 7, 8).

Two issues must be addressed: each subproblem requires an exterior facility (either existing already or added artificially) in order to adhere to the assumptions of the Graph Theoretic model within the subproblems; the linking or nesting of the subproblems must be consistent. Firstly let us consider the linking of the subproblems. Each subproblem must be nested within another subproblem, except for the *root* problem, which must contain the exterior facility (the root problem is the subproblem within which all other subproblems are nested - explicitly or otherwise). This leads to the formation of the *decomposition tree* which shows the hierarchical structure of the nested subproblems. If there exists more than one subproblem containing the exterior facility, the subproblem with the largest number of vertices containing the exterior facility is chosen as the root problem. The subproblems are considered in a top-down manner in order to ease the linking of nested subproblems, and to ensure the consistency of later subproblems. As one facility in a separating triangle acts as a placement host for subsequent subproblems within this separating triangle, we must ensure that in considering previous subproblems this can be accommodated. An example showing these ideas is given in Figure 9.3, where we have root problem A, within which are nested subproblems B, D, and E, and a subproblem C which is nested within subproblem B.

Every subproblem except the root problem has a pseudo exterior facility added to it, in order to satisfy the assumptions of the Graph Theoretic model within the subproblems. The pseudo exterior is a facility of degree three, which is adjacent to each of the vertices of the separating triangle. This addition of a pseudo exterior ensures the correct embedding of the facilities within the separating triangle *i.e.* the facilities of the separating triangle must be adjacent to the *outside* of the subproblem layout to ensure its subsequent placement within its *parent* subproblem. Subproblems which contain the exterior facility, and are not the root problem, still have the pseudo exterior added (with the actual exterior assigned zero area and placed as if it were not the exterior in the original problem). This is required to maintain the consistency within the linking of the subproblems (we will demonstrate this process with the aid of an example shortly). Note that each subproblem may be laid out using a different layout technique than is used for the

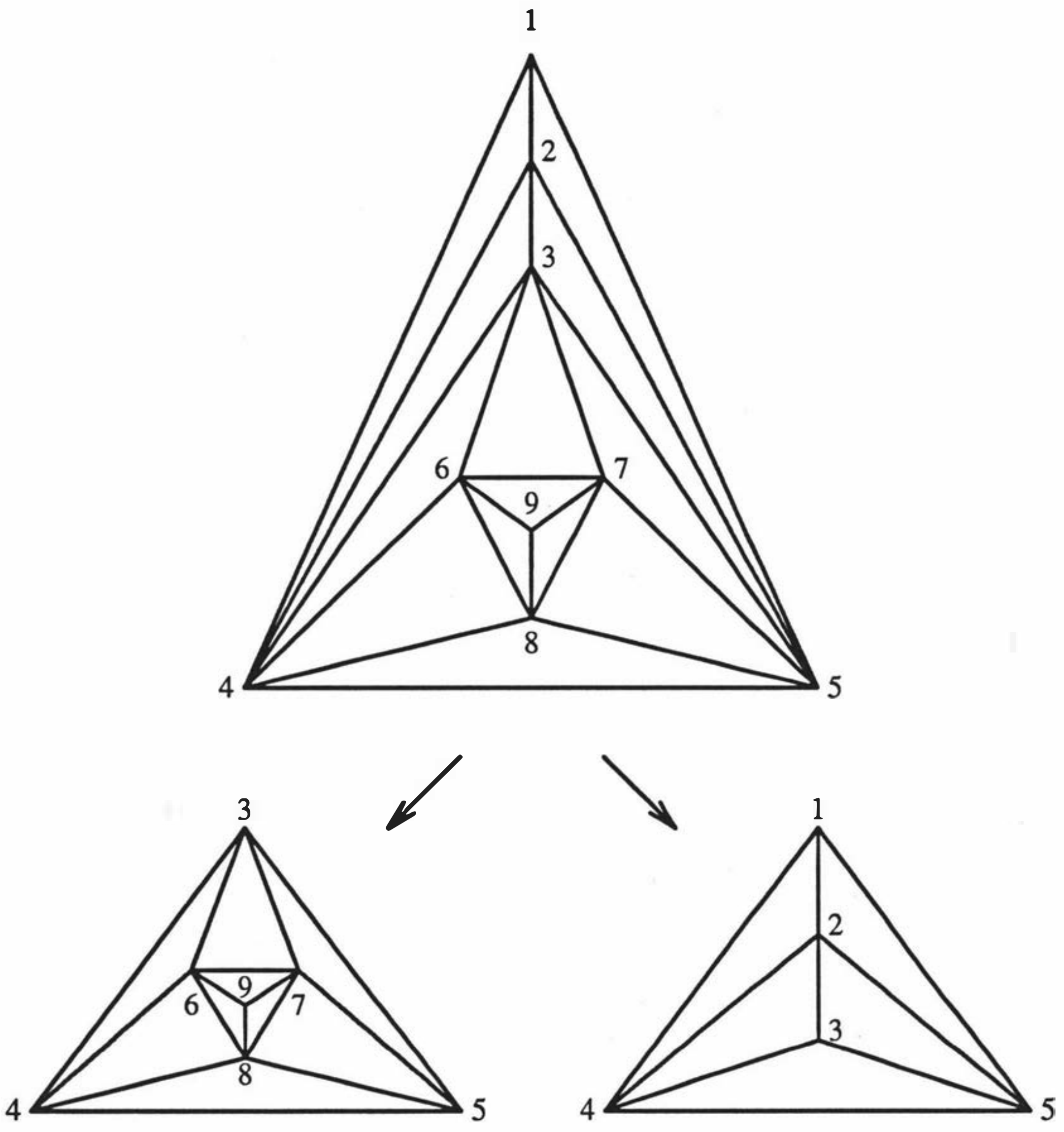


Figure 9.2: Decomposition of an MPG via the separating triangle (3, 4, 5)

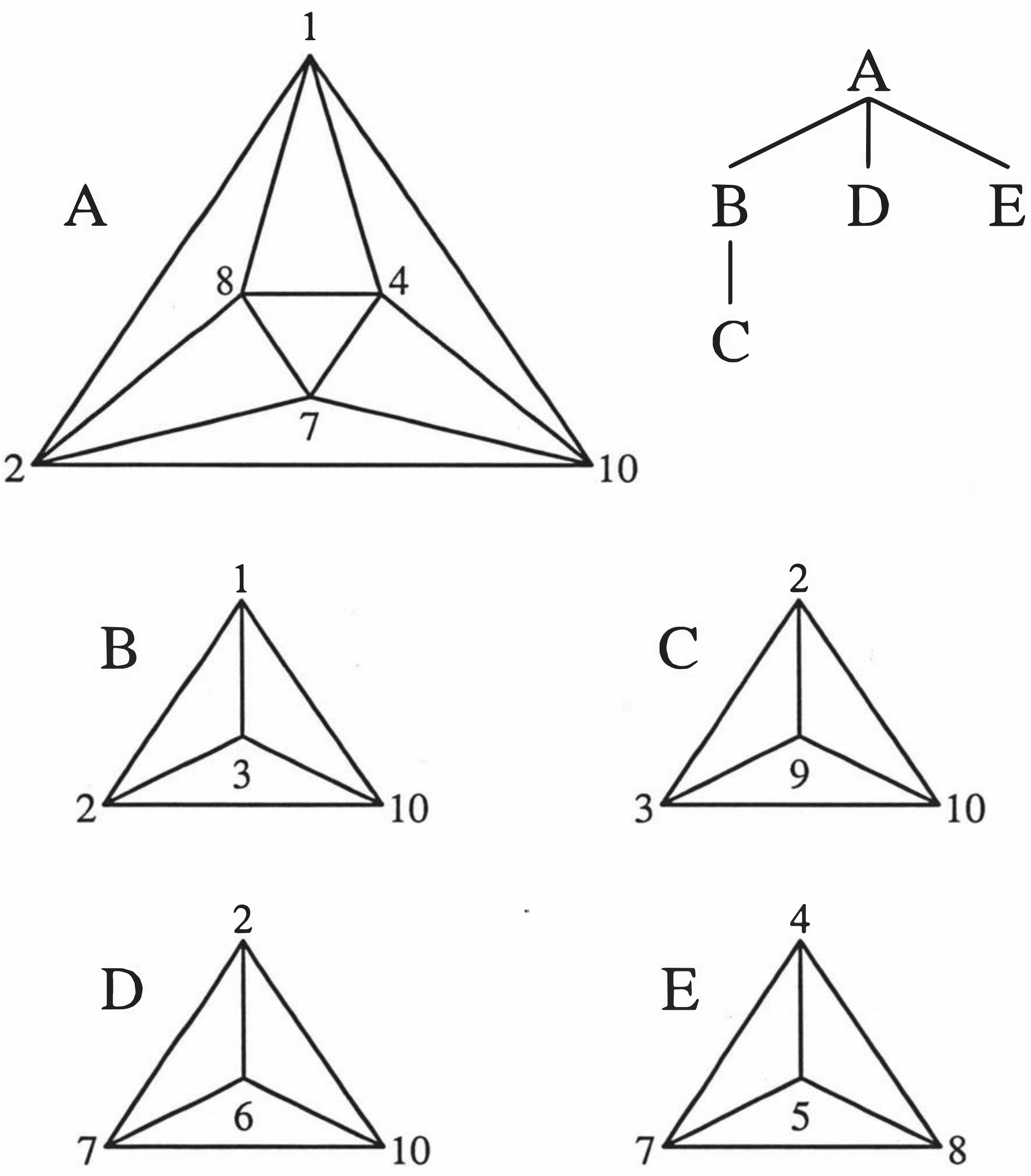


Figure 9.3: Example of the nesting of subproblems via a decomposition tree

other subproblems. In Chapter 7 we provided examples of problems amenable to one particular layout algorithm, and the structures exhibited by the corresponding MPGs which characterise these problems. By examining the structure of the MPG underlying each subproblem, we can choose the *best* method of constructing the layout for that subproblem, independent of the method of constructing layouts for the other subproblems. The VSA is the only method which we must be careful about using, as it does not guarantee to generate layouts with two of the facilities in D_1 rectangular, and the other L-shaped. All the other methods we have discussed can guarantee this (within the interactive framework we will discuss shortly). This leads to a *global* type of approach, which has the ability to consider all the mechanisms we have so far developed as subroutines within a more generic framework. We formally provide the process of identifying subproblems in Algorithm 9.1, where P_{ijk} is the subproblem based on separating triangle (i, j, k), P_{000} is the root problem, $Parent_{ijk}$ is the subproblem within which P_{ijk} is nested, and e^p represents a pseudo exterior. The process of creating the actual subproblem layouts, and recoupling them to create a layout for the original problem is provided in Algorithm 9.2.

Algorithm 9.1 *Decomposition Subproblem Identification*

```
Input: MPG G
Output: Set of subproblems $P_{ijk}$
$V_{P_{000}} \leftarrow \{e\}$
Determine all separating triangles $(i, j, k)$
for All separating triangles $(i, j, k)$ do
    $V_{P_{ijk}} \leftarrow \{i, j, k\}$
end
for $m = 1$ to $n$ do
    if shortest path from $m$ to $e$ passes through separating triangle $(i, j, k)$
    before any other separating triangle then
        $V_{P_{ijk}} \leftarrow \{m\}$
    end
    if shortest path from $m$ to $e$ passes through no separating triangle
    then
        $V_{P_{000}} \leftarrow \{m\}$
```

```
          end
      end
      for each pair $P_{ijk}$ and $P_{xyz}$ $(xyz \neq ijk)$ do
          if $(\{i,j,k\} \in V_{P_{xyz}})$ then
              $Parent_{ijk} \leftarrow P_{xyz}$
          end
      end
      for each $P_{mnp}$ do
          if $(P_{mnp} \neq P_{000})$ then
              $V_{P_{mnp}} \leftarrow V_{P_{mnp}} \cup \{e^p\}$
              $E_{P_{mnp}} \leftarrow \{(e^p,m),(e^p,n),(e^p,p)\}$
              $T_{P_{mnp}} \leftarrow \{(e^p,m,n),(e^p,m,p),(e^p,n,p)\}$
          end
          for $i = 1$ to $n-1$ do
              for $j = i+1$ to $n$ do
                  if $(\{i,j\} \in V_{P_{mnp}}, \{(i,j)\} \in E_G)$ then
                      $E_{P_{mnp}} \leftarrow E_{P_{mnp}} \cup \{(i,j)\}$
                  end
              end
          end
          for $i = 1$ to $n-2$ do
              for $j = i+1$ to $n-1$ do
                  for $k = j+1$ to $n$ do
                      if $(\{i,j,k\} \in V_{P_{mnp}}, \{(i,j,k)\} \in T_G)$ then
                          $T_{P_{mnp}} \leftarrow T_{P_{mnp}} \cup \{(i,j,k)\}$
                      end
                  end
              end
          end
      end


end
```

Algorithm 9.2 *Layout Construction via Decomposition/Recomposition*

```
Input: Set of subproblems $P_{ijk}$
Output: Layout dual to original MPG from which $P_{ijk}$s were derived
Generate layout for subproblem $P_{000}$ using best technique
for each $P_{xyz}$ such that $Parent_{xyz} = P_{000}$ do
    $t_{xyz} \leftarrow$ the top facility of the 3-joint $(x, y, z)$ in $Parent_{xyz}$
end
while (all $P_{ijk}$ layouts not constructed) do
    if $Parent_{ijk}$ layout constructed and $P_{ijk}$ layout not constructed then
        Generate layout for subproblem $P_{ijk}$ using best technique, with $t_{ijk}$
        as the top facility of the layout
        $c_{ijk} \leftarrow$ the $L$-shaped facility of $D_1$
        for each $P_{xyz}$ such that $Parent_{xyz} = P_{ijk}$ do
            $t_{xyz} \leftarrow$ the top facility of the 3-joint $(x, y, z)$ in $Parent_{xyz}$
        end
    end
end
for each $P_{ijk}$ do
    Calculate the total area of $P_{ijk}$, denoted $A_{ijk}$
    $L_{ijk} \leftarrow 0$
end
if $(\forall P_{xyz}, Parent_{xyz} \neq P_{ijk})$ then
    $L_{ijk} \leftarrow 1$
end
while (at least one $L_{ijk} = 0$) do
    if $\forall Parent_{xyz} = P_{ijk}, L_{xyz} = 1$ and $L_{ijk} = 0$ then
        if $Parent_{xyz} = P_{ijk}$ then
            $A_{ijk} \leftarrow A_{ijk} + A_{xyz}$
            $a_{c_{xyz}} \leftarrow a_{c_{xyz}} + A_{xyz}$   *in $P_{ijk}$ only*
        end
        $L_{ijk} \leftarrow 1$
    end
end
Reconstruct layouts with modified area specifications retaining the same $c_{ijk}$
```

```
        and $t_{ijk}$ for each $P_{ijk}$ as before
        Merge layouts (using rotations and reflections as required)

end
```

Let us now consider an example of the decomposition approach, by considering the twelve facility problem of Figure 9.4, with area conditions given in Table 9.1, with facility 1 as the exterior. There exist two separating triangles: $(1,3,8)$ and $(2,4,5)$. This creates three subproblems as shown in Figure 9.5, where facility 0 is the pseudo exterior facility that has been added, and in subproblem 2, facility 1 has been given area zero.

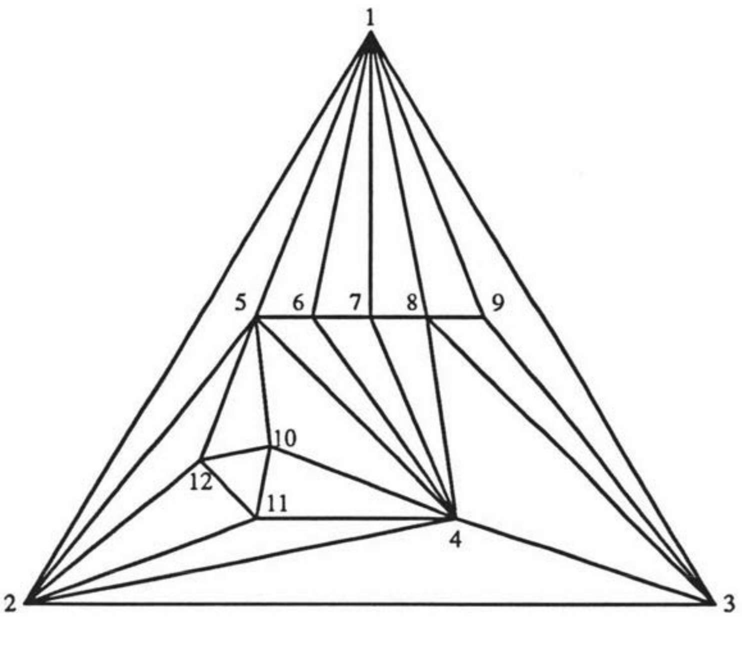


Figure 9.4: Decomposition Illustrative Example

Facility	1	2	3	4	5	6	7	8	9	10	11	12
Area	-	10	20	15	10	20	20	25	10	20	25	10

Table 9.1: Areas Specifications for the Decomposition Illustrative Example

Examining Figure 9.4 alone would make it difficult to discern how this MPG might be best dualised. However, Figure 9.5 makes this decision much clearer. Each of the three subproblems will have their layouts constructed from the templates

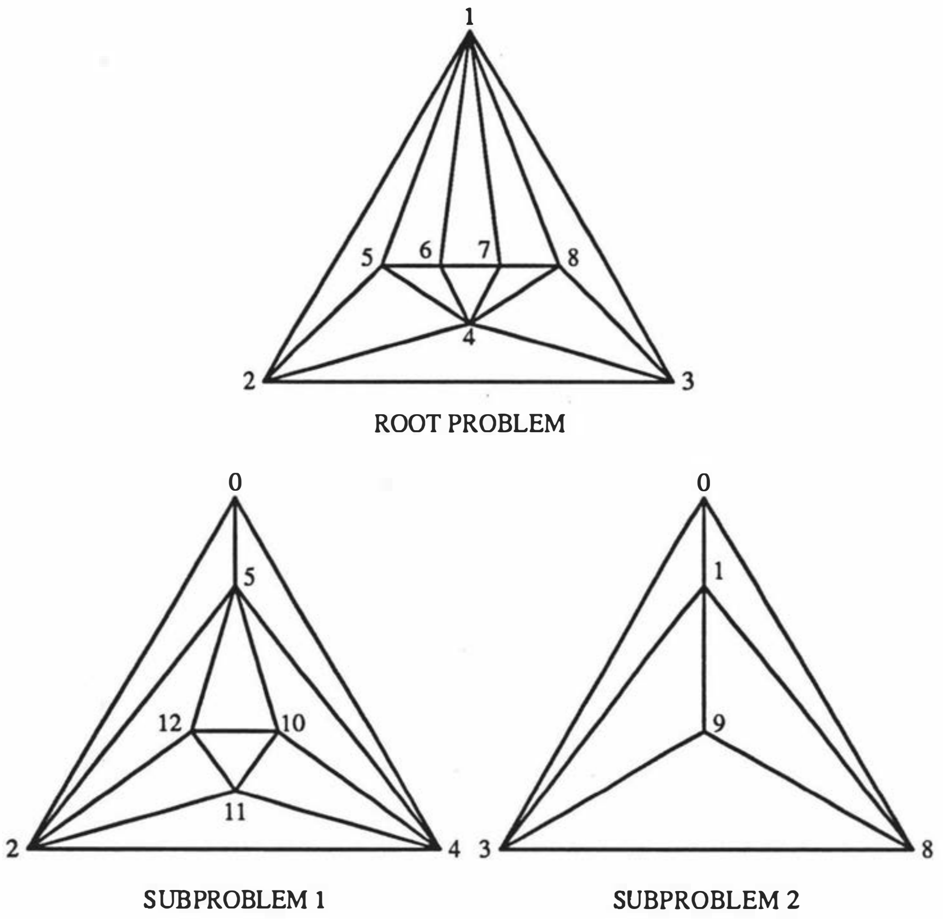


Figure 9.5: Decomposition of Figure 9.4

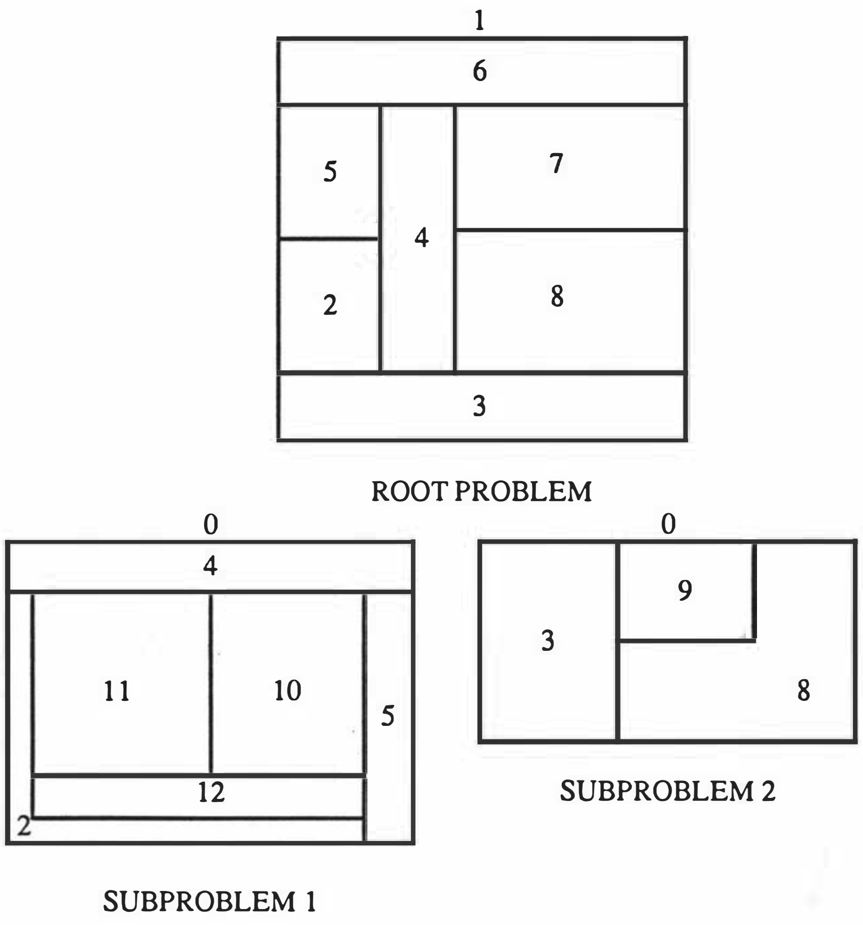


Figure 9.6: Layouts of the Subproblems of Figure 9.5

discussed in Section 4.7.5, and found in Appendix A, since each of the problems is of size eight or less. The resulting layouts of the three subproblems are shown in Figure 9.6. Firstly note that facility 1 is missing from the layout for subproblem 2. This is due to assigning facility 1 zero area, leading to facility 1 not being (explicitly) drawn. This is exactly what we require to maintain consistency. The top of the layout for subproblem 2 is where facility 1 lies, and it effectively ensures that facility 9 is adjacent to the exterior in the final layout. Note that the layout for subproblem 1 could have had facility 5 as the placement host for the block of facilities 10, 11 and 12; however, facility 3 could not be the placement host for facility 9 in subproblem 2, as this would lead to a faultline in the final layout.

We see here the basic structure of a *wave* type of approach, as shown formally by Algorithm 9.2, whereby we start at the top of the decomposition tree and work our

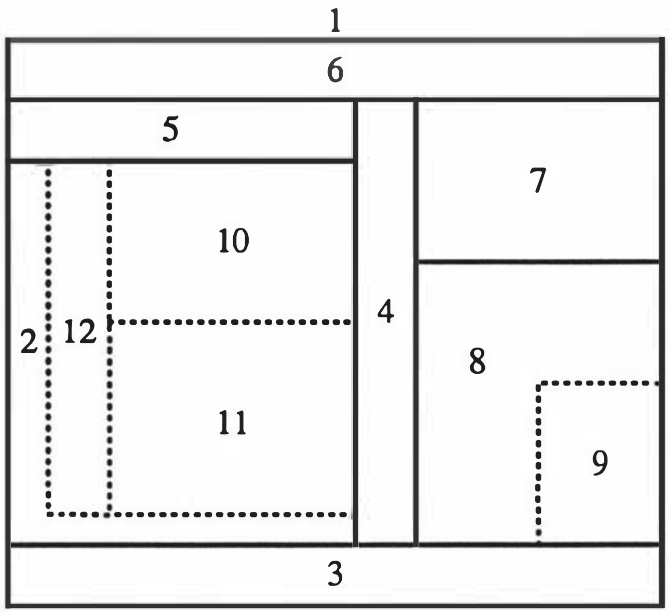


Figure 9.7: Actual Layout of Root Problem with Facility Areas changed to allow for nested Subproblems

way down, ensuring consistency of the subproblems, and ensuring that placement hosts are assigned so that no faultlines are created in the final layout; then having completed this, we must adjust the areas of all the placement hosts further up the tree so that they may accommodate the subproblems to be placed within them. Following this we can re-lay the subproblems which, upon completion, can be easily superimposed upon their parent subproblem. We see the modified root problem layout in Figure 9.7, as the subproblem layouts remain unchanged, where facility 2 now has area 65 in the root problem, so as to accommodate subproblem 1, and facility 8 now has area 35 in the root problem so as to accommodate subproblem 2. Subproblems 1 and 2 are superimposed on Figure 9.7, exhibiting the process of recoupling the subproblems to generate the layout corresponding to the original MPG. From Figure 9.6 we see that the layouts for subproblems 1 and 2 have been rotated anticlockwise 90°, and then reflected in the x-axis for placement in Figure 9.7. Note that we have only two L-shaped facilities; no single approach performs this well on this problem of those we have discussed.

We call a problem which can be decomposed into subproblems of size α or less *α-decomposable.* The decomposition process isolates vertices of degree three, so Deltahedron MPGs are 4-decomposable; the illustrative example in Figure 9.4 was

n	Benefit	Average Number of		Average	
		Sub Problems	Vertices of Degree 3	Largest Subproblem	Density (ρ)
10	4258.02	5.09	3.80	6.60	68.2%
15	7463.27	6.93	4.93	9.16	53.9%
20	11550.47	7.27	5.42	13.33	39.2%
25	15267.22	9.58	7.40	15.53	40.9%
30	18791.44	11.76	9.18	16.58	41.4%
40	26811.64	13.42	10.93	25.00	34.5%
50	34570.71	16.44	13.93	29.40	33.6%

Table 9.2: Decomposition as applied to the test problem set

8-decomposable.

Table 9.2 shows the effect of the decomposition theory on the set of test problems generated in Section 6.1. The effect of the decomposition process is a reduction to 60% the size of n for the largest subproblem, meaning that we can effectively treat 25 facility problems as 15 facility problems on average, hopefully leading to increased regularity, and usability.

9.1.2 Forcing Decomposition

The effectiveness of the decomposition process relies heavily on the *density* of separating triangles within an MPG. We define the density, ρ, of the separating triangles in Equation 9.1, where p is the number of separating triangles in the problem P.

$$\rho = 100\left(\frac{p}{n-4}\right)\% \tag{9.1}$$

There can exist at most $n-4$ separating triangles in an MPG (an MPG with exactly $n-4$ separating triangles is Deltahedron generateable as we initialise with the Tetrahedron, and each subsequent insertion creates a separating triangle). Furthermore the process is most effective when all the subproblems are of size 8 or less, as we can guarantee the existence of dimensionalisable layouts regardless of the size of n, by using the templates of Appendix A. Decomposition fails if there exist no separating triangles in the MPG, however we will discuss shortly ways in which we might tackle this problem. The density of the separating triangles within the set of test problems is exhibited in Table 9.2.

Two further experiments were undertaken, in an effort to generate MPGs which were 10-decomposable and 8-decomposable. The purpose of these was to examine the effect on the adjacency benefit of forcing decomposition. This was achieved by forcing TESSA to create a separating triangle when there were α vertices on the *outside* of all separating triangles. A separating triangle was created by allowing TESSA to only make case (b) operations (those which create a face using three existing vertices on the boundary), until a separating triangle was created. By only performing case (b) TESSA operations, existence of a separating triangle is guaranteed; if no separating triangle is created previously, a boundary of length three (creating a separating triangle) will be eventually attained. We will expand on this process shortly.

The results were disappointing as no improvement mechanism was able to be implemented. Having created the MPG, it was impossible to find a feasible move space (*i.e.* Γ-operations performed on an α-decomposable MPG could not guarantee the resulting MPG was also α-decomposable), hence forcing travel through infeasible solutions. This resulted in there being no guarantee of finding even one feasible solution once we moved into infeasible space, and so it turned out that in very few cases could the initial construction be improved upon. The results are disappointing, as Deltahedron, which after all generates 4-decomposable MPGs, could in all cases create MPGs of higher weight, while guaranteeing layouts with guaranteed worst-case facility shape. The results of this, along with the Deltahedron MPG benefits for comparison are given in Table 9.3.

n	Benefit		
	Deltahedron	10-Decomposition	8-Decomposition
10	4186.42	4056.33	3922.98
15	7441.69	6779.18	6651.18
20	11506.27	10536.29	10180.73
25	15373.38	13817.13	13398.36
30	19157.67	16694.93	16469.80

Table 9.3: Generation of 8- and 10-Decomposable MPGs on test problems

All is not lost, however, as the ability to force TESSA to create separating triangles can be very useful for the *enforced* decomposition of large undecomposable subproblems. Suppose we have an MPG which has large weight. We desire to have an MPG which is α-decomposable, but have a subproblem of size greater than α (or, indeed, the whole MPG cannot be decomposed at all). At this point we can initiate a TESSA routine which will generate the α-decomposition for us, and importantly does not affect the other subproblems. The process is initialised with the separating triangle as the first face chosen by TESSA, or equivalently the Tetrahedron, consisting of the three vertices of the separating triangle on the TESSA boundary, and the pseudo exterior already an interior vertex. Thereafter we can force TESSA to create separating triangles at any desired or specified point, as we described in performing the 8- and 10-decomposable experiments, until all vertices have been placed in the subproblem's MPG. The subproblem MPG must now be α-decomposable. Perhaps it is easier to then see the impact of this approach by changing the embedding so that the face consisting of the vertices in the separating triangle has all other vertices inside it, and then adding the pseudo exterior. An example of this process is given by Figure 9.8, where we see the separating triangle $(2,3,4)$ providing a subproblem of size 9 in Figure 9.8(a). In order to create an 8-decomposition, we initialise with the triangle $(2,3,4)$, followed by case (a) TESSA operations until we have eight vertices (2,3,4,5,6,8,9 and 10) in the current subproblem. At this point we allow only case (b) operations until a separating triangle (in this case $(8,9,10)$) is created. Vertex 7 is then added to complete the example in Figure 9.8(b). The reembedding is shown in Figure 9.8(c). The decrease in benefit in the MPG in this example would be $w_{5,7} + w_{6,7} - w_{8,10} - w_{9,10}$. This problem is now 8-decomposable with separating triangles $(2,3,4)$ and $(8,9,10)$.

If the root problem cannot be α-decomposed, initialisation for TESSA is as normal (*i.e.* the face of largest weight is chosen), since this problem is not nested within a separating triangle. There is an important caveat which must be considered when forcing decomposition using TESSA. If the non-α-decomposable subproblem has other subproblems nested within it we must ensure that the separating triangles which have subproblems nested within them are not deleted. This can be accomplished in two ways, both of which have their pitfalls. To illustrate these principles we suppose Figure 9.8(c) was required to be 6-decomposable, hence we are required to modify the subproblem consisting of the vertices $(2,3,4,5,6,8,9,10)$ which has the subproblem $(7,8,9,10)$ nested within it. The first approach is to include any subproblems which are nested within the non-α-decomposable subproblem when forcing the α-decomposition. For example we would impose the 6-decomposition

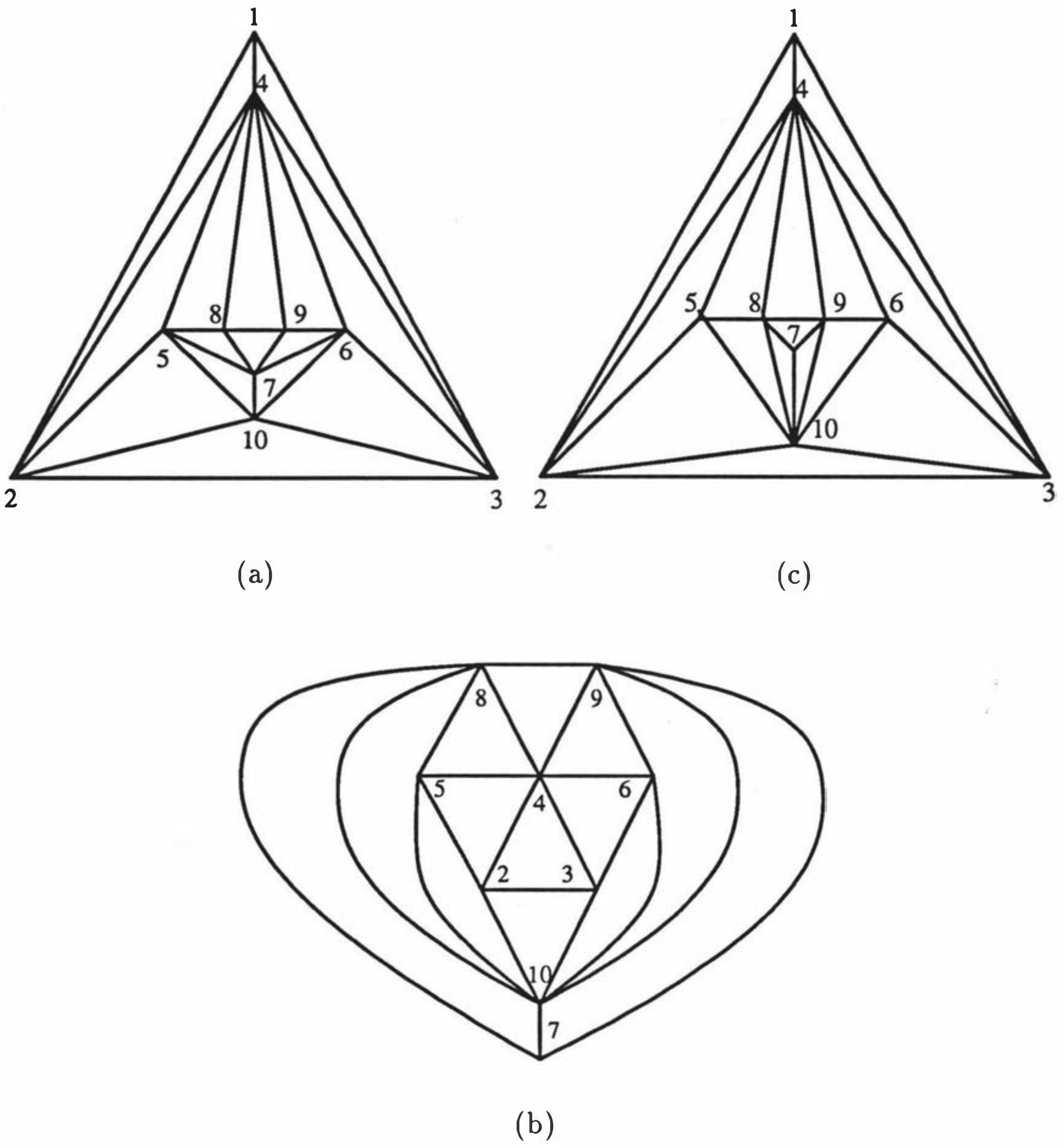


Figure 9.8: Forcing 8-decomposition upon a 9-decomposable MPG:(a) 9-decomposable MPG; (b) TESSA MPG to form 8-decomposability; (c) reembedding of 8-decomposable MPG

on the MPG consisting of $(2,3,4,5,6,7,8,9,10)$. This is undesirable if the non-α-decomposable subproblem is the root problem, or some other subproblem including a large proportion of the problem's vertices. The second approach is to force the inclusion of all faces which are separating triangles with nested subproblems. For example force the 6-decomposition on the MPG $(2,3,4,5,6,8,9,10)$, while ensuring the face $(8,9,10)$ is created, so that $(7,8,9,10)$ can be subsequently nested. The danger here is that if all the faces of the non-α-decomposable subproblem are also separating triangles in the original MPG, then we cannot change the subproblem. Therefore we must have some alternative strategy for dealing with these pathological problems. One possibility might be to force an α-decomposition on the non-α-decomposable problem, and afterwards place any subproblems nested within this one in a face of the new MPG for that subproblem (akin to the β-operation). For example we would create the 6-decomposable MPG on the vertices $(2,3,4,5,6,8,9,10)$, and add facility 7 in the best place subsequently. This provides a minimal disruption to the subproblems, by changing only the vertices of the separating triangle for each nested subproblem in the best way possible, *i.e.* at most three adjacency swaps for each nested subproblem.

A concern may be raised as to the detrimental effect these changes to the MPG may have on the overall adjacency benefit. This may become a problem, however utilising the adjacency perturbation ideas from Section 8.2 we can attempt to reconstruct the original underlying MPG. An example of this process is provided in Figure 9.9, where we force the 8-decomposition of a non-decomposable 10 vertex MPG with decrease in weight of $w_{5,8} + w_{6,9} - w_{4,7} - w_{4,10}$. Following construction of a layout dual to this new MPG (using Decomposition), application of adjacency perturbation will replace the adjacencies between facilities 5 and 8, and 6 and 9. Note that there is no other way of tackling this problem using the techniques we have explored.

The greater flexibility of this approach, allows us to take our analysis of problems further. Chapter 11 provides a full example, where we can see the Decomposition process in action.

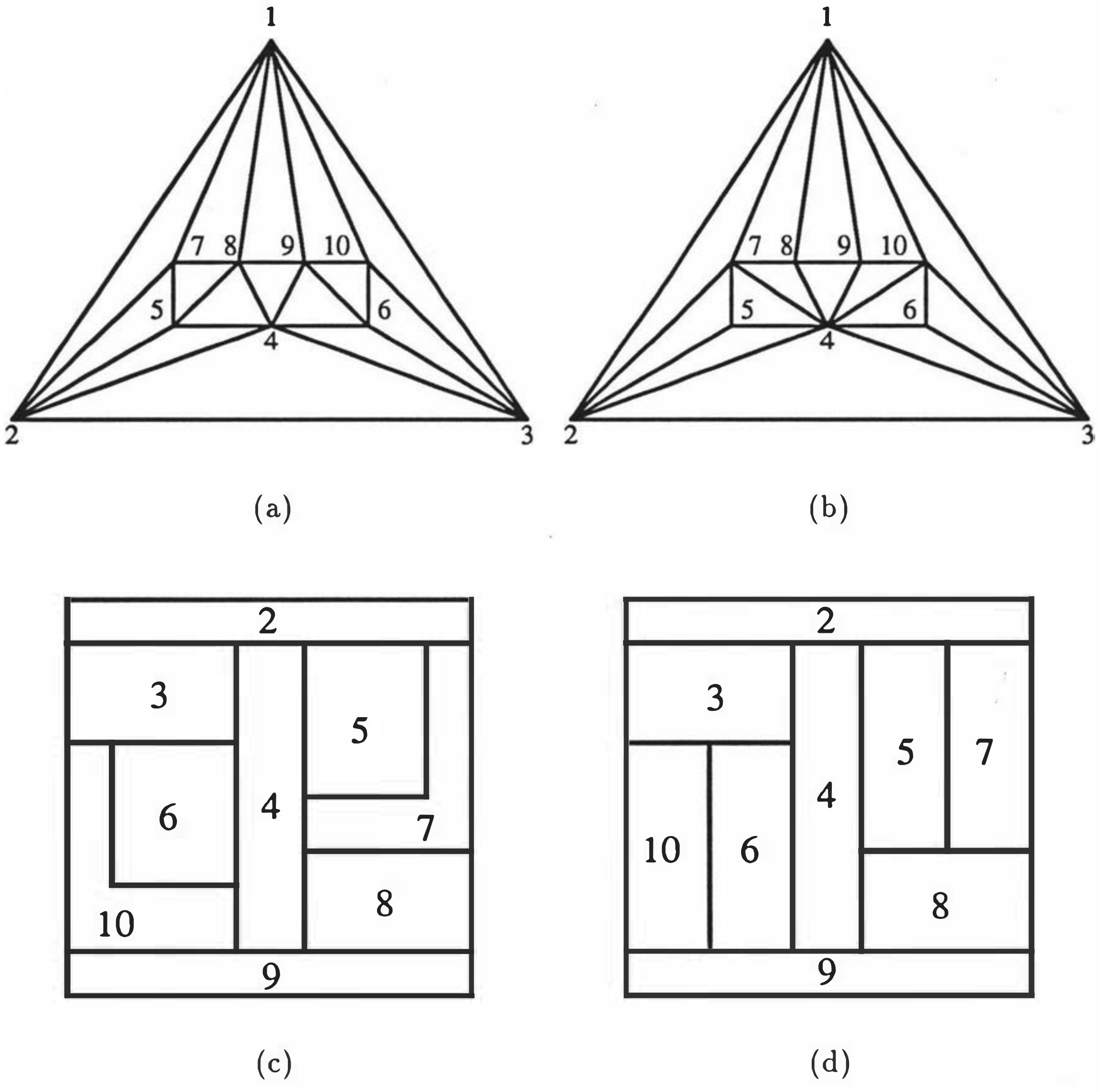


Figure 9.9: Forcing decomposition, followed by application of adjacency perturbation to obtain layout dual to original MPG: (a) non-decomposable MPG; (b) forced 8-decomposition on (a); (c) layout dual to (b); (d) adjacency perturbation applied to (c) to obtain layout dual to (a)

9.2 Interaction

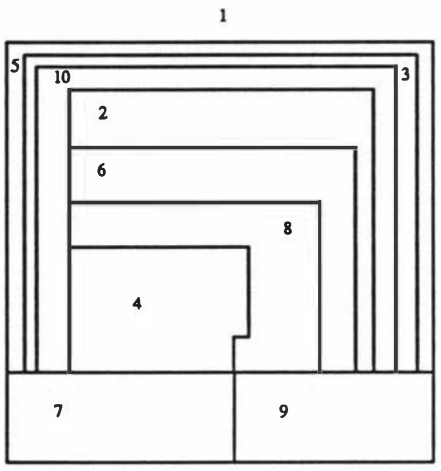
Another powerful concept in the layout process is to include the decision maker at a number of key steps in the creation of the layout, allowing a wide range of flexibility for the decision maker, and bypassing the need to quantify largely subjective factors such as aestheticity and practicality, which is necessary in order to automate layout construction techniques.

Firstly we must identify the important decisions facing the decision maker. Within specified MPGs these are the specification of the placement host for an entering facility, and (for a particular 3-joint) the designation of the top facility. However we also require manual intervention in implementing perturbations, and in selection of *promising* paths of investigation. The specification of the placement host becomes important in most of the methods, whereby we have two choices in general for the placement host, either the left or right facilities of a 3-joint. Every choice that must be made effectively doubles the number of layouts which could be generated from that MPG. This is especially important for the Contraction Algorithm, where the MPG is contracted originally with no foresight as to the implications on the layout. By implementing an interactive interface the Contraction Algorithm can be genuinely enhanced to allow the backtracking of decisions, so that the best choices, in the eyes of the decision maker, for the contractions and their order, is found. The ability to designate the top facility for 3-joints is important also. This has implications for the starting points of most methods, which rely on the Deltahedron initialisation, which uses the tetrahedron initially. The choice of the top facility out of the possible three facilities in the tetrahedron has repercussions throughout the remainder of the layout. This is also helpful when considering the decomposition approach, where in order to maintain consistency of the nested subproblems, we need to be able to determine the top facility from the facilities in the parent separating triangle.

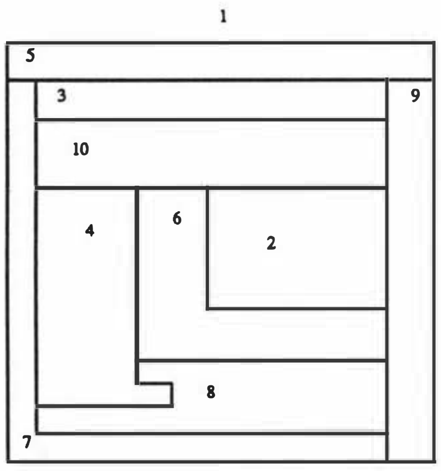
Figure 9.10 shows the steps taken to interactively change the contraction ordering of the MPG defined by the layouts, in order to obtain a more efficient layout. Initially, we see that every contracted facility is essentially within the top facility 5; therefore the only change which we make initially is to change the contraction of facility 10 into facility 7 instead of 5. The impact of this single change is significant, and is represented by the first two layouts of Figure 9.10. The adjacency between facilities 8 and 10 is small in the second layout, and facility 8 is rather irregular.

LAYOUT

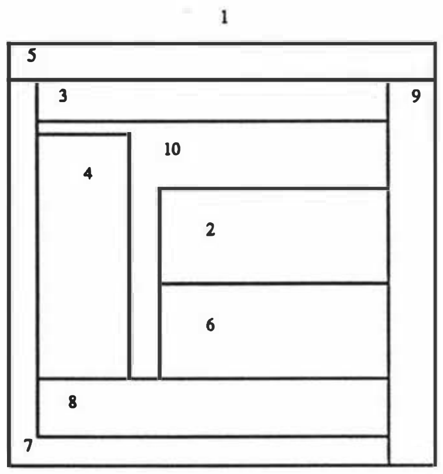
CONTRACTION ORDER



2	→	6
3	→	10
4	→	8
6	→	8
8	→	10
10	→	5



2	→	6
3	→	10
4	→	8
6	→	8
8	→	10
10	→	7



2	→	6
3	→	10
4	→	10
6	→	10
8	→	10
10	→	7

Figure 9.10: Example of Interaction working from the initial computer generated layout at the top through to a more acceptable layout at the bottom

We attempt to remedy this by contracting facilities 4 and 6 into facility 10 instead of 8. The change to the layout is seen immediately, in the bottom layout of Figure 9.10. Note that we have gone from an initial layout of four U-shaped and three L-shaped facilities, through one 3-concave-corner and three L-shaped facilities, to the final layout which has just one T-shaped facility and one L-shaped facility. Further note that no standard implementation of the Contraction Algorithm could be used to generate the final layout, as facility 10 has four other facilities contracted into it, whereas the standard implementations of the Contraction Algorithm state that each facility can be contracted into at most twice. The choices made in the example were made as intelligently as possible. Unfortunately the impact of a choice cannot usually be predicted when using this interactive approach, and it is therefore helpful to iterate through a number of arbitrary choices to find a good starting layout from which to refine. We can outline some general principles for this process however:

> Facilities which are a placement host to a large number of facilities (explicitly or implicitly) should not be the top facility of any 3-joint, as exhibited by the first change made in Figure 9.10.
>
> Selection of placement hosts should be made with respect to all facilities which will be placed (explicitly or implicitly) within that host, as shown by the selection of facility 6's placement host in Figure 9.10 - selection of facility 10 as the placement host allowed facility 2 to cover the 2-joint formed at the creation of facility 6.
>
> Facilities should be placed to minimise the number of faultlines created (to retain dimensionality as much as possible), as exhibited in Figure 9.10 by choosing the same placement host for facilities 4 and 6 to ensure the retaining of the adjacency between facilities 8 and 10.

Interaction allows algorithms such as the Contraction Algorithm to be relaxed in such a way that almost any vertex can be contracted into any other adjacent vertex. The only restrictions upon this are that there must exist three vertices in D_1, and that the exterior cannot be contracted or contracted into. These conditions simply ensure that the exterior is never a placement host, and that the Tetrahedron layout structure becomes the initial building block for the layout. In general it is wise not to contract facilities into a facility in a distance class further from the exterior, as this can cause substantial irregularity when the layout is constructed. This not only allows greater flexibility in the Contraction Algorithm, but also simplifies the theory behind this approach, leading to it being a more intuitively motivated method.

The VSA can also be improved by an interactive feature. For example, the deletion of shortcut edges (i,j), makes a choice for which of i or j will be the parent facility of the newly-created pseudo facility. This choice could be made in an interactive way, enabling the decision planner to the examine the outcome of each choice. Furthermore the choice of the first facility of each distance class which is placed in the layout again has an impact on the outcome of the final layout, this choice could be made by a planner, as could the determination of the region that facilities adjacent only to a corner facility can be placed, as they can be assigned to either region that this corner facility borders. Guidelines for the VSA are much more difficult, as each choice impacts on every subsequent facility placed, as opposed to the Contraction Algorithm where the alternatives in general have a much more local impact.

The flexibility and elegance of the decomposition approach, lends itself naturally to interaction with decision makers, and planners, due to the large degree of versatility inherent in the generic approach. Decision makers need not be restricted by a particular method, but rather are able to call upon a wealth of concepts which act within a structured framework. Importantly this removes any arbitrariness from the ideas and concepts developed within the GTFLP model. In short, interaction is useful at any point in facility layout methods where a choice must be made. The ability to successfully incorporate this interface is likely to lead to a layout framework which is applicable to virtually any layout design process, as the layout planner has control over all factors, and the knowledge of unquantifiable issues, coupled with knowledge of each facility's function.

Coda

In this chapter we have continued to build a framework which enables complex problems to be handled easily. Decomposition and interaction, along with the perturbations discussed previously, provide the GTFLP solution process with the flexibility it requires to be successful within a practical setting. The GTFLP model is now set within a versatile framework ensuring the greatest use is made of the concepts, rather than the rigid use of the algorithms. The ability to incorporate intangible factors such as aestheticity have been addressed through the use of interaction, ensuring that the Graph Theoretic model is as realistic as possible.

In the next chapter we examine an important evaluator of layouts: the Material Handling System (MHS). The GTFLP model does not in general consider relationships between non-adjacent facilities; the determination of the MHS allows the examination of relationships between such facilities, by considering flows between these facilities, and the route by which this interaction might be undertaken. Issues such as minimising the total flow on the MHS, and minimising the farthest distance of any flow on the MHS, arise. In Chapter 11, we attempt to put the methods and ideas we have described into practise, by considering two realistic examples, from conception.

Chapter 10

The role of Material Handling Systems in Evaluating Layouts

In the course of the work so far, we have considered only the benefit derived from adjacent facilities. In this chapter we are interested in relationships between non-adjacent facilities, and in particular, the cost of transporting materials between all pairs of facilities. In general there will exist desirable adjacencies which are unable to be included in the MPG, due to the constraint on the maximum number of edges permitted in the adjacency graph to maintain planarity. Therefore, while in general we accept that we have obtained the *best* adjacency structure available, we are unable to predict the ramifications on a final layout of any unsatisfied desirable adjacencies. This leads to the concept of a Material Handling System (MHS), which for our purposes is simply a mechanism which links facilities, for example a corridor or conveyor system. In this chapter we explore the role of the MHS in generating complete transportation cost data. Further we will see that the transportation cost of an MHS, provides another evaluator of the efficiency of a layout.

10.1 Previous Work on Near Adjacency

Giffin and Foulds [31, 40] examined the issue of omitting consideration of non-adjacent facilities when constructing a Deltahedron adjacency graph. The first paper, [31], examined the creation of the adjacency graph, under the objective of minimising the transportation cost, rather than maximising the adjacency benefit.

By changing the objective function, and assuming that all travel is rectilinear between facility centroids (by designating each facility to be square in shape) Foulds and Giffin, were able to insert vertices into the adjacency graph under the standard Deltahedron operation, such that the overall transportation cost between the vertices in the current MPG, and the candidate vertex was minimised. This heuristic, together with more modern improvement search routines such as Tabu Search, provides adjacency graphs, with overall transportation cost minimised, under the assumptions of Foulds and Giffin. However note that the assumption of square shaped facilities is overly optimistic, and therefore the final layout, may not accurately reflect the transportation cost derived by the MPG.

In the subsequent paper [40], Giffin and Foulds developed the concept of *near-adjacency*. Giffin and Foulds assumed M adjacency matrices, each depicting adjacency relationships of facilities k, $k = 1, \ldots, M$, walls apart. A tailored version of Dantzig's Shortest Path Algorithm [23] was incorporated, to calculate the shortest distance between two vertices. The objective was extended to choose the vertex to enter which induced the best overall adjacency benefit relative to all the vertices in the current partial MPG. Calculation of the derived adjacency between a candidate vertex with each vertex in the current MPG, was dependent on its edge distance from where the candidate vertex would be placed. However the objective function will still not in general accurately reflect the actual benefit of near adjacency, as the objective assumes that the distance between pairs of facilities is a constant.

These two modified heuristics provide an alternative method of constructing the MPG in order to incorporate non-adjacency. However as we will see in Section 10.5 they cannot hope to accurately reflect the actual benefit through this consideration.

In the next section we examine the literature which considers a model for generating representative transportation cost data on the designed MHS within the layout by explicitly designing the MHS (or communication path), rather than by using an approximation in the adjacency graph.

10.2 Previous Work on the MHS

Gawad and Whitehead [38] consider placing a communication path (which is a tree structure) onto the completed layout by sequentially considering the highest links between non-adjacent facilities, and building up the communication path by linking these facilities, with the paths only existing on facility boundaries, using already devised links where possible.

Baybars [7] presented two possible approaches for designing communication paths; the first was to delete edges from the MPG, so that during the layout construction facilities are non-adjacent, and hence circulation spaces result; the second approach is to consider the generated layout, and from this define a graph, where each node represents a 3-joint, and the lengths of the edges are the lengths of the walls joining these joints. The MHS is constructed by finding a tree structure (to provide minimal length of the MHS) in this graph, which is then embedded in the layout.

MacGregor-Smith [81] uses a Steiner node to represent the circulation system of a layout. In order to discuss the ideas of MacGregor-Smith further, we require the concept of a *Voronoi Cell*, which is the set of points in R^2 equidistant from all other cells in R^2, and closer to i, than any other cell j. Formally it is the intersection of all halfplanes defined by the bisectors of i and j. A *Voronoi Diagram* is a collection of all nonempty Voronoi Cells. Further a *Delaunay Triangulation* is the planar straight line dual graph of the Voronoi Diagram; this dual is a triangulation.

MacGregor-Smith obtains the network upon which the MHS is placed, by considering a Delaunay Triangulation of the layout. This Delaunay Triangulation, is equivalent to an MPG, except that the exterior is not considered, and the embedding is unique as it is dependent on the layout. From this Delaunay Triangulation, we derive the Voronoi Diagram, which parallels the layout; however, the building perimeter is missing, and the facility shapes are very distorted. A Steiner Tree is then able to be found using a heuristic devised by MacGregor-Smith, Lee, and Liebman [82], in $O(n \log n)$ time. The model used by MacGregor-Smith, requires a set of vertices which must be on the MHS.

For completeness we now also consider some other related work on the MHS. These papers consider the MHS to be the fundamental building block, with the layout being subsequently constructed around it. The layouts in general become irregular in their building perimeters, and in many cases the generated layouts have holes. Montreuil and Venkatadri [86] present a comprehensive linear programming model for generating a net layout. The main objective is to minimize the average aisle travel, while the constraints consider positions and shapes of facilities, as well as the location of Input/Output stations, and location and size of aisles. Montreuil, Venkatadri and Ratliff [87] present a linear programming approach to determining a layout from a design skeleton, which is essentially an underlying graph of the layout, specifying adjacencies for example. The linear program in this paper attempts to retain as much as possible the structure of the design skeleton, while still generating the best possible layout given the overall objectives. The formulation specifies the layout of rectangular facilities within a rectangular building perimeter. Welgama and Gibson [105] consider a layout problem in which each facility has pick-up and drop-off points, and dimensions specified *a priori*. The problem then is to arrange the facilities in such a way as to minimize the movement cost. The authors provide a construction heuristic along the same lines as many of the classical approaches; they note that a secondary objective is to make the layout as compact as possible, by considering the *sprawl* of the layout in the objective function. Finally, Langevin, Montreuil and Riopel [71] consider a spine layout design, in which the MHS is placed in the middle of the layout, and facilities are placed on either side of it, in order to minimize the movement costs. The method is divided into two steps, firstly to obtain a linear ordering of the cells, and then from this chain, to determine which facilities should be on which side of the spine. The second phase is completed optimally, however the solution to the first phase is heuristic, and hence the generation of a good solution to the first phase is crucial.

The ideas of Gawad and Whitehead, Baybars and MacGregor-Smith, design an MHS directly on an already constructed layout, as opposed to the other papers which consider the MHS as the fundamental component around which the facilities are placed. The methods outlined by these authors, have certain characteristics in common, the most important being the concept of an underlying network structure. The network that we will use, is the layout itself, where every wall, either in part or in full, is a potential edge to add to the MHS. We call this the *layout network*. The objective function is to minimise the total transportation cost, but there is also a cost associated with the length of the MHS. This is likely to lead to a tradeoff between the length of the MHS and the total transportation cost. Note that while we generally deal with adjacency benefit values, we assume that these values are proportional to the material flow between the facilities, and hence in general we assume the adjacency benefit between i and j equals the material flow between facilities i and j. Ignoring the cost associated with the length of the MHS, is likely to lead to an MHS where every pair of facilities has its own *private* MHS, whereas ignoring the transportation cost is likely to lead to a Minimal Spanning Tree within the layout network. An adaptation of the techniques discussed, incorporating a multi-criteria objective, is easily motivated, however not necessarily easily solved. The structure of the MHS is generally of two forms; a tree, or a more general form allowing cycles within the MHS.

10.3 Considering the MHS as a Facility

In this section we consider the impact of considering the MHS as a facility of the layout. In this approach, we consider that the MHS is adjacent to every facility, hence the MHS becomes a centre vertex. In Theorem 4.7.1, we proved that every MPG having a centre vertex, is Deltahedron generateable, therefore the layout can be easily constructed in this case. The difficulty that we have with this type of approach, is that adjacencies with the MHS incorporate (asymptotically) one third of the adjacencies of the adjacency graph, and thereby severely restricts the number of adjacencies the other facilities of the layout can attain. This indicates that the modification of Giffin and Foulds [40] is a preferable model in this case, as all facilities will be at most two walls apart, as the shortest path from any vertex in the MPG to any other can pass through the vertex representing the MHS. However, this approach will still not guarantee an accurate assessment of the actual transportation cost, as there is no provision for maintaining a compact MHS. This is better illustrated by considering how the layout would be constructed when using the Deltahedron Layout Algorithm. The MHS must be in the initial Tetrahedron, due to its large degree. We could place the MHS as the top facility, providing in most cases an elongated MHS, with facilities at opposite ends of the layout considered the same as two non-adjacent facilities near the middle of the layout. The alternative is to place the MHS as either the left or right facility. This is probably more efficient, as it is likely to lead to the MHS being T-shaped due to its large degree; however, it is more likely to be compact, and therefore provide a reduced transportation cost. If we desire to have more than one MHS, say two different production lines, then any subsequent MHSs can be easily incorporated into the problem data by adding as many extra vertices (representing each MHS) as required, following some of the ideas of circulation spaces as described by Baybars [7]. This leads to adjacency graphs with less structure, which are not guaranteed to be Deltahedron, but it is likely that the structure of the MPG could satisfy the structural conditions of Section 4.7.4, where we have a set of disjoint vertices in D_2 which could correspond to the disjoint MHSs.

As with most ideas and methodologies provided throughout this thesis, the consideration of the MHS as a facility can be examined on a problem-by-problem instance, where we can examine the transportation costs in the set of alternative layouts. As a simple guideline, if we are able to utilise the concepts of placement directions and placement hosts of Deltahedron and the special classes of Section 4.7, then the MHS should be used as a placement host as often as possible, as this allows the MHS to become (at worst) X-shaped while other facilities are able to retain (as much as possible) their rectangularity.

The area taken up by the MHS must be addressed, and in general we would require that the area of the MHS is sufficient to allow its effective functionality. For the dimensionalisable layouts of the Deltahedron Layout Algorithm, and the classes of MPGs of Section 4.7, this poses no difficulty, and indeed we can utilise the area perturbation ideas discussed in Section 8.1 to guide this.

10.4 Considering the MHS as a Post Construction Phase

In this section we examine ways in which we may incorporate an MHS into a constructed layout. There are several issues which need to be addressed. Firstly we must determine how *Input/Output Stations* can be assigned for each facility; an Input/Output (I/O) Station, is simply a position on a facility's perimeter, which enables material to enter or leave that facility *e.g.* doors. In order to meaningfully compare the MHS on different layouts on the same problem data, these I/O stations must be able to be assigned consistently between the layouts. Next we must determine under what conditions and measures we assess the performance of the MHS. In the absence of further information, assume that material destined for a facility is to be used uniformly throughout that facility. This leads to the use of the *centroid* of each facility. Therefore we assume centroid-to-centroid distances, unless otherwise stated, from this point onwards. To measure the distance between these centroids, we assume a rectilinear metric, which is the intuitive choice due to the orthogonality of the layout. Further we recognise that the derived distance on the MHS between all pairs of facilities will be a function of the flow of materials between those pairs of facilities, in order to reduce the total transportation cost of all material flows.

Under these assumptions, we define our model in terms of minimising the total transportation cost of an MHS. We impose no structural constraints on the MHS, other than that it is connected. There are two issues to address. The first is that, upon designation of the I/O stations, there is a fixed cost of transporting the material flow for a facility from the centroid to the I/O station for each facility. This appears to necessitate a 2-phase process for the determination of the MHS, the assigning of I/O stations, and secondly the actual determination with respect to these I/O stations of the MHS structure, with the cost of intra-facility flows a fixed cost within this second phase. The second issue is the development of a lower bound on the total transportation cost of the layout. If we assume that we could have a separate MHS for every pair of facilities' centroids, with a separate I/O station for each, then we can assign the lower bound on the MHS to be the shortest rectilinear distance in the layout between each pair of facilities, scaled by the flow between each pair of facilities. The I/O stations in this case would simply be located at the point at which each MHS leaves and enters each facility on the shortest paths.

In order to discuss this more easily, let us define some notation. Let f_{ij} be the material flow between two facilities i and j, d^*_{ij} be the shortest rectilinear path between facilities i and j in the layout network, and d_{ij} be the shortest rectilinear distance between facilities i and j on the MHS. Further, suppose that each facility i is defined by m_i corner points, with coordinates (x_{i_k}, y_{i_k}), $k = 1, \ldots, m_i$. The objective function for determining the MHS under this model is given by Equation 10.1, with the lower bound to this objective being given by Equation 10.2. Note that Equation 10.1 allows only one I/O station for each facility, whereas the lower bound of Equation 10.2 assumes as many as are required for each facility.

$$Min \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} f_{ij} d_{ij} \tag{10.1}$$

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} f_{ij} d^*_{ij} \leq \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} f_{ij} d_{ij} \tag{10.2}$$

Let us examine ways in which we might tackle this problem. Firstly in Equation 10.3, we formally define the centroid of facility i, denoted by (x_i^c, y_i^c).

$$(x_i^c, y_i^c) = \left(\sum_{k=1}^{m_i} \frac{x_{i_k}}{m_i}, \sum_{k=1}^{m_i} \frac{y_{i_k}}{m_i} \right) \tag{10.3}$$

Note that while the centroid of a non-convex facility may lie outside the facility, this poses no problem (as we will see the I/O station is derived to lie on the perimeter of the facility regardless), as the centroid is simply a point to which we can direct all material flow in order to calculate what the actual cost would be, under the assumption of uniform use of material throughout the facility. If this assumption is not valid, we can weight points in the facility, so that the centroid is closer to areas which have greater material flow. This adaptation can be easily achieved, and so is ignored henceforth.

The first issue we must address is the placement of the I/O stations, and two methods have been devised for achieving this. The first seeks to minimise the intra-facility travel cost, while the second attempts to strategically place the I/O station so that it is as close as possible to the other facility centroids. The placement of the I/O station under the first construction is simply to place the I/O station of facility i at the point on the perimeter of i closest to the centroid of i. For a convex facility, either the x or y coordinate of the I/O station will be the same as for the centroid. For non-convex facilities we must also consider the complete set of corners. The optimal point for each facility will be either a corner, or one of (at most) four points on the perimeter of the facility as for the convex facilities. Note that for centroids lying outside the respective facility, not all of these four points may exist. Denote the I/O station of facility i to be at the coordinate $(x_i^{I/O}, y_i^{I/O})$. Then this point is given by Equations 10.4 - 10.7, where X_i and Y_i, are the sets of all corner points j of i, where x_i^c, respectively y_i^c, lies between x_{i_j}, respectively y_{i_j}, and the next corner coordinate $x_{i_{j+1}}$, respectively $y_{i_{j+1}}$, encountered. Note that we assume $i_{m_i+1} = i_1$.

$$\begin{aligned} (x_i^h, y_i^h) &= (x_i^c, y_{i_k}) : \min_{i_k \in X_i}(|y_{i_k} - y_i^c|) && (10.4) \\ (x_i^v, y_i^v) &= (x_{i_k}, y_i^c) : \min_{i_k \in Y_i}(|x_{i_k} - x_i^c|) && (10.5) \end{aligned}$$

$$
\begin{aligned}
(x_i^r, y_i^r) &= (x_{i_k}, y_{i_k}) : \min_{k=1\ldots m_i} (|x_i^c - x_{i_k}| + |y_i^c - y_{i_k}|) && (10.6)\\
(x_i^{I/O}, y_i^{I/O}) &= (x_i^k, y_i^k) : \min_{k=h,v,r} (|x_i^c - x_i^k| + |y_i^c - y_i^k|) && (10.7)
\end{aligned}
$$

The second construction that we consider is concerned with strategically placing the I/O station, in an attempt to minimise the inter-facility flows. This is achieved by considering the weighted centroid of the distance from a facility to all other facilities. Therefore define (x_i^s, y_i^s) to be the point of the weighted centroid of the other facilities' flows with facility i, and call it the *sink* of facility i. For each facility we determine its sink via Equation 10.8; note that since $f_{ii} = 0$, there is no unnecessary biasing of this centroid.

$$
(x_i^s, y_i^s) = \frac{1}{\sum_{j=1}^n f_{ij}} \left(\sum_{j=1}^n f_{ij} x_j^c, \sum_{j=1}^n f_{ij} y_j^c \right) \qquad (10.8)
$$

To assign the locations of the I/O stations, we use Equations 10.9 - 10.12. Note that the I/O stations are determined independently of the actual route the material would have to traverse in going from the sink to the facility centroid. This creates no difficulty, and indeed Equations 10.9 - 10.12 can be easily modified so that the I/O station's placement is determined relative to the layout network.

$$
\begin{aligned}
(x_i^h, y_i^h) &= (x_i^s, y_{i_k}) : \min_{i_k \in X_i} (|y_{i_k} - y_i^s|) && (10.9)\\
(x_i^v, y_i^v) &= (x_{i_k}, y_i^s) : \min_{i_k \in Y_i} (|x_{i_k} - x_i^s|) && (10.10)\\
(x_i^r, y_i^r) &= (x_{i_k}, y_{i_k}) : \min_{k=1\ldots m_i} (|x_i^s - x_{i_k}| + |y_i^s - y_{i_k}|) && (10.11)\\
(x_i^{I/O}, y_i^{I/O}) &= (x_i^k, y_i^k) : \min_{k=h,v,r} (|x_i^s - x_{i_k}| + |y_i^s - y_{i_k}|) && (10.12)
\end{aligned}
$$

There are two further issues which we must consider when considering I/O stations. The first is the I/O station of the exterior facility. Obviously the centroid of this facility lies in the centre of the layout, yet the I/O station for the exterior facility can still be meaningfully determined, via the two constructions outlined. The second issue is the use of tie-breaking rules. There may exist more than one candidate I/O station location for a facility. In this instance an arbitrary choice is not generally desirable, as we wish to keep the I/O station as *central* as possible.

Therefore we could implement the alternative I/O construction in order to rank these candidate locations under the chosen construction.

Initial experimental evidence showed that the first construction based on minimising the intra-facility flows was the most realistic, as the sinks are in general closely clustered near the centre of the layout. The second construction (using the sinks) however proved very useful as a tie-breaking mechanism. Furthermore the intra-facility transportation costs overwhelmed the inter-facility transportation costs if the second construction was used as the main I/O placement routine. This was due to many pairs of adjacent facilities having their material flow travelling more than twice the distance that it could if the I/O stations were placed as close to the centroid as possible. Therefore we consider only the first construction, using the second as the tie-breaking routine. The two constructions outlined are able to be applied to any layout, and therefore the I/O stations of two layouts, although probably not at the same coordinates, are placed consistently, in order to facilitate comparison between the layouts.

The next step is to minimise the inter-facility transportation costs with respect to the I/O stations. Having determined the I/O stations, we can calculate the fixed costs of the intra-facility transportation costs, and this cost S is given by Equation 10.13. Note that under the first construction, the intra-facility costs are optimised.

$$S = \sum_{i=1}^{n}\sum_{j=1}^{n} f_{ij}(|x_i^c - x_i^{I/O}| + |y_i^c - y_i^{I/O}|) \tag{10.13}$$

The determination of which edges in the layout network should actually comprise the MHS is not a trivial task. Since our goal is to create a consistent evaluation only, we confine our discussion to that of underlying tree MHS structures; extensions to the more general case are easily obtained, by modifying the objective function to include a cost associated with the length of the MHS, and can be solved in essentially the same way. Note that even though the MHS has an underlying tree structure, the nature of the rectilinear metric implies that the placement of the MHS on the actual layout network may generate MHSs containing cycles.

As we are dealing with a tree structure, we require the identification of $n-1$ connected edges to construct the MHS. Unfortunately this is not simply a Minimum Spanning Tree Problem, as every 3-joint in the layout is a possible Steiner point, at which we can create intersections of the MHS which are not at I/O stations, leading to the possibility of the MHS allowing cycles. In order to derive the MHS, we must take account of the flows between the facilities. This weighting lends itself easily to a scaling of the length of each path connecting every pair of facilities. From the layout network, we determine the graph K_n, with every edge having weight proportional to the rectilinear distance between facilities, and the flow between the facilities. This is achieved by assigning the weight t_{ij} to each edge (i, j), where t_{ij} is defined by Equation 10.14. This definition of the edge weights provides an intuitive measure, of the flow per unit length.

$$t_{ij} = \frac{f_{ij}}{d^*_{ij}} \tag{10.14}$$

From this constructed graph K_n, we determine a Maximal Spanning Tree, using the method of Kruskal [67]. The edges of the K_n graph are chosen for inclusion in the MHS so that the edges of largest flow per unit distance are chosen first. The total transportation cost of the layout then becomes the fixed intra-facility costs added to the weighted cost of inter-facility travel between each facility. Note that the placement of the MHS in the layout can cause a reduction in the total length of the MHS, as edges added to the MHS from K_n may duplicate walls in the layout. This may lead to some 3-joints within the layout becoming Steiner points. Two improvement type procedures have been developed to remove some of the arbitrariness in the placement of the MHS; they guarantee not to increase the length or the cost of the MHS, and so can be performed without recourse to the ramifications of the changes. The first improvement procedure considers a facility whose complete material flow all travels through another facility's I/O station, and there exists another path of the same length between these two facilities, we choose the path which provides the most duplication of edges on the layout network. This cannot increase the transportation cost, and can decrease both the length of the MHS and the transportation costs for that facility, leading to a lower overall transportation cost. The second improvement assumes that the exterior I/O station may be placed at any point on the perimeter, as there are no intra-facility costs associated with the exterior. In this instance, if the exterior I/O station is a leaf node of the MHS system, we can move the I/O station to the first point encountered which is either a junction of the MHS, another I/O station, or the first point encountered on the layout perimeter on the path to the exterior's I/O station. The full MHS construction is given in Algorithm 10.1.

Algorithm 10.1 *MHS Construction*

```
Input: Layout, and inter facility flow data $f_{ij}$ for each pair of facilities
Output: Layout with corresponding MHS
for $i = 1$ to $n$ do
    $(x_i^c, y_i^c) \leftarrow \left(\sum_{k=1}^{m_i} \frac{x_{i_k}}{m_i}, \sum_{k=1}^{m_i} \frac{y_{i_k}}{m_i}\right)$
    $(x_i^s, y_i^s) \leftarrow \frac{1}{\sum_{j=1}^{n} f_{ij}} \left(\sum_{j=1}^{n} f_{ij} x_j^c, \sum_{j=1}^{n} f_{ij} y_j^c\right)$
    With reference to Equations 10.4 - 10.7 assign the I/O stations
end
Implement Floyd's Algorithm to determine the shortest path length $d_{ij}$ between
each pair of I/O stations
for $i = 1$ to $n - 1$ do
    for $j = i + 1$ to $n$ do
        $t_{ij} \leftarrow \frac{f_{ij}}{d_{ij}}$
    end
end
Determine the Minimal Spanning Tree under the $t_{ij}$ weights using Kruskal's
Algorithm
Place MHS on layout network via the shortest path and minimal spanning
tree data
Implement the two improvement procedures to reduce the cost and length of
the MHS

end
```

We proceed to illustrate the construction of the MHS through the use of a six facility example. The layout is given by Figure 10.1, while the material flow data is given in Table 10.1. The centroids are found, and then the I/O stations are assigned using the technique described earlier. From this set of I/O stations, we construct the K_6 network, with edge distances given by Table 10.2, and hence the corresponding weights for the edges given by Table 10.3.

With reference to Table 10.3, we see that the Minimal Spanning Tree associated with this network is the set of edges $\{(3,4), (4,5), (2,5), (5,6), (1,2)\}$. Overlaying

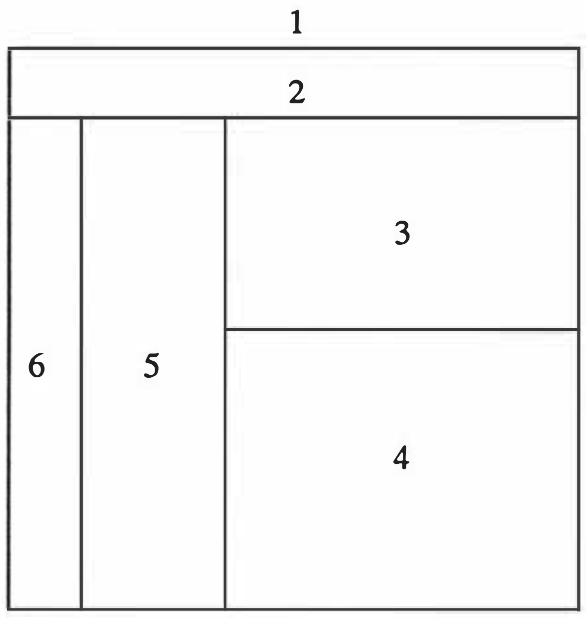


Figure 10.1: MHS Illustrative Example

	2	3	4	5	6
1	40	15	30	20	20
2		20	0	25	0
3			80	10	0
4				20	0
5					45

Table 10.1: Flow data for the MHS Illustrative Example

	2	3	4	5	6
1	22.50	26.25	26.25	28.75	23.75
2		16.25	16.25	11.25	16.25
3			0.00	7.50	27.50
4				7.50	27.50
5					22.50

Table 10.2: Inter-facility distances between I/O stations

	2	3	4	5	6
1	1.78	0.19	1.14	0.70	0.84
2		1.23	0.00	2.22	0.00
3			∞	1.33	0.00
4				2.67	0.00
5					2.00

Table 10.3: Inter-facility flows per unit distance between I/O stations

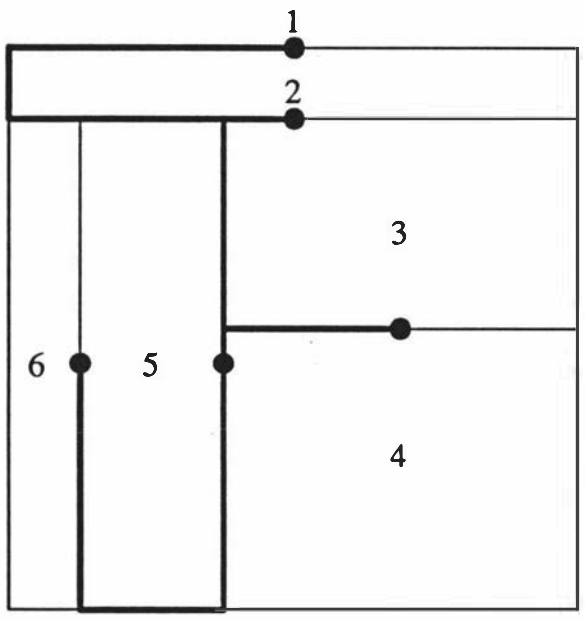


Figure 10.2: Initial MHS design

this MHS on the layout provides the MHS of Figure 10.2, with length 60, and total cost of 7468.75, of which the inter facility costs are 5862.50. Note that facilities 3 and 4 have their I/O station at the same point.

We now attempt to implement our two simple improvement procedures on this MHS. The path between facilities 5 and 6 can be changed for its alternative path, and the I/O station of the exterior can be moved to the first point at which the perimeter is touched en route to its current position. This results in the new MHS of Figure 10.3, with length 33.75, and total cost of 5356.25, resulting in the MHS length and inter-facility costs being significantly reduced.

Note that while we are unable to guarantee optimality, the effect of this is reduced when comparing layouts, as consistency between the two layouts is paramount

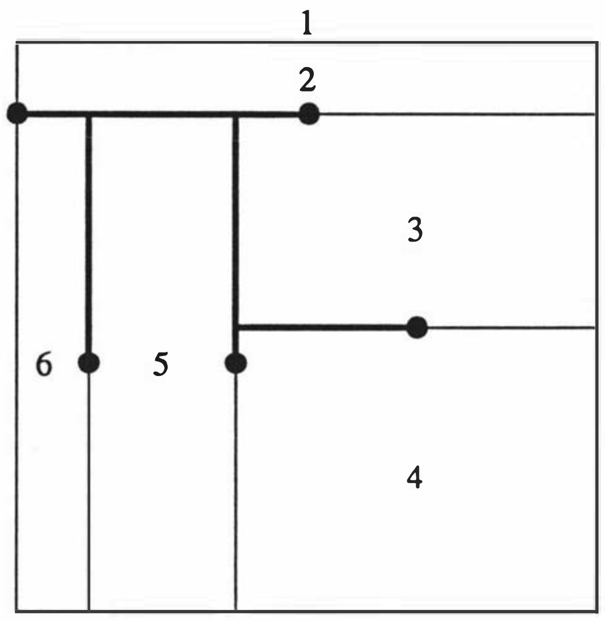


Figure 10.3: Final MHS design

for the effective comparison of them under the MHS evaluation scheme. For example, if we found an MHS for a rectangular layout heuristically, and a highly irregular layout optimally, we would be unable to make a valid comparison. If the MHS for each layout is constructed under the same mechanism, we can rank the layouts with respect to the derived total transportation costs, as another measure of the effectiveness or practicality of each layout. Therefore, while it is possible to implement a Tabu Search, or similar, routine in order to attempt to improve upon the initially constructed MHS, we do not implement this for comparison purposes, as we want to maintain as much consistency as possible. In attempting to determine the best MHS for some chosen layout, however, this improvement type of strategy can be performed.

10.5 MHS Computational Experiments

In this section we examine the MHS as an evaluation mechanism for comparing different layouts. The MHS construction was applied to the layouts generated in Chapter 6, and the results are shown in Tables 10.4 - 10.8. The VSA and Contraction Algorithm routines used the Deltahedron initialisation and the layout improvement routines.

We expected the results would overwhelming show that a good adjacency score corresponded to a good transportation cost. Surprisingly, the opposite occurred, with those layouts having lower adjacency scores having the most efficient MHSs. Throughout the problem instances the Spanning Tree Algorithm consistently outperforms the MHSs of the other layouts. This is due mainly to the structure of the Spanning Tree layouts, as the hierarchical layering of the facilities in the layout allows the MHS to closely mimic the underlying tree adjacency graph. The Contraction Algorithm outperforms the VSA in all cases, which is perhaps a little surprising, as we saw in Section 6.6 that the VSA outperformed the Contraction Algorithm under the regularity measures. Furthermore for $n = 10$, and $n = 15$, SIMPLE outperforms the VSA, the Contraction Algorithm, and surprisingly Deltahedron, yet has vastly inferior regularity scores. The reason for this is that the irregularity of many facilities in SIMPLE and Contraction Algorithm layouts, allows many non-adjacent facilities to be in close proximity. The Tiling Algorithm with a worst case T shape performs poorly in all problem instances, and this is due to the large proportion of facilities which are indeed T shapes within these layouts. The Contraction Algorithm performs surprisingly well for large n, surpassing the Deltahedron and Tiling Algorithms.

The results of this experiment are rather surprising, and the preliminary conclusion that the adjacency benefit is not a good predictor is both unexpected and disappointing. The apparent lack of correlation between the regularity of the layout and the efficiency of the MHS is also disappointing, as is the relationship between the transportation cost and the length of the MHS. Although there is the expected general trend of the transportation cost increasing as the length of the MHS increases, there is a large variance in these values, and indeed in many instances a longer MHS can provide a better transportation cost. These results are a function of the MHS form derived from this construction. As a direction for further work, alternative MHS constructions may prove further insights into the relationship between the regularity values and transportation costs.

As a second experiment, we examine the methods designed by Giffin and Foulds which incorporate near adjacency, in order to quantifiably analyse their performance at approximating the true transportation cost. We concentrate on the first

Measure	VSA	CA	SIMPLE	Deltahedron	TA(T)	TA(I)	ST
Adj Benefit	4258.02	4258.02	4258.02	4186.42	4304.02	4110.58	3619.22
Transport Cost	128546.03	124252.07	103449.84	122782.83	143733.10	100962.56	104307.25
MHS Length	80.76	81.75	70.41	81.23	92.25	69.60	67.83

Table 10.4: MHS Cost and Length for $n = 10$

Measure	VSA	CA	SIMPLE	Deltahedron	TA(T)	TA(I)	ST
Adj Benefit	7463.27	7463.27	7463.27	7441.69	7518.36	7266.96	6100.78
Transport Cost	407209.30	368713.18	311392.13	375169.26	485102.83	313266.97	310151.56
MHS Length	148.58	139.00	128.18	140.28	173.43	130.52	118.18

Table 10.5: MHS Cost and Length for $n = 15$

Measure	VSA	CA	SIMPLE	Deltahedron	TA(T)	TA(I)	ST
Adj Benefit	11550.47	11550.47	-	11506.27	11367.22	11099.82	9046.56
Transport Cost	1028984.58	933726.03	-	919898.24	1301800.77	836981.13	772929.81
MHS Length	220.16	214.59	-	220.63	283.78	214.13	173.64

Table 10.6: MHS Cost and Length for $n = 20$

Measure	VSA	CA	SIMPLE	Deltahedron	TA(T)	TA(I)	ST
Adj Benefit	15267.22	15267.22	-	15373.38	14963.78	14678.07	11683.40
Transport Cost	1927076.81	1598925.70	-	1665489.18	2392326.76	1618287.59	1452495.80
MHS Length	302.17	298.92	-	281.30	388.66	309.88	236.86

Table 10.7: MHS Cost and Length for $n = 25$

Measure	VSA	CA	SIMPLE	Deltahedron	TA(T)	TA(I)	ST
Adj Benefit	18791.44	18791.44	-	19157.67	18572.24	18295.87	14309.18
Transport Cost	3350072.28	2584713.62	-	2747796.95	4515771.14	2807231.77	2355569.53
MHS Length	412.31	371.28	-	352.80	527.50	413.21	293.95

Table 10.8: MHS Cost and Length for $n = 30$

Measure	Adj Benefit		Transport Cost	
	Greedy	S	Greedy	S
Transport Cost	122782.83	112410.92	127317.35	120252.87
MHS Length	81.23	77.37	88.11	80.53
Adj benefit	4186.42	4105.44	2159.47	4163.07
Transport Approx	35768.85	34164.54	49255.27	34766.56

Table 10.9: Deltahedron results under Adjacency Benefit and Transportation Cost Objectives for $n = 10$

Measure	Adj Benefit		Transport Cost	
	Greedy	S	Greedy	S
Transport Cost	375169.26	374264.26	383009.12	381954.10
MHS Length	140.28	133.52	160.67	143.83
Adj Benefit	7441.69	7258.44	3282.00	7238.18
Transport Approx	97643.98	92236.90	128128.34	89900.45

Table 10.10: Deltahedron results under Adjacency Benefit and Transportation Cost Objectives for $n = 15$

paper [31] which uses the transportation cost by approximating the centroid-to-centroid distances. Upon examination of the method of Giffin and Foulds we see that there is an inconsistency between the choice of the initial Tetrahedron, and the placement of the subsequent vertices. In order to remedy this, we brought the initialisation into line with the insertion process by greedily choosing the Tetrahedron of minimal cost. We will see that this led to some surprising results, and hence we also implemented the S-construction in order to generate an alternative MPG under this objective. The results of the constructions are given in Tables 10.9 - 10.13.

The most surprising feature of this experiment is the performance of the greedy approach under the transportation cost objective. In all cases the length of the MHS is longer than the other methods, while for n larger than 20, the true transportation cost is smaller. Furthermore, the adjacency benefit scores for this approach are significantly inferior to those constructions under the adjacency benefit objective, and the S-construction under the transportation cost objective, being only around 40% of the other adjacency benefit scores. Intuitively we would expect this method to perform well, however the method actually adds the vertices

Measure	Adj Benefit		Transport Cost	
	Greedy	S	Greedy	S
Transport Cost	919898.24	906889.55	889636.84	954459.40
MHS Length	220.63	209.51	244.64	230.77
Adj Benefit	11506.27	10943.91	4534.40	10553.31
Transport Approx	225307.59	208681.26	280399.99	195920.71

Table 10.11: Deltahedron results under Adjacency Benefit and Transportation Cost Objectives for $n = 20$

Measure	Adj Benefit		Transport Cost	
	Greedy	S	Greedy	S
Transport Cost	1665489.18	1672619.34	1576721.50	1724804.55
MHS Length	281.30	275.60	333.66	311.83
Adj Benefit	15373.38	14497.27	5580.44	13623.42
Transport Approx	402046.64	358686.44	480631.47	321901.44

Table 10.12: Deltahedron results under Adjacency Benefit and Transportation Cost Objectives for $n = 25$

Measure	Adj Benefit		Transport Cost	
	Greedy	S	Greedy	S
Transport Cost	2747796.95	2844532.97	2589838.73	2915410.56
MHS Length	352.80	348.77	429.40	423.73
Adj Benefit	19157.67	18509.93	7127.31	16584.04
Transport Approx	615854.13	561481.24	740859.48	479819.15

Table 10.13: Deltahedron results under Adjacency Benefit and Transportation Cost Objectives for $n = 30$

with *minimal* material flow first. For example the initial Tetrahedron can be constructed using four facilities having no material flow between any pair, resulting in a transportation cost of zero. However placing all of these vertices with smaller flows first makes the efficient placing of pairs of facilities with large material flows difficult. For this reason we implemented the S-construction, which ensures that the facilities with the largest flows are placed first, resulting in a better approximation of the transportation cost, and providing more realistic adjacency benefits. Note, however, that the actual transportation cost in these layouts is inferior to the other approaches. The effectiveness of the S-construction, which is usually outperformed in the Deltahedron methods under the adjacency benefit objective by the other constructions [13], is not surprising upon reflection, as it ensures vertices with large flows are placed early in the MPG. It is interesting to note that the two constructions under the adjacency benefit objective both provide efficient approximate and actual transportation costs. As a final comment we also see that the approximation of Foulds and Giffin of the transportation cost of the layout severely underestimates the actual cost of material flow through the MHS under the MHS design outlined. This is, of course, due primarily to the transportation cost approximation being calculated without recourse to the actual layout.

Coda

In this chapter we have investigated the incorporation of a Material Handling System, into a layout for the purpose of generating a further comparison scheme based upon transportation cost. The heuristic nature of the MHS construction is not a major concern, as we are interested in calculating the transportation cost of the MHS with respect to the MHS of another layout on the same problem constraints, under the same MHS construction. This enables a relative ranking of the layouts, and we assume (as the MHS construction will be a function of the layout) that the layout which produces the best transportation cost heuristically is also likely to produce the best transportation cost optimally. Therefore once we have chosen a preferred layout, we can concentrate more fully on obtaining the best MHS for that layout.

We have seen in this chapter that the adjacency benefit objective does not correlate with the transportation cost objective. Further we have seen that approximating the transportation cost does not necessarily provide a surrogate for the true transportation cost, by severely underestimating the impact of the layout structure on the transportation cost. There are perhaps a number of areas for future work within the design of MHSs. In this chapter we have been concerned with the role of the MHS as another evaluator for comparing layouts, and hence have concentrated more on the consistency of the MHS construction between layouts. We have considered only one MHS construction in this work; it would appear that the development of further constructions would be worthwhile, and indeed a characterisation of the relationship between the layout structure and the MHS construction would prove most useful. Other directions for research of the role of MHS could examine how the transportation cost might be more realistically approximated within the MPG. A useful starting point could be the examining of the EDIST measure proposed by Bozer and Meller [16] as an alternative to the centroid-to-centroid distance measure. A further interesting problem in MHS design is the possible decomposition of the flow matrix into sets of facilities with large interactions to provide a set of disjoint MHSs.

In the next chapter, we attempt to put the material covered so far into practise, by considering the design of a layout from conception through to the final construction of the set of alternative layouts. We will use the ideas of decomposition, interaction and perturbation to generate the layouts, and the MHS constructions and regularity measures to quantifiably rank the layout alternatives.

Chapter 11

Putting It All Together - A Tutorial

In this chapter we conduct two case studies from the raw data through to the generation of a set of alternative layouts, with the aims of showing the application of the many methods and to coalesce ideas discussed throughout this thesis.

11.1 Case Study 1: A Manufacturing Plant

This problem is an 18 facility problem given on *pg* 180 of Francis, McGinnis and White [36]. The problem given is for a manufacturing plant, with material flows between facilities provided in Table 11.1, with area specifications and facility functions in Table 11.2. We assume that the adjacency benefits are proportional to the material flows in accordance with our discussion in Chapter 10 on the relationship between adjacency and material flow.

TESSA coupled with a Tabu Search routine returns an MPG, as shown in Figure 11.1, which at first sight appears complicated and rather overwhelming, of overall adjacency benefit of 1234 (the upper bound is 1314), while Deltahedron generates an MPG of weight 1216. A quick test shows Figure 11.1 to be non-Deltahedron generateable, hence we will concentrate on generating a practical layout dual to this MPG, while remembering that we can guarantee a layout with a worst-case facility shape of T for a decrease in benefit of 18.

Layouts generated via myopic implementation of the VSA, Contraction and SIMPLE techniques are impractical and not worth pursuing further, as is plainly

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100
2		25	22	30	62	0	15	0	0	0	0	0	0	18	0	0	0
3			0	0	0	0	0	0	0	20	5	0	0	0	0	0	0
4				0	10	5	0	0	0	5	0	0	0	0	0	0	0
5					5	10	0	7	5	0	0	0	0	3	0	0	0
6						10	0	5	5	6	26	2	0	31	0	0	0
7							0	5	3	10	23	0	1	20	0	0	0
8								0	0	15	10	5	7	8	0	0	0
9									2	5	0	8	0	0	0	0	0
10										0	4	3	0	0	0	0	0
11											35	25	7	1	0	0	0
12												5	2	5	0	0	0
13													15	30	0	0	0
14														32	0	0	0
15															175	0	0
16																175	0
17																	175

Table 11.1: Adjacency Benefits for the Manufacturing Plant Problem

Facility	Function	Area	Facility	Function	Area
1	Receiving	500	10	Broaching	300
2	Raw Materials Storage	1000	11	Milling	3000
3	Shearing	200	12	Drilling	1200
4	Sawing	200	13	Heat Treating	500
5	Automatic Screw Machine	4000	14	Plating	300
6	Turret Lathe	2000	15	Assembly	700
7	Engine Lathe	500	16	Finished Goods Storage	1000
8	Punch Press	1000	17	Shipping	500
9	Hobbing	400	18	Exterior	-

Table 11.2: Function and Area of Facilities for the Manufacturing Plant Problem

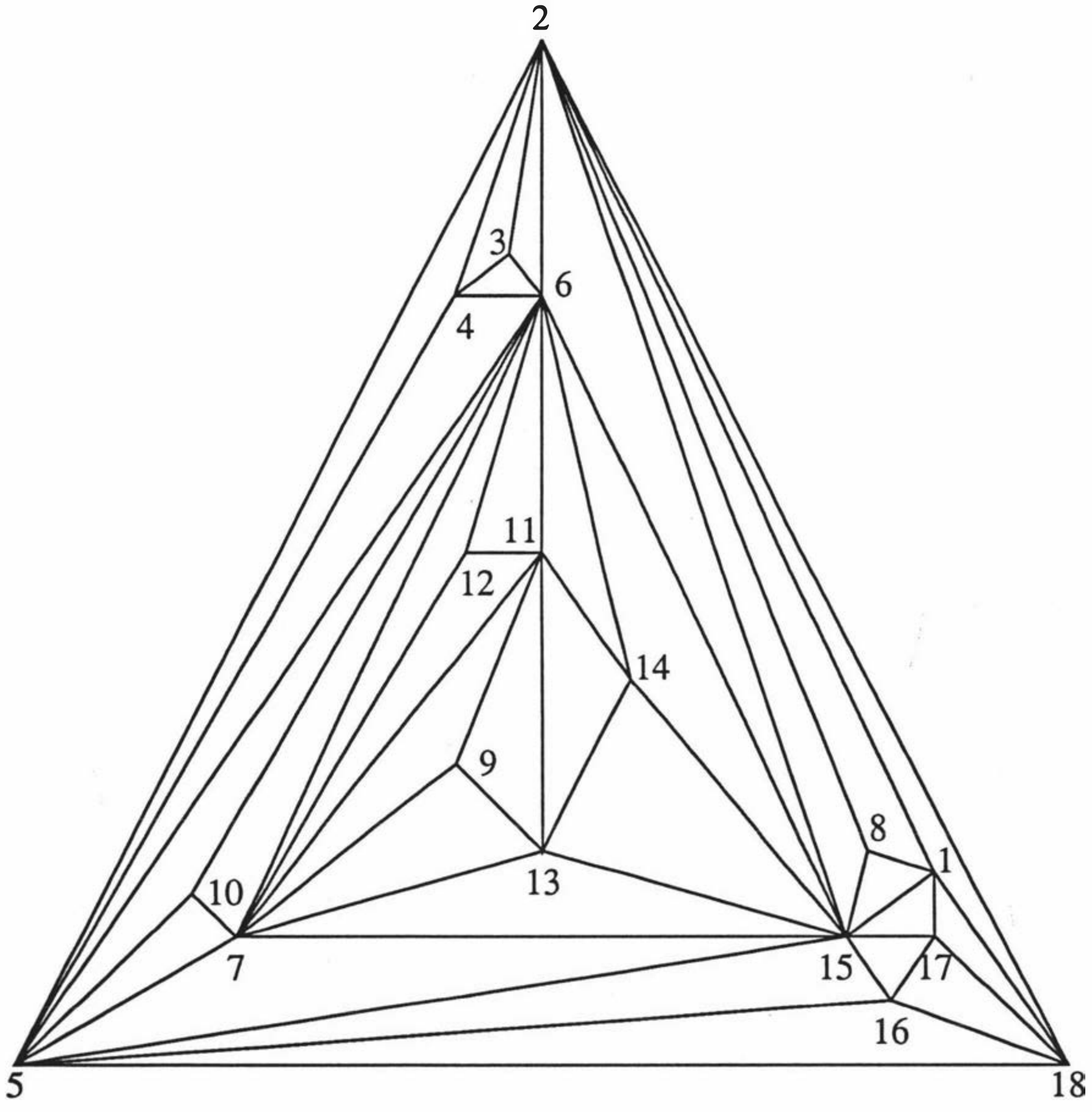


Figure 11.1: MPG generated by TESSA for the Manufacturing Plant Problem

Measure	VSA	CA	SIMPLE	Deltahedron	TA(T)	TA(I)	ST
Adj Benefit	1234	1234	1234	1216	1147	1177	1119
Ave C_i	9.41	11.06	20.82	4.71	4.59	4.00	4.00
Max C_i	18.00	26.00	74.00	6.00	6.00	4.00	4.00
Ave $a_i/\bar{S}_i$	0.201	0.186	0.208	0.137	0.163	0.175	0.407
Min $a_i/\bar{S}_i$	0.042	0.024	0.024	0.033	0.018	0.013	0.015
Ave $a_i/\bar{R}_i$	0.400	0.460	0.571	0.800	0.744	1.000	1.000
Min $a_i/\bar{R}_i$	0.053	0.037	0.028	0.046	0.029	1.000	1.000
Ave $4\sqrt{a_i}/P_i$	0.452	0.438	0.474	0.563	0.538	0.618	0.746
Min $4\sqrt{a_i}/P_i$	0.190	0.136	0.064	0.214	0.165	0.225	0.238
Transport Cost	179116.66	207961.65	134811.41	221020.49	168925.80	183940.48	149736.11
MHS Length	973.67	1228.85	1711.02	905.62	1141.19	1005.50	714.73

Table 11.3: Adjacency, Regularity and Transportation Data of the Layouts generated for the Manufacturing Plant Problem

	Decomposition 1 (Figure 11.7)	Decomposition 2 (Figure 11.8)
Adj Benefit	1234	1234
Ave C_i	4.70	4.70
Max C_i	8.00	6.00
Ave $a_i/\bar{S}_i$	0.316	0.233
Min $a_i/\bar{S}_i$	0.054	0.034
Ave $a_i/\bar{R}_i$	0.781	0.766
Min $a_i/\bar{R}_i$	0.075	0.085
Ave $4\sqrt{a_i}/P_i$	0.697	0.610
Min $4\sqrt{a_i}/P_i$	0.271	0.288
Transport Cost	210423.36	293618.10
MHS Length	810.26	1096.99

Table 11.4: Adjacency, Regularity and Transportation Data of the Layouts generated by Decomposition for the Manufacturing Plant Problem

seen in Table 11.3, where the worst case number of corners on each of these layouts is 18, 26 and 74 respectively. For brevity their layouts are not included. Layouts generated by the Tiling, Deltahedron, and Spanning Tree Algorithms are much more promising however, and are depicted in Figures 11.2 - 11.5, with their corresponding transportation, adjacency and regularity values provided in Table 11.3. We see that none of these layouts is initially ideal, especially with regard to the elongation of facilities 3 and 4. From Table 11.3 it appears that the Spanning Tree layout is the most regular having the best regularity value in every category, except the minimum bounding square, and further provides the shortest, and most cost effective (disregarding the SIMPLE layout as discussed already) MHS. Subjective examination of these layouts appears to confirm the superiority of the Spanning Tree layout.

From this juncture we also attempt alternative approaches for increasing the efficiency of the layouts. Interactive Decomposition is the obvious route to try. There exist nine separating triangles in the MPG of Figure 11.1, providing the decomposition tree of Figure 11.6 and hence the MPG is found to be 7-decomposable. Two alternative decomposition layouts, provided in Figures 11.7 and 11.8 with data values in Table 11.4, are created using The Templates of Appendix A, and the Contraction Algorithm with interaction. The regularity values of these Decomposition layouts are comparable to those of the Tiling, Deltahedron and Spanning

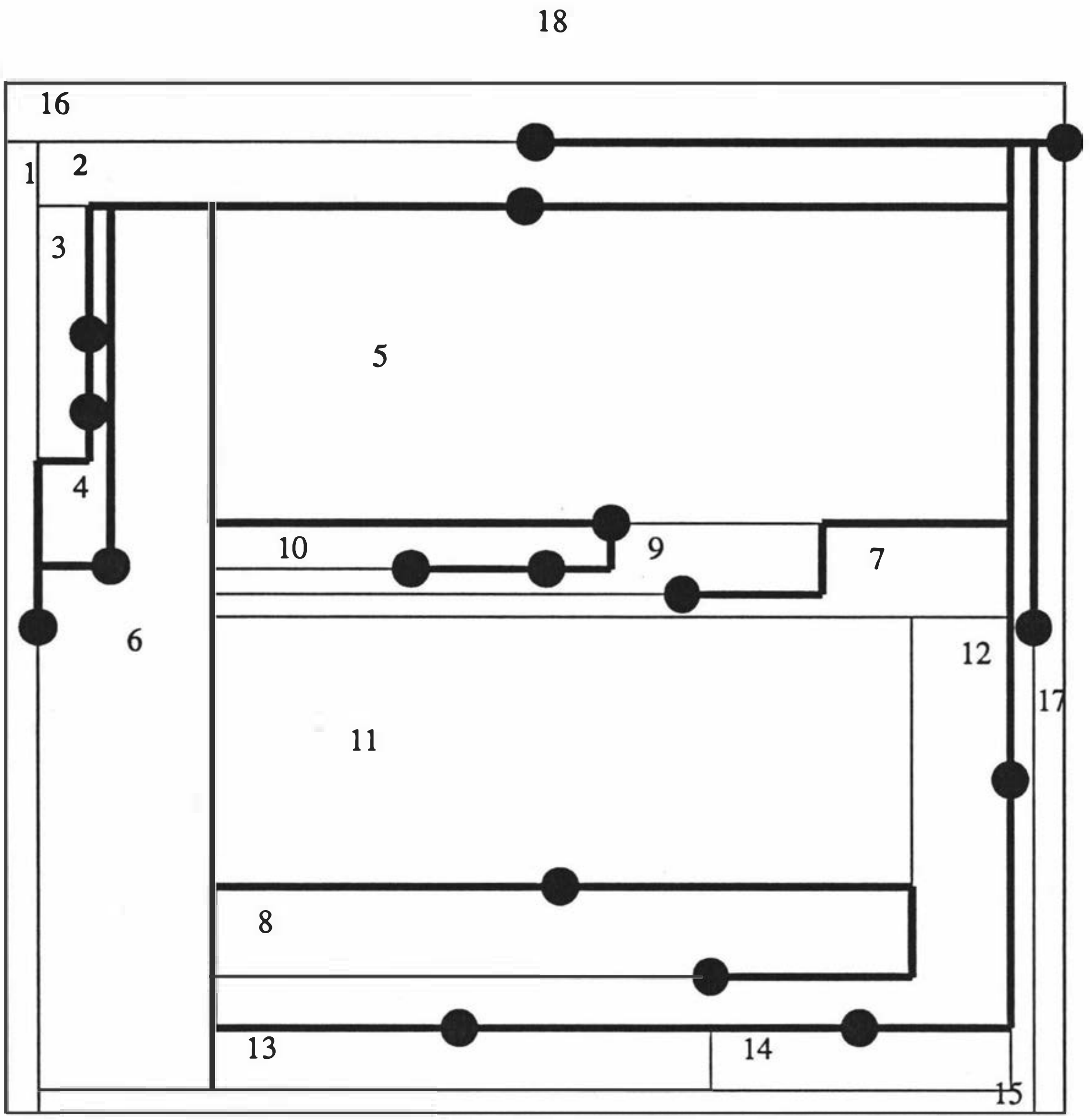


Figure 11.2: Deltahedron Layout for the Manufacturing Plant Problem

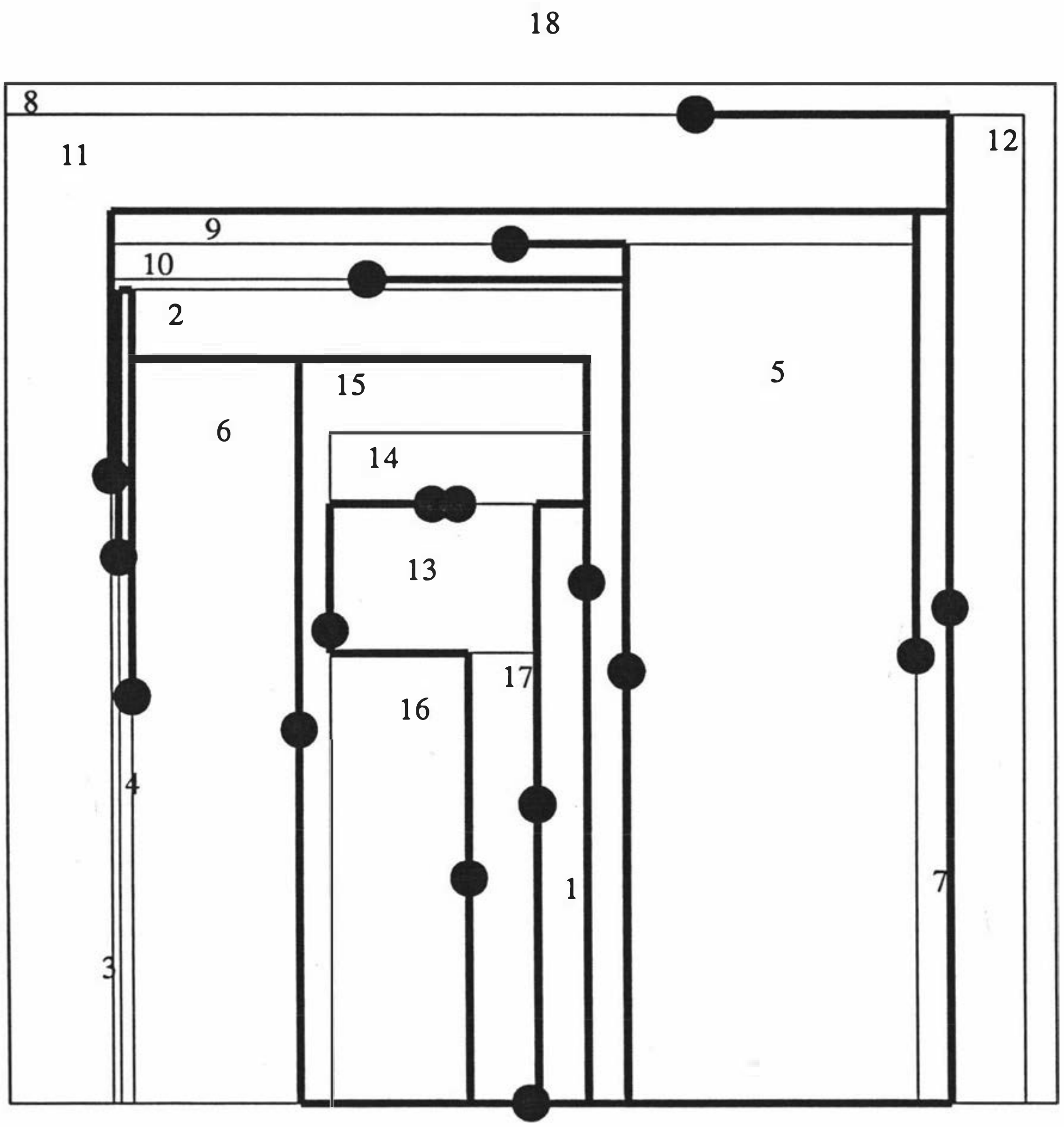


Figure 11.3: Tiling Algorithm(T) Layout for the Manufacturing Plant Problem

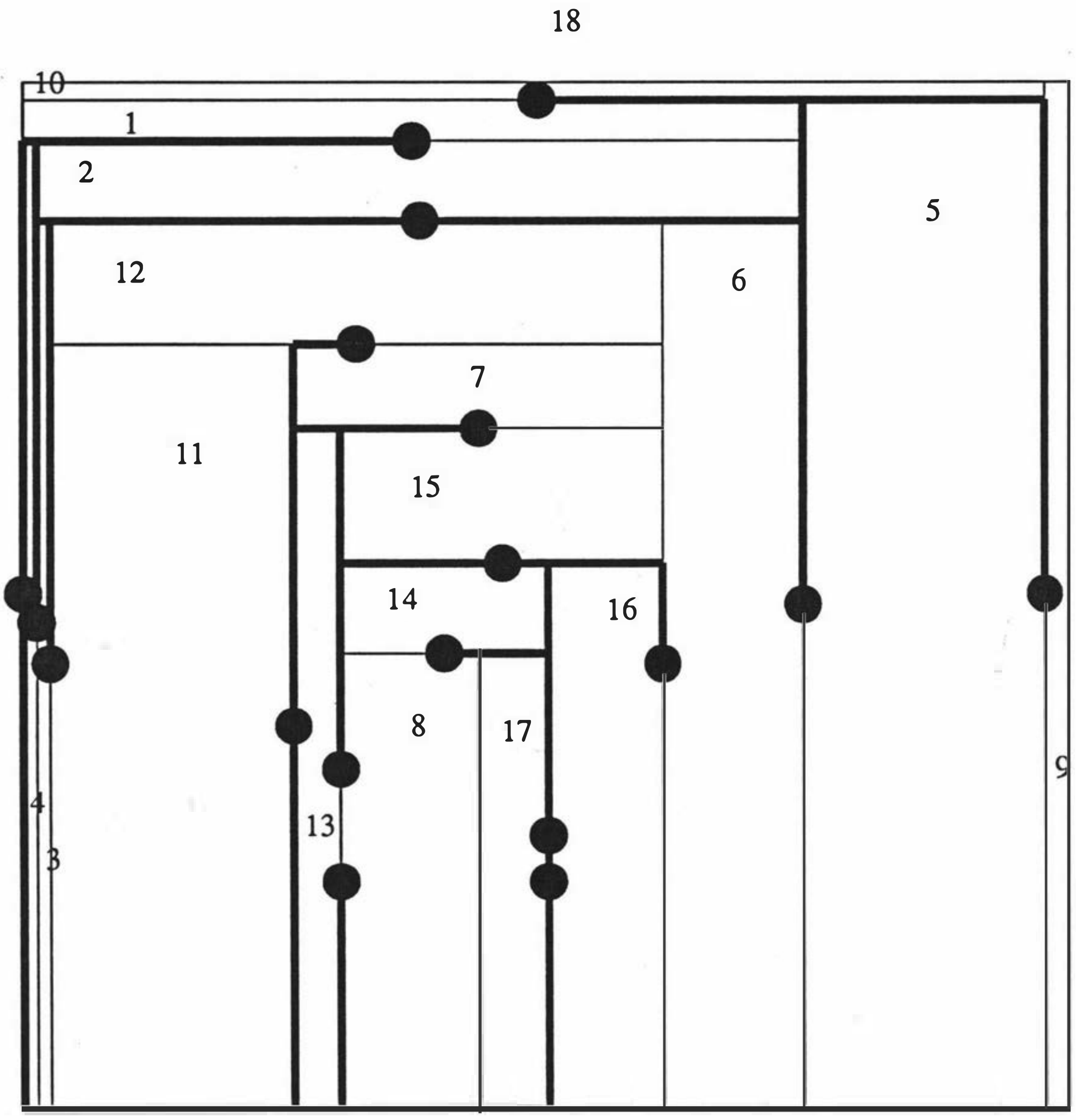


Figure 11.4: Tiling Algorithm(I) Layout for the Manufacturing Plant Problem

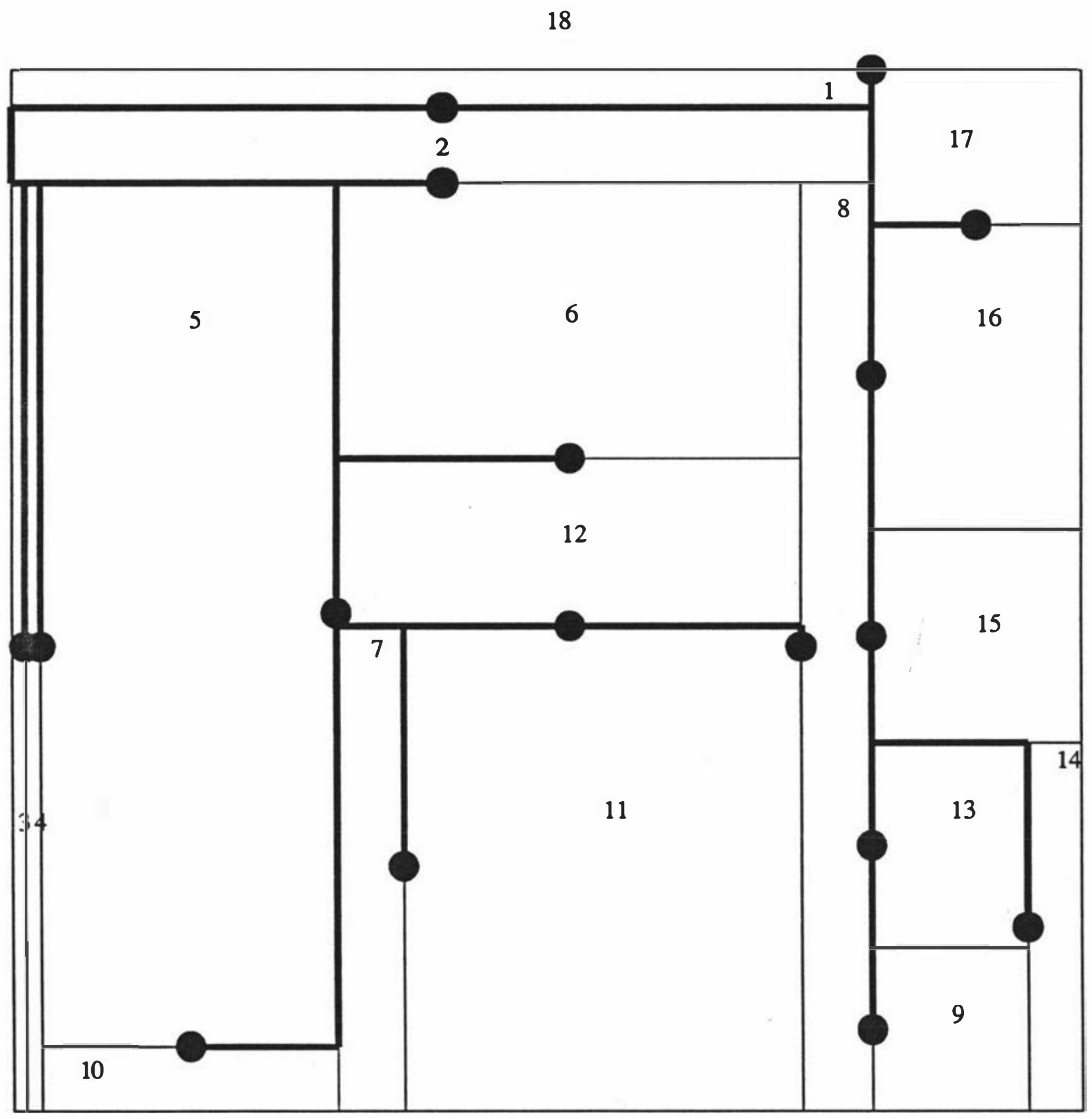


Figure 11.5: Spanning Tree Layout for the Manufacturing Plant Problem

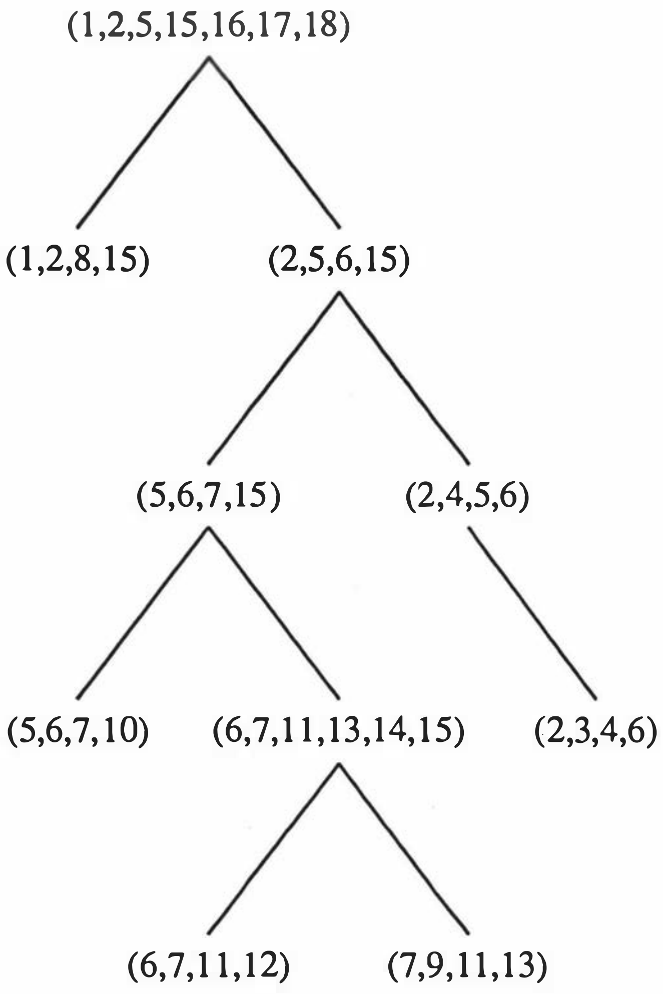


Figure 11.6: Decomposition Tree of the MPG for the Manufacturing Plant

Tree layouts, yet with a higher adjacency benefit. This is offset however by the substantially higher transportation costs. None of these layouts is currently ideal, so we attempt to examine the layouts for possible modifications that can be applied to increase their practicality.

The Deltahedron layout (Figure 11.2) has 6 L-shaped facilities, of which only facility 6 is really in a practical form. A number of the rectangular facilities are particularly elongated, and the MHS on this layout appears to have an unnecessarily high cost due to the spiral type structure around facilities 8 and 12. It appears that a reasonable amount of work is required to ensure this layout is practical, so while we do not completely discard it, we attempt to examine the other alternatives that are available first.

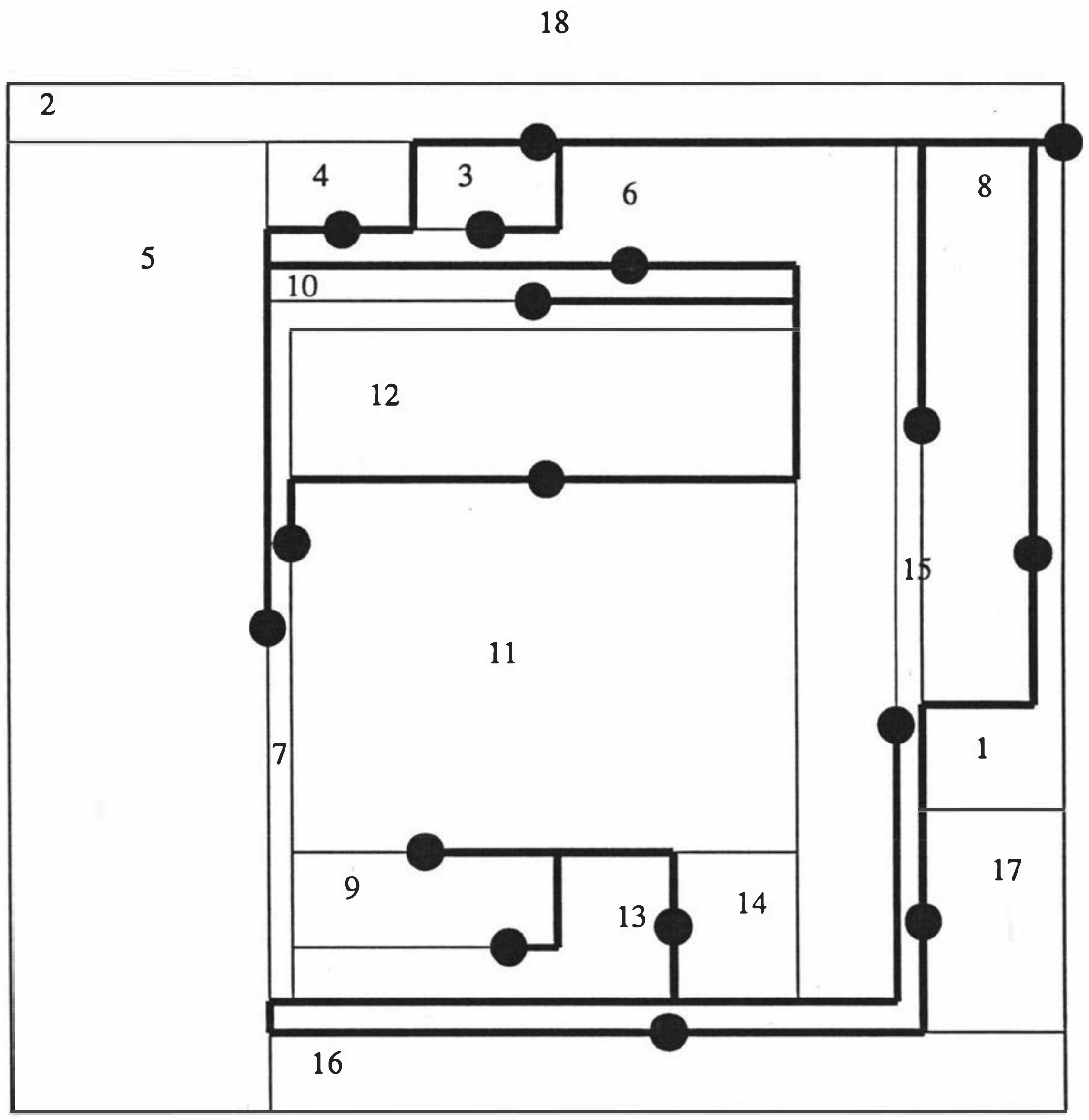


Figure 11.7: First Decomposition Layout for the Manufacturing Plant Problem

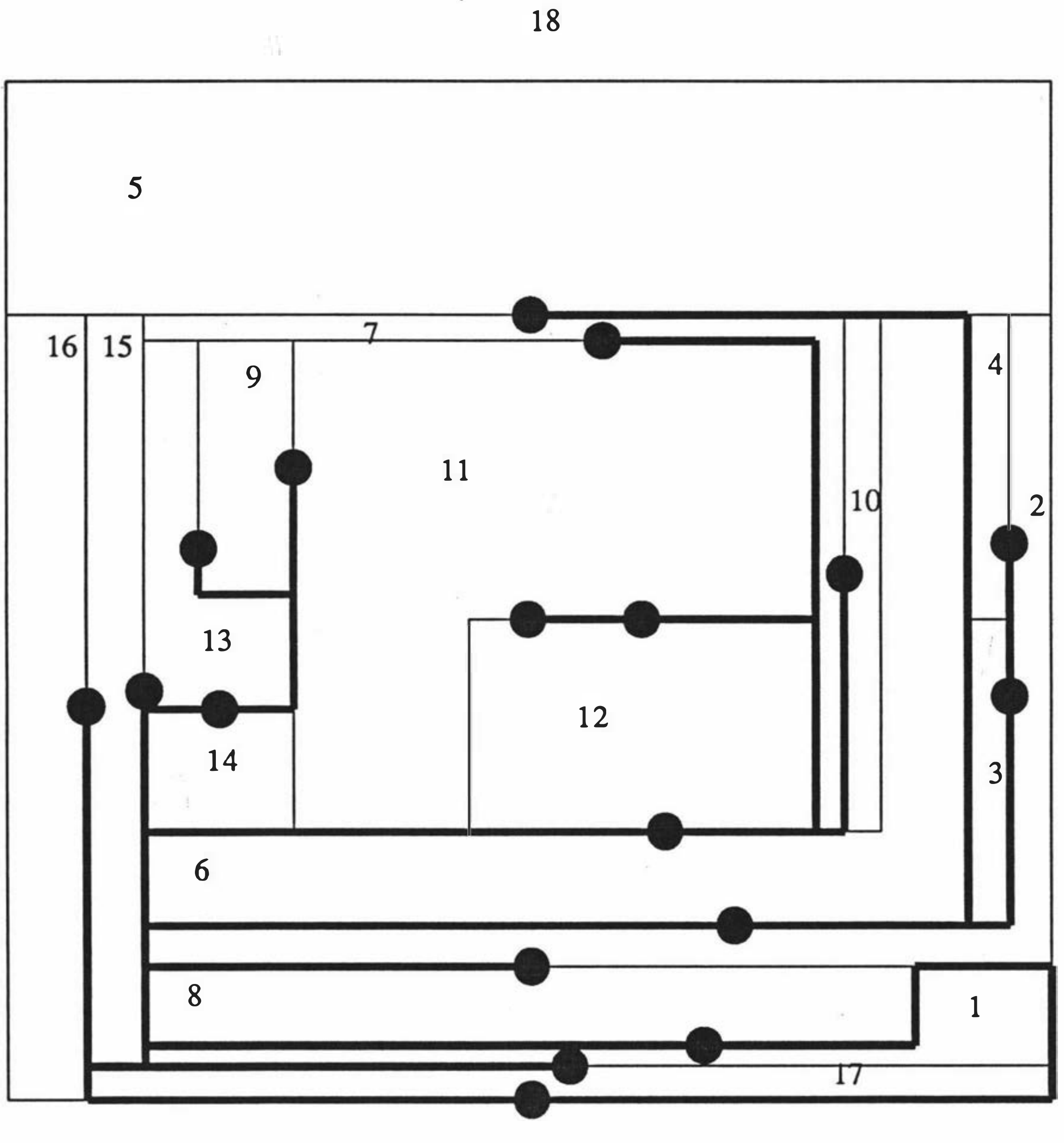


Figure 11.8: Second Decomposition Layout for the Manufacturing Plant Problem

Facility	Problem	Possible Remedy
3	Elongation	Repositioning to attain as many of the adjacencies (2,3), (3,11) and (3,12), the most important being the first two. Possible positioning in 12 adjacent to 2 and 11, but will create a faultline. Alternatives require either (2,3) or (3,11) to be not met.
4	Elongation	Repositioning such that adjacency (2,4) is maintained, with as many of (4,6), (4,7) and (4,11) as possible also being attained. Unable to satisfy all 4 adjacencies, but positioning in 12 adjacent to 2 and 6 is possible. Alternative of positioning in 12 adjacent to 7 as well creates a faultline.
9	Elongation	Has many small adjacencies of which a overall benefit of 7 is currently realised. Positioning adjacent to 13 allows increase of at least 1 to the overall benefit. Positioning in 8, deleting zero adjacency (8,18) retains dimensionalisability. Possible alternatives require deletion of highly weighted adjacencies.
10	Elongation	As for 9 has many small adjacencies of which only a total benefit of 5 is currently derived. Flexibility allows repositioning adjacent to any one of 5,6,7 or 12 at a cost of at most 2.

Table 11.5: Considerations for the modification of the Tiling Algorithm Layout (Figure 11.4)

The Tiling Algorithm layout with a worst-case of T shaped facilities (Figure 11.3), suffers from unnatural elongation. Even those facilities which are rectangularly shaped would require a substantial modification to ensure the regularity of this layout. Further the adjacency benefit of this layout is dominated by the adjacency benefit derived from the Tiling Algorithm where we only allow rectangular facilities.

The Tiling Algorithm layout with only rectangular facilities (Figure 11.4) requires the further development of facilities 3,4,9 and 10. These facilities can be made more regular by a simple area perturbation, or via an analysis of the zero weight adjacencies in the benefit matrix. Table 11.5 provides some insights into the directions these modifications might take.

From Table 11.5 we see that the repositioning of facility 4 is quite structured, and so is performed first (note we do not choose the alternative remedy as we initially want to retain dimensionalisability), in case it becomes unavailable as a result of other modifications. Facilities 9 and 10 are dealt with next due to their as they retain dimensionalisability. Facility 3, whose only realistic modification results in the loss of dimensionalisability, is performed last, as once dimensionalisability is lost modifications become much more difficult. Facility 4 is repositioned within facility 12, adjacent to facilities 2,3 and 6. This removes the zero weight adjacencies (2,12), (1,4), and (4,18), replacing them with other zero weight adjacencies, and the adjacency (4,6) of weight 10; hence this change increases the regularity, retains dimensionalisability, and increases the overall benefit by 10! Facility 9 is repositioned in facility 8 adjacent to facilities 13, 17 and 18 for a total decrease in adjacency of 1; meanwhile facility 10 is repositioned in facility 12 between facilities 4 and 12, resulting in an increase in benefit of 4. As we have already stated, repositioning of facility 3 creates a faultline which threatens the adjacency between 11 and 12. However the area specifications enable us to create the layout without violating that adjacency. This loss of dimensionalisability is not a concern if we are content with this as our final layout; however further modification may be difficult in the presence of the newly created faultline. This modified layout is provided in Figure 11.9, with regularity, transportation and adjacency values provided in Table 11.8.

The Spanning Tree layout of Figure 11.5 only requires the examination of facilities 3 and 4 as seen in Table 11.6. Placement of facility 3 within facility 6, adjacent to facility 8 provides an increase in benefit of 5 (through the adjacency (3,12)), while placement of facility 4 also in facility 6, adjacent to facility 3 has a net increase of 10. This improved layout is provided in Figure 11.10, with the appropriate data values in Table 11.8.

Upon the first Decomposition layout (Figure 11.7) we can perform adjacency swaps of zero net change to create rectangular facilities 1 and 13, yet there is little which can be achieved with facilities 6,7 and 15, due to their large number of non-zero material flows with other facilities.

The second Decomposition layout (Figure 11.8) suffers from a similar problem with the elongation and L-shape of a number of facilities, as shown by Table 11.7. From Table 11.7, we see that remedies applied to facilities 11 and 13 have no net effect on the overall benefit, and hence are performed first; indeed, facility 12 is

Facility	Problem	Possible Remedy
3	Elongation	Repositioning to attain as many of the adjacencies (2,3), (3,11) and (3,12); both (2,3) and (3,11) cannot be met without violating (8,12) and creating a faultline. Repositioning in 6 adjacent to 2 and 12 results in an increase in overall benefit of 5. Positioning in 5 adjacent to 2 will not reduce the elongation significantly without requiring 5 to become *L* shaped. Positioning in 17 adjacent to 2 is a valid alternative, but does not increase the overall adjacency benefit.
4	Elongation	Repositioning such that adjacency (2,4) is maintained, with as many of (4,6), (4,7) and (4,11) as possible also being attained. Unable to satisfy all 4 adjacencies, but positioning adjacent to 2 and 6 is possible, by placement of 4 in 6 while retaining (4,6). Alternatives are similar to those for facility 3.

Table 11.6: Considerations for the modification of the Spanning Tree layout (Figure 11.5)

now adjacent to facility 7 on two sides, hence this adjacency will be most likely retained. Modifications to facility 7 are now much cheaper, but significant sacrifices are made in modifying facilities 2 and 6. Final simple changes (including swapping facilities 3 and 4) create a rectangular layout shown in Figure 11.11. This results in a very practical layout, but at a decrease in adjacency of 29. The biggest loss is the inability to retain the adjacency (2,15).

In the absence of alternative possibilities, we return to the Deltahedron layout (Figure 11.2). We do not produce an extensive table of the possible remedies, as it should be becoming clearer by examining the layout and the benefit matrix, as to which options are available. The *L*-shape of figures 6 and 7 can be removed at a cost of 10 to the adjacency benefit, by repositioning the block (3,4) into facility 5, adjacent to facilities 2 and 6, and assigning 5 as the placement host of the block (9,10). Facility 5 becomes *T*-shaped, but this is removed by deletion of the adjacency between facilities 5 and 6. This leads to a solution of benefit only 1 greater than that of Figure 11.11, yet we must still deal with the elongation of facilities 1 and 17, and the *L* shapes of facilities 12 and 15. We do not pursue this further, as it is unlikely that we will obtain a realistic alternative.

Therefore we are left with the three modified layouts of Figures 11.9 - 11.11

Facility	Problem	Possible Remedy
1	L-shape, elongation	Reposition 8 which currently offers overall benefit of 18, adjacent to either 2 or 11 (for a maximum decrease of 3) while attempting to attain adjacencies with as many as possible of 12,13,14 and 15. Alternative involve the current placement of facility 8 with either $(1,15) \rightarrow (8,17)$ or $(1,2) \rightarrow (8,18)$, only the first of which is viable with respect to adjacency benefit.
2	L-shape, elongation	Deletion of either adjacencies with (6,8,15) or (4,5). Neither is particularly promising so we await further alternatives through the application of other modifications.
6	L-shape	Loss of adjacencies (4,6) and (5,6) at a cost of 15. Alternative of losing adjacency with 7,12 and 15 is more costly.
7	L-shape, elongation	Loss of (6,7) and (7,12) is costly, but modification on 11 first (as we will see) reduces the cost of this significantly. Alternative is to lose (7,15) which is also costly.
11	L-shape	Reposition 12 so that 12 covers the 2-joint currently covered by 11 improves regularity with no adjacency changes.
13	L-shape	Adjacency swap $(7,13) \rightarrow (9,15)$ at zero cost.

Table 11.7: Considerations for the modification of the Decomposition layout (Figure 11.8)

	Decomposition	Spanning Tree	Tiling Algorithm
Adj Benefit	1205	1134	1190
Ave C_i	4.00	4.00	4.00
Max C_i	4.00	4.00	4.00
Ave $a_i/\bar{S}_i$	0.304	0.434	0.239
Min $a_i/\bar{S}_i$	0.034	0.045	0.035
Ave $a_i/\bar{R}_i$	1.000	1.000	1.000
Min $a_i/\bar{R}_i$	1.000	1.000	1.000
Ave $4\sqrt{a_i}/P_i$	0.701	0.784	0.701
Min $4\sqrt{a_i}/P_i$	0.356	0.405	0.364
Transport Cost	152754.35	161934.04	157273.676
MHS Length	701.74	739.41	827.57

Table 11.8: Adjacency, Regularity and Transportation Data of the Modified Layouts for the Manufacturing Plant Problem

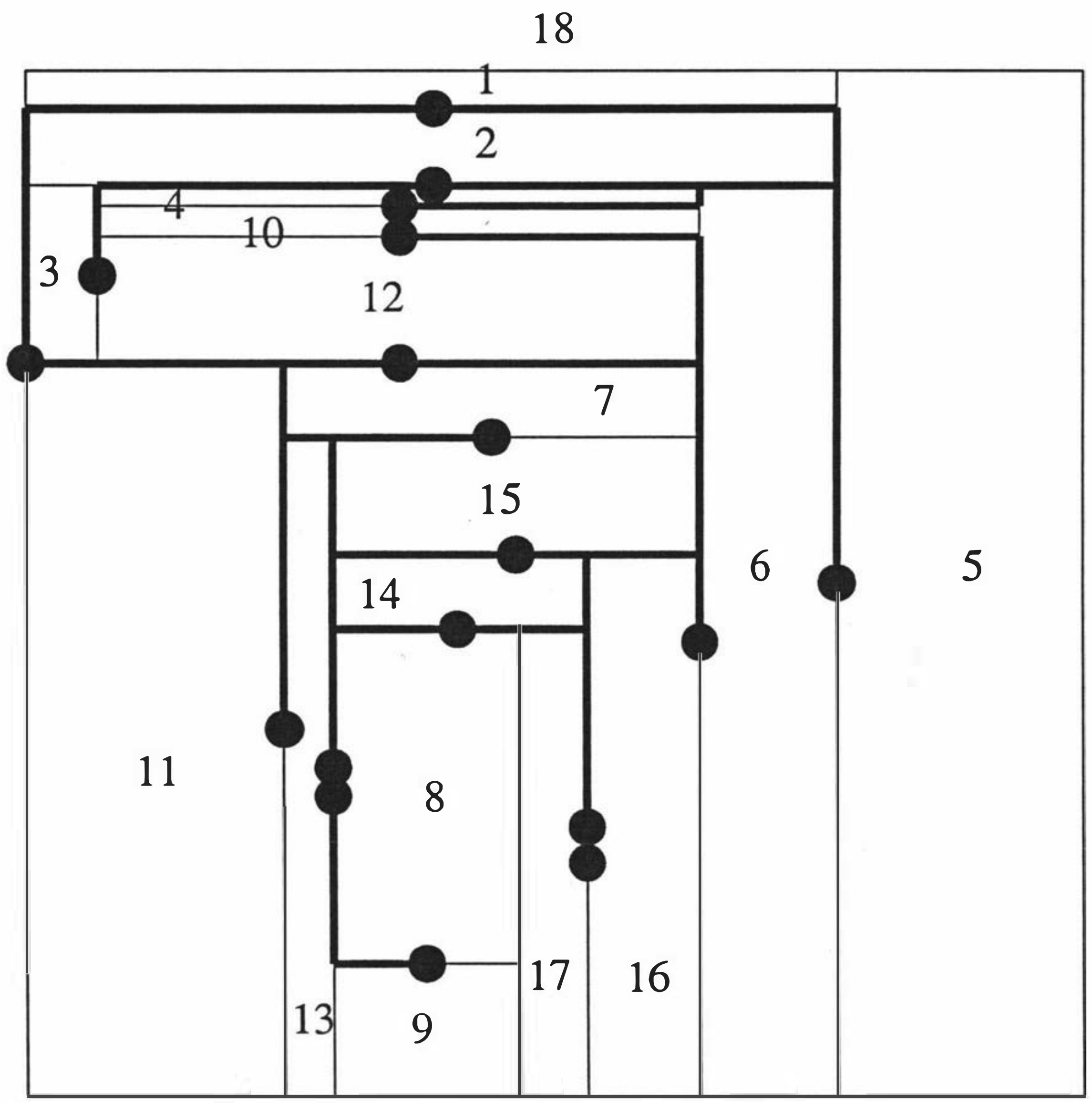


Figure 11.9: Modified Tiling Algorithm Layout for the Manufacturing Plant Problem

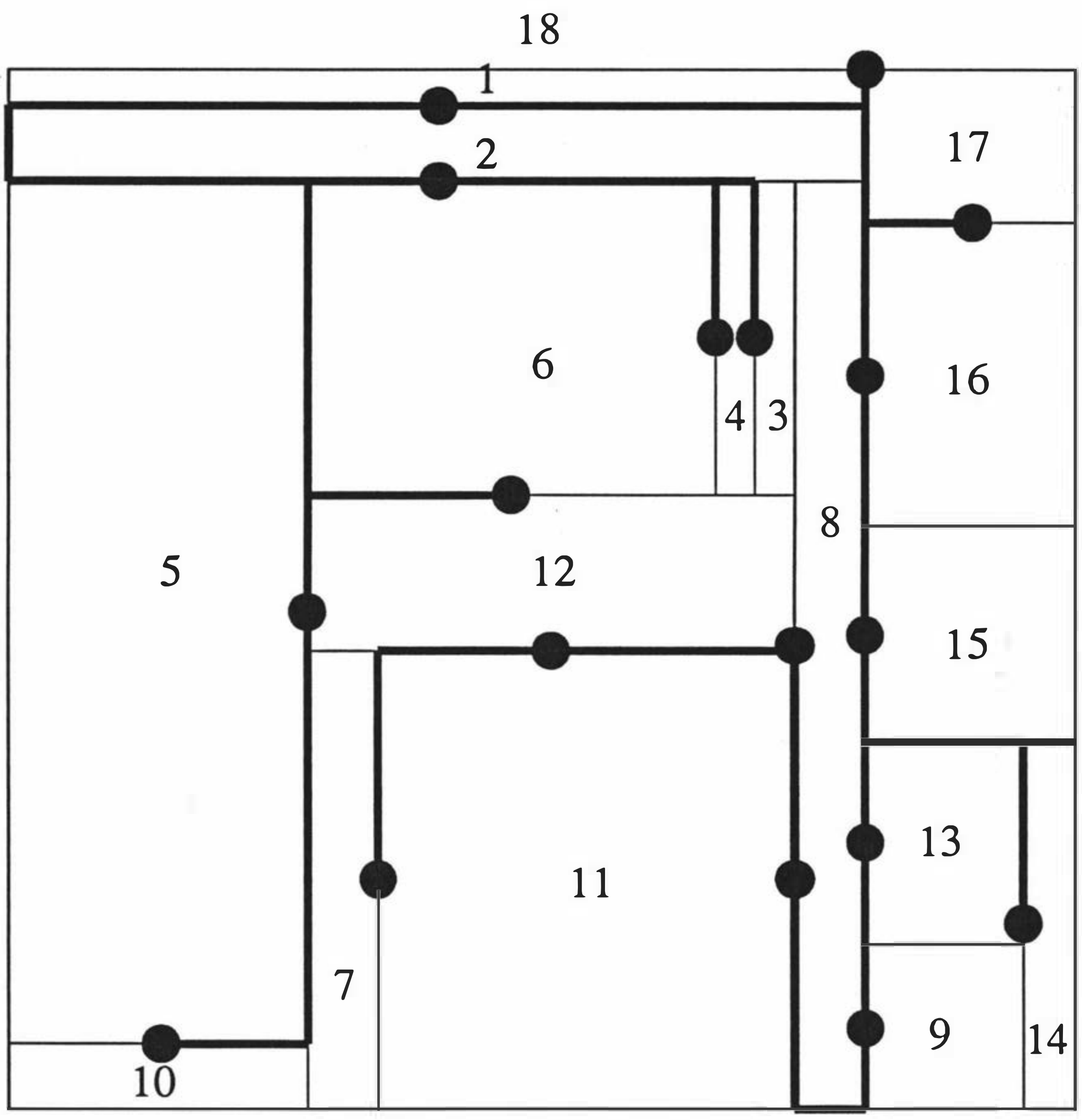


Figure 11.10: Modified Spanning Tree Layout for the Manufacturing Plant Problem

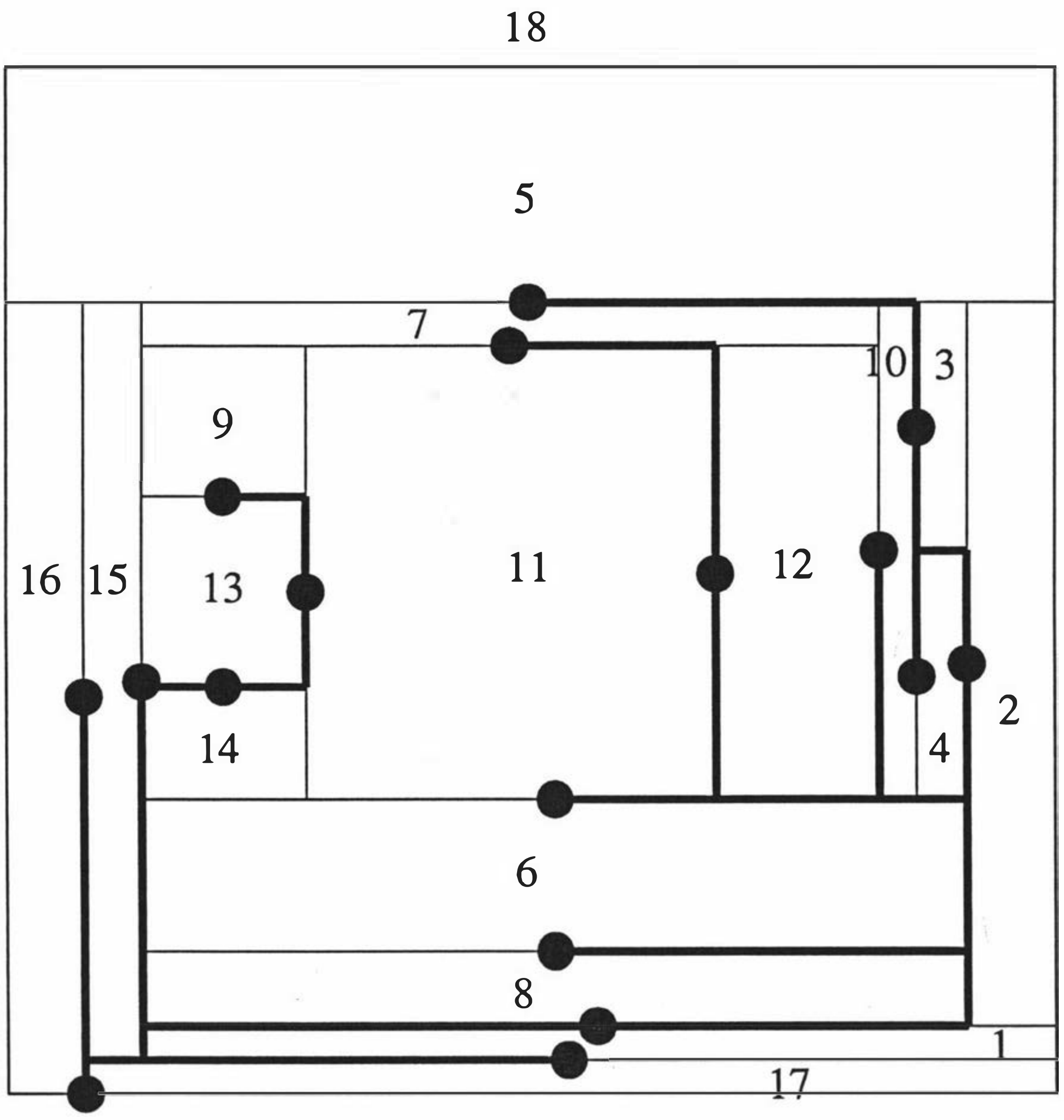


Figure 11.11: Modified Decomposition Layout for the Manufacturing Plant Problem

which provide a good set of alternative layouts (we could also retain the initial Spanning Tree layout (Figure 11.5) which has the best transportation cost). The three layouts are compared in Table 11.8, where we see that the Spanning Tree layout provides the best regularity values, but inferior adjacency benefit and transportation cost. The remaining two layouts are comparable with regard to regularity, but the Decomposition layout has the edge in both adjacency benefit, and transportation cost. Before a final decision could be made on which of these layouts would be the most practical, we would require the evaluation of the dimensions of the machines to be placed in each facility. For instance a long turret lathe (facility 6) could probably not be accommodated within the Spanning Tree layout (Figure 11.10), while assembly (facility 15) may not be efficient (due to elongation) within the Decomposition layout (Figure 11.11).

There exist other possible layouts which could be generated by Decomposition, and by Deltahedron; further, due to the large degree of non-zero elements in the benefit matrix, we can generate a vast number of MPGs of weight 1234 (the best we generated), each with different substructures and characteristics. The enumeration of all possibilities is in general impractical, however we can implicitly generate a number of these alternatives as we consider adjacency swaps within the layout which generate zero net change to the adjacency benefit.

11.2 Case Study 2: A Small Job Shop

The second problem which we will study is a 17 facility problem given on *pg* 134 of Francis, McGinnis and White [36]. The problem given is for a small job shop, with adjacency benefits determined by the A,E,I,O,U,X scheme as shown in Table 11.9, with functional and area data in Table 11.10.

Unlike the example of Case Study 1, the data is not expressed as transportation cost data, or as numerical adjacency benefit scores. It is difficult to place a numerical value on each of the A,E,I,O,U and X benefits, however as we have concentrated on numerical data throughout this thesis, it would appear prudent to attempt to determine numerical values for the adjacencies. We assign values to the classes as given by Table 11.11. We do not in general deal with negative adjacency benefits, and hence we add 256 to each benefit, to obtain the adjusted benefit scores. This does not change the objective function, simply adding a fixed value to it, as shown

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	U	O	A	X	U	A	X	I	U	E	I	U	U	X	O	U
2		O	U	E	E	U	O	O	U	E	I	U	I	X	O	U
3			U	I	U	U	O	U	U	U	O	U	O	O	O	U
4				U	E	E	O	U	U	U	U	U	U	U	U	U
5					I	U	E	I	U	O	O	U	I	O	O	U
6						U	I	U	U	U	U	U	U	U	U	U
7							O	U	U	U	U	U	U	U	U	U
8								U	I	I	E	A	U	I	I	U
9									U	A	U	U	U	U	U	A
10										U	U	A	U	U	U	A
11											U	U	U	U	U	U
12												U	U	U	U	U
13													U	U	U	U
14														U	O	U
15															I	U
16																U

Table 11.9: Adjacency Benefits for Small Job Shop Problem

Facility	Function	Area	Facility	Function	Area
1	Foundry	2000	10	Shipping	3000
2	Press	3000	11	Raw Material Storage	2500
3	Drill	1000	12	WIP Storage	2000
4	Grind	500	13	Finish Goods Storage	3000
5	Machine	3000	14	Maintenance	500
6	Weld	500	15	Offices	2000
7	Tumble	500	16	Locker Room	1000
8	Assemble	4000	17	Exterior	-
9	Receiving	2000			

Table 11.10: Function and Area of Facilities for the Small Job Shop

Benefit	Score	Adjusted Score
A	64	320
E	16	272
I	4	260
O	1	257
U	0	256
X	-256	0

Table 11.11: Benefit Scores for the Small Job Shop Problem

by Giffin [39]. Indeed all values shown relating to the benefit scores have this fixed cost removed to provide the true benefit. The adjacency graph derived from TESSA coupled with a Tabu Search routine is shown in Figure 11.12, with total adjacency benefit of 596, while the Deltahedron MPG is given in Figure 11.13, and has overall adjacency of 618. The two interesting points about the MPGs, is that the TESSA generated MPG is also Deltahedron generateable, and the MPG generated from Deltahedron has a better overall benefit than the TESSA MPG, hence we will use the Deltahedron generated MPG. The upper bound on the value of any MPG from this problem data is 648, involving all of the A, E and I adjacencies, the remainder being O adjacencies.

From the Deltahedron MPG we generate a layout via each of the techniques we have discussed for given MPGs, as well as generating (from the benefit matrix), the Tiling Algorithm and Spanning Tree Algorithm layouts. The adjacency benefits, regularity scores, and MHS costs are given in Table 11.12. We see from this table that the Tiling Algorithm with a worst room shape of a rectangle provides the best transportation cost, while the best regularity values are provided by the Spanning Tree Layout. Of the layouts designed from the Deltahedron MPG, the Contraction Algorithm does surprising well with respect to the transportation cost. Deltahedron however provides the best regularity scores for its layout. The layouts generated by each of these designs are shown in Figures 11.14 - 11.20.

We can see from the Figures that no one layout stands out as being very good in all respects. The VSA, CA and SIMPLE layouts are deceptive, as there are walls which cannot be seen as they are too close too each other. For example, facilities 10 and 11 do not appear to be adjacent in the VSA layout (Figure 11.14), 14 and 6 do not appear to be adjacent in the CA layout (Figure 11.15), and in the SIMPLE

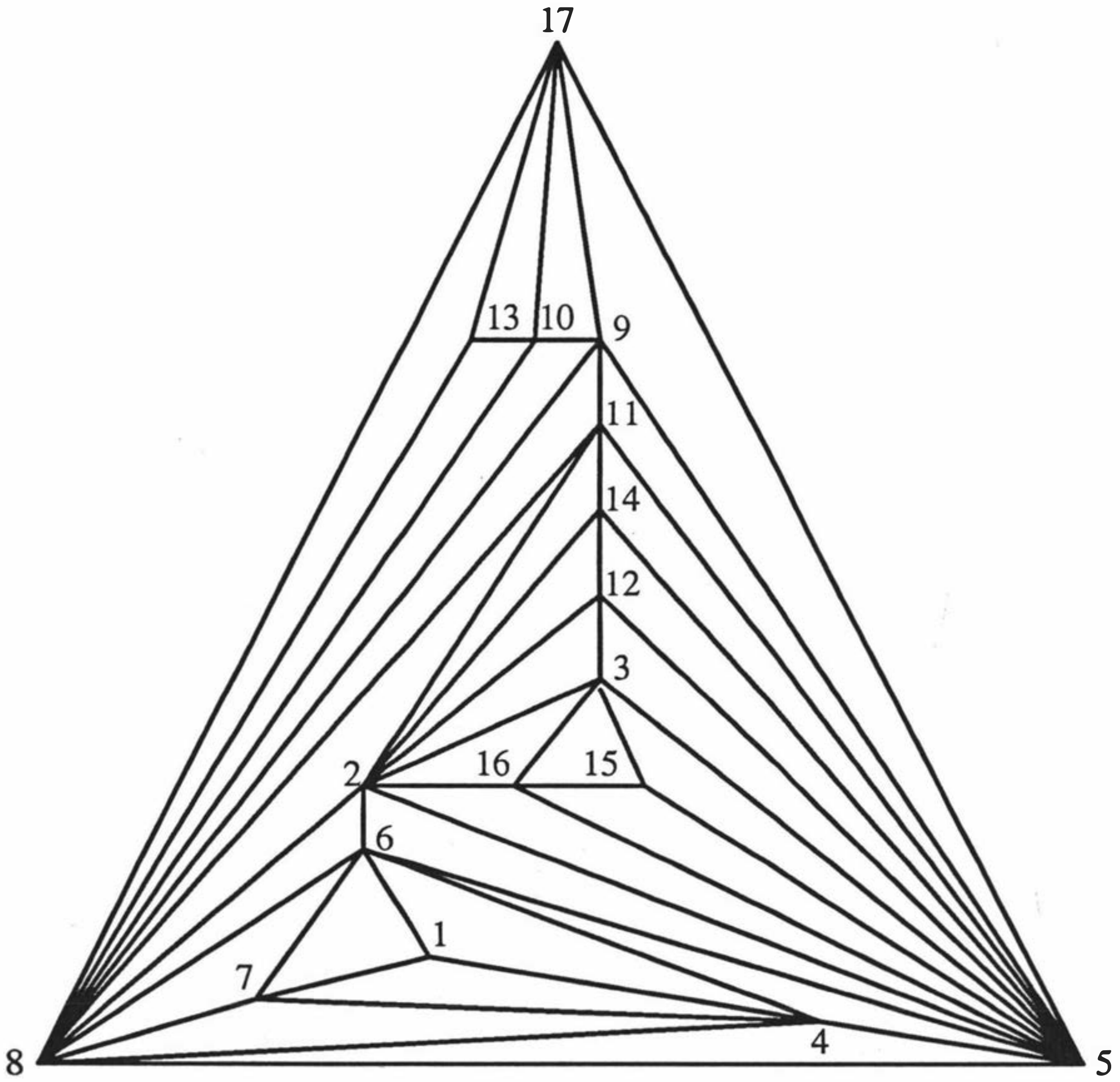


Figure 11.12: MPG generated by TESSA for the Small Job Shop Problem

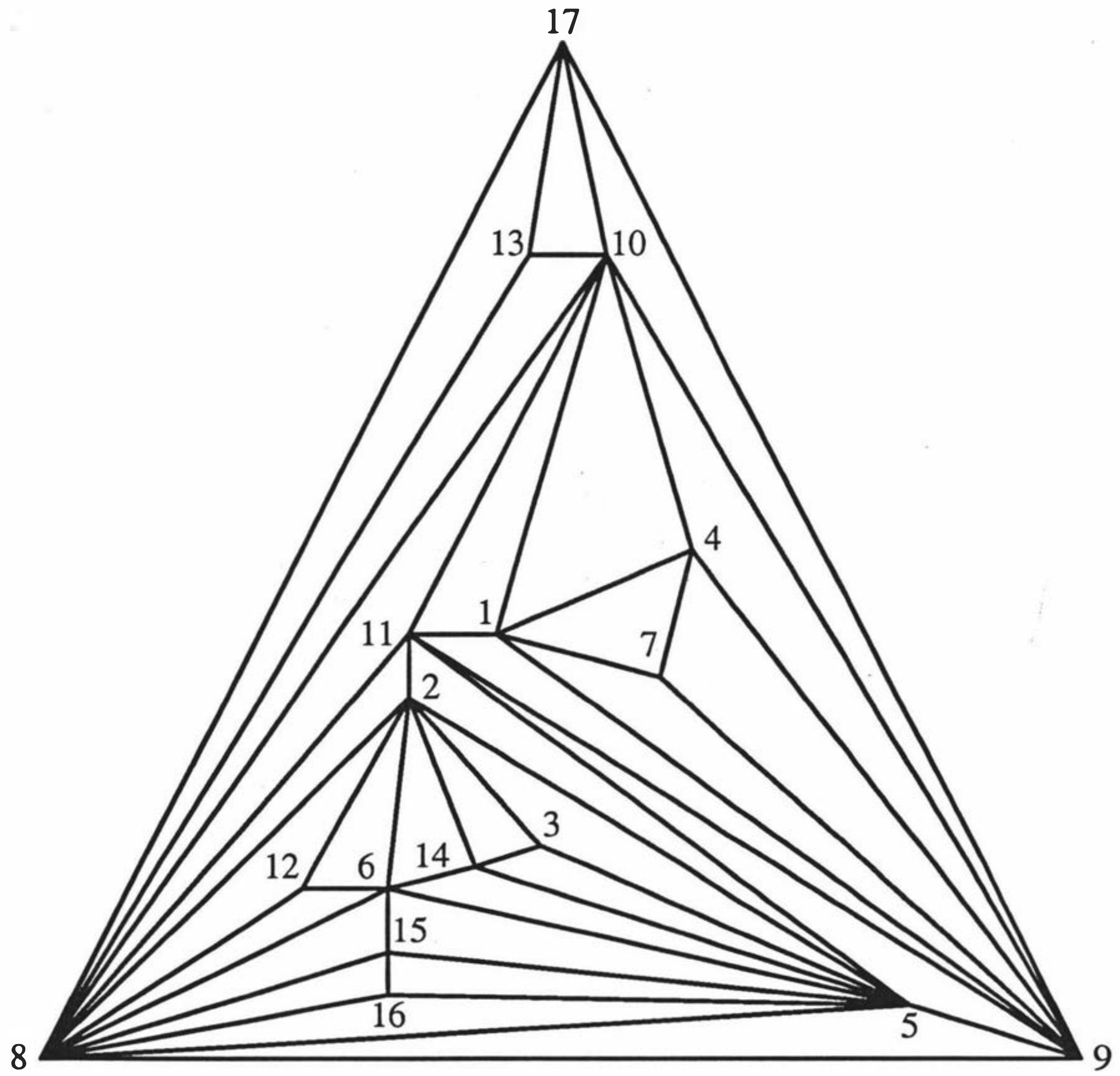


Figure 11.13: MPG generated by Deltahedron for the Small Job Shop Problem

Measure	VSA	CA	SIMPLE	Deltahedron	TA(T)	TA(I)	ST
Adj Benefit	618	618	618	618	602	603	599
Ave C_i	9.25	11.50	14.88	4.88	5.00	4.00	4.00
Max C_i	20.00	26.00	60.00	8.00	8.00	4.00	4.00
Ave $a_i/\bar{S}_i$	0.217	0.279	0.153	0.200	0.202	0.171	0.402
Min $a_i/\bar{S}_i$	0.069	0.054	0.024	0.029	0.016	0.019	0.083
Ave $a_i/\bar{R}_i$	0.558	0.582	0.501	0.741	0.636	1.000	1.000
Min $a_i/\bar{R}_i$	0.131	0.114	0.024	0.091	0.016	1.000	1.000
Ave $4\sqrt{a_i}/P_i$	0.561	0.591	0.458	0.598	0.509	0.617	0.853
Min $4\sqrt{a_i}/P_i$	0.235	0.210	0.126	0.301	0.128	0.270	0.532
Transport Cost	129213.85	106667.58	116755.43	127506.64	202498.44	105472.40	108106.02
MHS Length	1148.56	757.05	1084.23	981.13	1361.05	1091.19	842.39

Table 11.12: Adjacency, Regularity and Transportation Data of the Layouts generated for the Small Job Shop Problem

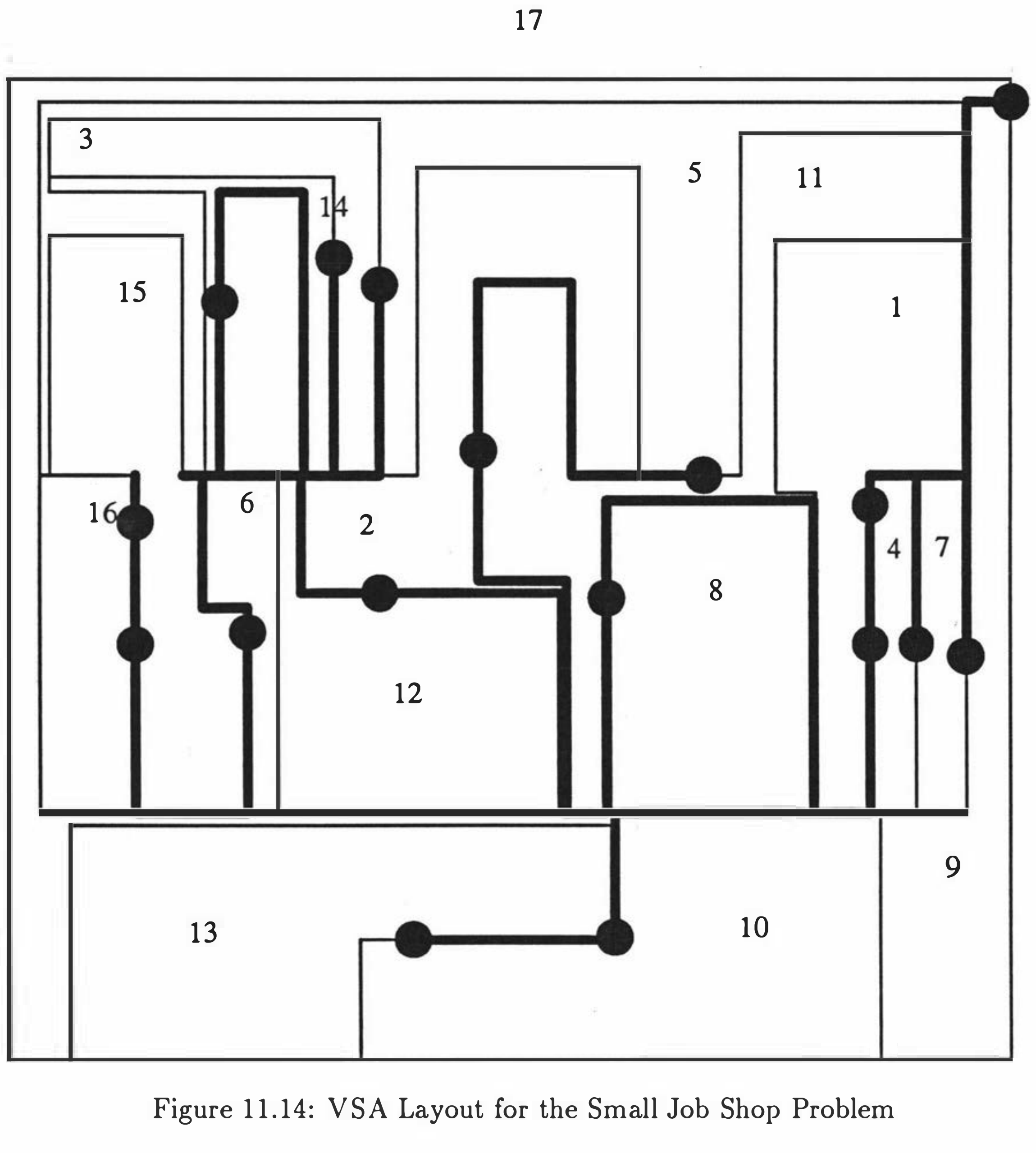


Figure 11.14: VSA Layout for the Small Job Shop Problem

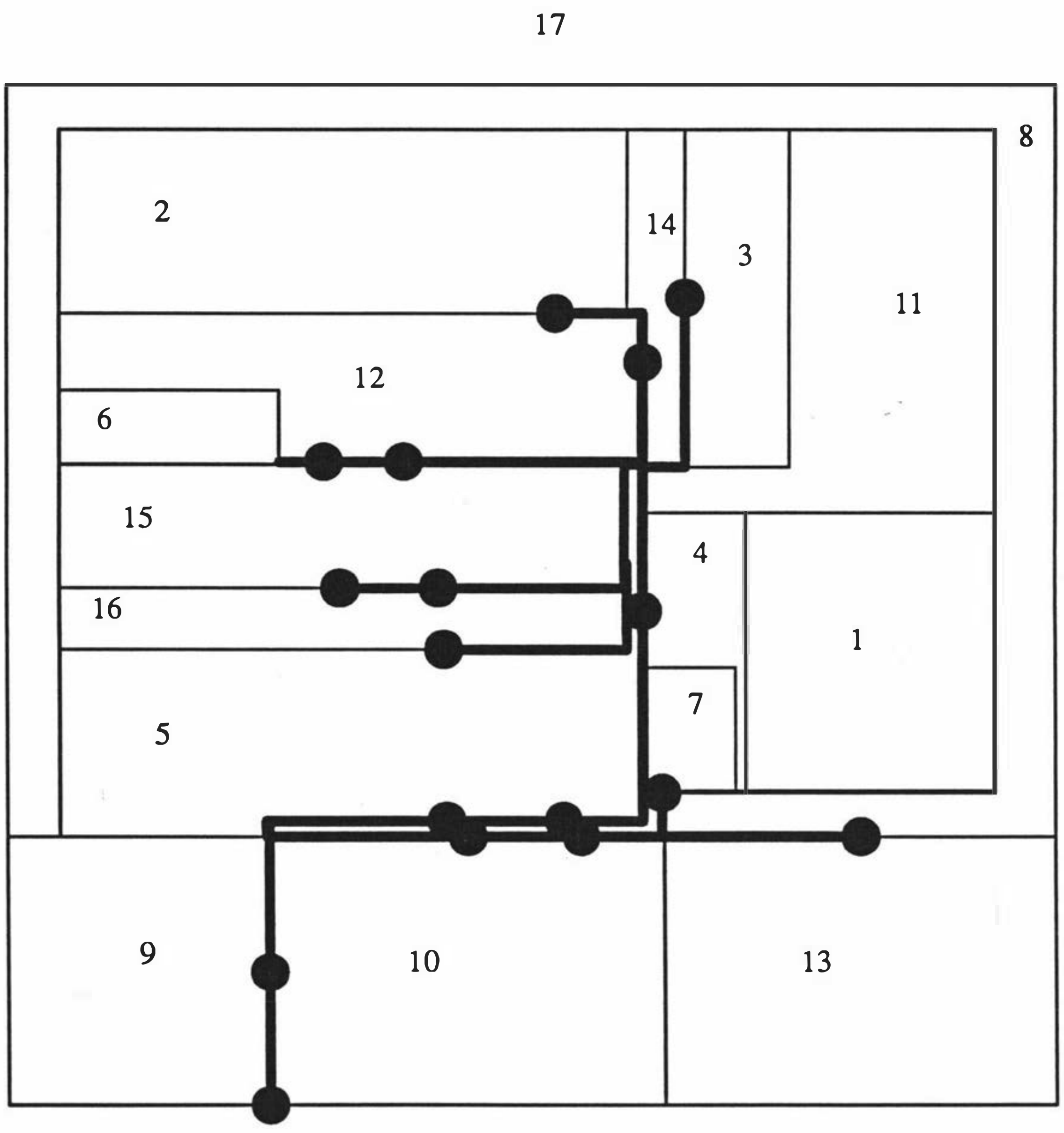


Figure 11.15: Contraction Algorithm Layout for the Small Job Shop Problem

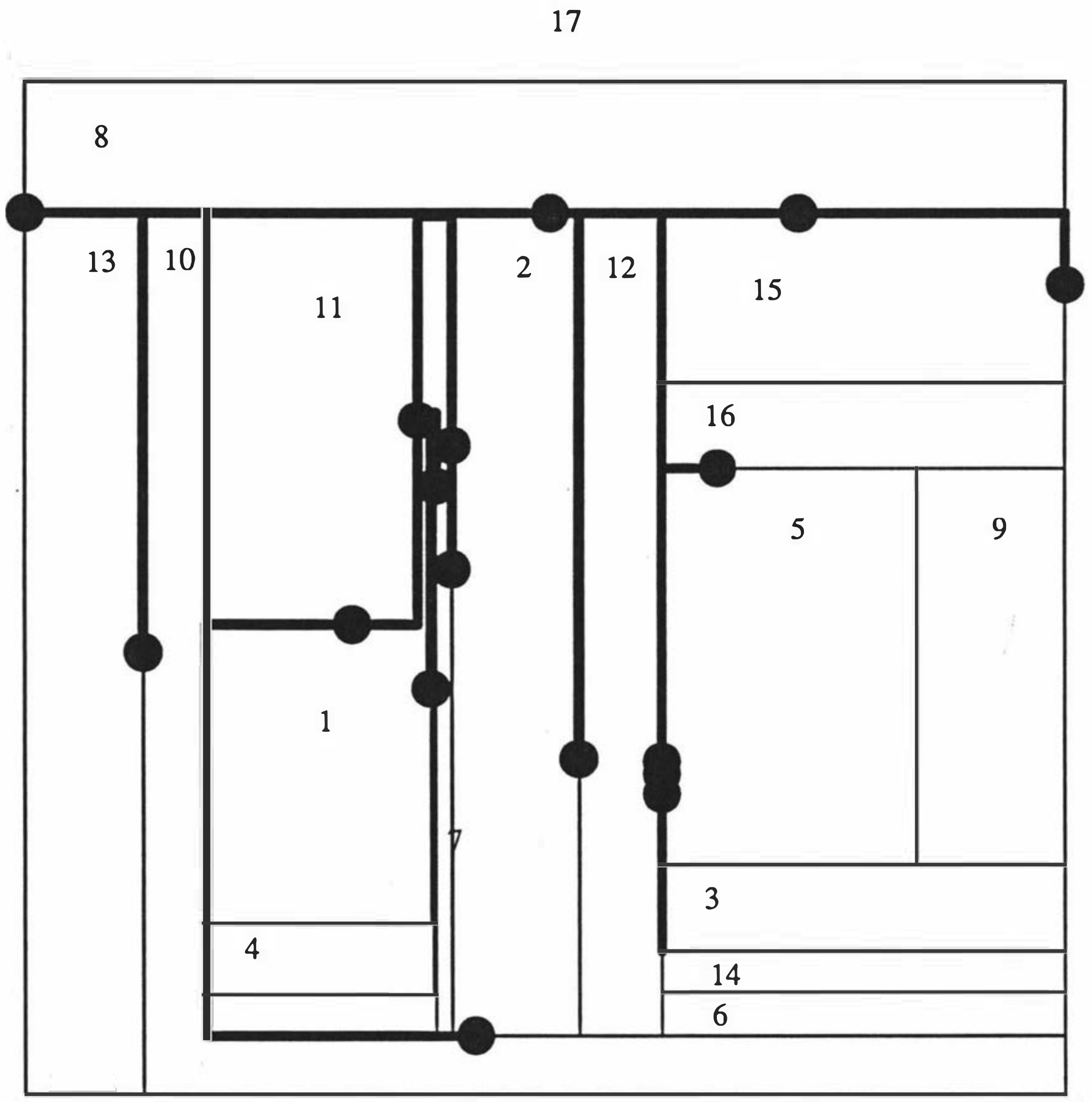


Figure 11.16: SIMPLE Layout for the Small Job Shop Problem

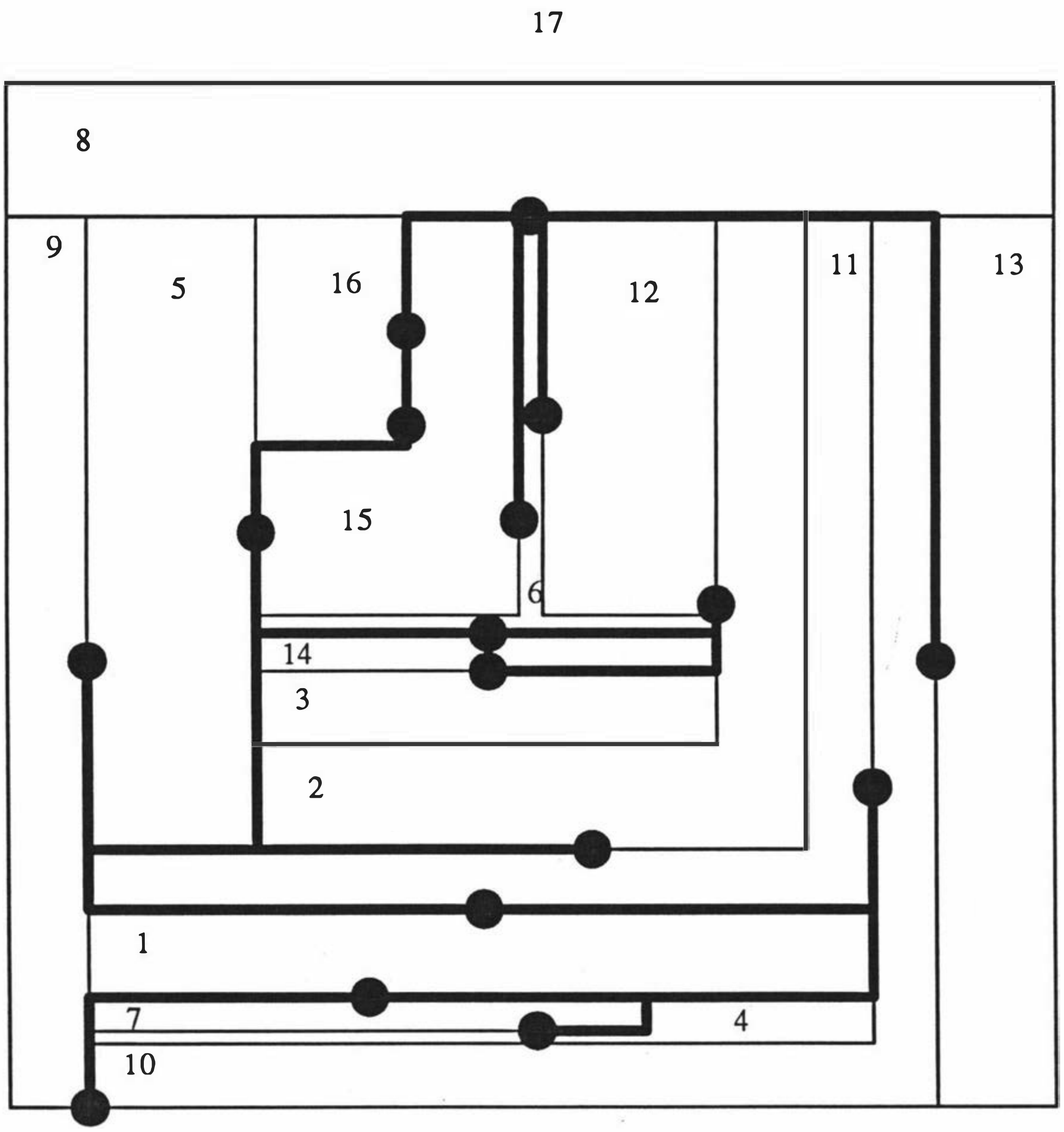


Figure 11.17: Deltahedron Layout for the Small Job Shop Problem

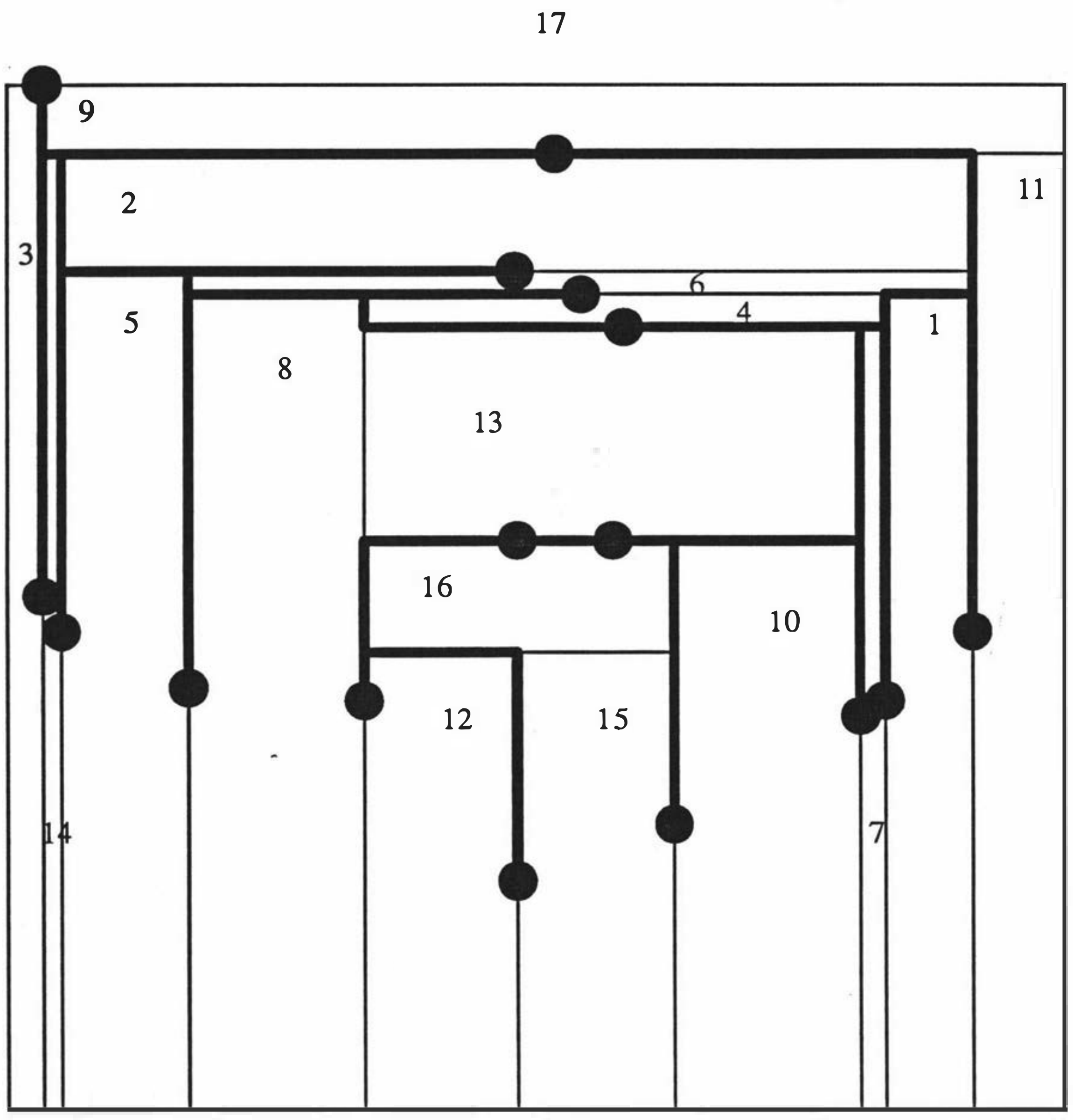


Figure 11.18: Tiling Algorithm(I) Layout for the Small Job Shop Problem

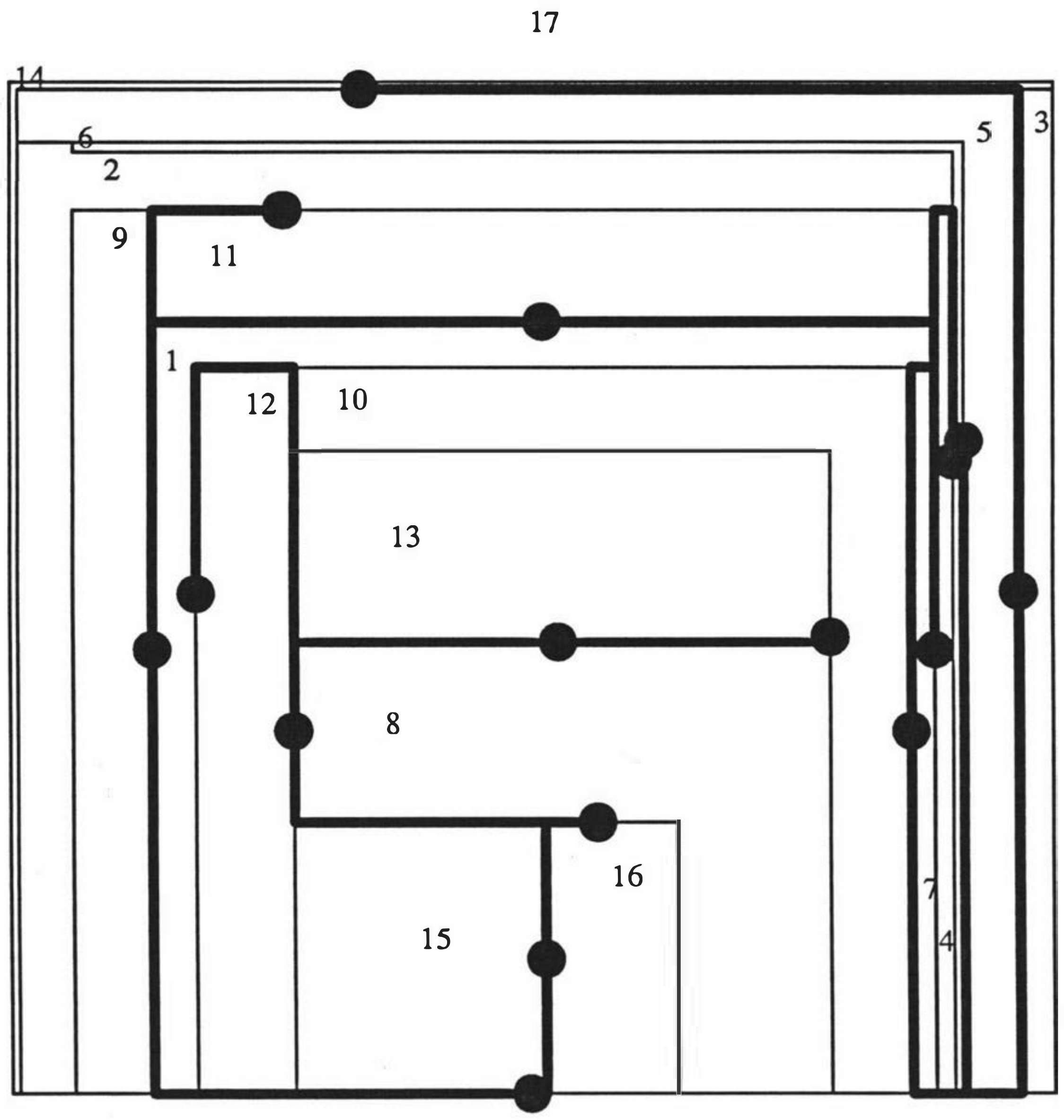


Figure 11.19: Tiling Algorithm(T) Layout for the Small Job Shop Problem

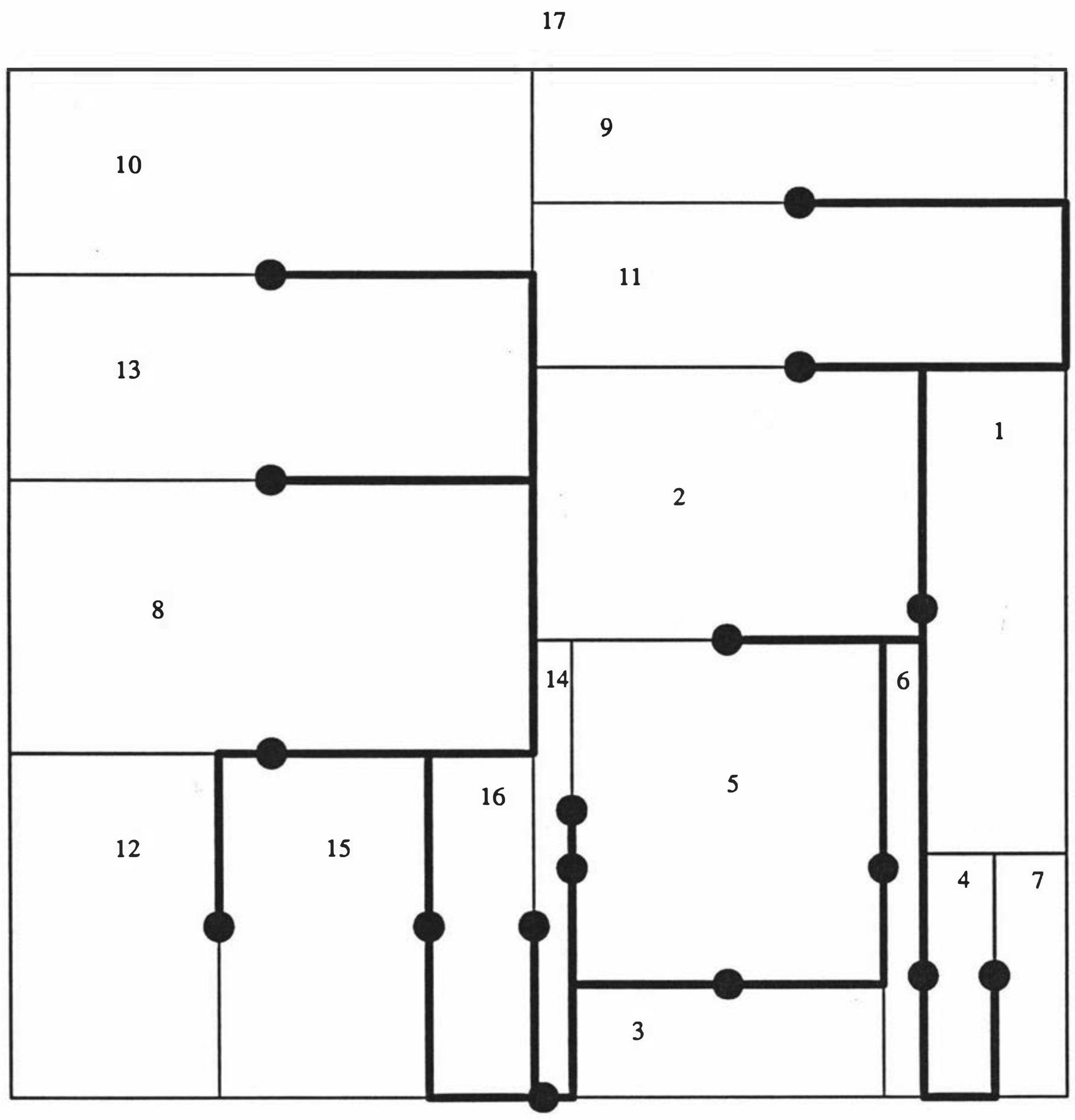


Figure 11.20: Spanning Tree Layout for the Small Job Shop Problem

layout (Figure 11.16), facilities 2 and 6 do not appear to be adjacent. The reality is that in fact all of these adjacencies are met, however the tentacles of one or both of these pairs of facilities are too narrow too be drawn. Note the maximum number of corners for each of these layouts in Figure 11.12. Therefore while these layouts *on paper* appear to perform reasonably well, they are not realistic alternatives. Of the other layouts, the Tiling Algorithm layout, with the worst facility shape of a T, (Figure 11.19) has an extremely high transportation cost; examining the actual layout, we see that facilities 4,6,7 and 14 are extremely elongated. The remaining three layouts appear promising. Note that since the highest weighted MPG is Deltahedron, decomposition of the MPG will not provide extra information, as we already know that the MPG is 4-decomposable.

Consider the Spanning Tree layout of Figure 11.20. The most concerning factor is the length-to-width ratio of facilities 6 and 14. The function of facility 6 is maintenance, while that of facility 14 is the weld. The elongation of facility 14 can be remedied by allowing the removal of the adjacencies $(2,14)$ and $(8,14)$, replacing them with $(5,8)$ and $(5,16)$. This allows facilities 5 and 8 to satisfy their essential adjacency (which was previously unmet) and creates a squarer facility 14. Furthermore this change results in an increase in adjacency of $E+O-I-U=13$. This creates a new Spanning Tree layout with adjacency benefit of 612. Note that this new layout can still be constructed by the Spanning Tree algorithm, as the underlying tree adjacency graph would replace edge (2, 14) with edge (5, 14). The elongation of facility 6 can also be remedied, by allowing the swap of adjacency $(6,17)$, for adjacency $(3,4)$. This change has a net effect of zero on the adjacency benefit, and provides a more regular facility 6. Note that this change could not be accommodated within the normal Spanning Tree routine. This perturbed Spanning Tree layout, given by Figure 11.21, with regularity, transportation and adjacency values given in Table 11.13, provides a valid alternative layout for the Small Job Shop, as each facility is able to function more efficiently.

The Deltahedron layout has similar concerns, in particular to the elongations of the facilities 4,6 and 7. Note that facility 6 is impractical in its current form. This applies to all the Deltahedron layouts which can be constructed from the Deltahedron MPG, using the alternative possibilities for the top facility, and insertions of subsequent facilities. Furthermore all of these possible layouts contain four L shaped facilities and facility 6 as a T shaped facility. In order to obtain a functional

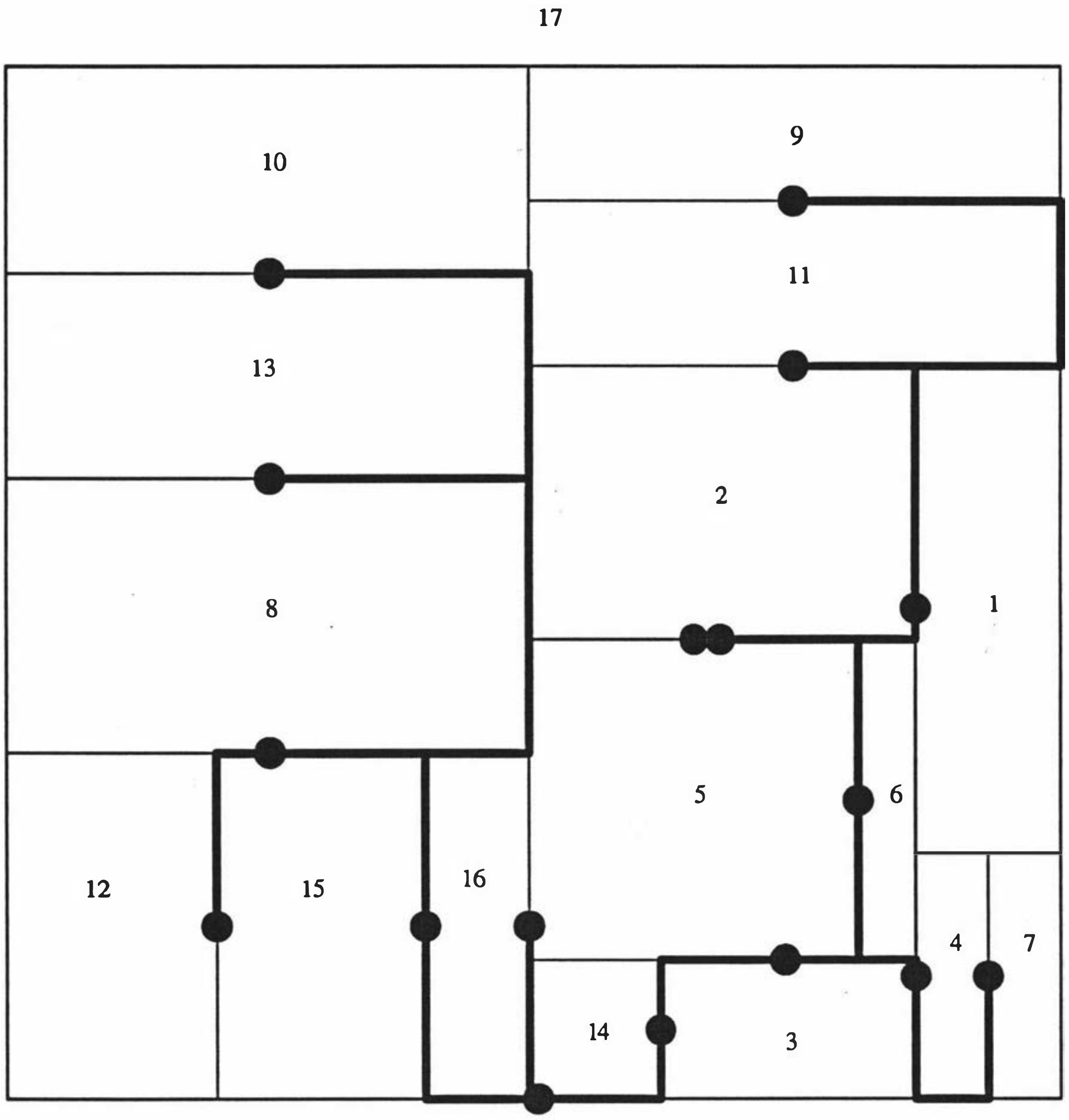


Figure 11.21: Modified Spanning Tree Layout for the Small Job Shop Problem

Measure	Modified ST
Adj Benefit	612
Ave C_i	4.000
Max C_i	4.000
Ave $a_i/\bar{S}_i$	0.477
Min $a_i/\bar{S}_i$	0.170
Ave $a_i/\bar{R}_i$	1.000
Min $a_i/\bar{R}_i$	1.000
Ave $4\sqrt{a_i}/P_i$	0.830
Min $4\sqrt{a_i}/P_i$	0.246
Transport Cost	109036.36
MHS Length	773.87

Table 11.13: Adjacency, Regularity and Transportation Data of the Modified ST Layout generated for the Small Job Shop Problem

layout using the Deltahedron layout of Figure 11.17, we would be required to swap $(6,8)$ for $(12,15)$, resulting in a decrease of four in the adjacency benefit. Facility 4 would also be required to lose its adjacency with facility 9, which would allow the adjacency $(7,10)$ to be met with no net change to the adjacency benefit. However we still have facilities 2, 10 and 11 L shaped after performing these perturbations. This is especially undesirable for facility 2, as the press would be required to fit within this facility, which is unlikely in its current form. Modifications to this layout to make facility 2 rectangular would result in one of its essential adjacencies, involving facilities 5 and 6, being lost. Alternatively, we could make facility 5 the placement host of the block of facilities $(3,6,12,14,15,16)$, creating an L shaped facility 5, however facility 5's function as the machine would become infeasible. Therefore although the Deltahedron layout initially appears to be promising, in hindsight the layout becomes impractical when functional considerations are examined.

The Tiling Algorithm layout with a worst case facility shape of a rectangle (Figure 11.18), exhibits again the elongation concerns of facilities 3,6,7 and 14. The area specifications of facilities 4,6,7 and 14 have been a consistent problem, due to their small size proportional to the remainder of the facilities' areas. Within the Tiling Algorithm layout the remedy to these elongations is to perform swaps such as swapping $(4,7)$ for $(1,13)$, in order to reduce the elongation of facility 7. This however removes the essential adjacency between facilities 4 and 7. Alternatively

Measure	Modified TA
Adj Benefit	597
Ave C_i	4.00
Max C_i	4.00
Ave $a_i/\bar{S}_i$	0.342
Min $a_i/\bar{S}_i$	0.066
Ave $a_i/\bar{R}_i$	1.000
Min $a_i/\bar{R}_i$	1.000
Ave $4\sqrt{a_i}/P_i$	0.803
Min $4\sqrt{a_i}/P_i$	0.481
Transport Cost	109152.31
MHS Length	800.15

Table 11.14: Adjacency, Regularity and Transportation Data of the Modified TA Layout generated for the Small Job Shop Problem

we swap $(7,17)$ for $(1,10)$, which has no net change and results in an even squarer facility 7. We must be careful in this instance that the dimensioning maintains the adjacencies, as the dimensionalisability of the layout is lost. Facility 3 has only one, currently unmet, important adjacency, and no essential adjacencies. This could result in the repositioning of facility 3, as it currently only contributes 1 to the overall adjacency benefit, within facility 5 say. This would result in an overall increase of $I + O - U - U = 5$. Further swaps to attain better facility length-to-width ratios for facilities 4,6 and 14 results in the layout of Figure 11.22 (which was actually created by the Spanning Tree Algorithm) as the structure of the layout after making these changes is amenable to it. Further we provide the transportation, regularity and adjacency values for this new layout in Table 11.14.

It would again be left to a decision planner to make the final decision between the two layouts of Figures 11.21 and 11.22, as their transportation and regularity values are very similar. It is likely that the modified Spanning Tree layout is likely to be chosen as it appears to be more robust, as facility 9 now appears a little elongated in the modified Tiling Algorithm layout. Further, the transportation costs are better for the Spanning Tree layout. It was perhaps disappointing that we were unable to implement the decomposition ideas for this problem, as in general that process creates the biggest impact. This was due to the stratified nature of the benefit matrix. By containing only six different values, the benefit matrix was able to generate a number of distinct, similarly weighted MPGs. This was exhibited

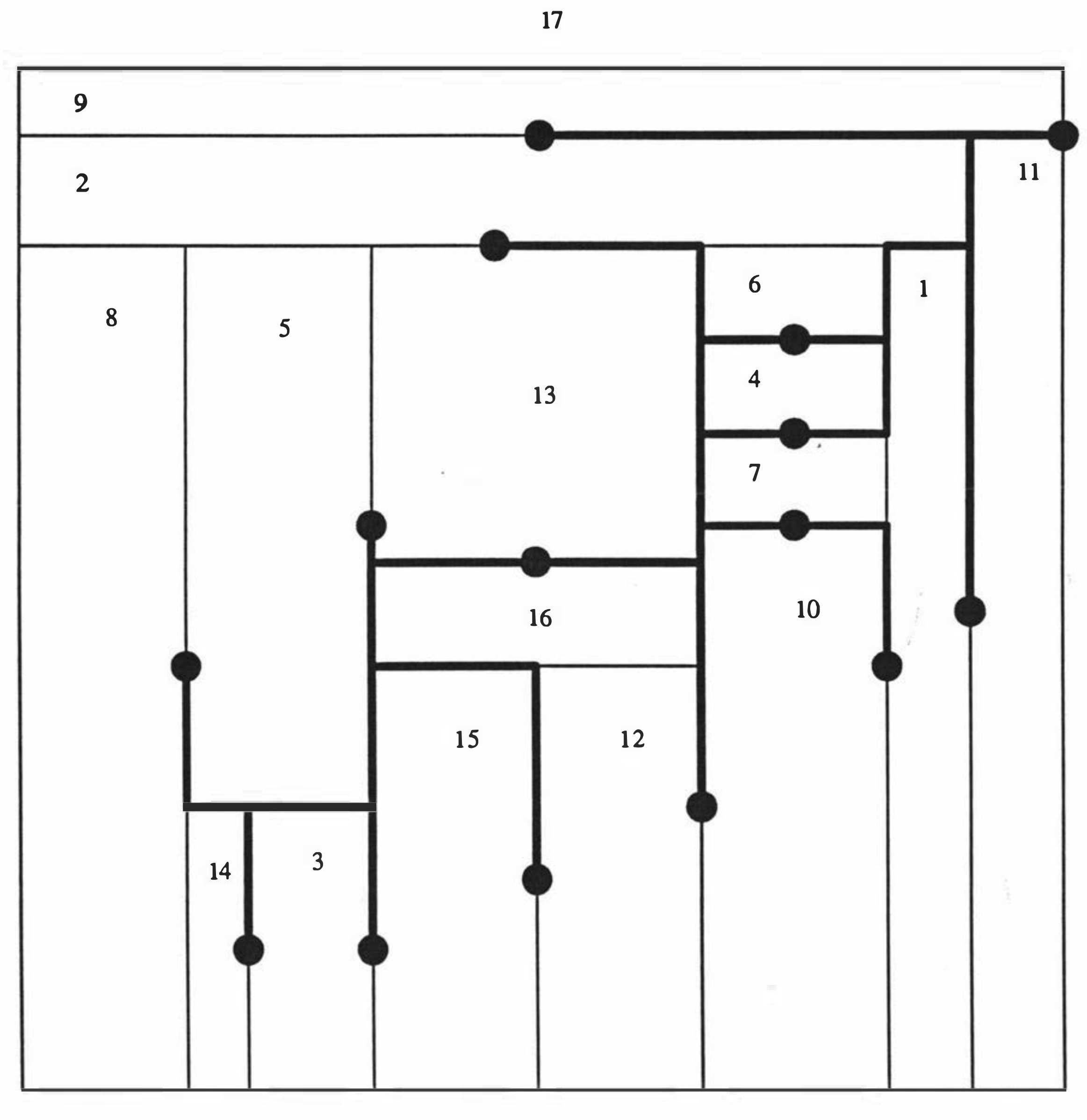


Figure 11.22: Modified Tiling Algorithm Layout for the Small Job Shop Problem

also when we were perturbing our final layouts. We were able to make a number of adjacency swaps which provided no net change to the overall adjacency benefit. It was therefore not surprising that we were able to obtain a Deltahedron MPG as the highest weighted MPG. The set of alternative layouts was disappointing, providing only two acceptable alternatives, and again this was due to the structure of the benefit matrix, as the essential adjacencies could not be compromised. It was no surprise really that the Spanning Tree layout algorithm provided the best layout, as the underlying tree was able to contain all the essential adjacencies, and from there we could construct a functional layout, of high weight. Further, following the construction of the layout we were able to obtain a number of important adjacencies to further boost the adjacency score, whilst maintaining the overall regularity of the layout.

Coda

In this section we have attempted to exhibit the process of determining a functional layout from the raw data. The problems chosen allowed us to make use of many of the techniques that we had previously discussed. This chapter has shown that the GTFLP techniques we have discussed are able to produce effective layouts, within a generic framework, where we concentrate on ensuring each facility, and the layout as a whole, performs efficiently.

In the next chapter we review the work that we have undertaken, reiterating the major components and contributions of this work. Further we offer some ideas where future research could be directed within the GTFLP.

Chapter 12

Conclusions and Areas for Further Study

In this thesis, we have examined the Graph Theoretic Facility Layout Problem. In particular we have investigated the construction of orthogonal geometric duals, or layouts, from defined adjacency and area specifications.

Chapter 4 reviewed and, in some instances, revised current methods for generating a layout. This chapter also included new techniques for the generation of the layout: the Vertex Splitting Algorithm, SIMPLE, and the Tiling Algorithm, as well as providing an initial characterisation of possible adjacency graphs. These algorithms were assigned to three classes, labelled, I, T and A, dependent on the guaranteed worst case facility shape of the method (where A stands for arbitrary). Chapter 5 provided two improvement techniques, Rectilinear Segment Reduction, and Linear Transformation, which attempted to enhance the effectiveness of layouts constructed by algorithms of Chapter 4. It was the applicability, or otherwise, of these techniques to *real world* situations which we wished to examine. This investigation initially considered the algorithms for generating a layout on a set of test problems.

A number of *regularity measures* were devised which quantifiably measured the effectiveness of each facility within a layout, and indeed the effectiveness of the layout as a whole. The results of the tests of Chapter 6, indicated that the best algorithm for generating a layout from a given adjacency graph with area specifications, was the Vertex Splitting Algorithm. However a comparison of all the methods of Chapter 4 indicated that a significant preference for adjacency benefit over regularity was required for these Class A methods to be effective in a general setting. Indeed we saw that algorithms such as the Tiling Algorithm, Spanning Tree Algorithm, and Deltahedron Layout Algorithm, which circumvent the need for an adjacency graph, provided the most effective layouts when regularity of the layout was given even small consideration.

Chapter 7 provided examples where each layout algorithm of Chapter 4 was more effective than the other available algorithms. The purpose of this was to show that even though the Class A algorithms in particular performed poorly on average, they could still provide helpful insights into the final structure of the layout. This chapter provided an important conclusion: that each Facility Layout Problem should be examined on its merits. As each method could assist in the construction of the layout, it proved that sweeping generalisations about the performance of the layout algorithms on a specific problem were dangerous. This chapter also enabled us to develop a set of rules on how each layout algorithm would perform, based on characteristics an MPG may exhibit. Chapters 8 and 9 examined three important concepts: Perturbation, Interaction and Decomposition. These two chapters provided a foundation for an interface between the rigidity of the algorithms and the subjectiveness and flexibility of a design planner.

Perturbation investigated the modification of the problem specifications. Chapter 8 outlined the factors which must be considered in modifying the problem specifications, and concluded that although theoretically possible in general, performed best within dimensionalisable layouts, and provided a framework for their implementation. By retaining the dimensionalisability of the layout, area perturbation became algorithmic under specified width-to-length ratios for each facility, for example. Furthermore, a family of adjacency swaps which would maintain the dimensionalisability property were devised, by which we could iterate through a variety of layouts perturbed from the originally constructed layout.

The concept of Decomposition was perhaps the most important contribution of this work, as it enabled a generic framework to be devised within which we could more efficiently construct the layout. The decomposition of the MPG was based on the separating triangles that existed within the adjacency graph. An adjacency graph consisting of k separating triangles could be decomposed into $k + 1$ subproblems which could be studied virtually independently. This enabled the closer examination of each subproblem, heightening the chances of generating efficient layouts. The concept of α-decomposibility was discussed, where α was the number of facilities in the largest subproblem. It was shown that if the adjacency graph was 8-decomposable, then a dimensionalisable layout could be constructed. Further we showed by experimentation, that decomposition on average reduces the largest subproblem to 60% the size of the original n. This allowed us to consider 25 facility problems as 15 facility problems within this framework, where the remaining 10 facilities could be added to the layout as a result of the separate consideration of the other subproblems. Further we showed that the non-existence of separating triangles does not necessarily prohibit the decomposition process, by providing guidelines where we can reconstruct parts of the MPG to enforce decomposition. An important development from Decomposition, was that each subproblem could be solved using the layout algorithm most appropriate to it. Therefore a single problem instance could utilise all of the techniques of Chapter 4 within this decomposition framework.

The third process developed was that of the interaction of the design planner to the overall process. By considering the arbitrariness of a number of important decisions within the layout algorithms, especially the Contraction Algorithm, we showed that practical layouts could be constructed by directing the algorithm in the most effective way via the development of some *rules of thumb*, which would help the decision making process. Interaction was also deemed to be important when attempting to perturb the problem specifications, as *local knowledge* can more effectively direct perturbations, rather than attempting to rely on the regularity measures for evaluation.

Interaction, Decomposition and Perturbation together provide a complete framework for generating a layout for a given set of problem specifications, as the degree of flexibility allowed by these concepts can incorporate as many factors into the problem as are deemed necessary, by allowing the examination of small components of the problem (Decomposition), the incorporation of subjective and local knowledge (Interaction), and the ability to change the problem specifications as the need arises (Perturbation).

Chapter 10 examined the role of the Material Handling System within the layout, and its effectiveness as an alternative evaluator of a layout. This chapter provided a review of the previous literature on the MHS, from which we were able to devise a heuristic for obtaining an MHS given the material flows between facilities in the layout. The construction of the MHS was based on a Minimal Spanning Tree between a set of Input/Output stations, determined with respect to the facility centroids and material flows. This heuristic for designing the MHS also admitted two simple improvement techniques, which could reduce the overall transportation cost of the MHS. The results of the experiment to evaluate layouts under this MHS scheme were somewhat surprising, with the Contraction Algorithm outperforming the Vertex Splitting Algorithm significantly, yet having inferior regularity scores, and even rivalling the Class I algorithms for large n. The Class I algorithms performed very well, and coupled with their exceptional regularity scores, produced the most practical layouts.

Chapter 11 worked through a tutorial on how we might tackle a *real world* problem. We were able to utilise the many ideas and algorithms we have developed, and were able to obtain good quality solutions with respect to adjacency, regularity and transportation cost. The step by step analysis of the two problems we investigated showed the power of the layout approaches we have used, within the generic framework discussed earlier.

It is somewhat disappointing that maximising the adjacency benefit does not appear to correspond to minimising the transportation cost within the layout. This is the basic assumption under which we use the Graph Theoretic model as an alternative to the Quadratic Assignment Problem. The transportation cost is dependent on the structure of the layout, hence as the adjacency graph incorporates no information about the layout, we cannot correlate the benefit of the adjacency graph with the MHS transportation cost of the layout. Further we have shown that the Class I and T algorithms in general will produce the most practical layouts, yet these methods circumvent the requirement of an adjacency graph. This does not nullify the Graph Theoretic model, as we saw in Chapter 7, where we provided examples of the effective working of this model. It appears that the best approach is one which incorporates the Graph Theoretic ideal of attaining $3n-6$ high benefit adjacencies whilst maintaining as much structure in the subsequent layout as possible. Methods such as the Spanning Tree Algorithm, which has an underlying tree adjacency graph, allow this interface to be fully utilised. Further the Maximal Spanning Tree adjacency structure has a worst case performance of $\frac{1}{3}$, as opposed to the arbitrary worst case performance of TESSA, and some versions of Deltahedron.

Although this thesis has in many ways tied up many of the *loose ends* in the GTFLP, there are several directions where future research may further enhance the development of the layout design. The complete characterisation of MPG structures which are amenable to dimensionalisable layout construction along similar lines to those of Section 4.7, would be an invaluable tool in developing standard layout templates. It is unlikely that the complete set of MPGs could ever be characterised in this way; indeed, it has not been possible to obtain a dimensionalisable layout for the Regular Dodecahedron, which is only of size $n = 12$.

Further work could also be focused on developing new layout algorithms along similar lines to the Class I algorithms, which as we have seen in general perform the most effectively. It would seem that more algorithms which can dualise arbitrary area specifications and MPG are unwarranted, unless they can be developed to perform with some degree of worst case analysis on the facility shape.

The final area of future research could be the further development of Material Handling System methodologies, which we briefly examined in Chapter 10. We provided one mechanism whereby we could construct the MHS, considering only one set of assumptions. There could exist a number Material Handling System models applicable to a variety of situations. Further investigation of the centroid-to-centroid distance metric, and the placement of Input/Output stations could provide many different ways of evaluating the transportation cost within a layout.

It has been difficult in the course of this work to make firm statements as to the effectiveness of the layout procedures. While we have seen general trends in Chapters 6 and 10 as to the practicality of layouts under the regularity measures and transportation costs, we cannot discard the importance of the (in general) inferior layout algorithms. These inferior algorithms can provide important information, especially within an interactive setting. It is therefore our conclusion that the Graph Theoretic model for the Facility Layout Problem contains important information for the effective development of any layout, and indeed the amalgam of the Layout Algorithms of Chapter 4 and the Interaction, Decomposition and Perturbation concepts of Chapters 8 and 9 is an important tool within any layout design process.

Bibliography

[1] O. M. Agraa and B. Whitehead. Nuisance restrictions in the planning of single-storey layouts. *Building Science*, 2:291–302, 1968.

[2] O. M. Agraa and B. Whitehead. A study of movement in a school building. *Building Science*, 2:279–289, 1968.

[3] L. A. Al-Hakim. Two graph-theoretic procedures for an improved solution to the facilities layout problem. *International Journal of Production Research*, 29(8):1701–1718, 1991.

[4] L. A. Al-Hakim. A modified procedure for converting a dual graph to a block layout. *International Journal of Production Research*, 30(10):2467–2476, 1992.

[5] L. A. Al-Hakim. A note on 'on TESSA'. *International Journal of Production Research*, 32(1):223–225, 1994.

[6] G. C. Armour and E. S. Buffa. A heuristic algorithm and simulation approach to relative allocation of facilities. *Management Science*, 9(2):294–300, 1963.

[7] I. Baybars. The generation of floor plans with circulation spaces. *Environment and Planning B*, 9:445–456, 1982.

[8] I. Baybars and C. M. Eastman. Enumerating architectural arrangements by generating their underlying graphs. *Environment and Planning B*, 7:289–310, 1980.

[9] J. Bhasker and S. Sahni. A linear time algorithm to check for the existence of a rectangular dual of a planar triangulated graph. *Networks*, 17:307–317, 1987.

[10] J. Bhasker and S. Sahni. A linear algorithm to find a rectangular dual of a planar triangulated graph. *Algorithmica*, 3:247–278, 1988.

[11] J. A. Bondy and U. S. R. Murty. *Graph Theorey With Applications.* North-Holland, New York, 1976.

[12] S. G. Boswell. Bounds on the number of Γ-operation edge substitutions required to transform a maximal planar graph into another. *Australasian Journal of Combinatorics*, 4:5–24, 1991.

[13] S. G. Boswell. *Graph theory for facilities layout planning: An investigation of the practicalities of heuristics for the adjacency problem.* PhD thesis, University of Newcastle, 1992.

[14] S. G. Boswell. TESSA - a new greedy heuristic for facilities layout planning. *International Journal of Production Research*, 30(8):1957–1968, 1992.

[15] R. Bowen and S. Fisk. Generation of triangulations of teh sphere. *Mathematics of Computation*, 21:250–252, 1967.

[16] Y. A. Bozer and R. D. Meller. A reexamination of the general facility layout problem. Technical report, Auburn University, 1993.

[17] Y. A. Bozer, R. D. Meller, and S. J. Erlebacher. An improvement-type layout algorithm for single and multiple-story facilities. *Management Science*, 40(7):918–932, 1994.

[18] T. Brunes. *The Secrets of Ancient Geometry - And its use*, volume 1. Rhodes International Science Publishers, Copenhagen, 1967.

[19] T. Brunes. *The Secrets of Ancient Geometry - And its use*, volume 2. Rhodes International Science Publishers, Copenhagen, 1967.

[20] B. D. Bunday. *Basic Optimisation Methods.* Edward Arnold Publishers, Baltimore, 1984.

[21] A. S. Carrie, J. M. Moore, M. Roczniak, and J. J. Seppanen. Graph theory and symbolic processors for computer aided facilities design. *Omega*, 6:353–361, 1978.

[22] V. Cerny. Thermodynamical approach to the travelling saleman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.

[23] G. B. Dantzig. All shortest routes in a graph. In *Theory of Graphs*. Gordon and Breach, New York, 1967.

[24] M. P. Deisenroth and J. M. Apple. A computerized plant layout analysis and evaluation technique. In *Annual AIIE Conference, Norcross, GA*, 1972.

[25] K. N. Dutta and S. Sahu. A multigoal heuristic for facilities design problems: MUGHAL. *International Journal of Production Research*, 20(2):147–154, 1982.

[26] M. E. Dyer, L. R. Foulds, and A. M. Frieze. Analysis of heuristics for finding a maximum weight planar subgraph. *European Journal of Operational Research*, 20:102–114, 1985.

[27] P. Eades, L. Foulds, and J. Giffin. An efficient heuristic for identifying a maximum weight planar subgraph. In *Lecture Notes in Mathematics 952 (Combinatorial Mathematics IX)*. Springer-Verlag Berlin, 1981.

[28] R. B. Eggleton and L. A. Al-Hakim. Maximal planar graphs and diagonal operations. *Australasian Journal of Combinatorics*, 3:93–110, 1991.

[29] R. B. Eggleton, L. A. R. Al-Hakim, and J. MacDougall. Braced edges in plane triangulations. *The Australasian Journal of Combinatorics*, 2:121–133, 1990.

[30] L. R. Foulds. Techniques for facilities layout: Deciding which pairs of activities should be adjacent. *Management Science*, 29(12):1414–1426, 1983.

[31] L. R. Foulds, P. B. Gibbons, and J. W. Giffin. Facilities layout adjacency determination: An experimental comparison of three graph theoretic heuristics. *Operations Research*, 33(5):1091–1106, 1985.

[32] L. R. Foulds and J. W. Giffin. A graph-theoretic heuristic for minimising total transport cost in facilities layout. *International Journal of Production Research*, 23(6):1247–1257, 1985.

[33] L. R. Foulds and D. F. Robinson. A strategy for solving the plant layout problem. *Operational Research Quarterly*, 27:845–855, 1976.

[34] L. R. Foulds and D. F. Robinson. Graph theoretic heuristics for the plant layout problem. *International Journal of Production Research*, 16:27–37, 1978.

[35] L. R. Foulds and D. F. Robinson. Construction properties of combinatorial deltahedra. *Discrete Applied Mathematics*, 1:75–87, 1979.

[36] R. L. Francis, L. F. McGinnis Jr., and J.A. White. *Facility Layout and Location: An Analytical Approach.* Prentice Hall, 2nd edition, 1992.

[37] M. R. Garey and D. S. Johnson. *Computers and Intractability: A guide to the theorey of NP-completeness.* Freeman, San Francisco, 1979.

[38] M. T. A. Gawad and B. Whitehead. Addition of communication paths to diagrammatic layouts. *Building and Environment*, 11:249–258, 1976.

[39] J. W. Giffin. *Graph Theoretic Techniques for Facilities Layout.* PhD thesis, University of Canterbury, 1984. Unpublished.

[40] J. W. Giffin and L. R. Foulds. Facilities layout generalized model solved by n-boundary shortest path heuristics. *European Journal of Operational Research*, 28:382–391, 1987.

[41] J. W. Giffin, L. R. Foulds, and D. C. Cameron. Drawing a block plan with graph theory and a microcomputer. *Computers and Industrial Engineering*, 10:109–116, 1986.

[42] J. W. Giffin, K. H. Watson, and L. R. Foulds. Orthogonal layouts using the Deltahedron heuristic. *Australasian Journal of Combinatorics, (to appear)*, 1994.

[43] F. Glover. Tabu search - part I. *ORSA Journal on Computing*, 1:190–206, 1989.

[44] F. Glover. Tabu search - part II. *ORSA Journal on Computing*, 2:4–32, 1990.

[45] F. Glover. Tabu search: A tutorial. *Interfaces*, 20:74–94, 1990.

[46] M. Goetschalckx. An interactive layout heuristic based on hexagonal adjacency graphs. *European Journal of Operational Research*, 63:304–321, 1992.

[47] R. H. Green and L. Al-Hakim. A heuristic for facilities layout planning. *Omega*, 13(5):469–474, 1985.

[48] A. Hammouche and D. B. Webster. Evaluation of an application of graph theory to the layout problem. *International Journal of Production Research*, 23(5):987–1000, 1985.

[49] R. Hashimshony, E. Shaviv, and A. Wachman. Transforming an adjacency matrix into a planar graph. *Building and Environment*, 15:205–217, 1980.

[50] M. M. D. Hassan. Some observations on converting a dual graph into a block layout. *International Journal of Production Research*, 30(10):2477–2482, 1992.

[51] M. M. D. Hassan and G. L. Hogg. A review of graph theory application to the facilities layout problem. *Omega*, 15(4):291–300, 1987.

[52] M. M. D. Hassan and G. L. Hogg. On converting a dual graph into a block layout. *International Journal of Production Research*, 27(7):1149–1160, 1989.

[53] M. M. D. Hassan and G. L. Hogg. On constructing a block layout by graph theory. *International Journal of Production Research*, 29(6):1263–1278, 1991.

[54] M. M. D. Hassan, G. L. Hogg, and D. R. Smith. SHAPE: A construction algorithm for area placement evaluation. *International Journal of Production Research*, 24:1283–1295, 1986.

[55] S. S. Heragu and A. S. Alfa. Experimental analysis of simulated annealing based algorithms for the layout problem. *European Journal of Operational Research*, 57:190–202, 1992.

[56] S. S. Heragu and A. Kusiak. Machine layout: an optimization and knowledge-based approach. *International Journal of Production Research*, 28(4):615–635, 1990.

[57] S. S. Heragu and A. Kusiak. Efficient models for the facility layout problem. *European Journal of Operational Research*, 53:1–13, 1991.

[58] F. S. Hillier and M. M. Conners. Quadratic assignment problem algorithms and the location of indivisible facilities. *Management Science*, 13:42–57, 1966.

[59] J. E. Hopcroft and R. E. Tarjan. Efficient planarity testing. *Journal of the Association for Computing Machinery*, 21:549–568, 1974.

[60] S. Irvine and I. Rinsma-Melchert. Lampg - implementation of a new facilities layout algorithm. Research Report 42, Waikato University, 1995.

[61] S. Jajodia, I. Minis, G. Harhalakis, and J. Proth. CLASS: Computerized layout solutions using simulated annealing. *International Journal of Production Research*, 30(1):95–108, 1992.

[62] S. Kirkpatrick, C. D Gelatt, and M. P. Vecchi. Optimization by simmulated annealing. *Science*, 13:671–680, 1983.

[63] T. C. Koopmanns and M. Beckmann. Assignment problems and the location of economic facilities. *Econometrica*, 25:52–76, 1957.

[64] K. Kozminski and E. Kinnen. An algorithm for finding a rectangular dual of a planar graph for use in area planning for vlsi integrated circuits. In *21st Design Automation Conference*, pages 655–656, 1984.

[65] K. Kozminski and E. Kinnen. Rectangular duals of planar graphs. *Networks*, 15:145–157, 1985.

[66] K. Kozminski and E. Kinnen. Rectangular dualization and rectangular dissections. *IEEE Transactions on Circuits and Systems*, 35(11):1401–1416, 1988.

[67] J. B. Kruskal. On the shortest spanning subtree of a graph and the travelling salesman problem. *Proceedings of the American Mathematical Society*, 7:48–50, 1956.

[68] A. Kusiak and S. S. Heragu. The facility layout problem. *European Journal of Operational Research*, 29:229–251, 1987.

[69] T. A. Lacksonen. Static and dynamic layout problems with varying areas. *Journal of the Operational Research Society*, 45(1):59–69, 1994.

[70] Y. Lai and S. M. Leinwand. A theory of rectangular dual graphs. *Algorithmica*, 5:467–483, 1990.

[71] A. Langevin, B. Montreuil, and D. Riopel. Spine layout design. *International Journal of Production Research*, 32(2):429–442, 1994.

[72] E. L. Lawler. The quadratic assignment problem. *Management Science*, 9:586–599, 1963.

[73] R. Lee and J. M. Moore. CORELAP - computerized relationship layout planning. *Journal of Industrial Engineering*, 18:195–200, 1967.

[74] J. Lehel. Deltahedra are realizable as simplicial convex polyhedra. *Discrete Applied Mathematics*, 2:81–84, 1980.

[75] S. M. Leinwand and Y. Lai. An algorithm for building rectangular floorplans. In *21st Design Automation Conference*, pages 663–664, 1984.

[76] A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. *Theorie des Graphs*, 1966.

[77] J. K. Lenstra. *Sequencing by enumerative methods.* Mathematisch Centrum, Amsterdam, 1976.

[78] J. Leung. A new graph-theoretic heuristic for facility layout. *Management Science*, 38(4):594–605, 1992.

[79] W. P. Lewis and T. E. Block. On the application of computer aids to plant layout. *International Journal of Production Research*, 18(1):11–20, 1980.

[80] P. C. Liu and R. F. Geldmacher. On the deletion of non-planar edges of a graph. Technical report, Stevens Institute of Technology, Hoboken, New Jersey, 1976.

[81] J. MacGregor-Smith. Cellular arrangement problems and steiner-star duals. Tech reprot, University of Massachusetts, 1993.

[82] J. MacGregor-Smith, D. T. Lee, and J. S. Liebman. An o(n log n) heuristic algorithm for the rectilinear Steiner minimal tree problem. *Engineering Optimization*, 4:179–192, 1980.

[83] C. J. Malmborg. A heuristic model for simultaneous storage space allocation and block layout planning. *International Journal of Production Research*, 32(3):517–530, 1994.

[84] R. D. Meller and K-Y Gau. The facility layout problem: A review of recent and emerging research. Technical report, Auburn University, Alabama, 1995.

[85] B. Montreuil, H. D. Ratliff, and M. Goetschalckx. Matching based interactive facility layout. *IIE Transactions*, 19(3):271–279, 1987.

[86] B. Montreuil and U. Venkatadri. From gross to net layouts: An efficient design model. Technical report, Laval University, 1988.

[87] B. Montreuil, U. Venkatadri, and H. D. Ratliff. Generating a layout from a design skeleton. *IIE Transactions*, 25(1):3–15, 1993.

[88] J. M. Moore. Facilities design with graph theory and strings. *Omega*, 4:193, 1976.

[89] Q. Ning. On a conjecture of Foulds and Robinson about deltahedra. *Discrete Applied Mathematics*, 18:305–308, 1987.

[90] O. Ore. *The Four-Color Problem.* New York: Academic Press, 1967.

[91] P. E. Radcliffe, D. E. Kawal, and R. J. Stephenson. *Critical Path Method.* Cahner, Chicago, Ill, 1967.

[92] A. D. Raoot and A. Rakshit. An experimental comparison of systematic placement procedures for facility layout design. *International Journal of Production Research*, 31(7):1735–1756, 1993.

[93] A. D. Raoot and A. Rakshit. A "linguistic pattern" approach for multiple criteria facility layout problems. *International Journal of Production Research*, 31(1):203–222, 1993.

[94] I. Rinsma. *Existence Theorems for Floorplans.* PhD thesis, University of Canterbury, 1987. Unpublished.

[95] I. Rinsma. Rectangular and orthogonal floorplans with require room areas and tree adjacency. *Environment and Planning B*, 15:111–118, 1988.

[96] I. Rinsma, J. W. Giffin, and D. F. Robinson. Orthogonal floorplans from maximal planar graphs. *Environment and Planning B*, 17:57–71, 1990.

[97] M. J. Rosenblatt. The facilities layout problem: a multi-goal approach. *International Journal of Production Research*, 17(4):323–332, 1979.

[98] J. Roth, R. Hashimshomy, and A. Wachman. Turning a graph into a rectangular floorplan. *Building and Environment*, 17:163–173, 1982.

[99] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the Association of Computing Machinery*, 23(555-565), 1976.

[100] J. M. Seehof and W. O. Evans. An automated layout design program. *Journal of Industrial Engineering*, 18(2):690–695, 1967.

[101] J. M. Seppanen and J. M. Moore. Facilities planning with graph theory. *Management Science*, 17:242–253, 1970.

[102] J. S. Shang. Multicriteria facility layout problem:an integrated approach. *European Journal of Operational Research*, 66:291–304, 1993.

[103] Y. Sun and M. Sarrafzadeh. Floorplanning by graph dualization: L-shaped modules. *Algorithmica*, 10:429–456, 1993.

[104] J. A. Tompkins and R. Reed Jr. An applied model for the facilities design problem. *International Journal of Production Research*, 14(5):583–595, 1976.

[105] P. S. Welgama and P. R. Gibson. A construction algorithm for the machine layout problem with fixed pick-up and drop-off points. *International Journal of Production Research*, 31(11):2575–2590, 1993.

[106] P. S. Welgama, P. R. Gibson, and L. A. R. Al-Hakim. Facilities layout: A knowledge-based approach for converting a dual graph into a block layout. *International Journal of Production Economics*, 33:17–30, 1994.

[107] B. Whitehead and M. Z. Eldars. The planning of single-storey layouts. *Building Science*, 1:127–139, 1963.

[108] K. Yeap and M. Sarrafzadeh. A unified approach to floorplan sizing and enumeration. Northwestern University Manuscript, 1992.

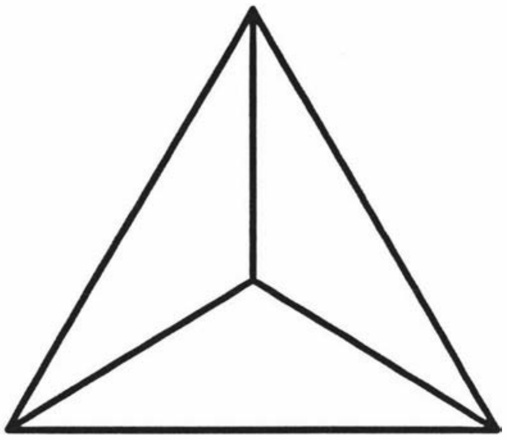
[109] K. Yeap and M. Sarrafzadeh. Floor-planning by graph dualization: 2-concave rectilinear modules. *Siam Journal of Computing*, 22(3):500–526, 1993.

Appendix A

Small MPGs and their layouts

In Section 4.7.5, we stated that all MPGs with $n \leq 8$ could generate dimensionalisable layouts with at worst T shaped facilities. This appendix provides the templates for these layouts. Bowen and Fisk [15] show that the number of unlabelled MPGs on $n = 4, 5, 6, 7$, and 8 are 1,1,2,5, and 14 respectively. In order to generate the templates, we need only specify the exterior facility, as usual labelled 1, and then impose an arbitrary labelling on the remaining $n - 1$ facilities. Firstly we show that the set of MPGs generated for each n is indeed the complete set of unlabelled MPGs on that n. The *vertex degree numbering* of an MPG is a ordering of each distinct degree in the MPG with superscript dependent on the number of vertices having that degree. For example, an MPG with four vertices of degree four, and two vertices of degree five, would have vertex degree numbering $4^4 5^2$. For $n = 4$ and $n = 5$, there exists only one MPG, while for $n = 6$ and $n = 7$, the vertex degree numberings for each MPG are distinct, and hence cannot be isomorphic. The set of unlabelled MPGs on $n = 8$ has 13 distinct vertex degree numberings. Of the two MPGs with the vertex degree numbering of $3^2 4^2 5^2 6^2$, one is Deltahedron generateable, and the other is not, hence we obtain the set of 14 non-isomorphic unlabelled MPGs on $n = 8$. MPGs on $n \leq 8$ which are Deltahedron generateable, obviously have a worst case facility shape of a T. Less obvious, is that MPGs on $n \leq 8$ which are Extended Deltahedron generateable, also have a worst case facility shape of T. The reason for this is that Y and X shaped facilities can only be created by placing 3 or 4, respectively, facilities within a placement host, which has four placement directions directed into it. In order to create a valid placement host which has this condition requires at least a six facility partial layout, including the exterior, but at that point we have only two facilities remaining to be placed. Furthermore, in the case where an S shaped facility could be created, we can redirect the placement directions on one side of the placement host to circumvent this.

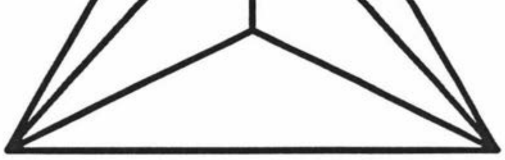
$n = 4$



Deltahedron

Figure A.1: Template 3^4

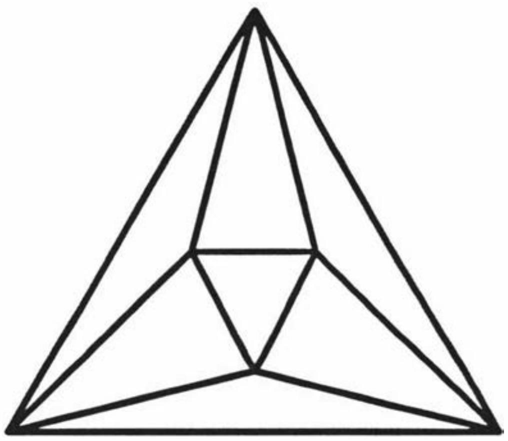
$n = 5$



Deltahedron

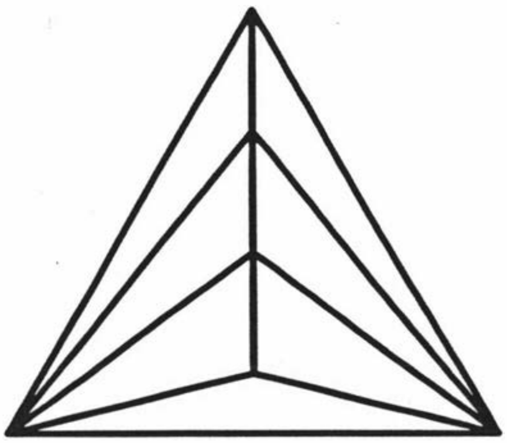
Figure A.2: Template $3^2 4^3$

$n = 6$



Extended Deltahedron

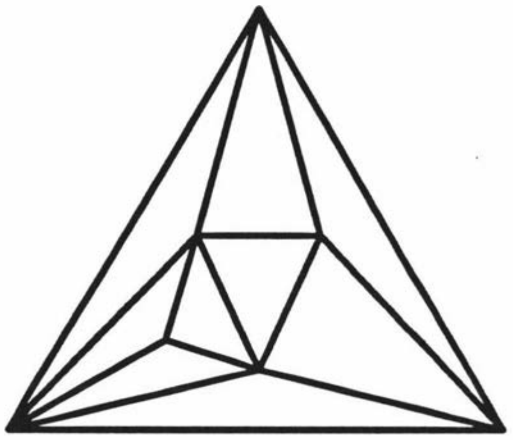
Figure A.3: Template 4^6



Deltahedron

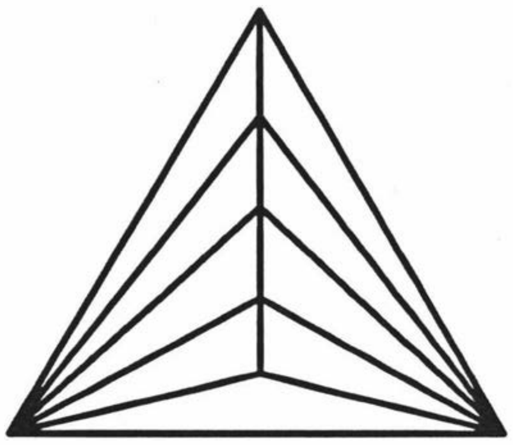
Figure A.4: Template $3^2 4^2 5^2$

$n = 7$



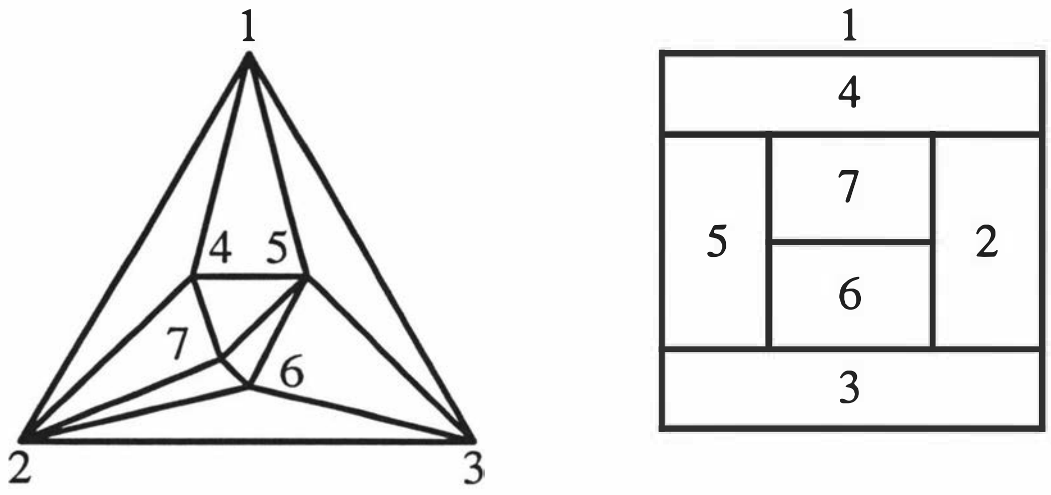
Extended Deltahedron

Figure A.5: Template $3^1 4^3 5^3$

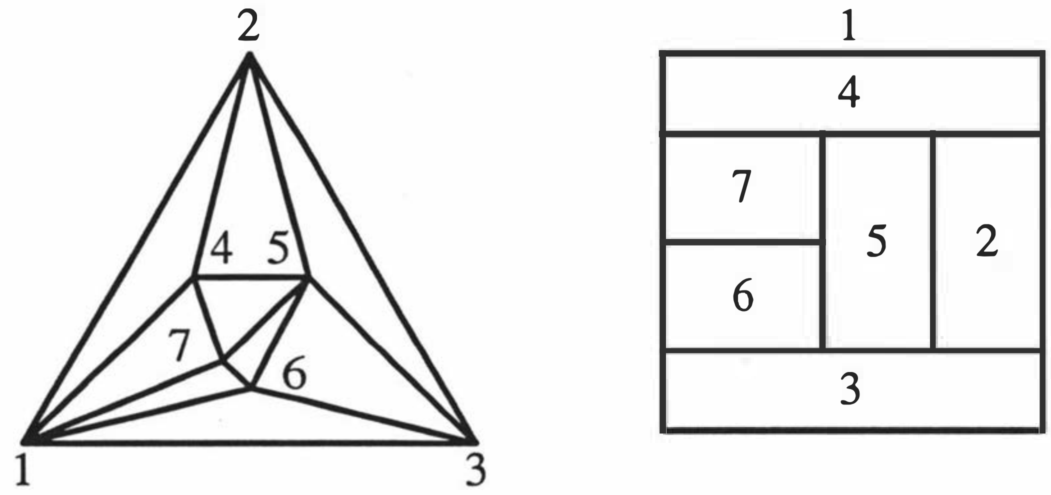


Deltahedron

Figure A.6: Template $3^2 4^3 6^2$

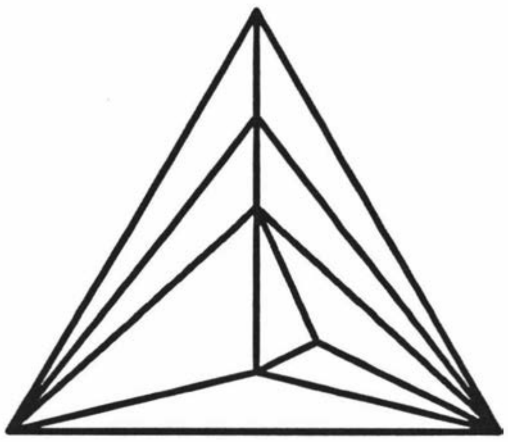


(a)



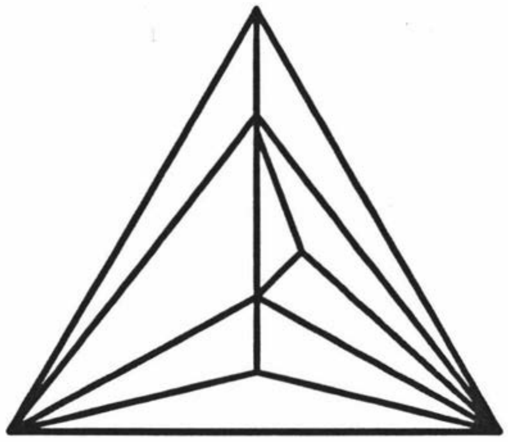
(b)

Figure A.7: Template $4^5 5^2$; (a) exterior vertex of degree 4; (b) exterior vertex of degree 5



Deltahedron

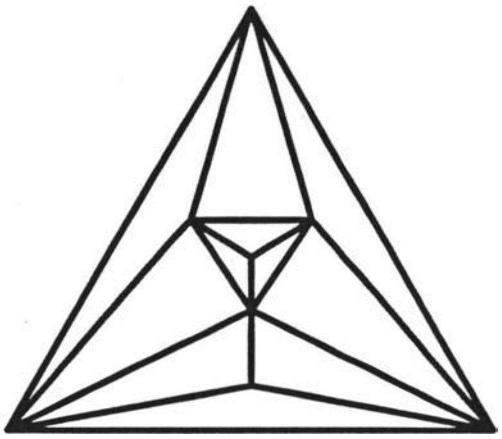
Figure A.8: Template $3^2 4^2 5^2 6^1$



Deltahedron

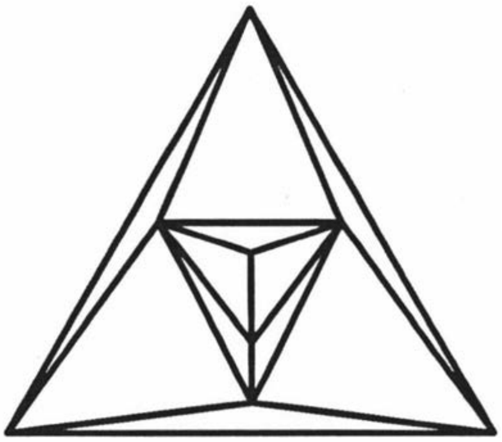
Figure A.9: Template $3^3 5^3 6^1$

$n = 8$



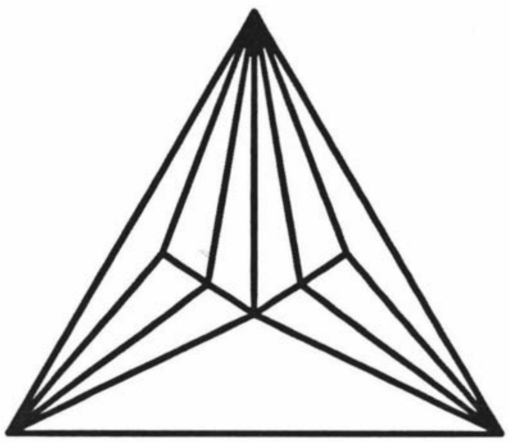
Extended Deltahedron

Figure A.10: Template $3^2 4^1 5^4 6^1$

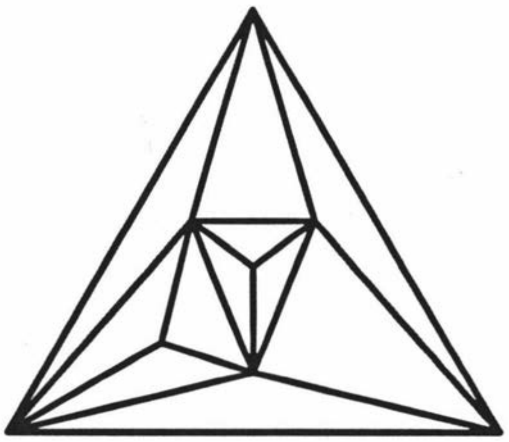


Extended Deltahedron

Figure A.11: Template $3^1 4^4 5^1 6^2$



(a)



(b)

Figure A.12: Template $3^2 4^2 5^2 6^2$; (a) Deltahedron Generateable (b) Not Deltahedron Generateable

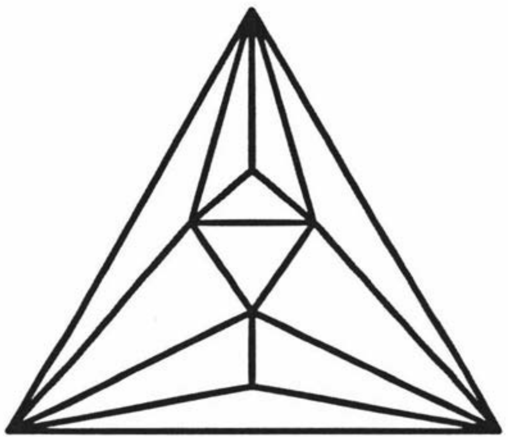


Figure A.13: Template $3^2 5^6$

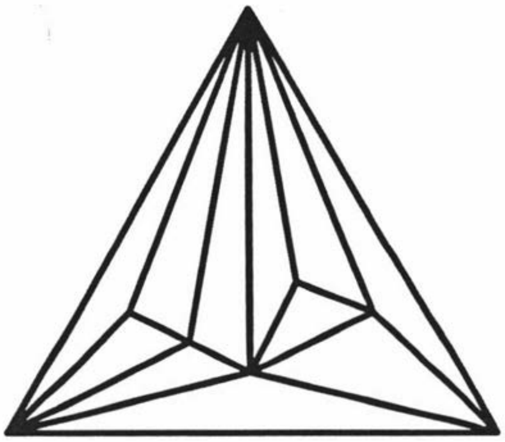
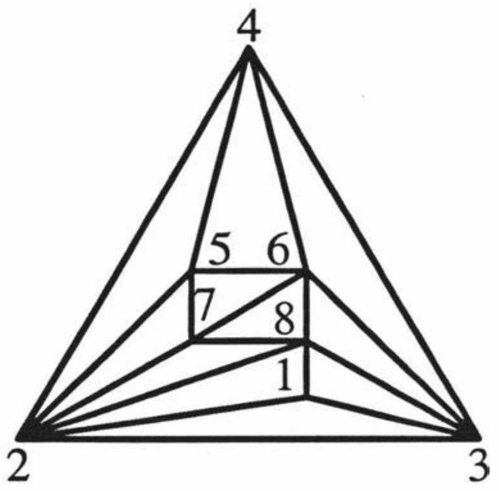
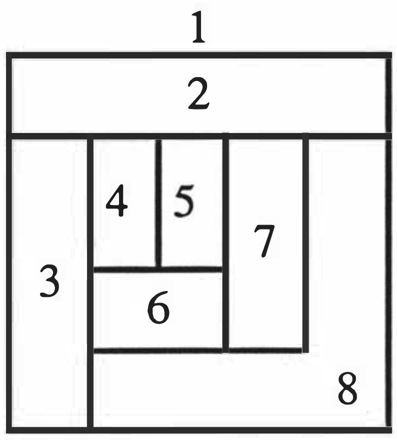
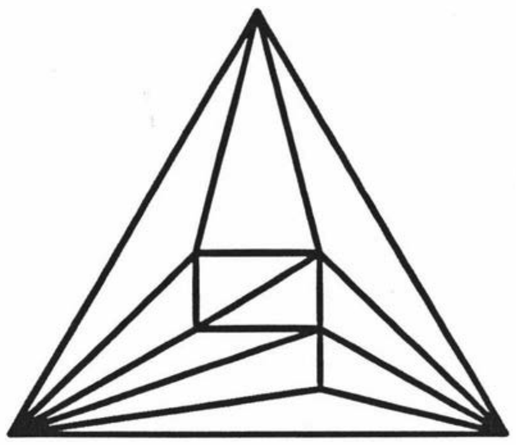


Figure A.14: Template $3^2 4^3 5^1 6^1 7^1$





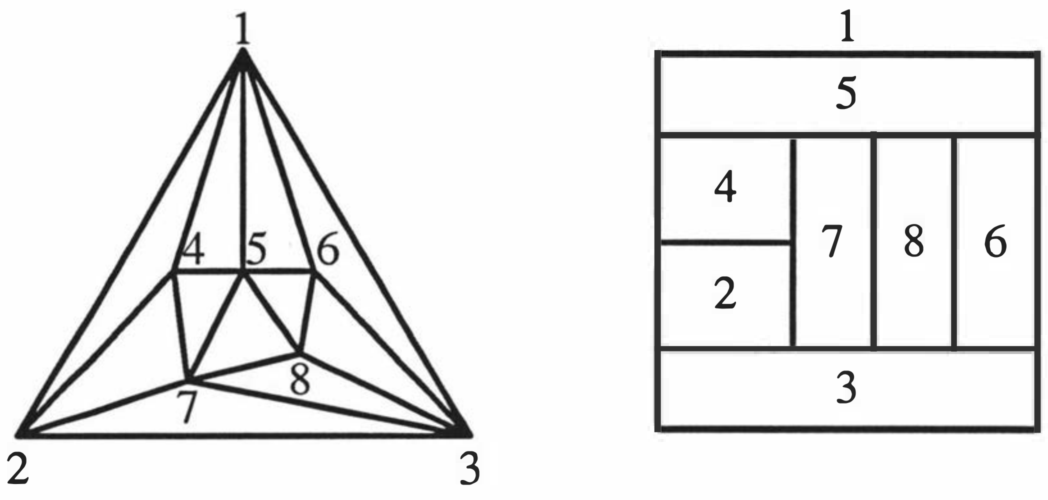
(a)



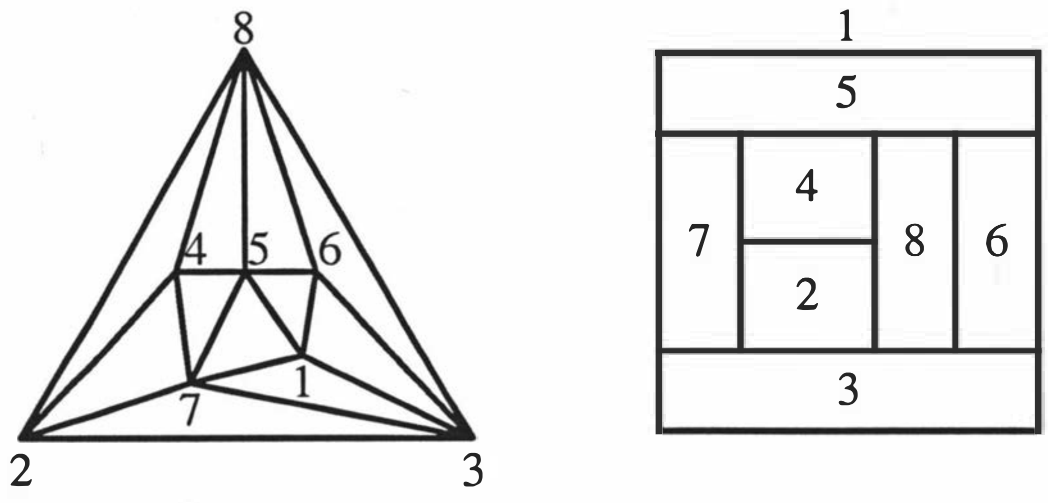
Use Templates of Figure A.7, and perform a Deltahedron insertion to place the vertex of degree 3

(b)

Figure A.15: Template $3^1 4^3 5^3 6^1$; (a) exterior vertex of degree 3; (b) exterior vertex of not of degree 3

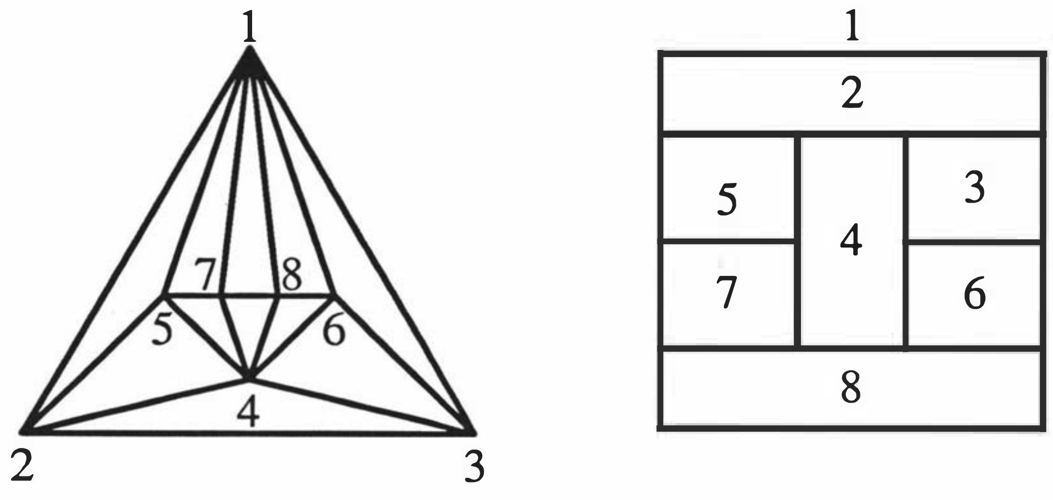


(a)

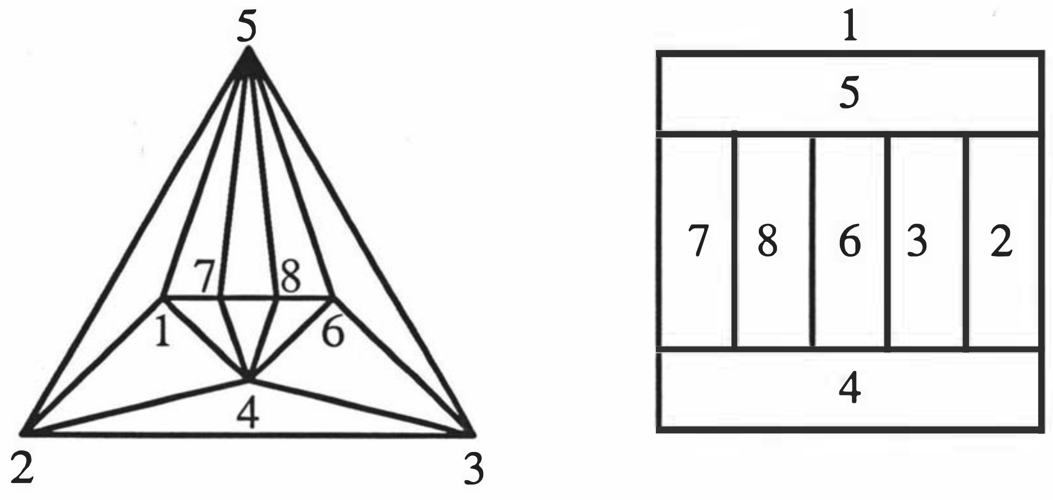


(b)

Figure A.16: Template $4^4 5^4$; (a) exterior vertex of degree 5; (b) exterior vertex of degree 4



(a)



(b)

Figure A.17: Template $4^6 6^2$; (a) exterior vertex of degree 6; (b) exterior vertex of degree 4

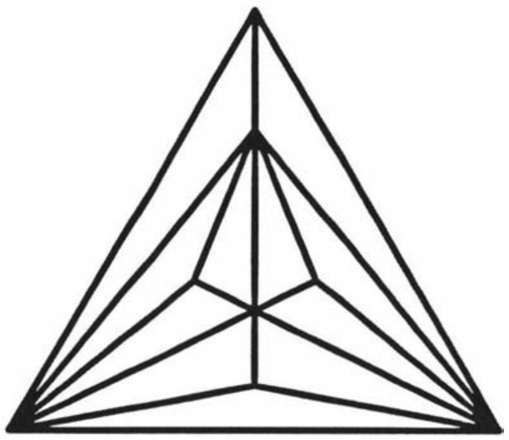


Figure A.18: Template $3^4 6^4$

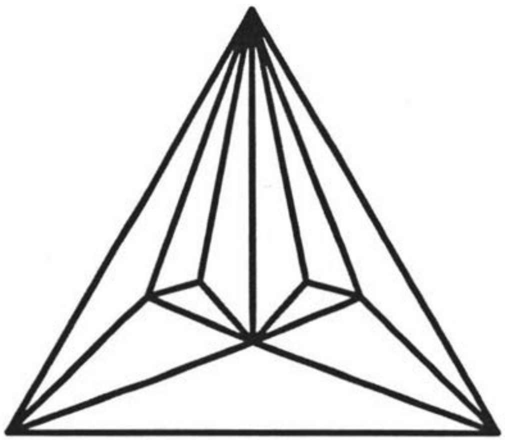
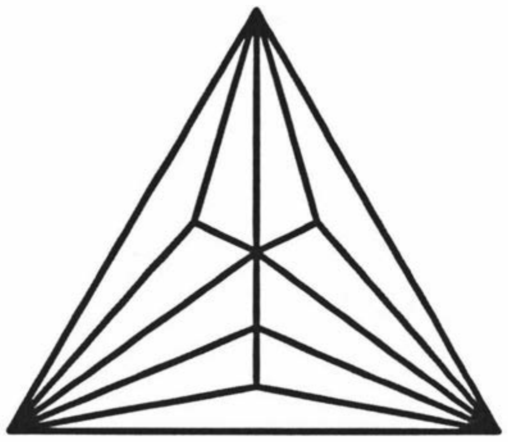
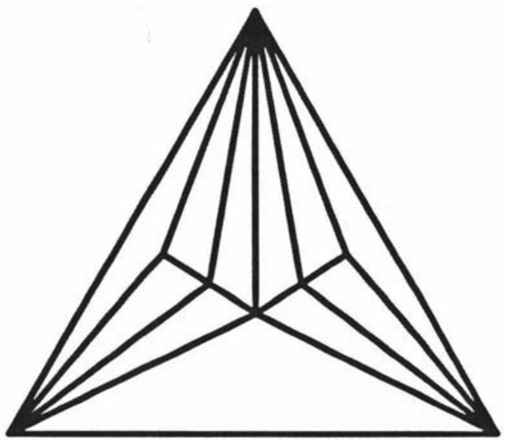


Figure A.19: Template $3^2 4^4 7^2$



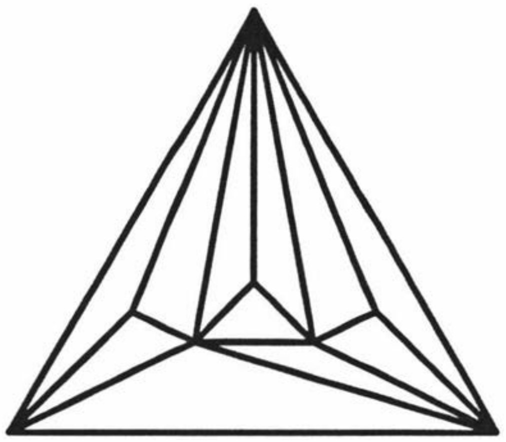
Deltahedron

Figure A.20: Template $3^3 4^1 5^1 6^3$



Deltahedron

Figure A.21: Template $3^2 4^2 5^3 7^1$



Deltahedron

Figure A.22: Template $3^3 4^1 5^2 6^1 7^1$