Genetic Network Programming with Reinforcement Learning and Optimal Search Component

A thesis presented in partial
fulfilment of the requirements

for the degree of

Master of Science in Computer Sciences

at Massey University, Auckland, New Zealand

Mona Abdulrahman M Alshehri

2019

Contents

Figures

Equations

Pseudo Codes

Tables

Abstract

This thesis presents ways of improving the genetic composition, structure and learning strategies for a graph-based evolutionary algorithm, called Genetic Networking Programming with Reinforcement Learning (GNP-RL), particularly when working with multi-agent and dynamic environments. GNP-RL is an improvement over Genetic Programming, allowing for the concise representation of solutions in terms of a networked graph structure and uses RL to further refine the graph solutions. This work has improved GNP-RL by combining three new techniques: Firstly, it has added a reward and punishment scheme as part of its learning strategy that supports constraint conformance, allowing for a more adaptive training of the agent, so that it can learn how to avoid unwanted situations more effectively. Secondly, an optimal search algorithm has been combined in the GNP-RL core to get an accurate analysis of the exploratory environment. Thirdly, a task prioritization technique has been added to the agent's learning by giving promotional rewards, so they are trained on how to take priority into account when performing tasks. In this thesis, we applied the improved algorithm to the Tile World benchmarking testbed, which is considered as one of the standard complex problems in this domain, having only a sparse training set. Our experiment results show that the proposed algorithm is superior than the best existing variant of the GNP-RL algorithm [1]. We have achieved 86.66% test accuracy on the standard benchmarking dataset [2]. In addition, we have created another benchmarking dataset, similar in complexity to the one proposed in [1], to test the proposed algorithms further, where it achieved a test accuracy of 96.66%; that is 33.66% more accurate.

Acknowledgement

I would like to express my thanks and appreciation to all the people who have supported me to finish this research…

To my parents: Thank you for all the care, attention, and prayers you have given me until I became what I am today.

To my husband (Bandar): Thank you for all the support, motivation and for all the times you stand by my side.

To my daughters (Reema & Sara): I wait for the day when I see you reading this thesis.

To my sister (Nada): Thank you for all of your spiritual and moral support.

A grateful to my supervisor Dr. Napoleon Reyes and co-supervisor Dr. Andre Barczak: Thank you for your time, guidance and invaluable advices you provided for me to complete this research.

This thesis is a tribute to the place I belong, hope and home; the Kingdom of Saudi Arabia, for giving me the opportunity and full support to complete my higher education.

Chapter 1 Research Description

1.1 Overview of the current state of technology

Artificial intelligence revolution is one of the qualitative developments in computing, which increased computer performance and its work. Over the years, scientists have tried to simulate human and animal intelligence in the machine, such as the ant algorithm, neural networks, and genetic algorithm.

In 1975 [3], Holland began working on the genetic algorithm (GA) as an algorithm used to find the best solution using random search method. This algorithm was inspired by simulating the transmission of genes from generation to generation in the living organisms. Holland represented the set of solutions with a set of chromosomes for each chromosome group of genes that represent the properties of the solution. By crossing the chromosomes and working on the mutations, Holland was able to generate new solutions, generation by generation. By using the solution function to evaluate individuals, the optimal solution can be obtained after several generations. The genetic algorithm has been used to solve many problems in productive ways. The chromosome represented with a series of ones and zeros to simplify the operations that have been used on them. This algorithm evolved over the years until in 1992 and 1994 [4] [5] Koza changed the composition of the chromosome to be a tree structure instead of a series of binary numbers as in (GA), and he called it Genetic Programming (GP). The new structure has proved it's accomplished in many areas such as (robot soccer by Vic in 2002 [6] and tile world problem by Li in 2010 [7]).

In 2002, a new algorithm produced from the (GP) which called genetic network programming (GNP), the algorithm was presented by Katagiri [8] which uses a graph structure to represent the chromosomes, each graph has a number of nodes including one start node and without finishing node, each node could be visited more than one time and from more than one agent. In 2007, the reinforcement learning (RL) had been added to the (GNP) to allow the algorithm to solve more complex problems, that by combining several sub-nodes to each node and run the RL to learn how to use the graph and which subnode is the best. This algorithm called (GNP-RL), and it was addressed by Mabu [9]. This algorithm has achieved better results than previous genetic algorithms in solving many problems such as making mobile robot behavior [10] and in the Stock trading model [11] especially the problem of tile world which the algorithm was applied on it by Mabu [9].

Since 2007, GNP-RL has been improved, and many other algorithms and techniques were combined to it. One of these implementations is the distributed (DGNP-RL) and the variable-sized GNP-RL (VSGNP-RL). These two techniques are working to divide the problem into small tasks by distributing the chromosome structure to some subprograms; each subprogram works with a task. VSGNP-RL differs from the DGNP-RL in which the VSGNP-RL could have different size for each subprogram. These two algorithms were addressed and compared by Mabu in 2014 [2]. When Mabu applied VSGNP-RL on the tile world benchmark, the algorithm was able to achieve an acceptable result on the training stage but not in the testing stage. Li came with a new method which extracts the shared rules from the best individuals and uses them with others to get the final serial of rules for the problem. This mechanism was applied on the tile world problem and addressed in 2018 [1], it got the best results so far on using GNP-RL with the tile world problem which is (19/30) (63.33%) of success on the testing stage.

This work aims to further improve the composition and learning paradigm of the GNP-RL algorithm when working on a multi-agent and dynamic environment. As a testbed, we aim to get the highest results in tile world problem by working on enhancing the training and learning method by adding the reward and punishment and by integrating optimal search algorithm to obtain more accurate feedback from the environment to the agents.

1.2 Research objectives

The main objective of this work is to develop an improved variant of the GNP-RL algorithm that utilizes an optimal search algorithm at its core, in order to solve more effectively in a multi-agent, dynamic path-planning benchmarking problem. The following outlines the specific objectives:

- To improve the GNP-RL chromosome structure to allow for faster learning and higher performance.
- To improve the reinforcement learning technique on the GNP.
- To improve the fitness function, allowing for adaptive assigning of rewards and punishments, proportional to the agent's goal achievements.
- To improve the way that the agent used to search for the objects in the grid world, by adding the optimal search algorithm to the GNP-RL.
- To add a learning facility for training the agent on how to avoid getting into trapped situations.
- To add the priority tasks implementation technique to the learning stage.
- To check if the GNP-RL may suffer from overfitting during training.

- To compare the performance of the proposed algorithm against other existing variants of GNP-RL on the standard benchmarking testbed known as the Tile World problem, using the standard dataset as well as using our extended and more complex environments.

1.3 Scope and limitations of research

- This work focuses on improving the GNP-RL-based algorithm for solving a multi-agent path-planning problem, operating in a dynamic environment.
- It has been tested in the Tile World Problem, which is one of the complex problems, because of the sparse training set. The agents are designed to turn left, turn right, go forward, but not go backward. And, they can push the tiles, but can't pull them.
- For all implementations in this thesis, the parameters in Table 1.1 have been chosen. Most of these parameters were set in [9] [2] [12] [11] and [13].

Population Size	**Crossover**	**Mutation**	**Elite**	**Max # sub nodes**
300	120	175	5	4
Crossover rate Pc	**Mutation rate Pm**	**ε**	**γ**	**α**
0.1	0.01	0.1	0.9	0.9
Delay time	1 → Judgment node		5 → Processing node	

Table 1.1 Parameters Table

- For all the implementations, the training has been stopped before 5000 generations if one of these cases happened:
  - The algorithm has not improved for a long time.
  - The algorithm enters into an over trained stage.
- For all the implementations, the test phase has not been implemented if the algorithm failed in the training stage.
- Using optimal search (A* algorithm in finding the shortest path) need a comprehensive vision from the agents to the environment.
- The agents don't have direct communication between them.
- Using A* algorithm spend more time.
- All the experiments in this work have been done on a machine with (Intel® Core ™ i7-7700HQ CPU @ 2.80GHz – RAM 16 GB).

1.4 Overview of the problem domain

This work was implemented on the Tile world problem, which is one of the multi-agents, dynamic system benchmarks problem. Pollak described this application in 1990 [14], which is a grid world with 5 different objects, Agents, Holes, Tiles, Obstacles, and Floors. In this work, the training and testing sets are produced from this benchmark. There are 10 environments; each one of them is a two-dimensional grid world with 3 agents, 3 tiles, 3 holes, and some obstacles and floors, as shown in Figure 1.1.

The rules of this problem are:

1- The agent can turn left, turn right, or go forward (which called Processing node in this work, as described in page 31).
    a. The agent can go forward if the next vertex is floor or tile.
    b. If the next vertex is tile, the agent can push the tile to the next vertex except if the next vertex is an obstacle or another agent.
    c. If the tile is dropped into a hole, the tile and the hole disappear.
2- Each agent has sensors that allow making the agents answer some questions such as (what is in the right, where is the nearest tile …etc., which described in page 31) (this called Judgment node in this work).
3- Each agent has 60 steps, and each step contains 8 points. So, each agent has a sequence of 8 actions on the grid world. These actions could be Processing like (turn left) or Judgment as (what is in the right), each Processing action costs 5 points, and each Judgment costs 1 point (This called Delay time in this work which is described in page 31-32).
4- In the simulation, in each environment, the three agents work to drop all the three tiles into the three holes. If the agents could drop all the tiles into the holes, or if the steps are finished before that, the algorithm will close this environment and start the next one.
5- There are two different training sets in this work (Figure 1.1 & Figure 1.2)
6- There is one testing set in this work, which was used by Mabu in [2] (Figure 1.3).

Training Set (1) [2]

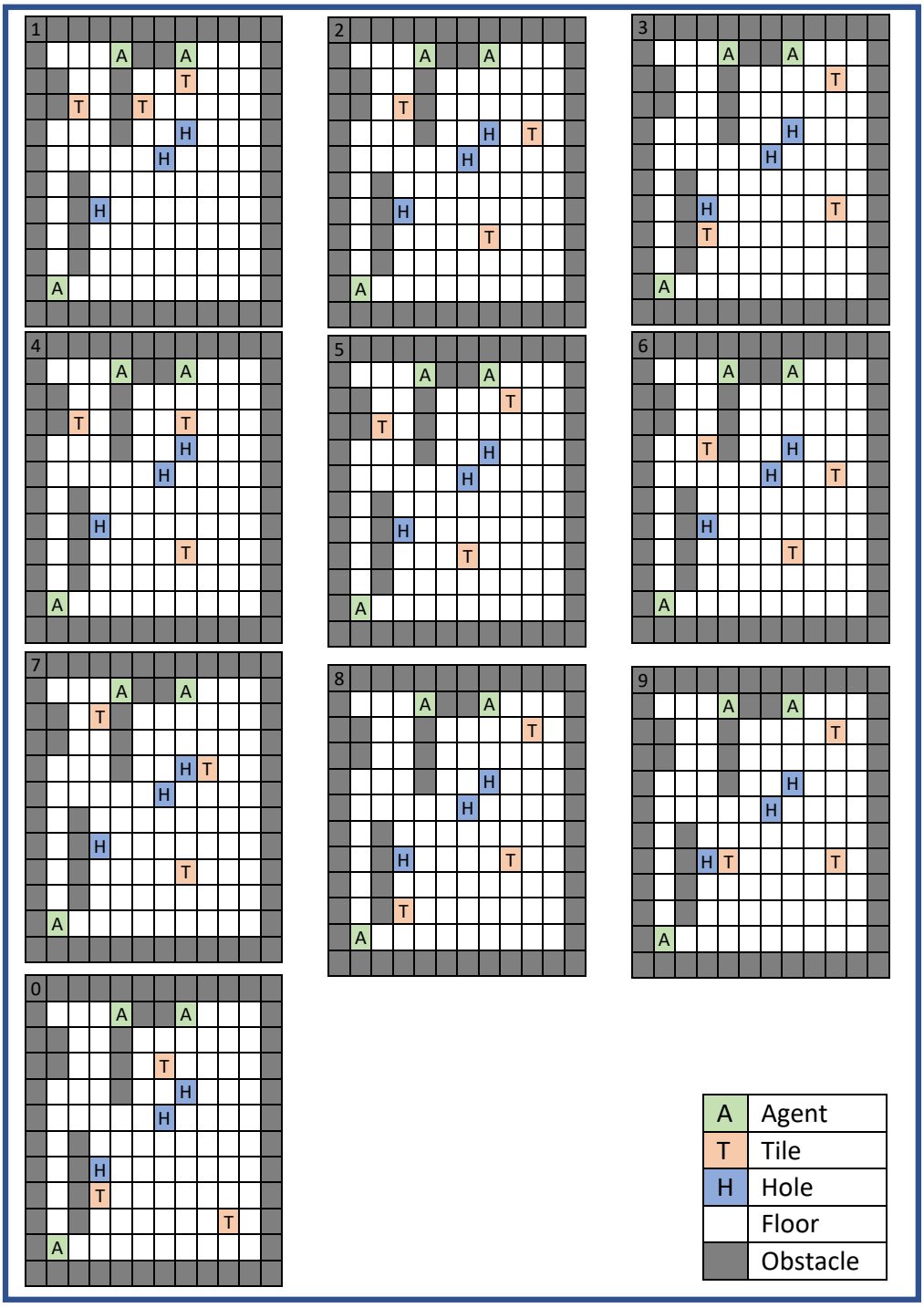


Figure 1.1 Training Set (1) [2]

Training Set (2)

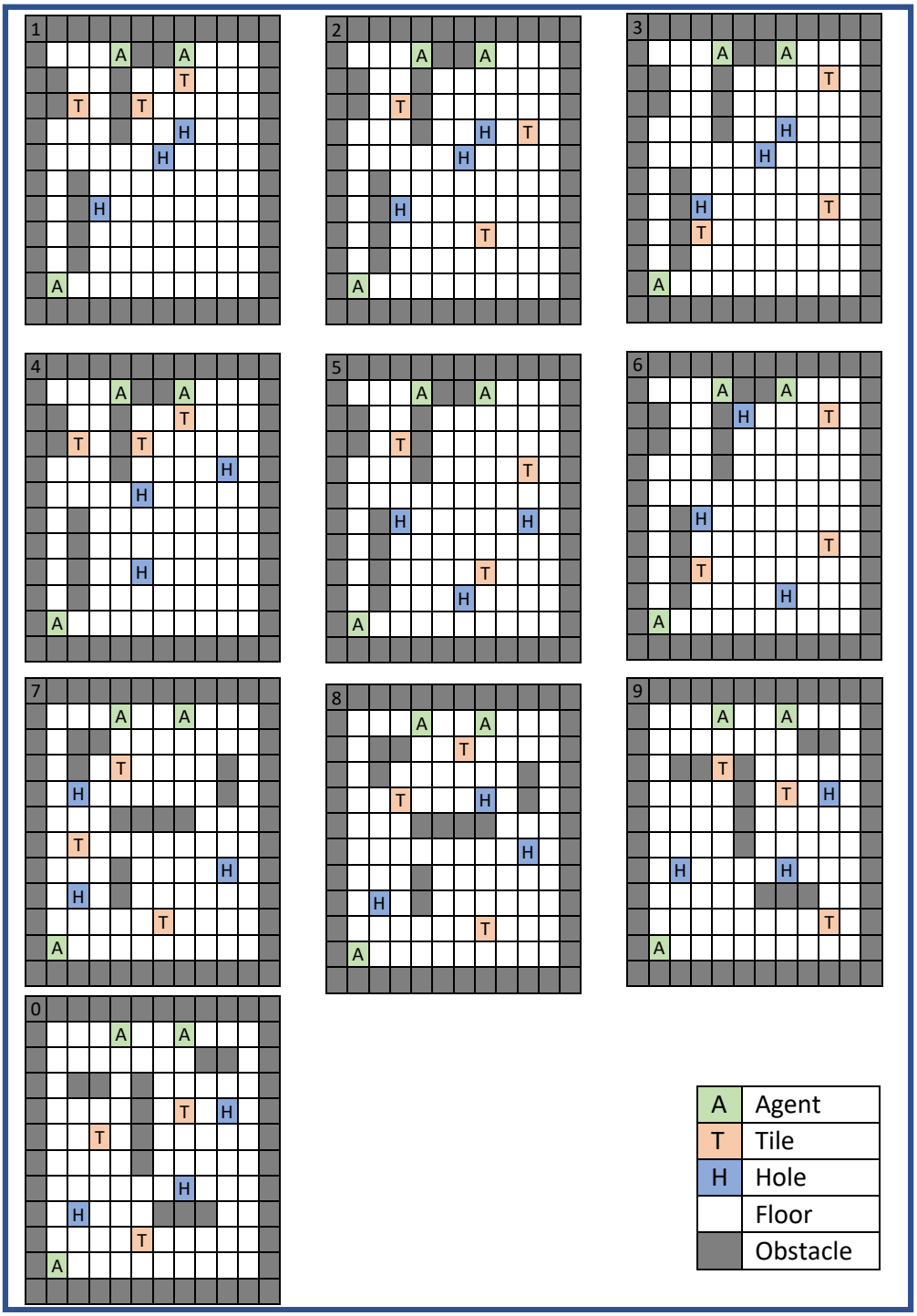


Figure 1.2 Training Set (2)

Testing Set (1) [2]

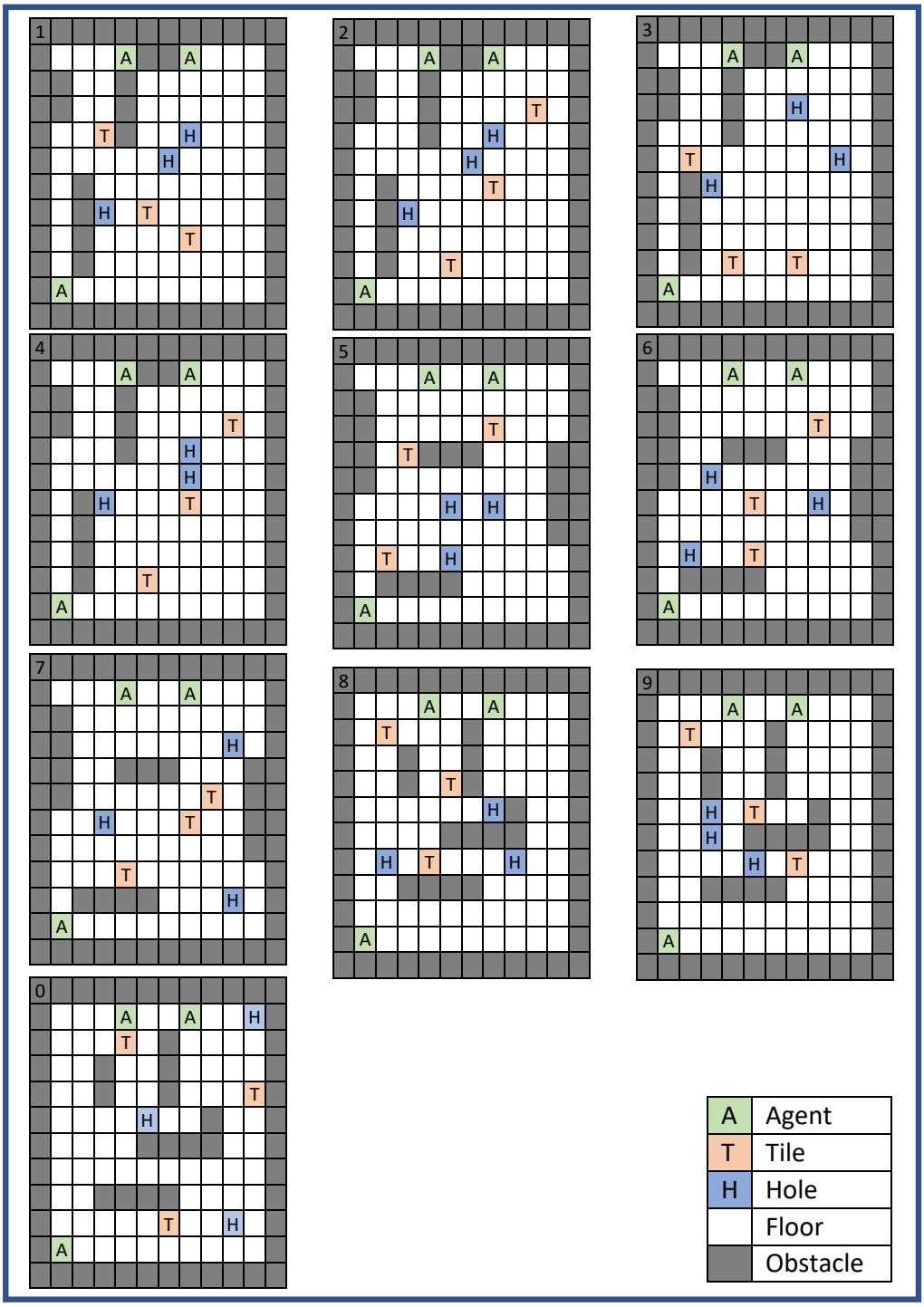


Figure 1.3 Testing Environment (1) [2]

Testing Set (2)

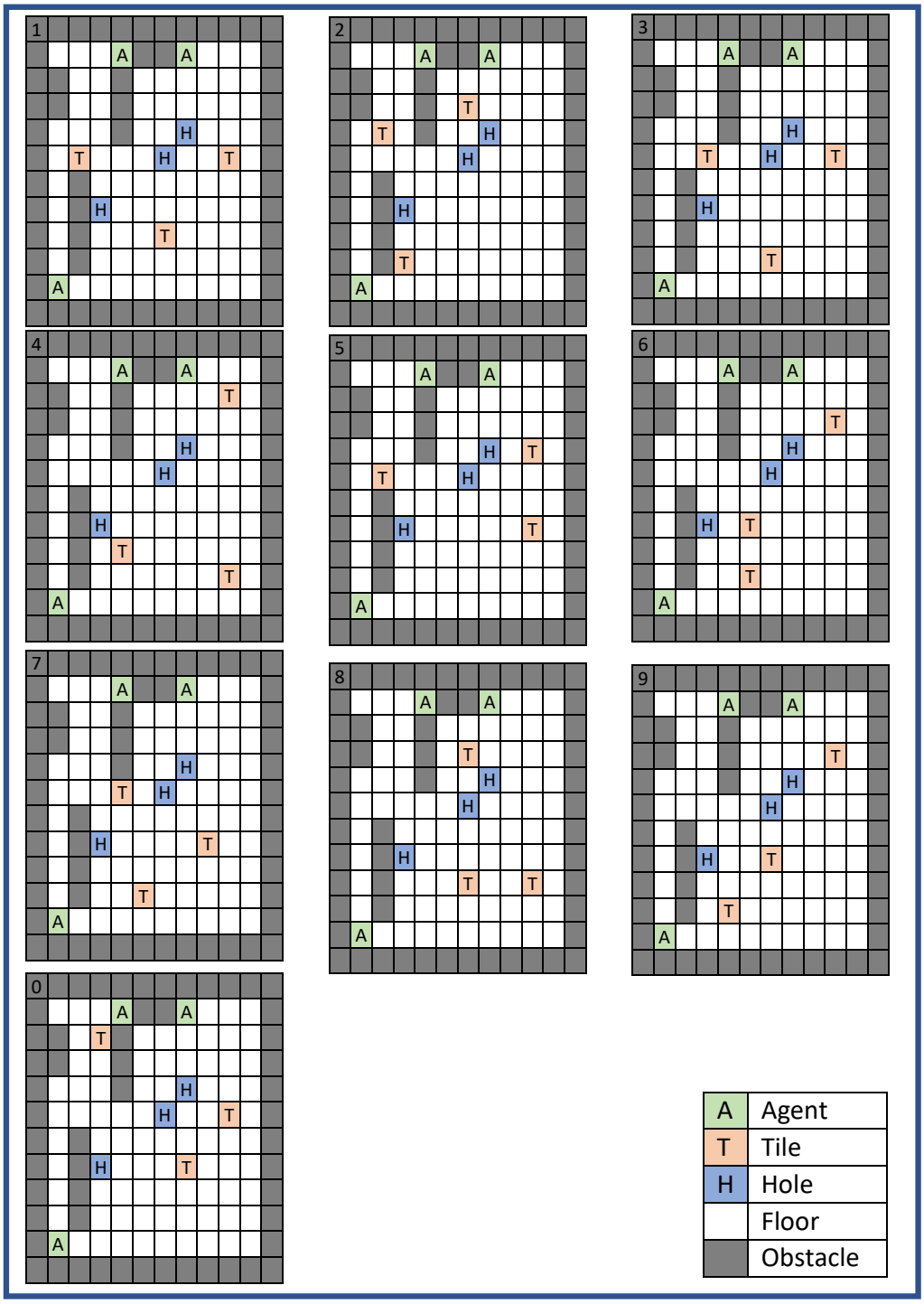


Figure 1.4 Random Tiles initialised position (from Training Set (1))

1.5 Significance of the research

It has been observed that the best performing GNP-RL-based algorithm on a multi-agent, dynamic environment problem such as the tile world problem is still not sufficiently adequate enough to solve the problem at an acceptable level, the best results so far are in 2018 (19/30 = 63.33% [1]). To address this problem, we have developed an improved GNP-RL architecture that uses an optimal search algorithm as one of its judgment nodes. Also, we have devised a new fitness function that can both reward and penalize an agent's action accordingly, while giving a score proportional to the agent's goal achievements. Most of the previous works on the tile world problem with GNP-RL did not consider using penalty, nor address the actual results on this problem, but the results were presented based on the fitness function which was not useful in comparing the results especially when changing the fitness function, so in this work we will show the results in both ways. Adding to this, we have improved the RL in the algorithm to train the agents more sufficiently to use the graph, by teaching them how to avoid some troubling situations and to prevent collision between the objects. Moreover, we have added a priority tasks execution technique to the agents, so they trained how to implement the tasks prioritize. We have tested the algorithms on two kinds of environments; simple and complex environments. Comparing with the other works on GNP-RL with tile world problem [9], [2], and [1] , the implementation in this work demonstrates the effectiveness of the produced techniques on the GNP-RL with (100%) of training accuracy with the training set (1), (96.66%) of testing accuracy on test set (2) which randomly initialized the tile locations, and (86.66%) of testing accuracy in testing set (1) from [2].

1.6 Research methodology

In the first stage of this work, for direct comparison against other existing GNP-RL architectures, we have tested and compared all proposed algorithms using the same multi-agent benchmarking problem, known as tile world (see section 1.4 Overview of the problem domain), including the reported training and test environments, as well as the same fitness functions. Next, we analyzed the performance of the algorithms based on fitness and goal achievement. It was observed that the fitness score is not proportional to the number of goals achieved by the agent. We rectified this problem (page 47) by changing the weights assigned to the different components of the fitness function, as well as a penalty component.

1. Study and implement the original GNP-RL algorithm [9]. Test algorithm on the tile world problem. Analyse experiment results. Identify strengths and weaknesses. Here, the objective is to replicate the same results (same accuracy) as reported in the literature.

2. Study and implement the Variable-Sized GNP-RL algorithm [2]. Test algorithm on the tile world problem. Analyse experiment results. Identify strengths and weaknesses. Here, the objective is to replicate the same results (same accuracy) as reported in the literature.
3. Study and implement the diversity using rank space method.
4. Apply the diversity on the crossover in the GNP-RL. Test algorithm on the tile world problem. Analyse experiment results. Identify strengths and weaknesses. Here, the objective is to improve the results.
5. Apply the diversity on the crossover and on choosing elite in the GNP-RL. Test algorithm on the tile world problem. Analyse experiment results. Identify strengths and weaknesses. Here, the objective is to improve the results.
6. Use the reward and penalise when updating the Q-value for the GNP-RL. Test algorithm on the tile world problem. Analyse experiment results. Identify strengths and weaknesses. Here, the objective is to improve the results by training the agent on how to avoid trapped in trouble situations. (Constraint conformance GNP-RL)
7. Add A* with using Manhattan Heuristics to the GNP-RL. Test algorithm on the tile world problem. Analyse experiment results. Identify strengths and weaknesses. Here, the objective is to improve the results by detecting goals accurately.
8. Add A* with using Manhattan Heuristics to the Constraint conformance GNP-RL. Test algorithm on the tile world problem. Analyse experiment results. Identify strengths and weaknesses. Here, the objective is to improve the results by detecting goals accurately.
9. Use performing tasks by priority on the (A* & Constraint conformance GNP-RL). Test algorithm on the tile world problem. Analyse experiment results. Identify strengths and weaknesses. Here, the objective is to improve the results by training the agent on how to execute the tasks by priority to avoid the conflict and collision between them.
10. Implement the experiments on two kinds of environments (simple, complex). Test algorithm on the tile world problem. Analyse experiment results. Identify strengths and weaknesses. Here, the objective is to check which is the best in training the multi-agent (simple or complex environments).
11. Test the experiments from the first generation to the last one. Test algorithm on the tile world problem. Analyse experiment results. Identify strengths and weaknesses. Here, the objective is to check if the algorithm enters into the over-training phase.

1.7 Structure of the thesis

This thesis consists of 8 chapters. Chapters 3 through 8 introduces a specific variant of the GNP-RL algorithm. We provide details on the evolutionary mechanisms employed, the related literature for that variant of the algorithm, as well as the empirical results and analysis.

Chapter 2 shows a background of the (GA), (GP), and (GNP) algorithms.

Chapter 3 provides an implementation of the GNP-RL on the Tile world problem following the same parameters on [9], and detect the limitation of this implementation.

Chapter 4 addresses the VSGNP-RL algorithm as in [2] on the same benchmark and detects the strengths and limitations of the algorithm and the used fitness function.

Chapter 5 presents our own novel extension to the GNP-RL architecture, using the diversity in the crossover and with choosing the elites on the GNP-RL. Shows the results, detects the strengths and limitations of this technique.

Chapter 6 provides our own novel extension to the GNP-RL architecture incorporating a new reward and punishment scheme when updating the Q-value. Shows the results, detects the strengths and limitations for using this technique.

Chapter 7 shows our own novel extension to the GNP-RL architecture, adding A* algorithm as a judgment node. Shows the results, detects the strengths and limitations of this technique.

Chapter 8 presents our own novel extension to the GNP-RL architecture, adding a task prioritization method. Shows the results, detects the strengths and limitations of this technique.

Chapter 9 summarizes the results and analysis of all the previous chapters, then states our conclusions, as well as identifies some future works.

Chapter 2 Background

2.1 Genetic Algorithm (GA)

2.1.1 Introduction

The series of genetic algorithms began in 1975 by Holland [3], who mimicked genetic evolution in humans by producing a genetic algorithm (GA). This algorithm was used to find the best solution through a random search, by creating a sample of random solutions and generating new solutions by applying two operations; crossover, and mutation. Holland represented solutions with the chromosome structure so that each solution had a set of properties represented by genes. Chromosomes were represented in the genetic algorithm with a series of binary numbers. Each problem has an equation through which solutions are evaluated called Fitness. The success of this algorithm depends heavily on the composition of the chromosome and on the used Fitness that are produced to solve a particular problem [15]. The genetic algorithm used to solve a wide range of problems, such as; credit card fraud detection [16], edge detection [17], and breast cancers detection [18].

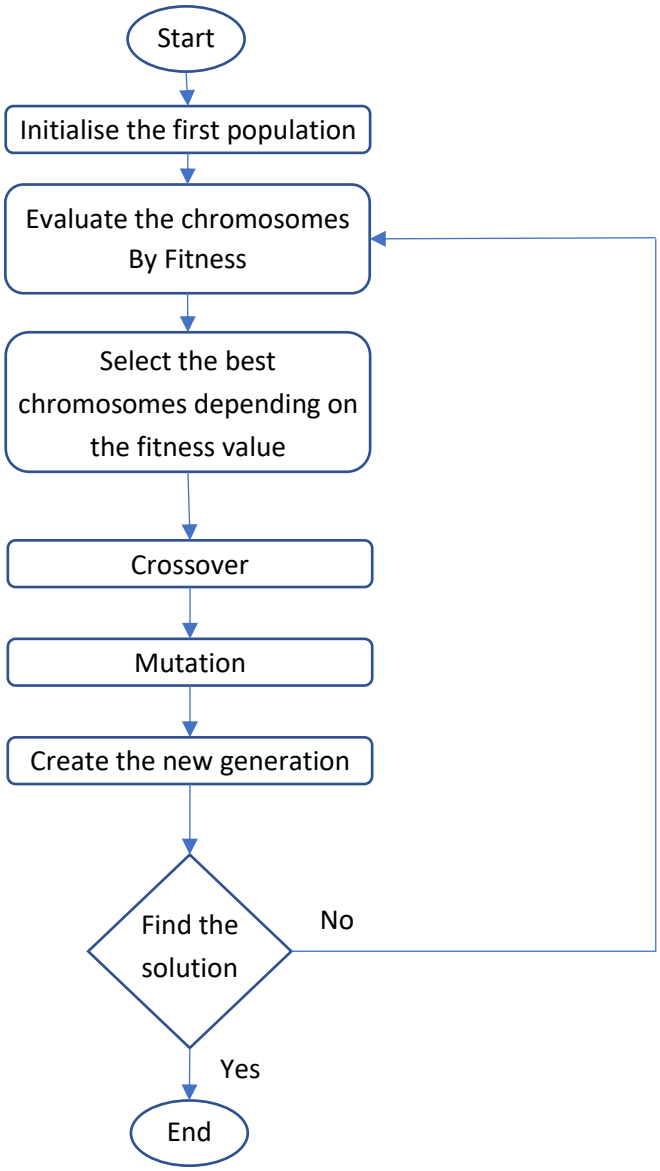


Figure 2.1Genetic Algorithm Flow chart [3]

As presented in Figure 2.1 the genetic algorithm steps. Firstly, the algorithm starts with initializing the first population; then it evaluates all the chromosomes in the current generation by using a suitable Fitness function. After that, the algorithm selects the best solutions by using one of the selection methods. The next step, it applies the evaluation operation on the chromosomes to generate new ones by using the crossover and mutation. If the algorithm finds the best solution or if it reaches a specific number of generations, it stops, otherwise, it returns the steps from the evaluation step.

2.1.2 Chromosome structure

Each individual is a string of zeros and ones, which are the encoding of the chromosome properties [15]. For example, if the solution is a rectangle, the chromosome will be as Figure 2.2.

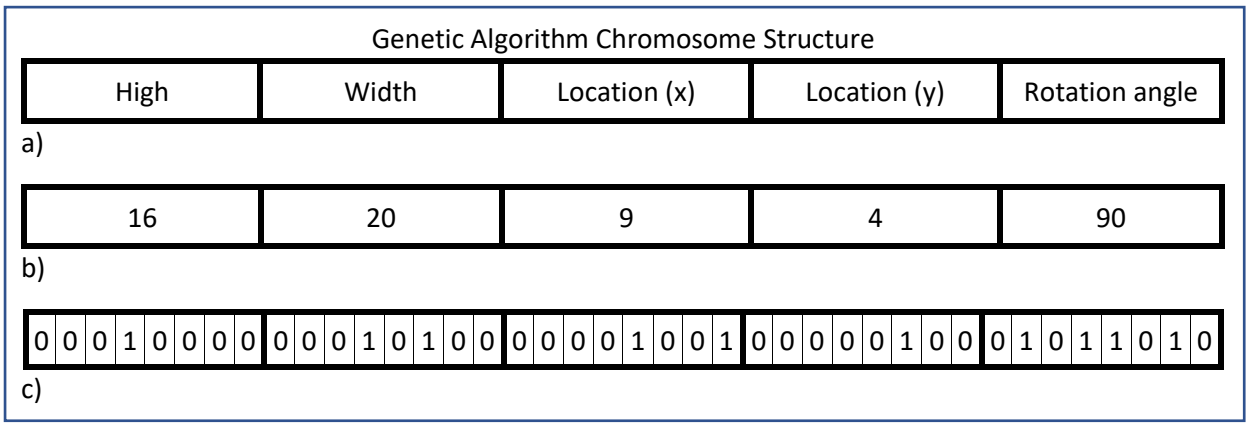


Figure 2.2 Genotype Genetic Algorithm Chromosome Structure. a) The chromosome features. b) The chromosome features' values. c) The encoding for the chromosome

As the rectangle could be described with the high, width, location (x,y), and rotation angle, these properties can be the genes of the chromosome that represent the solution.

2.1.3 Initialise the first population

When the algorithm starts, the first population is initialized randomly. So, each chromosome is a string of zeros and ones which are produced randomly depending on the solution conditions [15].

2.1.4 Evaluating and Selection

Evaluating

The Fitness function is used to evaluate the individuals to choose the best chromosomes to send them to the new generation. There are different kinds of Fitness functions, the fitness type depends on the problem domain and which one is suitable. In 2009 Nelson analyzed seven different types of Fitness function and compared them [19]. For example, the number of correctly dropped tiles in the Tile word problem [9], or Euclidean distance function as in Travelling Salesman Problem [20].

Selection

After evaluating the chromosomes in a generation, the algorithm uses one of the selection methods to select specific individuals to send them to the new generation. There are

different types of selection such as; truncation selection [21], tournament selection, linear and exponential ranking selection [22]

2.1.5 Evolutionary Operations

Crossover

Crossover is an operation that is applied to two chromosomes to generate new two chromosomes which share some features from each parent [3]. These two parents can be chosen depending on one of the selection methods. The algorithm detects a point in the string to divide the parents on it and to crossover them on this point. In some cases, the algorithm set two or more points that the chromosomes could be divided on. In [23], Schaffer proved how much the location of the divided point could affect the results. Figure 2.3 shows the way that the algorithm crossover two parents and generate two new offspring.

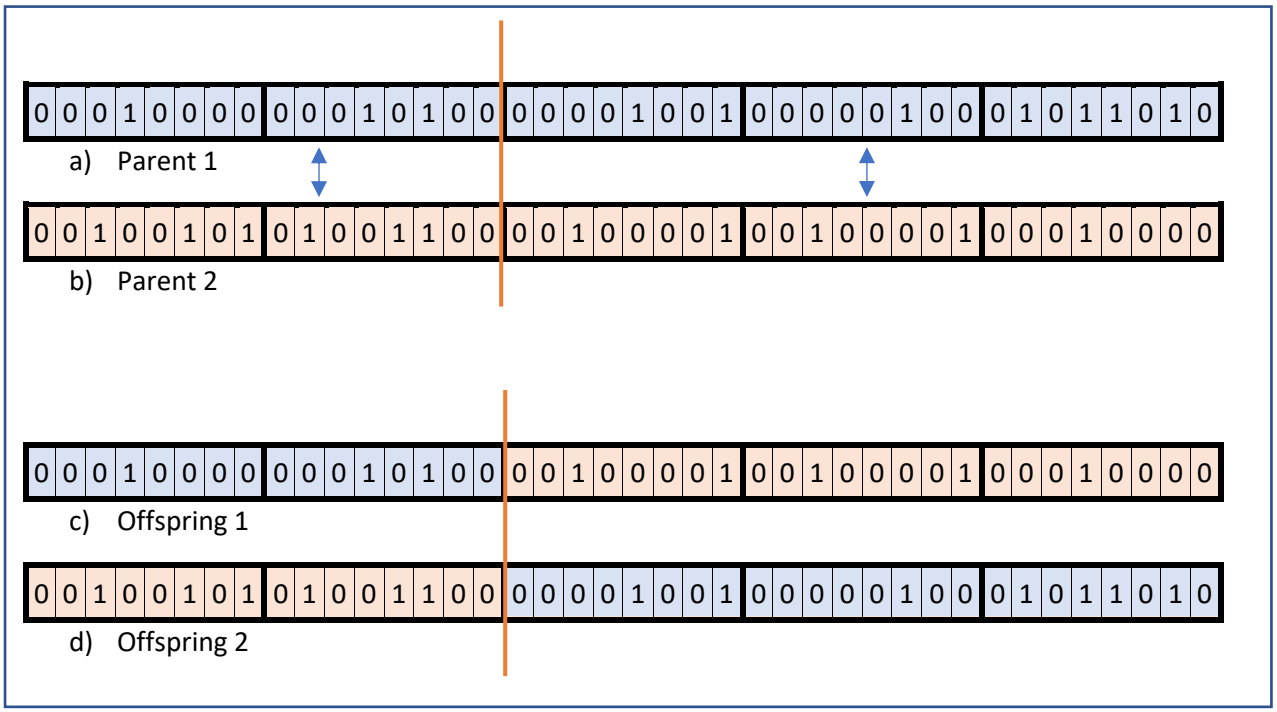


Figure 2.3 Phenotype Genetic Algorithm Crossover Operation

Mutation

The mutation is the operation that can be applied on one chromosome to create a new one offspring by changing a specific feature inside the string [3]. Usually, mutation uses to ensure some diversity in the population. There are many types of GA mutation such as; Fogel produced Gaussian and uniform mutation in linear systems in 1990 [24], Larranaga tested swap and insertion method in the Travelling Salesman Problem in 1999 [25], and polynomial and

power mutation which was introduced by Deep in 2007 [26]. Figure 2.4 illustrates mutation operation in the genetic algorithm with swapping the genes inside the string.

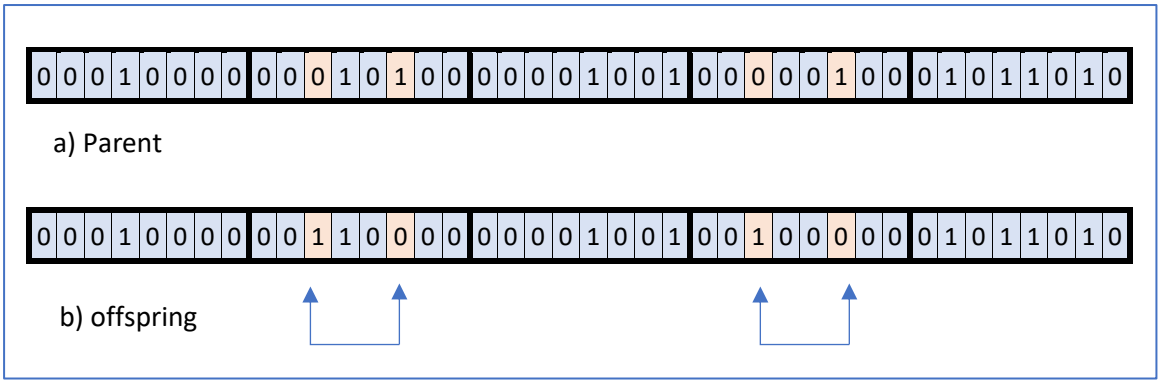


Figure 2.4 Genetic Algorithm Mutation Operation (swap genes)

2.1.6 Summary

The genetic algorithm was implemented to simulate genetic evolution in living organisms. It uses the chromosome structure to represent the solutions, and with the evolutionary operations, new solutions are created to generate a new population.

Since the GA has a liner structure, it is difficult for it to solve the more complicated problems, especially the ones that need a complex instruction to solve [27]. To overcome that problem, a new algorithm was produced which called Genetic Programming (GP). The next section will discuss it.

2.2 Genetic Programming (GP)

2.2.1 Introduction

In 1992 [4] [5], Koza came with a new structure to represent the solution which uses the solutions that have if-then states with a non-linear individual. He came with a structure that simulates making decision rules, so he used a tree structure to represent the chromosome. This algorithm uses with a more complex problem to find a set of rules that solve the problem. It used to solve a biochemistry application [28], Breast cancer diagnosis [29], in the financial area [30], and so in.

2.2.2 Chromosome structure

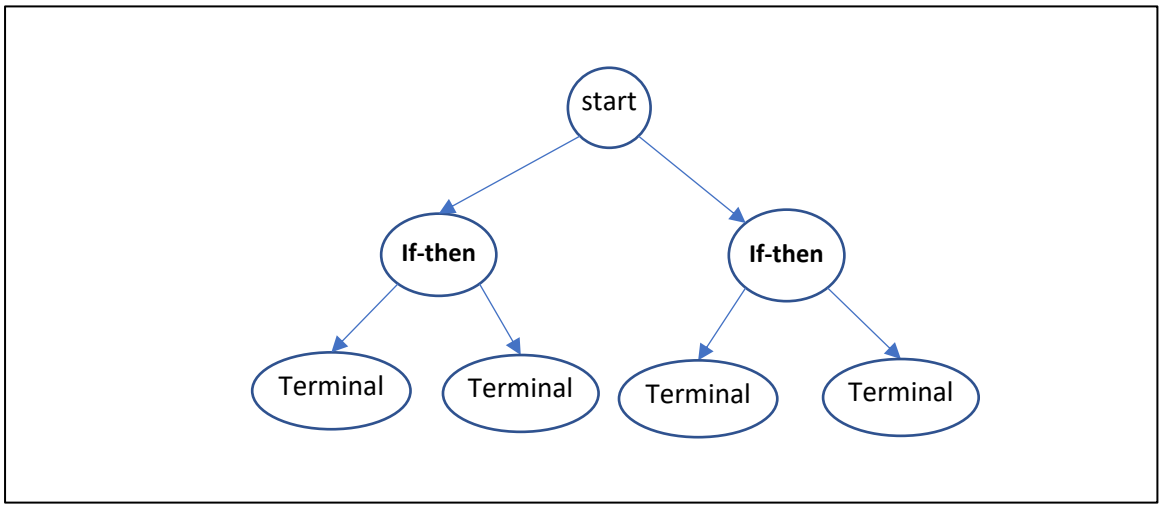


Figure 2.5 Phenotype Genetic Programming Chromosome Structure

Each chromosome represented with a tree structure with nodes. Tree root is the start node for the solutions, all the terminal nodes are an action node, and if-then nodes are judgment nodes that use to take a decision.

2.2.3 Initialise the first population

Before the algorithm starts, the first population is initialized by randomly creating a set of trees that represent the solutions. In the first population, the tree length and depth are either equal to a predefined number (called Full method) or less than or equal to a predefined value (called Grow method) [27]. The depth of each tree could be changed by using crossover and mutation.

2.2.4 Evaluating and Selecting

After initialized the first population, a Fitness function should be applied to each chromosome to test the chromosomes and evaluate them. The Fitness function could be any type of evaluated functions that suited to the problem. For instance, the Fitness can be determined as the difference between its result error and the optimal value of the solution [31], in the classification problem, the fitness was calculated as the rate of the correct classes on the total number of classifications [32], and so on. The chromosome with the higher fitness value is the chromosome with a higher probability to be selected [32].

2.2.5 Evolutionary Operations

Crossover

The algorithm selects two parents using one of the selection methods and applies the crossover between them to create a new two children. In the tree structure individual, a node from the tree is chosen randomly from each parent to exchange the nodes and their offspring between them [27].

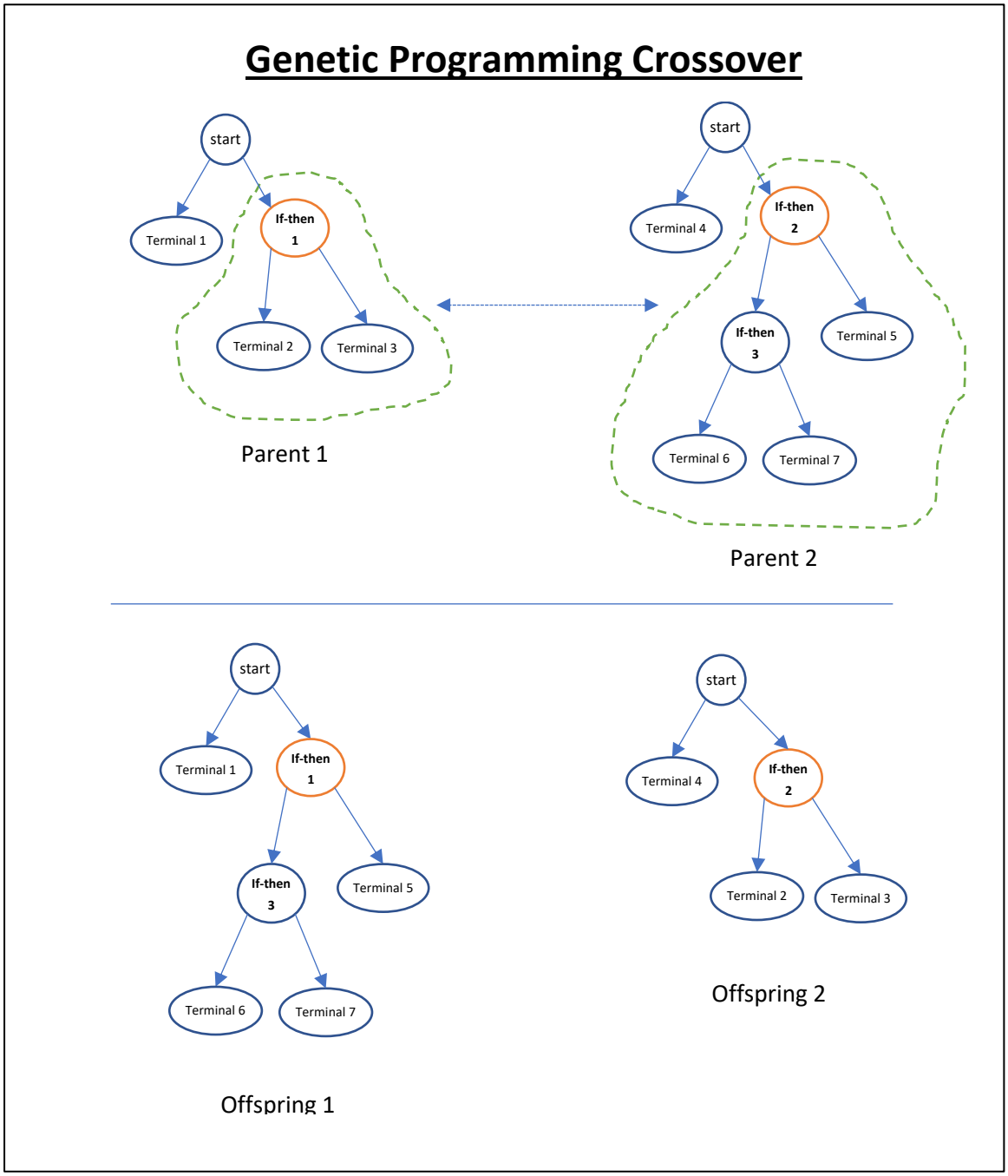


Figure 2.6 Genetic Programming Crossover Operation modified from [27]

Mutation

A mutation is an operation which uses to ensure the diversity in the population by making a change inside the individual itself. In GP a random node from the tree is selected with its children to change it with a new randomly generated node. Since the chromosome has a tree structure, losing diversity is not a major problem in the GP. Because of that, the crossover in the GP is more effective than the mutation [27].

Hence the crossover and mutation can increase the tree depth and length, an important problem can occur, which is a significant increase in the search space and in the needed time to evaluate the individuals. That need more memory size and high processor performance. To avoid this problem, a new algorithm was implemented, which is Genetic Networking Programming. This algorithm will be discussed in the next section.

2.2.6 Summary

Genetic programming algorithm was produced as a non-linear chromosome structure to solve decision-making problems. It uses a tree structure with start root, if-then nodes, and terminal nodes. One of the major limitations for the GP is the possibility of the tree to become oversized because of the evolutionary operations. To solve this issue, a Genetic Networking Programming Algorithm (GNP) has been produced.

2.3 Genetic Networking Programming (GNP)

2.3.1 Introduction

To overcome the oversized problem in the GP, a new algorithm was presented by Katagiri in 2000 called Genetic Networking Programming (GNP) [33]. GNP was implemented to allow for multi-used of nodes. So, each node can be used more than one time. And there is no terminal node. By using this mechanism, the oversized problem is solved. GNP used with multi-agents systems, such as ants behavior [34], online learning [35], and Prisoner's Dilemma Game [36].

2.3.2 Chromosome structure

The structure for the GNP chromosomes is a network structure (graph) which contains a set of nodes see Figure 2.8. There are three types of nodes; judgment node which match (if-then) node in the GA, Processing node which match (terminal) node in the GP, and start node [8]. Judgment node uses to check for a specific condition, Processing node uses to apply an action, start node is the node which the simulation starts with. When the algorithm begins, it visits the start node; then it follows the connections to transfer to the next node. Each node could be visited more than one time, and there is no terminal node. That means, the algorithm will continue working on the graph until one of the two states happened; either the available time for the individual is finished, or the algorithm finds the solution. Each node has connections to guide to the next node, the processing node could be connected from more than one node, and it has just one connection to the next node. The judgment node could be connected from only one node, and it has more than one connection, each one of them direct to a specific node that is the answer to this judgment node [36].

Node-id	Node type	Delay time	Connections
Gene(node)#1	Judgment node	Dt#1	C#1, C#2, …
Gene(node)#2	Processing node	Dt#2	C#1, C#2, …
Gene(node)#3	Processing node	Dt#3	C#1, C#2, …
. . .	. . .	. . .	…. …. ….

Figure 2.7 GNP Genotype expression for Chromosome Structure modified from [33]

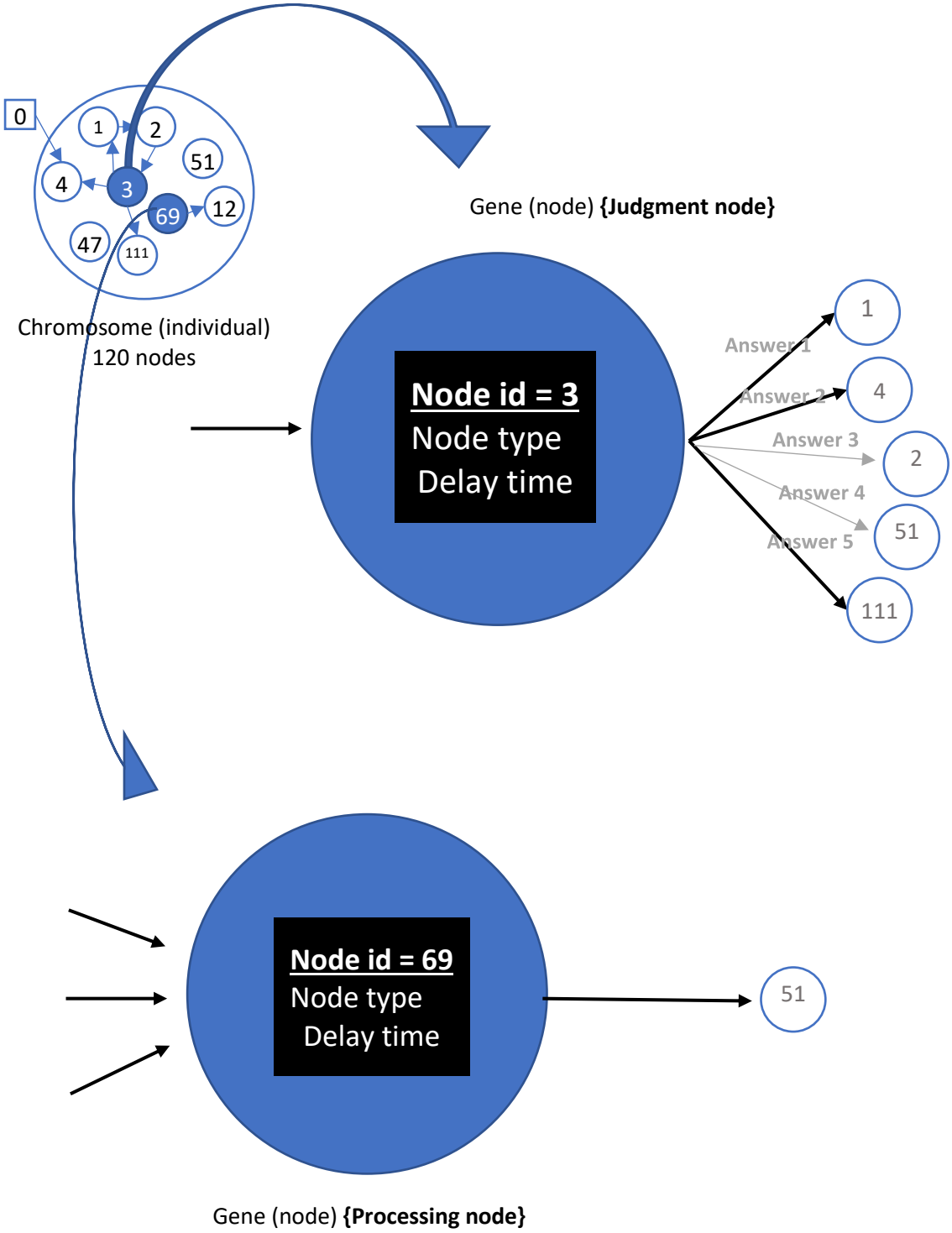


Figure 2.8 GNP Phenotype Chromosome Structure modified from [33]

2.3.3 Initialise the first population

For the first population, a specific number of nodes per chromosome is generated randomly. So, each chromosome has the same amount of judgment and processing nodes, but the connections between them are different from each other.

2.3.4 Evaluating and selection

To evaluate the chromosome, the algorithm should run on the graph by starting with the starting node, then continue following the connections and apply the actions of the nodes on the problem environment with considering the delay time which is a predefined time that the algorithm spends it on a specific node. The delay time uses to give a time limitation to the algorithm to stop working on one chromosome and work on another [36]. To calculate the fitness value, GNP can use any type of Fitness function that suited to the problem and checks for the feedback from the environment to get a final evaluated value [35].

2.3.5 Evolutionary Operations

Crossover

Crossover in GNP used to combine some features from parent 1 with some features from parent 2 to generate a new two offspring which inherit some genes from their parent. In GNP, two chromosomes are chosen with one of the selection methods and used as a parent. Some nodes from each parent are selected randomly with a crossover probability to be exchanged between the parents to generate the children, as shown in Figure 2.9 [37].

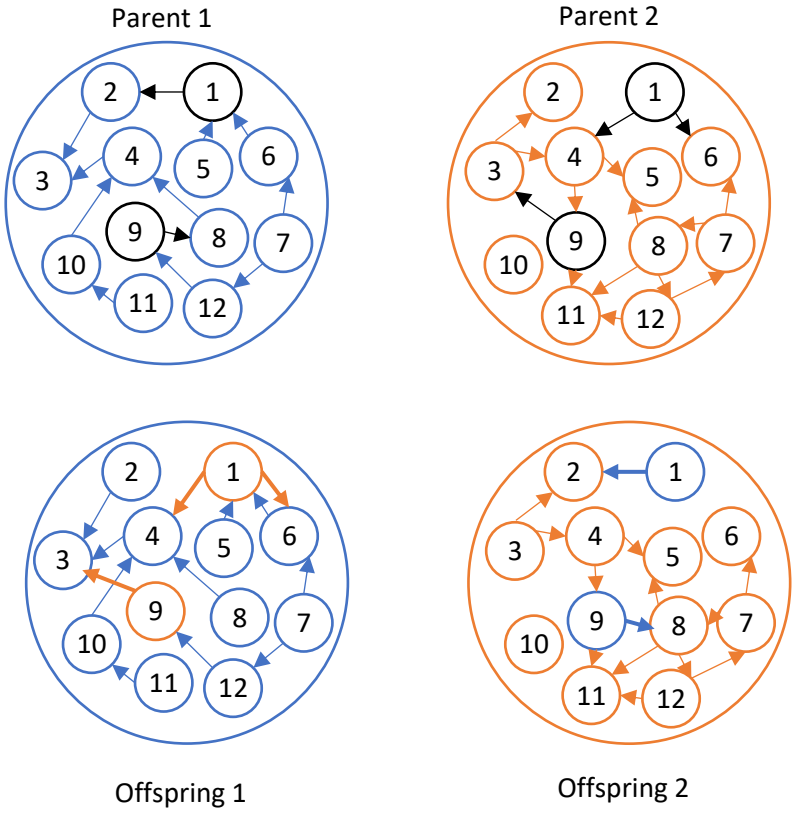


Figure 2.9 GNP Crossover modified from [37]

Mutation

There are many ways to apply mutation on the GNP. One of them used by Mabu in 2010, which change the connection for a randomly selected node from the graph with a new random connection [37]. Figure 2.10 illustrates that.

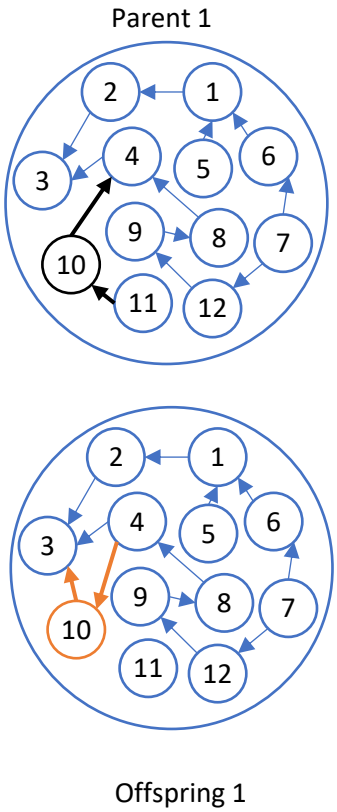


Figure 2.10 GNP Mutation modified from [37]

2.3.6 Summary

In the GNP algorithm, the problem of increasing the chromosome size has been solved by implementing a chromosome with network structure (graph) that able to solve a complex problem by allowing the multi-using nodes and infinity graph.

Although the algorithm has been used to solve some complex problems, in some problems it needs to increase the probability by increasing the number of nodes so that the size of the chromosome becomes large and to solve this problem a new algorithm has been produced. This algorithm will be discussed in detail in the next chapter.

Chapter 3 (Review) GNP-RL

3.1 Motivation

Using GNP-RL to find the most suited trained graph for the Tile world problem is an exciting way to combine the genetic networking programming randomization with the reinforcement learning algorithm for multi-agent systems. GNP is working to generate random individuals (graphs), crossover, and mutate them to create generation after generation, while the RL is used to evaluate each chromosome and train the agent how to use this chromosome to solve the problem. We choose this algorithm because of its extensibility and flexibility.

3.2 Related work

Shingo Mabu was the first one who implemented GNP with reinforcement learning algorithm using the Sarsa algorithm in the tile world problem [9] in 2007. He divided each node to some sub nodes and ran the RL algorithm to train the agent on the graph to choose the best sub node with the maximum Q-value. This method was active with the dynamic environments and multi-agents' systems, and it was used in many applications such as in making mobile robot behavior [10] and in the Stock trading model [11]. In 2013 Siti SENDARI combined fuzzy with the GNP-RL [38] for Mobile Robot. A distributed graph with a fixed and variable sized distributed graph was proposed by Mabu in 2014 [2] by dividing the graph to subprograms as a way to separate the problem to small situations this work showed better results than the previous ones.

3.3 The algorithms

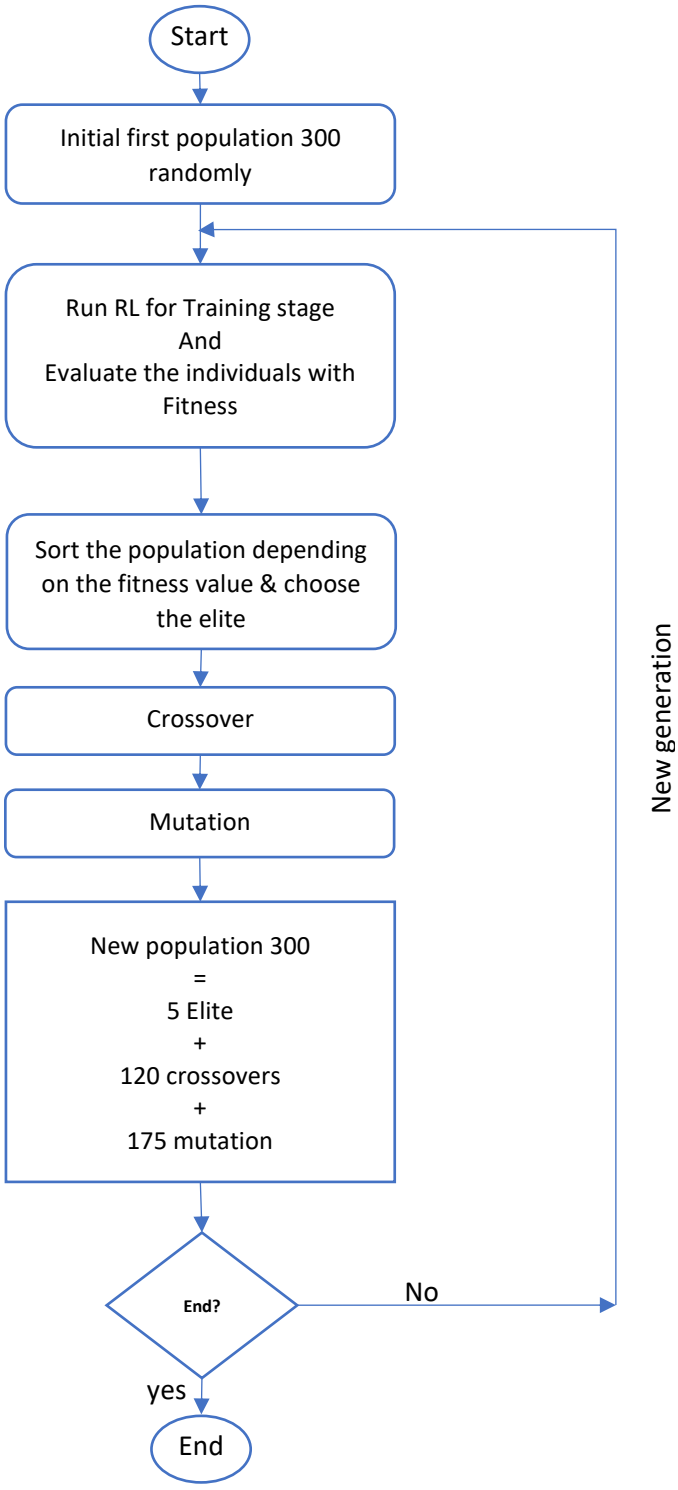


Figure 3.1 GNP-RL flow chart from [13] with modifying

The GNP-RL algorithm works to find the best solution randomly, which represent as a graph. This algorithm starts with a random population which has 300 chromosomes (solutions); these random individuals are created depending on some conditions (see section Initialise the first population).

After generation the first population, each chromosome will be evaluated by applying a fitness function on it and by running the Reinforcement Learning on it to train the agents which sub node is the best (see section Evaluate the chromosome). To produce the new generations, evolutionary operations should be applied on the chromosomes to generate new descendants for the new population which have better features than the parents. These operations are crossover and mutation, the crossover is the method that uses to create new two children from two parents in this implementation the number of offspring that produce from the crossover is 120 individuals, and the mutation is used to make a change on one individual to create another, the number of individuals that produced using mutation is 175 (see section Evolutionary Operators).

The new generation has 300 individuals; the best 5 chromosomes from the current generation, 120 offspring which are produced from the crossover, and 175 individuals from the mutation. After creating the new generation, the algorithm repeats from the evaluation step until finding the best solution or after a specific number of generations.

Population Size	**Crossover**	**Mutation**	**Elite**	**Max # sub nodes**
300	120	175	5	4
# Nodes	**# Judgment nodes**	**# Processing nodes**		**Tournament Size**
120	80	40		7
Crossover rate Pc	**Mutation rate Pm**	**ε**	**γ**	**α**
0.1	0.01	0.1	0.9	0.9
Delay time	1 → Judgment node 5 → Processing node		**Agent Steps**	60 for each agent

Table 3.1 GNP-RL parameters

3.3.1 Individual (chromosome) structure

<table>
<tr><td rowspan="2">Node-id</td><td rowspan="2">Node type</td><td rowspan="2">Delay time</td><td colspan="5">SubNodes (SN)</td></tr>
<tr><td colspan="2">SN#1</td><td>SN#2</td><td>SN#3</td><td>SN#4</td></tr>
<tr><td rowspan="7">Gene(node)#1</td><td rowspan="7">0=start node.
1=judgment node.
2=processing node.</td><td rowspan="7">1→ JN.
5→PN.</td><td>Q-value</td><td>SN1-Q</td><td rowspan="7">….</td><td rowspan="7">…..</td><td rowspan="7">…..</td></tr>
<tr><td>ID</td><td>SN1-ID</td></tr>
<tr><td rowspan="5">Connections</td><td>SN1-C#1</td></tr>
<tr><td>SN1-C#2</td></tr>
<tr><td>SN1-C#3</td></tr>
<tr><td>SN1-C#4</td></tr>
<tr><td>SN1-C#5</td></tr>
<tr><td>Gene(node)#2</td><td></td><td></td><td colspan="5"></td></tr>
</table>

Table 3.2 GNP-RL Genotype Chromosome Structure (JN → Judgment node) (PN → Processing node) modified from [9]

Each chromosome composed of 120 genes. Each gene (node) has four features (node-id, node type, delay time, and sub-nodes). Each node has a maximum of four sub-nodes; each one of them has three characteristics (Q-value, sub-node ID, and connections).

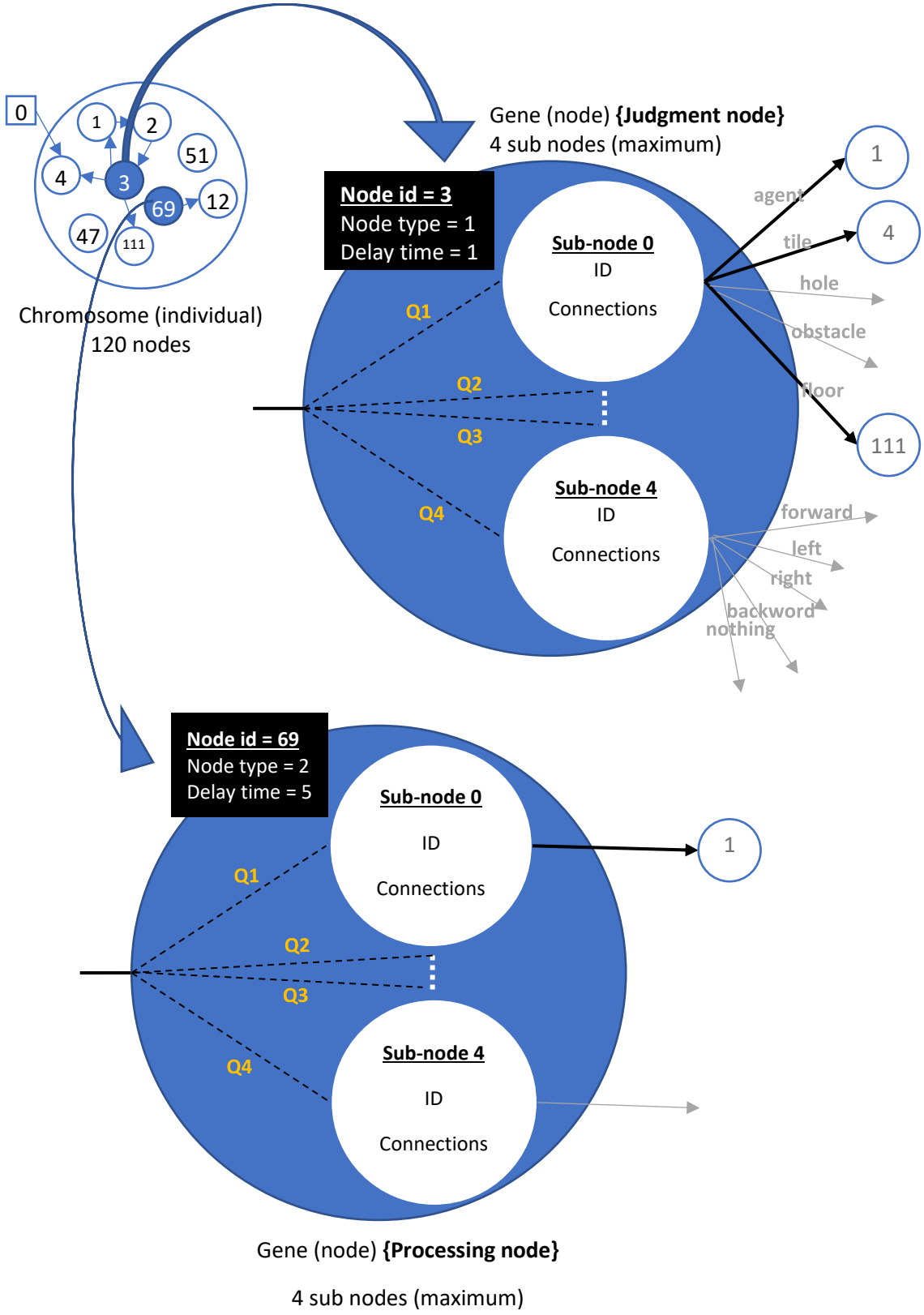


Figure 3.2 GNP-RL Phenotype Chromosome Structure modified from [9]

Node-id: identical number for each node.

Node type: there are three types of the node:

1- Start node → there is one start node in each chromosome which directs to the node where the algorithm will start. [39] In 2006, Tadahiko Murata used more than one start node for the multi-agent system (one start node for each agent).
2- Judgment node → this kind of node is used to ask and answer some questions according to the agent situation. There are eight functions of judgment node:

   J1- What is in the FORWARD (JF).

   J2- what is in the BACKWORD (JB).

   J3- what is in the LEFT (JL).

   J4- what is in the RIGHT (JR).

   J5- what is the direction to the nearest TILE from the agent (TD).

   J6- what is the direction to the nearest HOLE from the agent (HD).

   J7- what is the direction of the nearest HOLE from the closest TILE (THD).

   J8- what is the direction of the second nearest TILE from the agent (STD).

   There are five possible answers for each one of these judgment nodes:

   - For the nodes (J1, J2, J3, and J4), the answer should be one of (AGENT, TILE, HOLE, FLOOR, or OBSTACLE).
   - For the nodes (J5, J6, J7, and J8), the answer should be one of (FORWARD, BACKWARD, LEFT, RIGHT, or nothing).

3- Processing node → this type of node takes one of these actions (functions) on the environment:
   a. Move Forward (MF).
   b. Turn Right (TR).
   c. Turn Left (TL).
   d. Stay (ST).

Delay time: the delay time is a chosen time for each type of node to give a theoretical value for the node execution, which was firstly used by Shingo Mabu in 2007 [9]. It is used to calculate the agent's steps when the algorithm runs, by giving each agent 60 steps, each step could have up to eight delay time, which are total of the delay time for a sequence of the nodes. In all the experiments in this work, the delay time for the judgment node costs 1, and for the processing node costs 5 as it was chosen by Shingo Mabu in 2014 [2].

Q-value: hence, each node has more than one sub-node, reinforcement learning (Sarsa) has been used to train the agents to use the graph. So, each sub node has a Q value, which initialized by 0. Each time the agent visits the node, it chooses one sub-node to execute. The

way that the agent uses to select the sub node will be explained in section (Evaluate the chromosome) below.

ID: each sub node has a function or (ID). This function should be one of the processing and judgment functions. If the node is a judgment node, all its sub nodes should be judgment and vice versa.

Connections: for every sub-node, there are some connections to direct the agent to the next node. The number of connections for each sub-node depends on the sub node function.

1- For J1, J2, J3, and J4 the answer is one of (AGENT, TILE, HOLE, FLOOR, or OBSTACLE), and for J5, J6, J7, and J8 the answer is one of (FORWARD, BACKWORD, LEFT, RIGHT, or nothing). There are five possible answers, so there are five connections from this sub-node to other nodes in the graph. When the algorithm works, it chooses the connection that reflects the answer for this node to go to the next node.
2- For FW, TR, TL, and ST, there is just one connection that because this type of node is organised to execute the process. So, after processing the action, the algorithm directly goes to the next node by following the one connection for this type of node.

3.3.2 Initialise the first population

Before the algorithm starts, 300 individuals are created randomly using the same technique in [2]. Each chromosome has one start node and 120 (80 judgment & 40 processing) nodes. The start node is chosen randomly from the 120 nodes. Each gene (node) is created depending on these rules:

1- Node type: randomly created (judgment or processing) (80 & 40) respectively.
2- Delay time: 1 for the judgment node and 5 for the processing node.
3- Sub node number: each node has several sub-nodes. This number is initialised randomly from 1 – 4.
    a. Q-value: each sub node has 0 as initial value for the Q.
    b. ID: for the judgment node, the ID for its sub nodes could be randomly any number from 1 to 8. For the processing node, it could be randomly from 1 to 4.
    c. Connections: each sub node has several connections, as described previously. Each connection direct to a randomly node-id this node-id should not be the same node that this sub-node is part of to prevent an infinite loop.

3.3.3 Evaluate the chromosome

To evaluate any chromosome, the reinforcement learning runs on this chromosome for the three agents on the tile world environments that is one of the complex problems in which

many algorithms have been applied on this problem which was described by Pollack in 1990 [14]. And apply the fitness function to evaluate and calculate the fitness value for everyone. There are ten environments as shown in Figure 1.1 Training Set (1) which was the same training environment that used in [2], each environment has three tiles, three holes, and three agents. The three agents should push the three tiles into their nearest holes. Once the agent pushes the tile to the hole, the tile and the hole disappear, and a floor will be in their locations. After the agents push all the three tiles to their holes, the environment will be changed to the next one. If the agents spent all their steps before accomplishing the goal, the environment would be changed to the next one.

Each agent starts from the start node and follows the connection to visit the next nodes. When visiting the node, the agent should choose one of the sub-nodes. To select one sub-node Sarsa with ε-greedy technique is used, which was first implemented by Sutton [40]. By setting the probability of randomly selected the subnode with 0.1 which called exploration, and 0.9 for the sub node with the maximum Q-value which called exploitation.

After visiting the node, the Q-value will be updated according to Equation 3.1. Where Qip is the Q value for the visited sub node, Qjq is the Q-value for the chosen sub-node in the next node, α is a learning rate it could be any value from 0 to 1, in all the experiments in this work a 0.9 was chosen for it, and γ is a discount rate it is a value from 0 to 1 in this work it is 0.9. The Reward is equal to 1 for each time a tile is pushed into a hole. The previous method is described in Pseudo Code 3.1 GNP-RL .

$$Qip = Qip + (\alpha * (Reward + (\gamma * Qjq) - Qip)) \quad \textit{Equation 3.1 Update Q-value [9]}$$

Finally, after finishing the ten environments, the fitness function is calculated by Equation 3.2 Fitness (1), which was produced by Mabu in 2014 [2]. Where D_{tile} is the number of correctly dropped tile in the hole for each environment, $D_{distance}$ is how many times the agent pushes the tile closer to the hole, and T_{remain} is the remain agent steps when all tiles are dropped in the holes.

$$Fitness\ (1) = \sum_{env=1}^{ENV} [(100\ \times D_{tile}) + (20\ \times\ D_{distance}) + (T_{remain})] \quad \textit{Equation 3.2 Fitness (1) [2]}$$

The difference between the training stage and the testing stage is that the training phase uses the ε-greedy technique which has both the exploration and the exploitation, whereas in the testing phase the agent chooses only the sub node with the maximum Q-value that has been settled in the training stage. The second difference is that the Q-values will not be updated any more on the testing (see Pseudo Code 3.2 GNP-RL Testing).

GNP-RL Simulation

```
Int Apply_Judgment(ID)
1.      Switch(ID):
2.              Case(1): answer ← what is in the Forward
3.              Case(2): answer ← what is in the Backword
4.              Case(3): answer ← what is in the Left
5.              Case(4): answer ← what is in the Right
6.              Case(5): answer ← what is the direction to the nearest tile
7.              Case(6): answer ← what is the direction to the nearest hole
8.              Case(7):answer ← what is the direction from the nearest tile to the nearest
                hole
9.              Case(8): answer ← what is the direction to second nearest tile
10.     Return answer

Int Apply_Processing (ID)
1.      Switch(ID):
2.              Case(1): Go Forward; IF (tile is dropped into the hole) reward ←1
3.              Case(2): Turn Left
4.              Case(3): Turn Right
5.              Case(4): stay
6.      Return reward

Main()
1.      double RL ← 0.1
2.      NEXT_NODE ← START_NODE
3.      CURRENT_NODE ← NEXT_NODE
4.      rand ← RAND(0-1)
5.      IF rand<RL
6.              SELECTED_SUBNODE ← RAND[1-CURRENT_NODE.subnodenum)] //exploration
7.              IF CURRENT_NODE type is Judgment
8.                      ANSWER←answer the judgment with the current subnode ID
9.                      NEXT_NODE←the connection that reflects the ANSWER
10.             ELSEIF CURRENT_NODE type is Processing
11.                     REWARD←Apply_Processing(current subnode ID);
12.                     NEXT_NODE←the first connection from the SELECTED_SUBNODE
13.     ELSE
14.             SELECTED_SUBNODE ← MAX(subnode_Q-value) // exploitation
15.             IF CURRENT_NODE type is Judgment
16.                     ANSWER←answer the judgment with the current subnode ID
17.                     NEXT_NODE←the connection that reflects the ANSWER
18.             ELSEIF CURRENT_NODE type is Processing
19.                     REWARD←Apply_Processing(current subnode ID);
20.                     NEXT_NODE←the first connection from the SELECTED_SUBNODE
21      Qip = CURRENT_NODE.subnode[SELECTED_SUBNODE].Q
22.     [SELECTED_SUBNODE Q value ← Qip+(0.9*(REWARD+(0.9*MAXQNEXT)-Qip))
23.     IF (agent.steps finished) or (all tiles pushed to the holes)
24.             Change the environment to the next one
25.             GOTO line2
26.     ELSE
27.             GOTO line3
```

Pseudo Code 3.1 GNP-RL Training, produced from [9]

GNP-RL Testing Phase

```
Int Apply_Judgment(ID)
1.      Switch(ID):
2.              Case(1): answer ← what is in the Forward
3.              Case(2): answer ← what is in the Backword
4.              Case(3): answer ← what is in the Left
5.              Case(4): answer ← what is in the Right
6.              Case(5): answer ← what is the direction to the nearest tile
7.              Case(6): answer ← what is the direction to the nearest hole
8.              Case(7):answer ← what is the direction from the nearest tile to the nearest
                hole
9.              Case(8): answer ← what is the direction to second nearest tile
10.     Return answer

Apply_Processing (ID)
1.      Switch(ID):
2.              Case(1): Go Forward; IF (tile is dropped into the hole)
3.              Case(2): Turn Left
4.              Case(3): Turn Right
5.              Case(4): stay
6.

Main()
1.      NEXT_NODE ← START_NODE
2.      CURRENT_NODE ← NEXT_NODE
3.              SELECTED_SUBNODE ← MAX(subnode_Q-value) // exploitation
4.              IF CURRENT_NODE type is Judgment
5.                      ANSWER←answer the judgment with the current subnode ID
6.                      NEXT_NODE←the connection that reflects the ANSWER from the
                        SELECTED_SUBNODE
7.              ELSEIF CURRENT_NODE type is Processing
8.                      Apply_Processing(current subnode ID);
9.                      NEXT_NODE←thew firset connection from the SELECTED_SUBNODE
10
11.     IF (agent.steps finished) or (all tiles pushed to the holes)
12.             Change the environment to the next one
13.             GOTO line2
14.     ELSE
15.             GOTO line
```

Pseudo Code 3.2 GNP-RL Testing produced from [9]

3.3.4 Evolutionary Operators

Crossover

The crossover is an operation that applied on two chromosomes to generate two new offspring. These two children have the same features from both parents. To perform the

crossover, firstly two parents are picked out from the 300 individuals using the torment selection technique with size 7. That means randomly 7 chromosomes are selected then pick the best one with the highest fitness value and use it as parent1. Secondly, the same mechanism is implemented to select the second parent. Finally, randomly, a gene (node) from parent one exchanged with the gene (node) that have the same id from parent2 with crossover probability (Pc = 0.1) as in [9]. These two selected genes should be with the same node type. As shown in Figure 3.3 GNP-RL Crossover. It had mentioned in [9] that when the two nodes exchanged the Q-value for each one will exchange with them also, that means the Q-value will go with the nodes through generations and will be updated in the evaluation step. So, we have adopted this method.

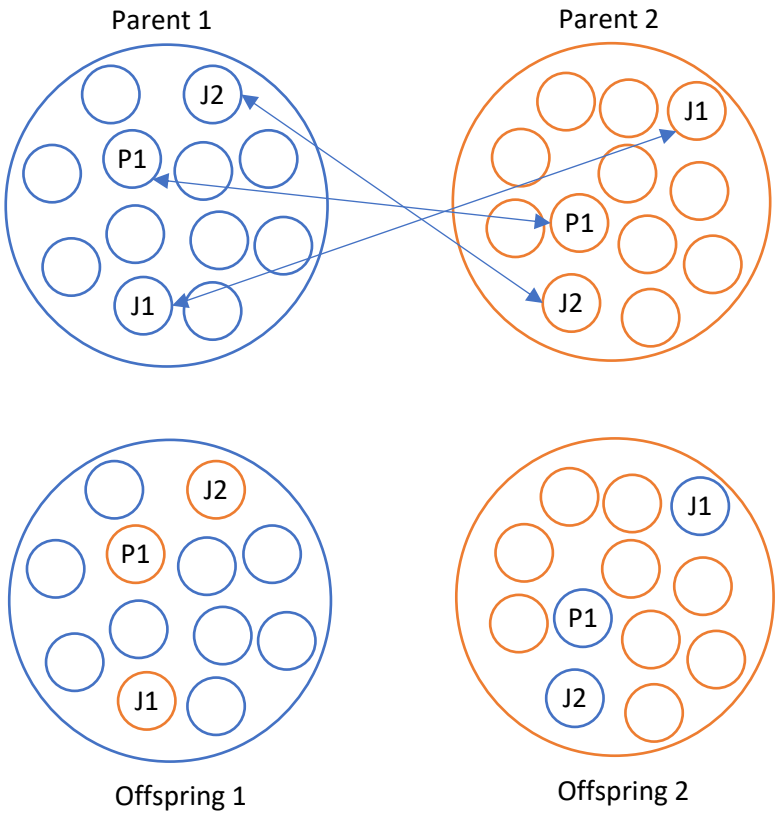


Figure 3.3 GNP-RL Crossover. In this representation, colours uniquely identify the different processing and judgement nodes that crossed over between the parents. [9]

Mutation

The mutation is an operation that is executed in one chromosome to make a changing in some of its genes. In this thesis, three types of mutation are used, which was used by Mabu in 2014 [2].

1- Randomly choose a gene and change its sub nodes functions (ID) to a new function with a probability (Pm = 0.01).
2- Randomly choose a sub-node from a node with a probability (Pm = 0.01) and change its connections to new ones with some conditions:
    a. The new connection should not direct to the same node.
    b. The new connection should not be equal to any connection from this sub node.
3- Randomly choose a node with a probability (Pm = 0.01) and change the number of its sub nodes to be more or less. More with adding new sub-nodes or less with deleting sub nodes with keeping the maximum sub-nodes number is 4.

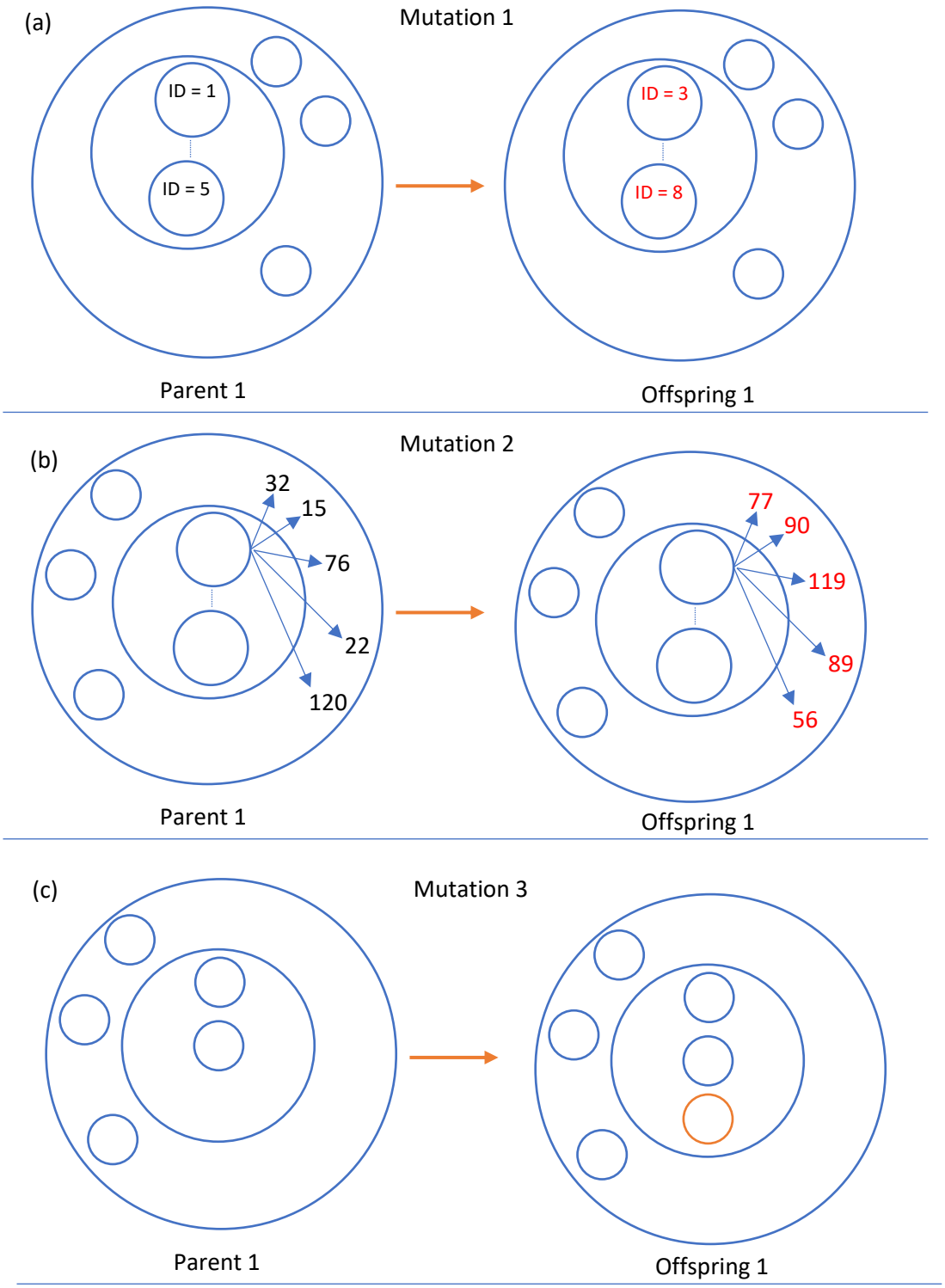


Figure 3.4 GNP-RL Mutation. (a) Mutation 1 allows for an ID number change. (b) Mutation 2 enables the connection to be changed. (c) Mutation 3 allows for insertion/deletion of sub nodes. [2]

3.4 Empirical testing and analysis

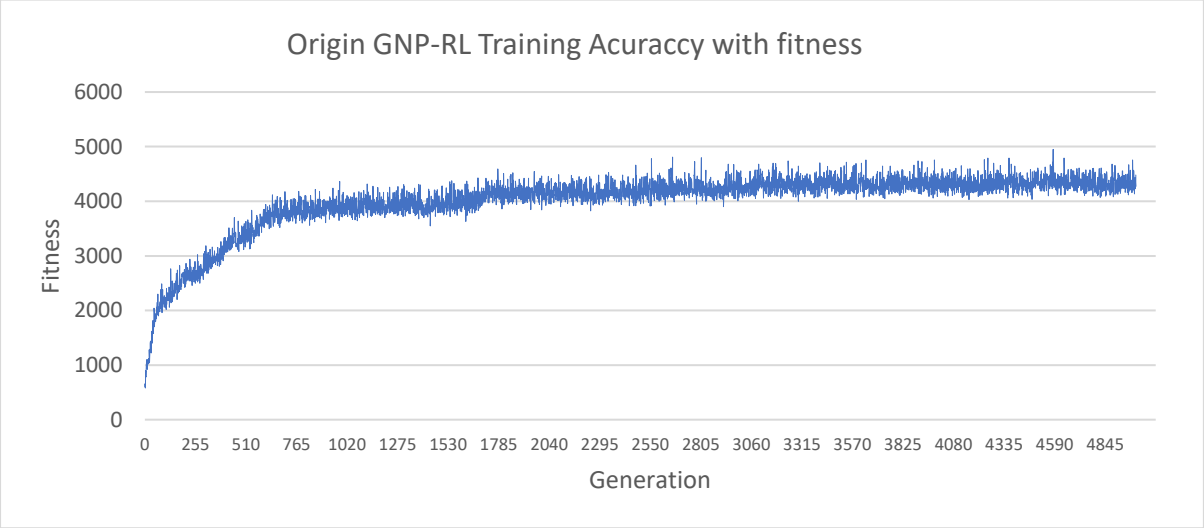


Figure 3.5 GNP-RL training accuracy by fitness

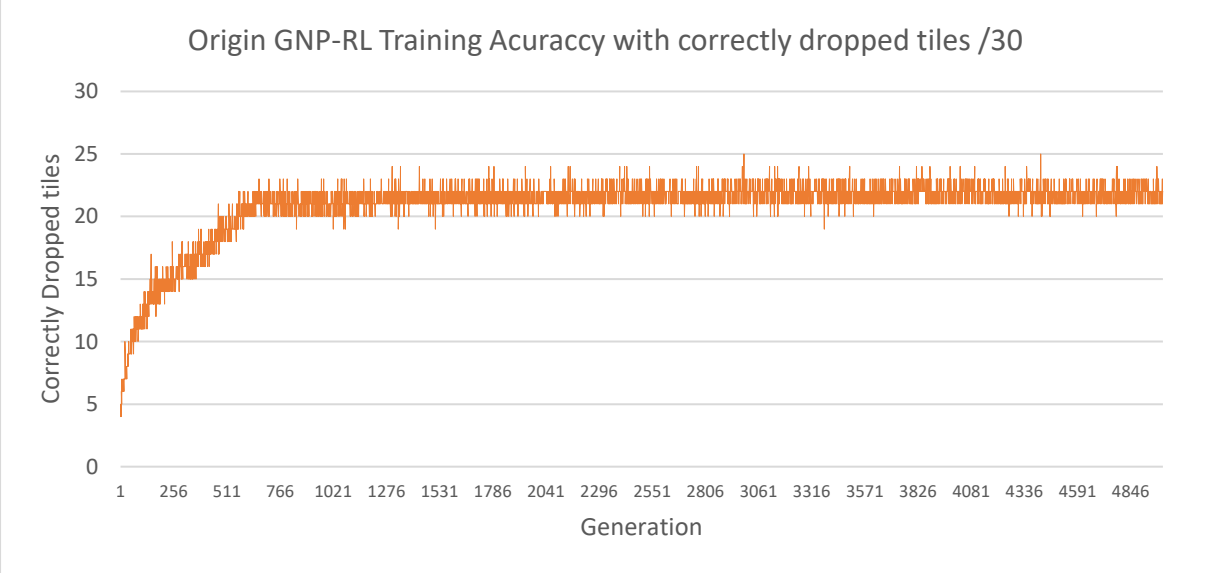


Figure 3.6 GNP-RL training accuracy by a number of correctly dropped tiles

The tile world problem simulation generates 5000 generations with 300 individuals for each one and records the best chromosome for each generation in the training phase. From the figures Figure 3.5 and Figure 3.6 we noticed that the results increased within the first 800 generations with a maximum of 25 correctly dropped tile out of 30 tiles in the training phase, and this result is not enough to be adopted in the real world problem that because the algorithm failed in the training stage.

3.5 Summary

In this chapter, we have analyzed the learning ability of the original GNP-RL by plotting the fitness values as well as the number of tiles successfully used to cover the holes (correctly dropped tiles) for every 1 training epoch. We show that our implementation of the algorithm is able to obtain the same level of accuracy, as reported in Mabu [9]. However, upon further testing (counting the number of correctly dropped tiles), it was observed that the algorithm performed poorly in the training and test sets, covering only 25/30 tiles (83% success) in the training stage, while correctly covering only 19/30 tiles (63.33 % success) when testing the training set and 9/30 tiles (30 % success) when testing the test set (1). We would like to note that in the literature, the used fitness function was the number of correctly dropped tiles while we used the fitness function from [2] Equation 3.2 Fitness (1) . Also, the training set that used in [9] was a grid world with 30 tiles, 30 holes and 3 agents, whereas the testing set that chosen in this implementation was as in [2] which has 10 environments each one of them has 3 tiles, 3 holes, and 3 agents. Even we changed the Fitness function and the training set, the results matched the implementation in [9].

As a conclusion, GNP-RL needs to be improved to be able to solve such a complex problem like tile world application. Mabu worked on this algorithm and enhanced it by using variable sized chromosome in 2014 [2]. The next section will discuss this method.

Chapter 4 (Review) VSGNP-RL

4.1 Motivation

In 2014 Mabu came with a new chromosome structure that divides each individual into variable sized subprograms to separate the problem to small tasks and connect them with a transferred connection to ensure that there is a way to navigate from one subprogram to another. So, we decided to work with this new construction and compare the results with the previous one.

4.2 Related work

Yang was the first one who proposed the divided structure method [11] with the GNP-RL, and he proved the efficiency of this technique with a stock trading model. This method divides the problem to small tasks to facilitate the chromosome structure and to make the algorithm more functional. Distributed GNP-RL got better results than GNP-RL when applying them in the stock trading model. After that, in 2014, Mabu implemented the variable sized genetic networking program with reinforcement learning (VSGNP-RL) on the tile world problem [2]. This time the distributed structure could have variable sized subprogram. So, each subprogram could have a different number of nodes (genes) than the other subprogram. In this work, Mabu demonstrated that this mechanism had enhanced the results in the tile world system.

4.3 The algorithms

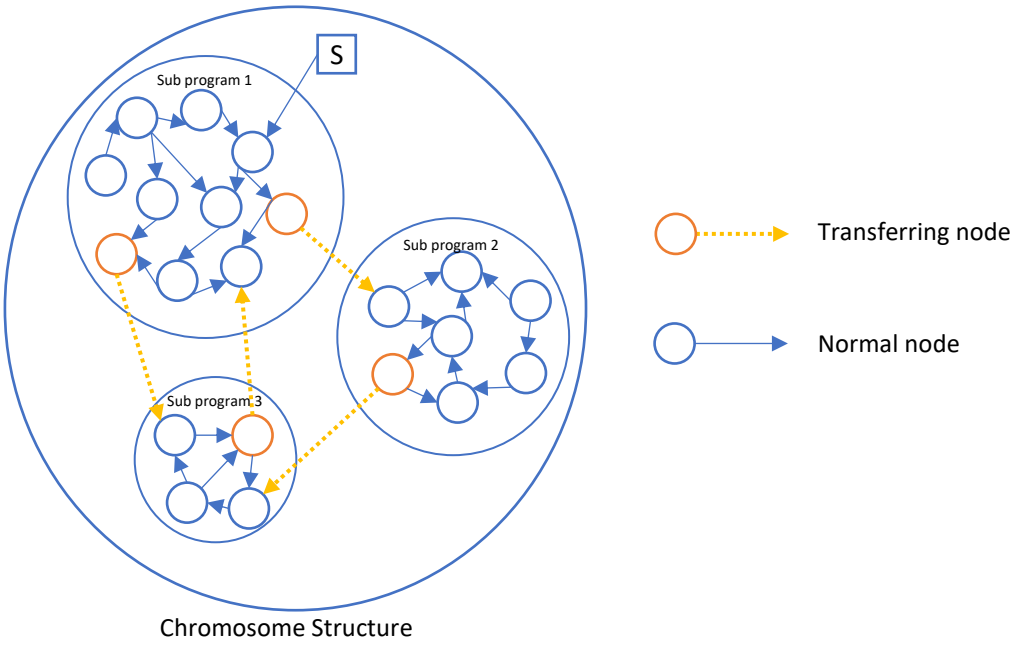


Figure 4.1 VSGNP-RL Phenotype Chromosome Structure. The orange-coloured nodes identify the nodes that directly connects to another subprogram modified form [2]

Population Size	**Crossover**	**Mutation**	**Elite**	**Max # sub nodes**
300	120	175	5	4
# Nodes	**# Judgment nodes**	**# Processing nodes**	**# Sub Programs**	**Tournament Size**
120	80	40	3	7
Crossover rate Pc	**Mutation rate**	**ε**	**γ**	**α**
0.1	Pm 0.01 Pmtr 0.001 Pmtaff1 0.002 Pmtaff2 0.01	0.1	0.9	0.9
Delay time	1 → Judgment node 5 → Processing node		**Agent Steps**	60 for each agent

Table 4.1 VSGNP-RL Parameters

4.3.1 Individual (chromosome) structure

As shown in Figure 4.1 the individual has the same structure as GNP-RL Figure 3.2, but it is divided into some subprograms in this implementation, there are 3 subprograms. The total number of nodes in the chromosome is 120 nodes. They are distributed between the subprograms. Each node is either a transfer or a normal node. The transfer node is the node that direct to a node from another subprogram and the normal node is a node that navigates to a node from the same subprogram.

4.3.2 Initial the first population

To initialize the first population, the same technique which used to initialize the first population in GNP-RL will be used here with some additions. Each node has 2 more features: subprogram number (AF) and node connection type (TR). AF refers to the subprogram that this node is part of, and TR refers to if this node is transfer or normal node. In the first generation, each subprogram has the same number of nodes, then within the mutation, the number of nodes in every subprogram is changed. Each node can be a transfer node with a probability of 0.1, and all the other are normal nodes.

4.3.3 Evaluate the chromosome

The same mechanism which was used with GNP-RL.

4.3.4 Evolutionary Operators

Crossover

To create a new two children, two parents are chosen with a tournament selection with size 7, then crossed over by selecting randomly two nodes one from each parent with a probability 0.1 and exchange them between the chromosomes. These two selected nodes should be with the same id, type and from the same subprogram. That means, if the selected node from parent one is from the subprogram 2, the second selected node from parent 2 should be from subprogram 2 also. To make a balance between the subprograms, the same number of judgment and processing nodes should be crossed over from each subprogram. For instance, three judgment node and one processing node from every subprogram should be crossed over between the parents. As shown in Figure 4.2.

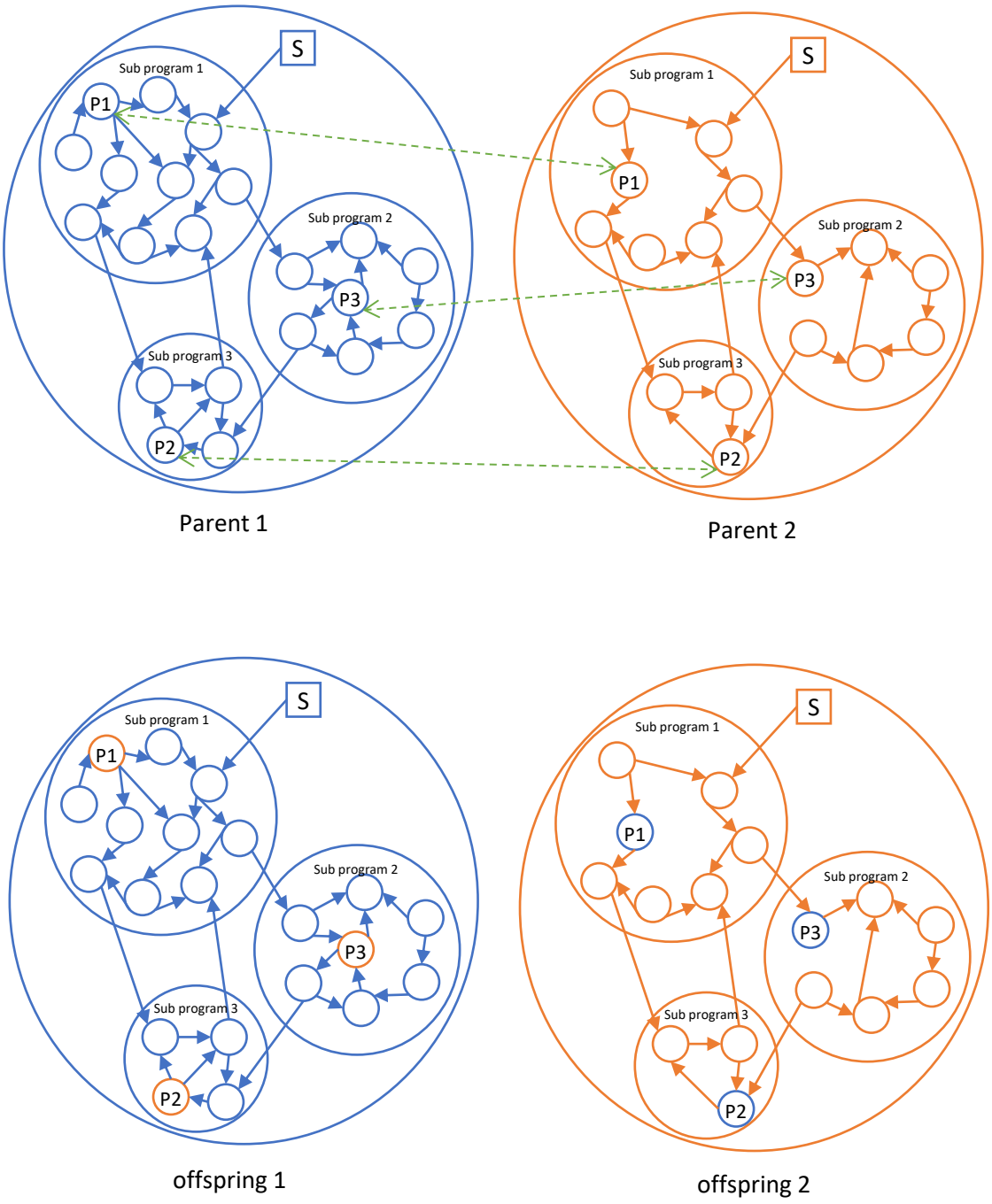


Figure 4.2 VSGNP-RL Crossover the different colours indicate the nodes that crossed over between the parents. The nodes crossed over with another same type node from the same subprogram. [2]

Mutation

Two types of mutation are used in this implementation, as in Figure 4.3:

1- Choose randomly node with a probably 0.01 and change the node connection type (TR). If the node is transferring node, it is changed to a normal node and vice versa with considering the connections. If the node is converted to a normal node, the connections should also be adjusted to direct to a node from the same subprogram. And if the node is converted to a transfer node, the new connections should be changed to lead to nodes from another subprogram.

2- Choose a node randomly with a probability of 0.01 and change the number of a subprogram that it belongs to (AF). After changing the subprogram number, the connections of this node should also be changed that because if the node is transferring node, the new connections should adjust with the new subprogram and the same for the normal node.

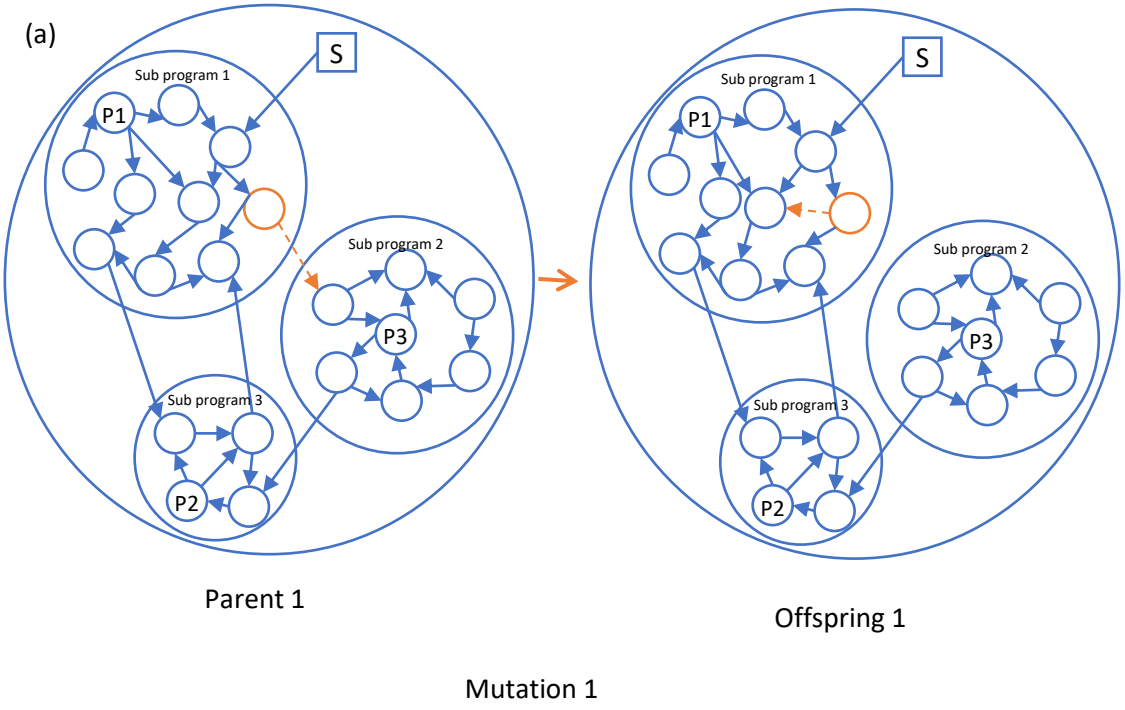


Figure 4.3 VSGNP-RL Mutation. (a) Mutation 1 allows changing the connection type from normal to transfer or vice versa. (b) Mutation 2 allows changing the subprogram that the node belongs to considering changing the connections also. [2]

4.4 Empirical testing and analysis

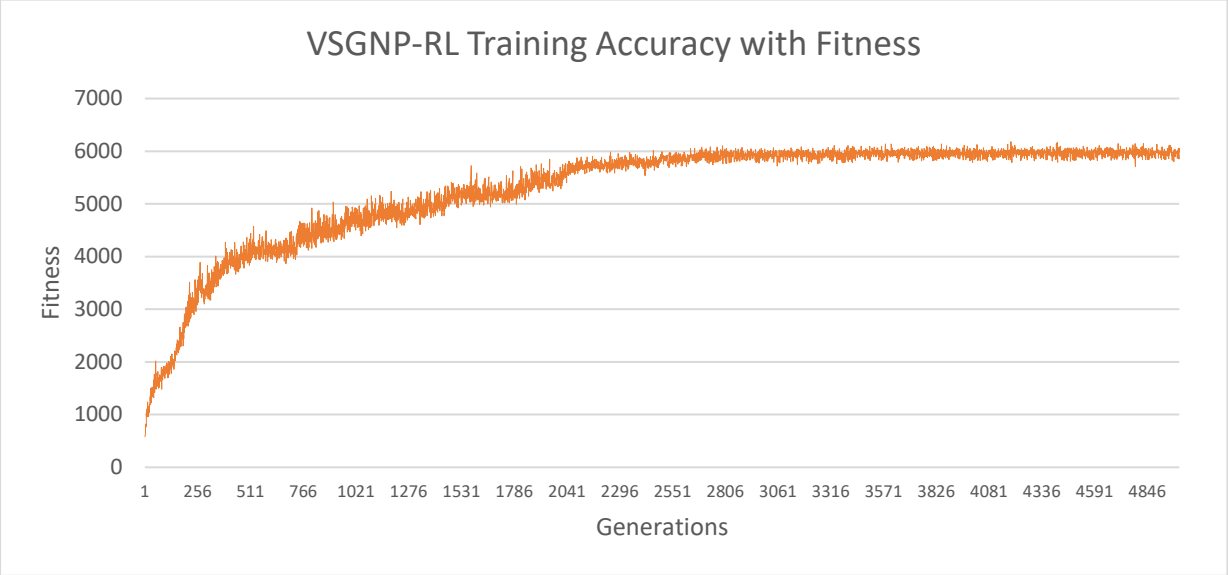


Figure 4.4 VSGNP-RL training accuracy by Fitness

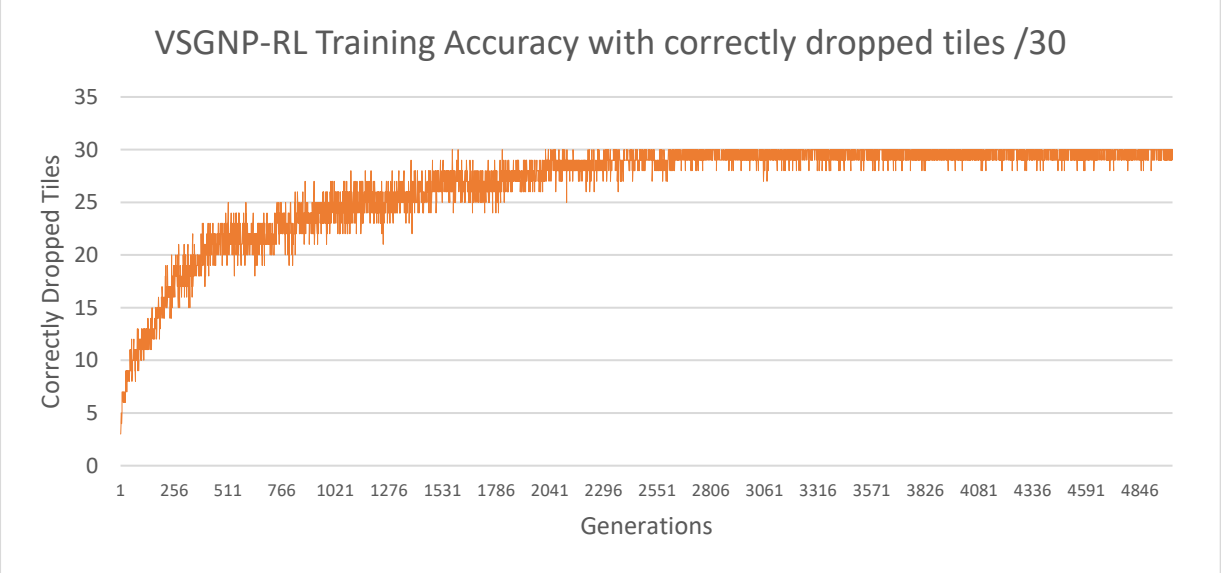


Figure 4.5 VSGNP-RL training accuracy by correctly dropped tiles

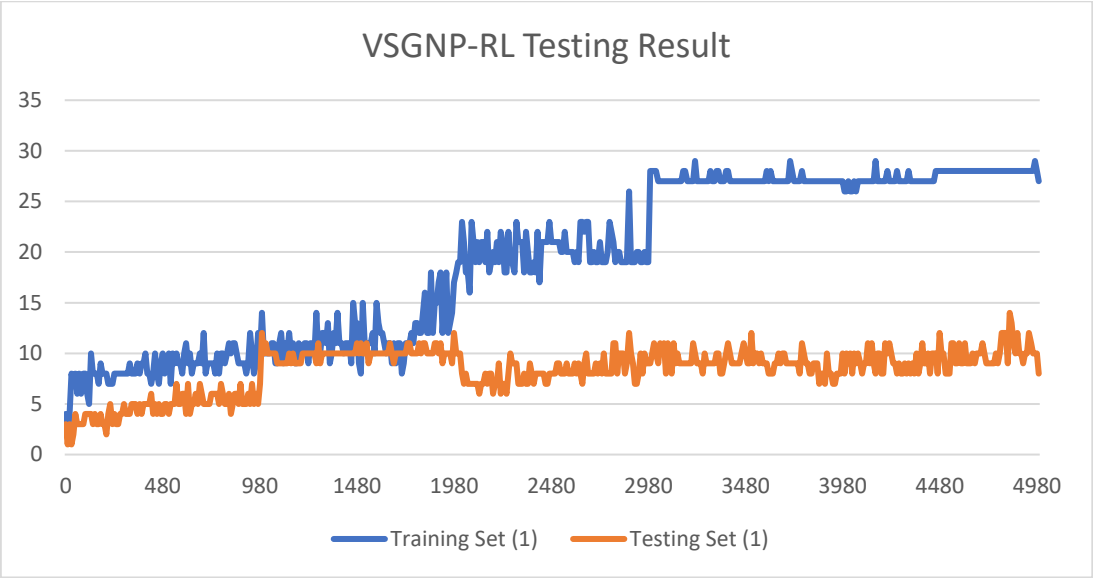


Figure 4.6 VSGNP-RL Testing Accuracy on the Training Set (1) and Testing Set (1)

After following the same strategy and parameters, which was used by Shingo Mabu in 2014 [2], the training result was great depending on the same fitness function but in the testing stage, the best chromosome gave 13/30 correctly dropped tiles when testing it on the test environment as in Figure 1.3 Testing Environment which is the same test environment that used in [2]. And some problems have appeared in this testing:

1- The agent tries to push the tile near to the hole and then pushed it away from the hole then again pushed it near to the hole and that to gain more points. In this way, the fitness function will increase, but the number of correctly dropped tiles will not raise. (See this video https://youtu.be/IYHeDbLh9ls)
2- Because of the distributing inside the chromosome, each agent work with a subprogram. And that caused to put all the work on one or two agents and the third agent was not doing anything. (See this video https://youtu.be/cPnKVMNU70A)

The first problem above appeared because of the used fitness (Equation 3.2 Fitness (1)). This fitness adds 20 points to the fitness value each time the agent pushes the tile near to the hole and gives 100 points each time the agent drops the tile into the hole. After running the algorithm for some generations, it finds that pushing the tile closer to the hole more times will gain more points than dropping it. For example, when pushing the tile 6 times closer to the hole, it will earn 120 points and that more than 100, which is for dropping it. Se, we decided to update the function and change the weights to prevent such problems. Here is the new fitness function:

$$Fitness\ (2) = \sum_{env=0}^{ENV} [(1000 \times D_{tile}) + (D_{distance}) + (T_{remain})] \qquad \textit{Equation 4.1 Fitness (2)}$$

Where D_{tile} is the number of correctly dropped tiles. $D_{distance}$ is (2) points if the agent pushes the tile closer to the nearest hole and (-1) when the agent pushes the tile far away from the nearest hole. T_{remain} is the remain steps for the three agents if they push the three tiles into the holes before the end of the steps (each agent has 60 steps). The reason for choosing 1000 as a weight for the correctly dropped tile is that each environment has 10 rows and 10 cols, so the maximum time that the agent can push the tile closer or far away is 10 steps, and because there are three tiles so each environment needs:

- $3 \times 10 \times 2 = 60$ when pushing the tile closer to the hole.
- $3 \times 10 \times -1 = -30$ when pushing the tile far away from the nearest hole.

For the 10 environments ($60 \times 10 = 600$) and ($-30 \times 10 = -300$) these are the maximum and minimum limit for each number of correctly dropped tile.

By using this equation, the fitness value and the number of correctly dropped tiles will be scalable, that means each number of correctly dropped tile has a range of fitness value that is not crossed with another number of correctly dropped tiles. The table (Table 4.2) and the chart (Figure 4.7) below demonstrate that.

# Correctly Dropped tiles	Min distance - Max distance	Max Tremain	Fitness
0	-300	180	-120
0	600	180	780
1	-300	180	880
1	600	180	1780
2	-300	180	1880
2	600	180	2780
3	-300	180	2880
3	600	180	3780
4	-300	180	3880
4	600	180	4780
5	-300	180	4880
5	600	180	5780
6	-300	180	5880
6	600	180	6780
7	-300	180	6880
7	600	180	7780
8	-300	180	7880
8	600	180	8780
9	-300	180	8880
9	600	180	9780
10	-300	180	9880

10	600	180	10780
11	-300	180	10880
11	600	180	11780
12	-300	180	11880
12	600	180	12780
13	-300	180	12880
13	600	180	13780
14	-300	180	13880
14	600	180	14780
15	-300	180	14880
15	600	180	15780
16	-300	180	15880
16	600	180	16780
17	-300	180	16880
17	600	180	17780
18	-300	180	17880
18	600	180	18780
19	-300	180	18880
19	600	180	19780
20	-300	180	19880
20	600	180	20780
21	-300	180	20880
21	600	180	21780
22	-300	180	21880
22	600	180	22780
23	-300	180	22880
23	600	180	23780
24	-300	180	23880
24	600	180	24780
25	-300	180	24880
25	600	180	25780
26	-300	180	25880
26	600	180	26780
27	-300	180	26880
27	600	180	27780
28	-300	180	27880
28	600	180	28780
29	-300	180	28880
29	600	180	29780
30	-300	180	29880
30	600	180	30780

Table 4.2 Fitness (2) limitations

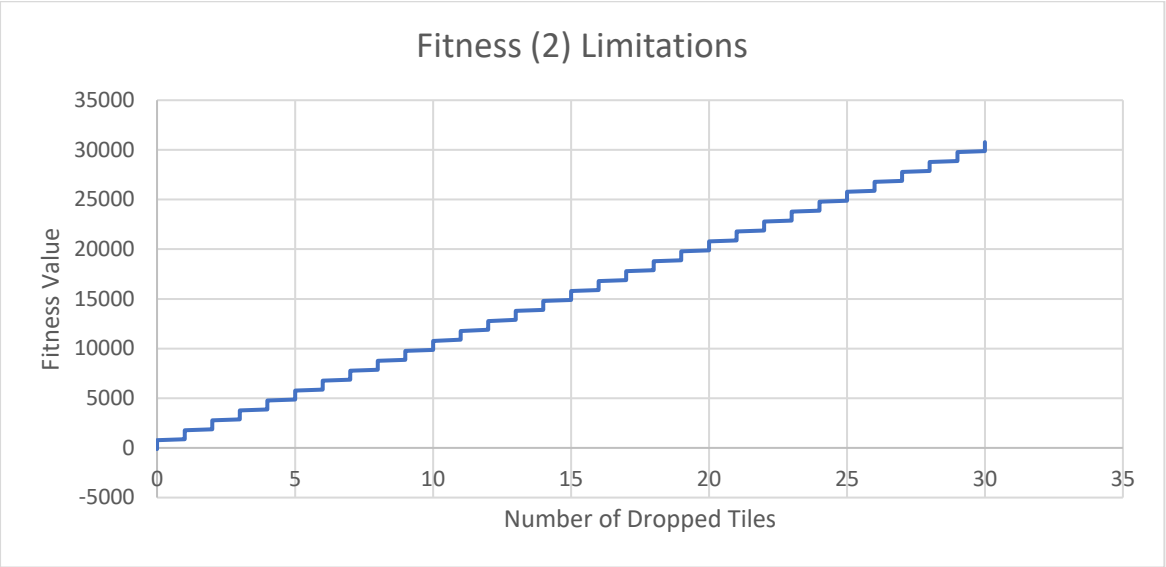


Figure 4.7 Fitness (2) Limitations

As noticed from Figure 4.7 that there is no overlap between the maximum and the minimum values for each number of correctly dropped tiles.

4.5 Summary

VSGNP-RL is an improvement over the GNP-RL, which divides the chromosome into several subprograms to divide the problem into smaller and simpler tasks, this enhancement was addressed in [2].

After implementing the same strategies in [2], the training results achieved the top number of correctly dropped tiles which 30/30; however, the testing results could not achieve 50% of accuracy. We also found some issues in the used function, so after finding the defect in the first fitness and editing it, the new fitness will be used in all the next implementations. Even the VSGNP-RL pass in the training stage but its test results still under 50% of success. So, the following chapters will discuss some of the new techniques that have not been used yet and compare the results with the previous ones.

Chapter 5 (Proposed Architecture #1) GNP-RL with Elitism and Gene Diversification

5.1 Motivation

One of the most effective operations on the progress of the genetic algorithm is a crossover. The crossover was used to improve the next generation of new individuals. On the other hand, some researchers suggested the benefits of ensuring diversity in the populations. In this section, we will implement a mechanism which using the crossover to apply the diversity on the population.

5.2 Related work

Losing the population diversity in the genetic algorithms one of the limitations in improving the results. In 2006 [41] Shapiro has studied the population diversity loss in the distributed algorithms. In his introduction to the Needle problem, he showed that increasing the size of the sample is one of the most critical factors in increasing its diversity. Create the variety in the population individuals has not been used yet in the GNP-RL, while the probability of diversity loss was calculated for GA, GP, and GNP with Khepera robot by Li in 2010 [42].

Selection individuals based on the highest fitness values will result in the appearance of samples with similar chromosomes, and this may lead to a decline in the evolution of individuals genetic characteristics, which leads to the cessation of the algorithm to improve the solutions. Creating diversity among individuals will ensure that solutions are rapidly enhanced. Rank space method is one of the ways that used as a technique to ensure the difference in the population which selects the individuals by two rankings; the quality rank and the diversity rank. This method was addressed on the GA by Winston in 1993 [43]. But it was not used yet with the GNP-RL.

In this chapter, we used the rank space method with two operations in the GNP-RL. The first one, the rank space method has been used to select the second parent in the crossover operation, and the second operation is selection the second 5 elites by using the rank space method.

5.3 The algorithms

Population Size	Crossover	Mutation	Elite	Max # sub nodes
300	120	170	10	4
# Nodes	**# Judgment nodes**	**# Processing nodes**		**Tournament Size**
120	80	40		7
Crossover rate Pc	**Mutation rate Pm**	**ε**	**γ**	**α**
0.1	0.01	0.1	0.9	0.9
Delay time	1 → Judgment node 5 → Processing node		**Agent Steps**	60 for each agent

Table 5.1 GNP-RL with Elitism and Gene Diversification Parameters

5.3.1 Individual (chromosome) structure

This implementation has 300 individuals for each population,5 of them is the best five chromosomes from the previous generation depending on the fitness value, 5 are selected by the rank space method by using the fitness rank and the diversity rank, 120 of them created by the crossover with using rank space method, and 170 are generated by the mutation operation.

Crossover and Elite diversity have been applied on the GNP-RL, so the chromosome structure is the same structure for the GNP-RL see Figure 3.2.

5.3.2 Initialize the first population

The same conditions for initiating the first population in the GNP-RL are used here.

5.3.3 Evaluate the chromosome

The individuals have been evaluated with Fitness (2) see Equation 4.1.

5.3.4 Evolutionary Operators

The mutation had the same operations as in the GNP-RL. The crossover had a new technique that is addressed below.

This research will apply the diversity in the GNP-RL crossover by choosing parent1 with the tournament selection. After that, use parent1 as a reference to choose parent2 with using rank space method to find the most diverse chromosome from the reference. In this step, the algorithm calculates the diversity between all the chromosomes in the population and the reference which is parent1. To calculate the diversity, two genes picked up to detect the distance between them. The first one is the node type, and the second one is the number of sub-nodes. We choose these two genes after some experiments, and these two genes won. To

calculate the distance, Euclidean distance Equation 5.1 [44] and Equation 5.2 have been used, where *totalnodenum* is the number of the nodes in each chromosome. *Reference[i]* is the node number i in the reference (parent1). *Ch[i]* is the node number i in the chromosome (ch). These equations should be calculated for all the individuals in the population.

After that, each chromosome should have two ranks (diversity rand & quality rank). Where diversity rank is depending on their diversity distance (lowest diversity → highest rank), and quality rank is depending on their fitness value (most top fitness → highest rank). The next step takes the total of these two ranks then rank the total again depending on their sum ranks (lowest sum → highest rank considering breaking tie in favour of diversity). In the end, calculate the rank fitness by using Equation 5.3 [45] , where reference.fitness is the fitness value for parent1, ch.sumrank is the summation of (diversity and quality ranks) for the chosen chromosome. Once each has a rank fitness, the chromosome with the highest rank fitness is determined to be parent2. In this way, parent 2 is the most diverse chromosome with parent1. So, when crossover these two diverse chromosomes, the new children have different features and various genes. This technique is coded in Pseudo Code 5.1 below.

Equation 5.1 Calculate Diversity Distance [44]

$$\boldsymbol{dis}+ = \sqrt{\sum_{i=0}^{totalnodenum} \left(\boldsymbol{refrence}[i].\boldsymbol{node}_{type} - \boldsymbol{ch}[i].\boldsymbol{node}_{type}\right)^2 + \left(\boldsymbol{refrence}[i].\boldsymbol{subnode}_{number} - \boldsymbol{ch}[i].\boldsymbol{subnode}_{number}\right)^2}$$

$$dis(t+1) = \frac{1}{dis(t)^2}$$

Equation 5.2 Diversity Proportion

$$stdFitness = \frac{\text{refrence.fitness}}{\text{totalfitness}};$$

$$C = \frac{stdFitness}{1-\text{stdFitness}};$$

$$rank_{fitness} = C * (1-C)^{ch.sumrank-1}$$

Equation 5.3 Calculate Rank Fitness [45]

GNP-RL Crossover Diversity

GNPRL_Crossover(vector<Chromosome> population)

1. Chromosome Parent1 = Tournament_Selection(population);
2. **For** all ch as chromosomes in population:
3. ch.diversity_distance = Calculate_Diversity_Distance (Parent1 , ch);
4. Diversity_Rank_population_individuals (population); // lowest diversity → highest rank
5. Quality_Rank_population_individuals (population) ;// highest fitness → highest rank
6. Sum_two_ranks(population);
7. Sum_Rank_population_individuals(population); // lowest sum → highest rank
8. **For** all ch as chromosomes in population:
9. ch.rankfitness = Calculate_rank_fitness(parent1, ch);
10. Chromosome Parent2 = individual_highest_rank_fitness;
11. Apply_Crossover(Parent1 , Parent2);

Calculate_Diversity_Distance(Chromosome reference, Chromosome ch)

1. $dis+= \sqrt{\sum_{i=0}^{totalnodenum} (refrence[i].node_{type} - ch[i].node_{type})^2 + (refrence[i].subnode_{number} - ch[i].subnode_{number})^2}$
2. $dis(t+1) = \frac{1}{dis(t)^2}$

Calculate_rank_fitness(Chromosome reference, Chromosome ch)

1. $stdFitness = \frac{\text{refrence.fitness}}{\text{totalfitness}}$;
2. $C = \frac{stdFitness}{1-\text{stdFitness}}$;
3. $rank_{fitness} = C * (1-C)^{ch.sumrank-1}$;

Pseudo Code 5.1 GNP-RL Crossover Diversity

To simulate this experiment, the same (chromosome structure and initial population) are used. The evaluation has been done using Equation 4.1, Figure 1.1 Training Set (1), and Pseudo Code 3.1 GNP-RL . And with the same crossover (Figure 3.3 GNP-RL Crossover) and mutation (Figure 3.4 GNP-RL Mutation) operations.

5.4 Empirical testing and analysis

When applying the crossover diversity on the GNP-RL, the results increased fast, but they stopped growing after about 500 of generations. That's because, after some generations, most of the chromosomes become similar to each other as a result of implementing the crossover diversity. To solve this problem, another technique used in this stage, which is the elite diversity. Apply the diversity with rank space method on the best five individuals and find the most 5 different chromosomes with using the same technique that used with choosing parent2 in the crossover but this time by setting the best 5 elites as references. Then add them to the next generation to ensure some differences in the epochs. The results grew more than with using only the crossover diversity, as shown in Figure 5.1 and Figure 5.2.

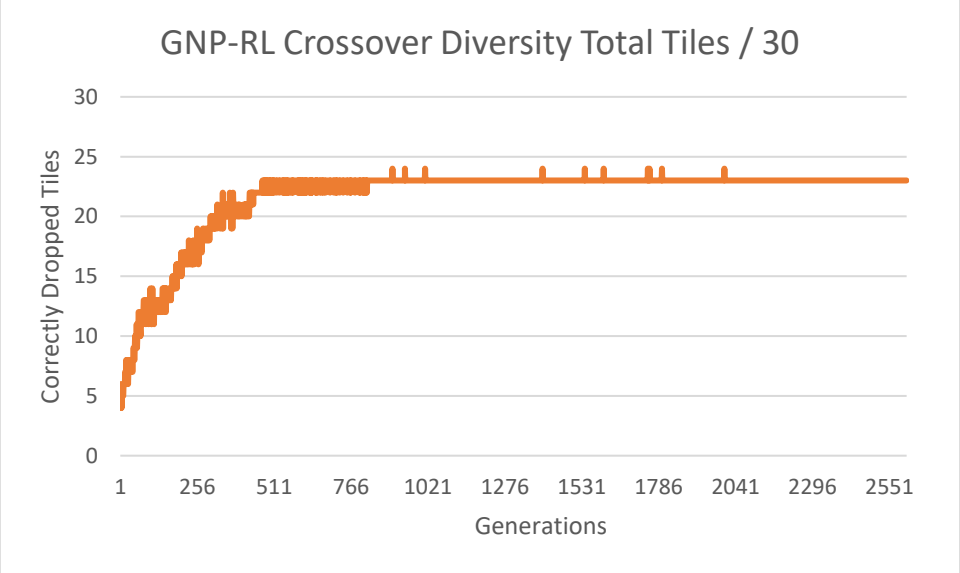


Figure 5.1 GNP-RL Crossover Diversity Training results

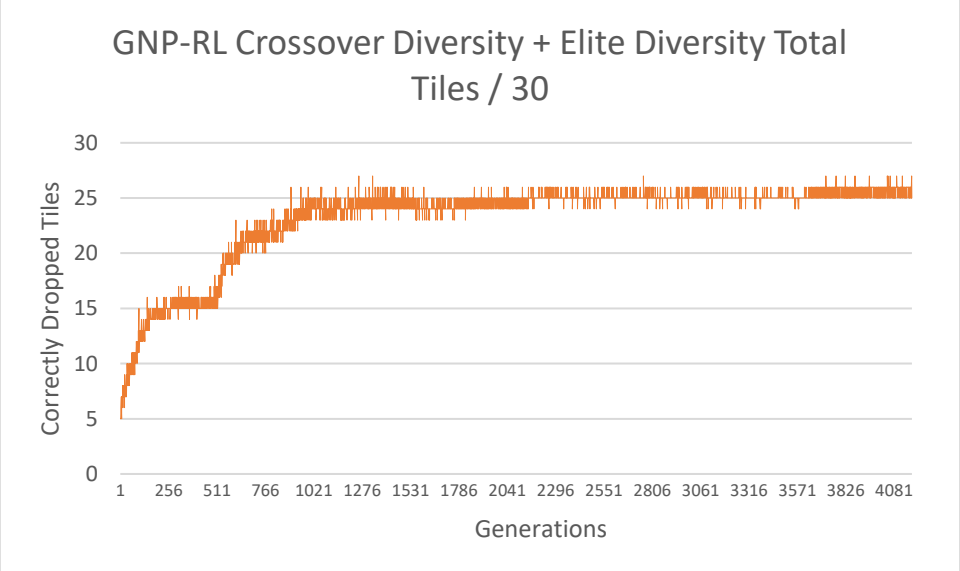


Figure 5.2 GNP-RL Crossover Diversity + Elite Diversity Training results

5.5 Summary

In this chapter, we applied the rank space method on the GNP-RL crossover to ensure that the two parents that used in the crossover are different and have a variety of features. Using crossover diversity make the results increased in early generations, but as a result of using this mechanism for many generations, the algorithm loses the difference between the individuals in the population. When applying the elite diversity in the implementation with the crossover diversity, the algorithm has ensured some variety in individuals, and that gave some enhancement. However, the results were not good enough to start the test stage. So, the next chapters will work with more new techniques to improve the algorithm.

Chapter 6 (Proposed Architecture #2) GNP-RL with Constraint Conformance

6.1 Motivation

This technique has been implemented after noticing that most of the tiles unaccounted in the test stage have been pushed to a trapped location such as one of the states in Figure 6.1, which makes them trapped and could not be pushed again to anywhere and that make the Agent loos this Tile (See this video https://youtu.be/g7NsiK_Fueg). So, to help the algorithm dealing with these cases, we came with a technique that trains the agent how to deal before pushing the tile to one of these situations and how to avoid trapped in one of them. We chose to apply this technique on the basic GNP-RL. This technique uses two methods. The first one, changing the number of connections in the processing node (changing the chromosome structure). The second method introduces a reward and punishment scheme in order to allow the agents to learn from their mistakes.

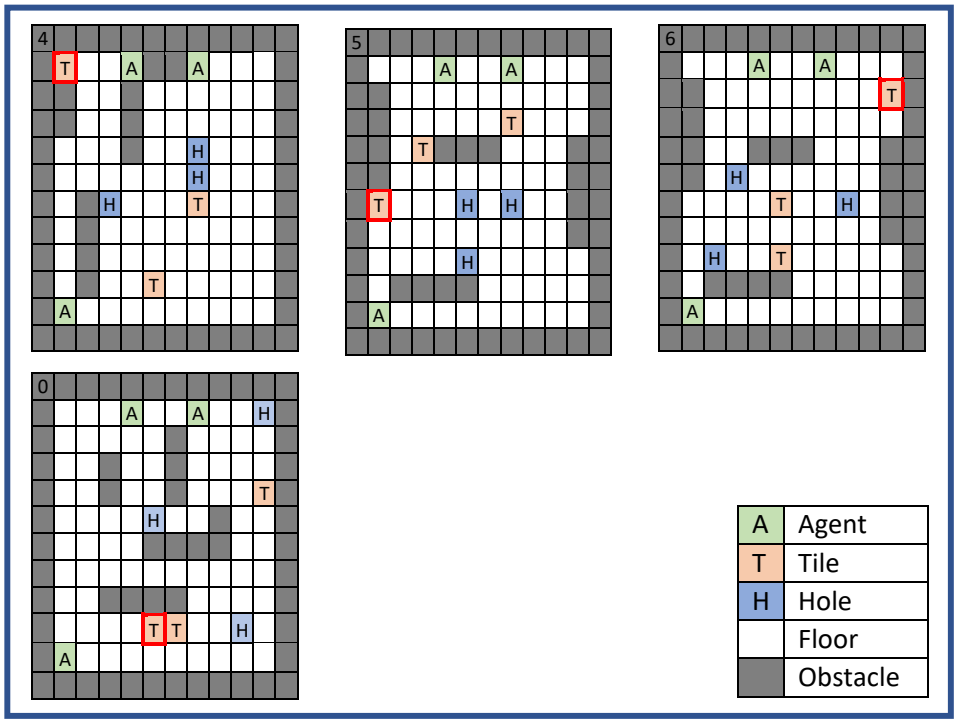


Figure 6.1 some Tile trapped locations (4) the Tile surrounded by obstacles. (5) the Tile has two borders with obstacles. (6) the Tile in the grid border with no Hole in the same edge. (0) the Tile stuck with another Tile.

6.2 Related work

In 1998, Watkins started using the Q-learning method with reward and punishment, which allows the agents to learn from their actions without needing to know the environment [46]. Such as Sutton in 1984 [47], who adopted the method of learning based on the rewards

and penalties received by the agent immediately after the implementation of the actions, and by the accumulation of values in the long term, the best states could be identified. After Mabu came with the idea of adding the RL to the GNP [9], we have observed through our experiments that bad actions are not efficiently penalized by the GNP-RL. In this chapter, the reward and punishment feedback learning method will be used with the GNP-RL.

6.3 The algorithms

Population Size	**Crossover**	**Mutation**	**Elite**	**Max # sub nodes**
300	120	175	5	4
# Nodes	**# Judgment nodes**	**# Processing nodes**		**Tournament Size**
120	80	40		7
Crossover rate Pc	**Mutation rate Pm**	**ε**	**γ**	**α**
0.1	0.01	0.9 → 0.1	0.9	0.9
Delay time	1 → Judgment node 5 → Processing node		**Agent Steps**	60 for each agent

Table 6.1 GNP-RL with Constraint Conformance Parameters

First point (two different possible paths)

The basic GNP-RL has only one connection emanating from a processing node. Therefore, straight after applying an action defined in the processing node, the link will direct the agent to the same next node, regardless of whether the action was right or wrong. In the proposed constraint conformance technique, we have incorporated two connections on the processing node. This ensures that there are two different possible paths for the agent to take, after executing an action. The presence of the two paths allows the algorithm to do the necessary corrections, whenever an action leads towards an unwanted or hazardous state.

Second point (punishment)

GNP-RL uses equation $Qip = Qip + \big(\alpha * (Reward + (\gamma * Qjq) - Qip)\big)$ to update the Q-learning algorithm by using a positive/negative reward. Using the positive reward ensures an increasing in the Q-value and thus will increase the probability to choose this sub node the next time an agent visits this node. Using negative rewards when updating the Q-value guarantees decreasing the Q-value and thus will reduce the probability of choosing this sub node again when visiting the same state.

These two techniques work together to make the agents learn from their actions.

6.3.1 Individual (chromosome) structure

The chromosome structure for this experiment is similar to the GNP-RL chromosome structure except for the number of connections used with the Processing node. In GNP-RL there is just one connection from the Processing node and that direct to the next node after executing the processing action, whereas in this algorithm the processing node, which has (Go Forward) function, has 2 connections. The first connection will be used when the Tile is pushed to a safe vertex, but the next connection will be used when the following location is a trapped vertex. As shown in Figure 6.2.

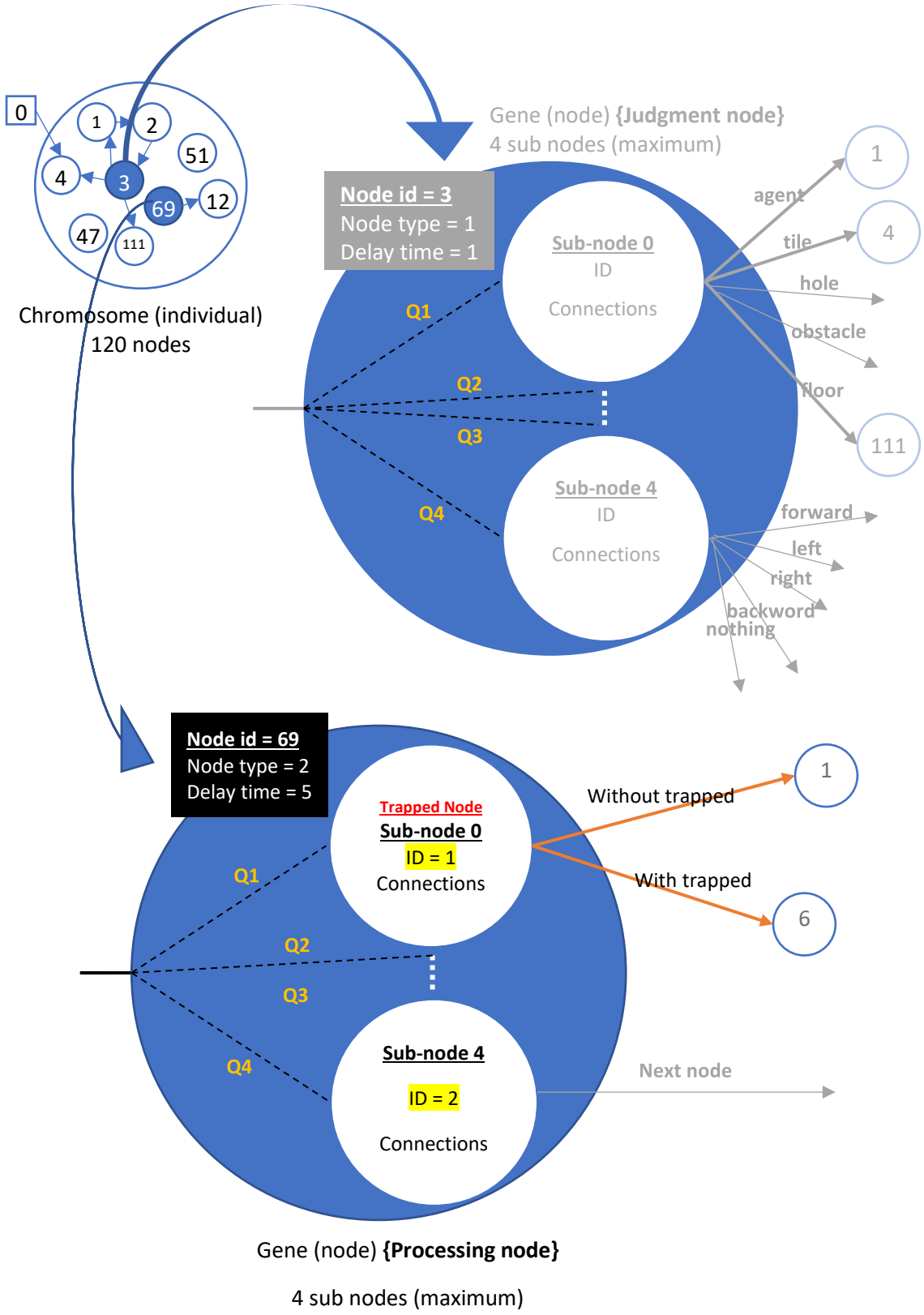


Figure 6.2 Constraint conformance Chromosome Structure

6.3.2 Initial the first population

To generate the first population, the same instructions that have been used with the GNP-RL initial population (see section 3.3.2) are used here, adding to this that the connections for the Trapped node are chosen randomly from the chromosome nodes (120 nodes).

6.3.3 Evaluate the chromosome

To evaluate the chromosome, Fitness function (2) Equation 4.1 is used to run two experiments. One of them on the Training Set (1) (see Figure 1.1) which was produced by Mabu in 2014 [2], and the second one on the Training Set (2) (see Figure 1.2) which was designed by us. In the training set (2) we arranged the environments to make them have different locations for the Tiles, Holes, and Obstacles, while Training set (1) has 10 similar environments with changing only the Tile's locations. We produced different set to make sure that the algorithm is correctly working even with the more challenging environment, and it is not memorized the environment.

From this chapter and later, At the beginning of training, the rate of exploration is set at (0.9), while the rate of exploitation is set at (0.1), thus, giving the algorithm the liberty to try out the different actions available within the chromosomes which have been initialized. By decreasing the rate of exploration and increasing the rate of exploitation by (0.1) after each 100 generations, at later stages of training (900^{th} generation), the rate of exploration will eventually be (0.1) and the rate of exploitation at (0.9). This switch in parameters ensures that the agent will be utilizing the knowledge learned through interaction than exploring a random sub node.

Since the chromosome's fitness evaluation is carried out at the same time as the reinforcement learning process, reducing the exploration rate and increasing the exploitation rate over the time can have a significant impact on the stability of the evaluation results and efficiency of training.

When running the algorithm, the agents start implementing the nodes from the graph. If the agent faces a (Trapped node) Processing node with ID →1 (function → go Forward), before applying the node, the agent check the next location for the Tile (see Pseudo Code 6.1):

- If it is trapped location, there are three actions the algorithm will take:
  1- The algorithm will prevent the Agent from pushing the Tile.
  2- Punish the Agent, that means to give -1 for the Reward when updating the Q-value in Equation 3.1.
  3- Choose the second connection in the trapped node to go to the next node.
- If it is a safe location:

1- The algorithm will apply the go forward action.
2- If the Agent pushes the Tile into the Hole, reward the Agent by giving 1 as a reward in Equation 3.1.
3- Choose the first connection in the trapped node to go to the next node.

Processing node in the Constraint conformance GNP-RL

```
Int Apply_Processing(ID)
1.  Switch(ID):
2.      Case(1):
3.          If (the next location is trapped location)
4.              Don't Go Forward;
5.              Reward ← -1;
6.              NEXT_NODE←the second connection from the selected subnode of the
                current node
7.          Else If(the next location is not trapped location)
8.              Go Forward;
9.              NEXT_NODE← the first connection from the selected subnode of the
                current node
10.         If (the agent dropped a tile into a hole)
11.             Reward ← 1;
12.     Case(2): Turn Left;
13.     Case(3): Turn Right;
14.     Case(4): stay;
15. Return Reward;
```

Pseudo Code 6.1 Processing node in the Constraint conformance GNP-RL

6.3.4 Evolutionary Operators

Crossover

The crossover operation has done using the same technique which was used in the basic GNP-RL (see page 36). When choosing the trapped node to be crossed, it will be crossed over with another Processing node from the second parent. Adding to this, in this implementation when choosing node1 from parent1 randomly, a node2 will be chosen randomly also from parent2, unlike the crossover in GNP-RL which exchange a node1 from parent1 with a node2 that has the same id as node1 from parent2.

Mutation

The same Mutation operation that used with GNP-RL (see page 36) has been used here. Considering that the trapped node connections could also be changed in the mutation.

6.4 Empirical testing and analysis

Two experiments have been done on the constraint conformance GNP-RL. One of them with the training set (1) Figure 1.1 [2], and the other with training set (2) Figure 1.2. After running both implementations for 5000 generations with the same parameters used in the basic GNP-RL (see page 32), we noticed that both experiments gave similar results. The differences were in the time that the algorithm needs to reach the best individuals. In the simple set, which was a training set (1), the algorithm could get the best individuals earlier than when using the more complex environment, which was a training set (2). This concludes that the simplest the set was, the easier it is for the agent to learn and thus find the best results faster.

Conversely, if the environment is more challenging, it takes the agent a period to understand and extract the rules from the data. Adding to this, we found that agents learned how to keep the tiles from losing, resulting in higher results in a shorter period than the basic GNP-RL when using a simple method such as Constraint conformance. This technique was able to get 28 Tiles over 30 as the highest result in the training phase, and this result was better than the original algorithm, which achieved a maximum of 25/30 Tiles.

Experiment (1) – Training Set (1) [2] **Figure 1.1**

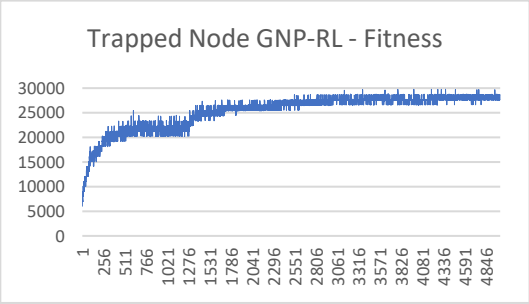


Figure 6.3 Constraint conformance GNP-RL Training Accuracy with Fitness (Training Set (1))

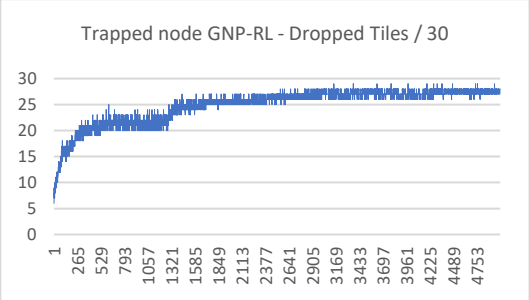


Figure 6.4 Constraint conformance GNP-RL Training Accuracy with correctly Dropped Tiles (Training Set (1))

Experiment (2) – Training Set (2) Figure 1.2

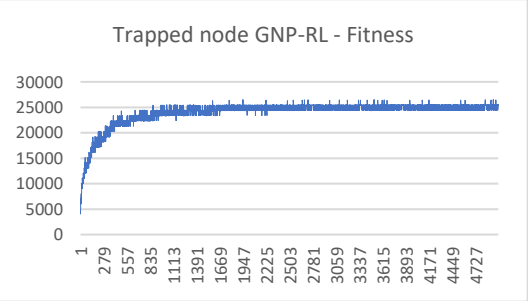


Figure 6.5 Constraint conformance GNP-RL Training Accuracy with Fitness (Training Set (2))

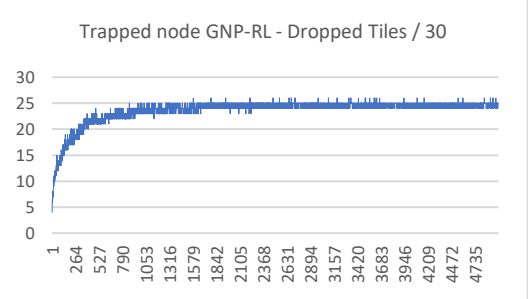


Figure 6.6 Constraint conformance GNP-RL Training Accuracy with correctly Dropped Tiles (Training Set (2))

Testing on Experiment (1) – Training Set(1) Figure 1.1+ Testing Set (1)Figure 1.3

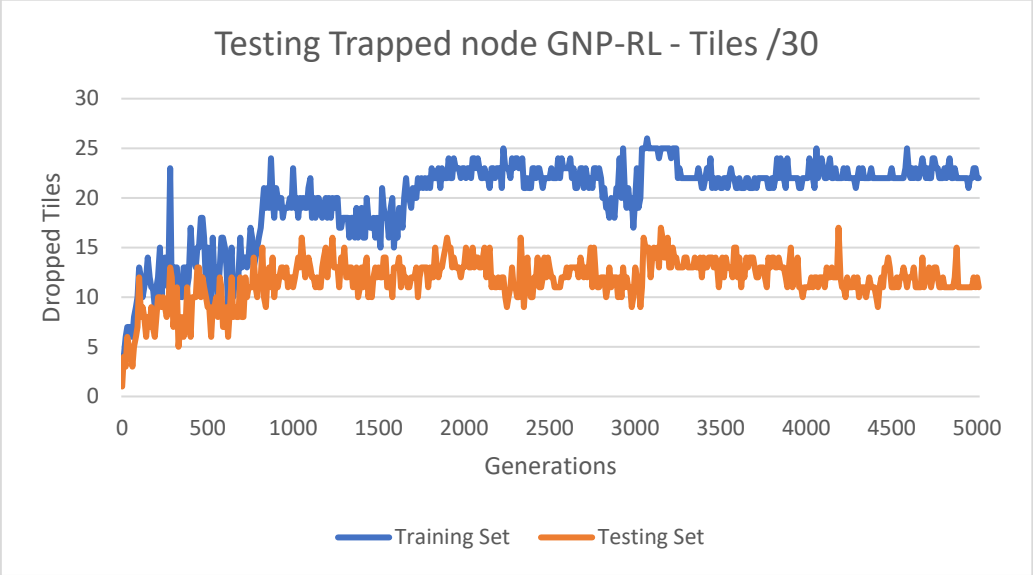


Figure 6.7 Testing Constraint conformance GNP-RL on the Experiment (1)

Testing on Experiment (2) – Training Set(2) Figure 1.2 + Testing Set (1)Figure 1.3

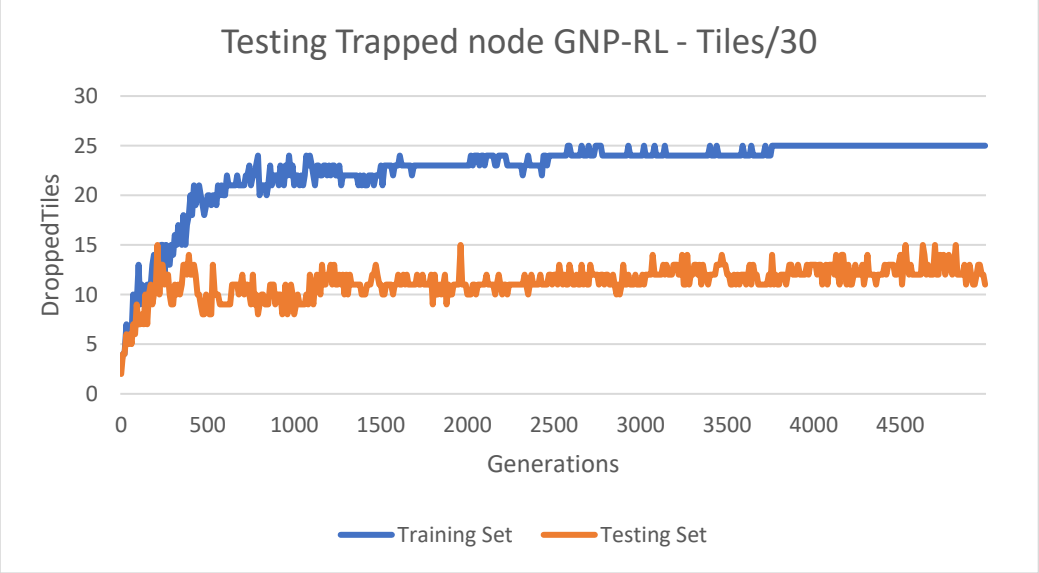


Figure 6.8 Testing Constraint conformance GNP-RL on the Experiment (2)

To verify the effectiveness of the algorithm, we tested the best chromosomes every ten generations once on the training set and again on the test group. We tested over the first generation to the 5000 generation to make sure that the test results did not get a drop-in over-training. It is noted that when the test results are increased on the training group, the test results also increase in the test group to a certain extent, then the test results stop growing and get fluctuated. From Figure 6.7 and Figure 6.8, we note that the test results on the first experiment

were better than the test results on the second experiment. The best individuals achieved 26 results over 30 in the training group the equivalent of (86.6%), and 17 over 30 in the test group (56.6%).

6.5 Summary

After noting the reason for losing the Tiles which was when the Tile trapped in some locations that make it could not be pushed again, we came with a Constraint conformance technique which was applied in the basic GNP-RL. This technique has two main methods; the first one is a simple changing in the chromosome structure by giving two connections for the Processing node with ID = 1 (function go forward) and call this node Trapped node. The second method trains the agents on how to act when facing a trapped location by rewarding and punishing them when updating the Q-value for the RL. When implementing this mechanism, a better result is shown in the training and testing phases with (86.6%) in the training set, and (56.6%) of testing success.

Even this technique achieved a better result than the basic GNP-RL, but it still needs to be improved. One of the problems observed when testing the best chromosome on the environment is that the agent tries to reach the tiles without taking into consideration the obstacles between them, we noticed that the agent tried to across the obstacles to access the tiles, and this made it lose steps without actually reaching the tile (See this video https://youtu.be/u7Ht4V_qhvk). This issue will be discussed in the next chapter.

Chapter 7 (Proposed Architecture #3) GNP-RL with Optimal search and Constraint Conformance

7.1 Motivation

One of the reasons why we use the optimal search method in the GNP-RL algorithm is how the Agent determines the direction of the nearest tile or hole. We have noticed that the technique used by Mabu in 2007 with GNP-RL [9] to determine the direction of objects is the way shown in Figure 7.1 (a) by specifying the direction of the goal for the agent regardless of any obstacles between them. That led to the agent trying to reach this direction and lose many steps, especially if there are obstacles between them. To solve this problem, we used the Optimal Search algorithm (A*) in determining the shortest path between the agent and the target, whether Tile or Hole. When determining the shortest way, the first step in the path is used to determine the direction to the nearest Hole or Tile. As shown in Figure 7.1 (b).

7.2 Related work

In the previous implementations for the GNP-RL on the Tile world problem, the way that used to answer the judgment node (5,6,7, and 8) for detecting the directions to a specific objects is way (a) in Figure 7.1 which detects the directions from the agent to the object without considering any other objects between them. This way was used by Mabu in 2007 [9] to solve the tile world benchmark with GNP-RL. He also used the same way in 2014 with the VSGNP-RL [2]. On the other hand, there is another way to find the path between two objects which uses the heuristic methods to find the shortest path with considering the obstacles and walls. In 1971, Nilsson used heuristics to take advantage of feedback from the environment to make the process of finding the shortest path with A* algorithm faster [48]. The heuristics could be found by using many ways such as Manhattan distances, Euclidean distance [44]…etc. In this chapter, a Manhattan heuristic and A* algorithm have been used to answer the judgment nodes (5,6,7, and 8) by finding the shortest path to the goal then detecting the direction to the first point in this path as the answer.

7.3 The algorithms

Apply A* algorithm with Manhattan Distance to answer the Judgment node

```
Apply_A_Star(start, goal, grid_world)
1.      calculate the heuristics for the all vertexes in the grid world
        h ← |vertex.row – goal.row | + |vertex.col – goal.col |
2.      Q←Ø
3.      ExpandedList ← Ø
4.      start.f ← start.h
5.      start.g ← 0
6.      Q.push(start)
7.      While(Q not empty)
8.          Q.pop(path with the smallest f)
9.          If the goal is found Then stop and return the shortest path
10.         For all the neighbours of the vertex (except the obstacles)
11.             If the neighbour is not in the expandedList
12.                 g+=1
13.                 f = h+g
14.                 Add this neighbour to the current path
15.                 Q.push(newpath)
16.                 expandedList.push(neighbour)
```

Pseudo Code 7.1 A algorithm with Manhattan heuristics modified from [49]*

A* guarantees finding the optimal path provided that the heuristic function used is admissible. Using the optimal search with the heuristic function allows detecting the nearest object to the agent (the object with the smallest cost path) and then detecting the direction to it.

To make A* more efficient; that is, to prevent it from expanding states more than once, we made use of the strict expanded list and made sure that the heuristic is also consistent. This approach guarantees that in the worst case, if there are N states in the problem domain, there is a maximum of N state expansions.

Population Size	**Crossover**	**Mutation**	**Elite**	**Max # sub nodes**
300	120	175	5	4
# Nodes	**# Judgment nodes**	**# Processing nodes**		**Tournament Size**
120	80	40		7
Crossover rate Pc	**Mutation rate Pm**	**ε**	**γ**	**α**
0.1	0.01	0.9 → 0.1	0.9	0.9
Delay time	1 → Judgment node 5 → Processing node		**Agent Steps**	60 for each agent

Table 7.1 GNP-RL with Optimal Search and Constraint Conformance Parameters

7.3.1 Individual (chromosome) structure

There is no changing on the chromosome structure from the previous chapter (see Individual (chromosome) structure).

7.3.2 Initial the first population

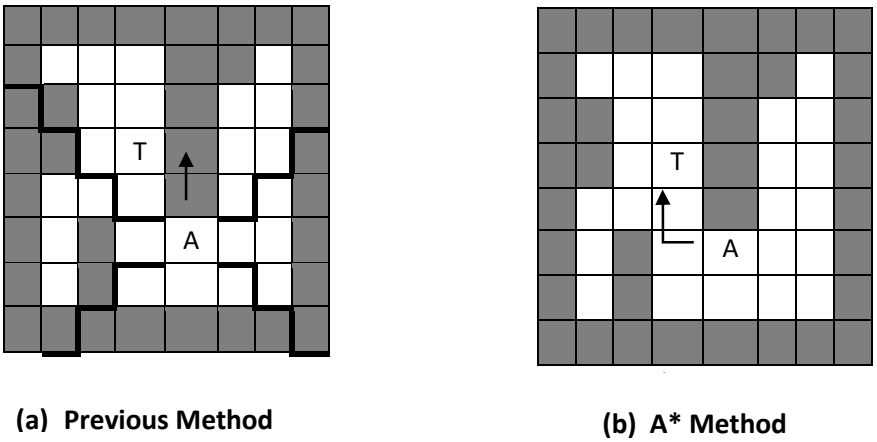
According to the chromosome structure, and the initial population technique, they have implemented exactly as the Constraint conformance GNP-RL implementation (see page 29).

7.3.3 Evaluate the chromosome

Using A* algorithm inside the GNP-RL graph has produced to give more accurate directions in the four judgment nodes (5,6,7, and 8) which asked for the directions. This method has been implemented after noting that in some cases in J5, J6, J7, or J8 the answer gives the direction regardless of the presence of obstacles or agents in its way. So, by using A*, it will provide the shortest path to the detected goal and then it will provide the direction to the first vertex in that path as the answer to the judgment node. For instance, if the judgment node is (what is the direction to the nearest tile), A* algorithm will set the agent as the start point and the tiles as the goals. By finding the shortest paths to the available tiles, the nearest tile will be the tile with the smallest cost path, and the direction will be the direction from the agent to the first point in the shortest path. As in Pseudo Code 7.1 A* algorithm starts its working by calculating the heuristic with using Manhattan distance for the all vertex in the environment as in Equation 7.1. It starts with an empty queue and an expanded list. Then the start vertex with f = h and g = 0 will be pushed to the queue. The algorithm works on a loop until one of the two conditions is met. The first is to find the shortest path for the target and the second when the queue is empty. In each loop, the path with the smallest f value will be picked to explore its neighbors and add them with the picked path as a new one to the queue to explore it another time.

$$\text{Heuristic} = |\text{vertex}_{\text{row}} - \text{goal}_{\text{row}}| + |\text{vertex}_{\text{col}} - \text{goal}_{\text{col}}|$$

Equation 7.1 Heuristic Manhattan Distance



(a) Previous Method **(b) A* Method**

Figure 7.1 Find the directions ways: (a) Using Global information to detect the direction. (b) using A to determine the direction.*

Depending on the figures above Figure 7.1 (a) and (b), if the judgment node asks about the direction to the nearest tile in the previous method, the answer will be Forward. But with using A*, the answer will be Left, which is the direction to the first vertex in the shortest path. So now the answer will be more accurate, and the agent can go to the correct direction instead of trying to cross the obstacle.

7.3.4 Evolutionary Operators

The crossover and mutation in this implementation have been executed as the Architecture #2 evolutionary operation (see page 63).

7.4 Empirical testing and analysis

To test the effectiveness of using A* in the judgment nodes with (ID = 5,6,7, or 8), an implementation of GNP-RL with A* has been done on training environments set (1) Figure 1.1. The results as in Figure 7.2, which have achieved a maximum of 23/30 correctly dropped tiles. This result was not good enough to start the testing stage.

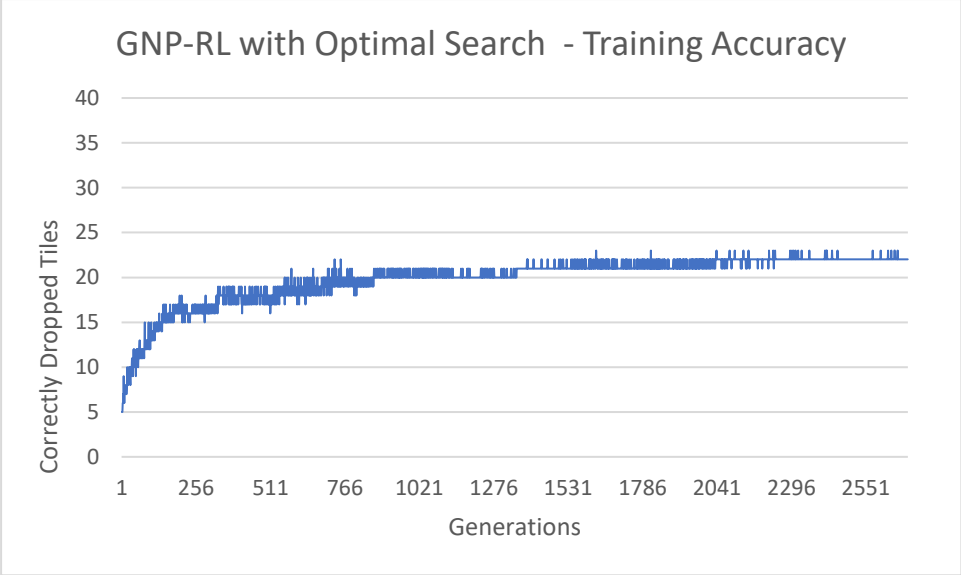


Figure 7.2 A & GNP-RL Training Accuracy with the Training Set (1) by Correctly dropped tiles /30*

After that, we decide to apply A* technique on the Constraint conformance GNP-RL. That means use trapped node in the processing nodes and use A* node with the judgment ones, combine both methods had great results in the training and testing phases.

In an experiment (1) the training phase (on the training set (1) Figure 1.1), the results reached the highest point in only 1,000 generations. Through this result the ability of this technique to find the best results in a faster time is proved by the strength of the A* algorithm to find the target accurately, making the agent reach the goal faster. However, in the experiment (2) the training on the training set (2), the results have not been able to reach the top within the first 1000 generation. That because training set (2) is more complicated, and that made the learning difficult for the agent as what happened on the constraint conformance experiments (see page 64).

Experiment (1) – Training Set (1) [2] **Figure 1.1**

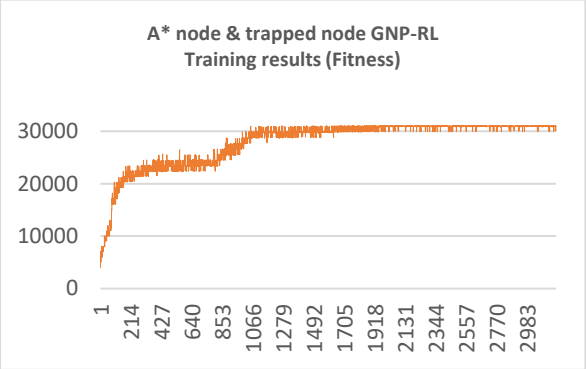


Figure 7.3 A node & Constraint conformance GNP-RL- Training Results (training set (1)) with the Fitness*

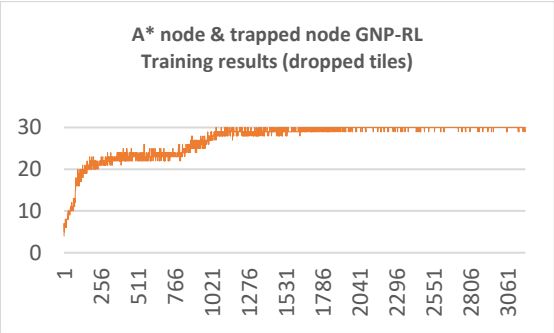


Figure 7.4 A node & Constraint conformance GNP-RL- Training Results (training set (1)) with the number of correctly dropped tiles*

Experiment (2) – Training Set (2) Figure 1.2

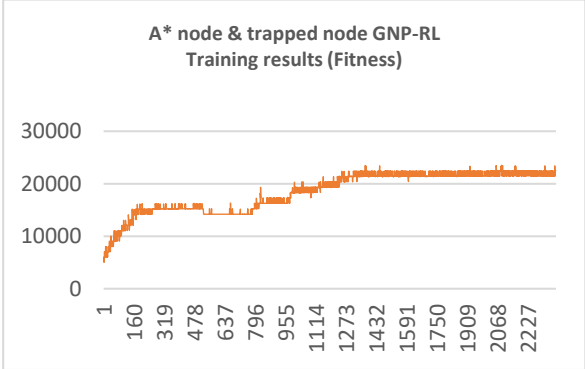


Figure 7.5 A node & Constraint conformance GNP-RL Training Results (Training Set(2)) with the Fitness*

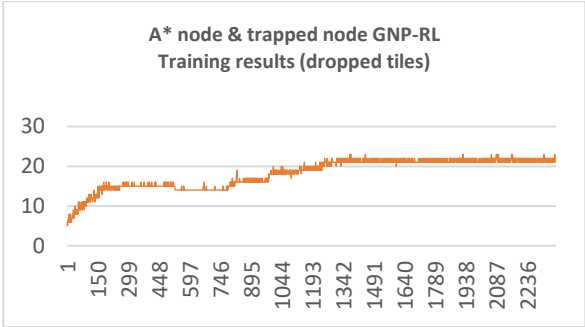


Figure 7.6 A node & Constraint conformance GNP-RL Training Results (Training Set(2)) with the number of correctly dropped Tiles*

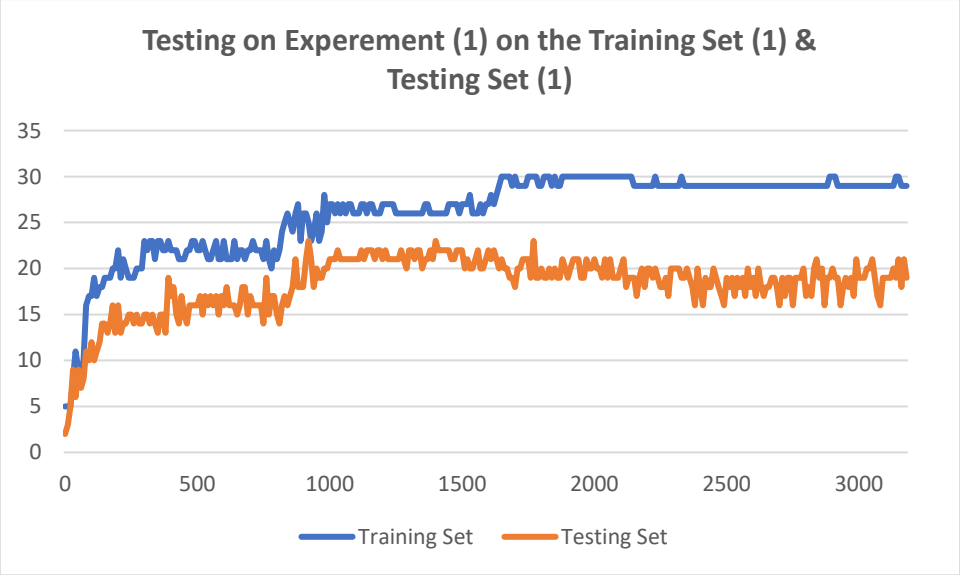


Figure 7.7 A node & Constraint conformance GNP-RL Testing Accuracy (Experiment (1)) with the number of correctly dropped Tiles*

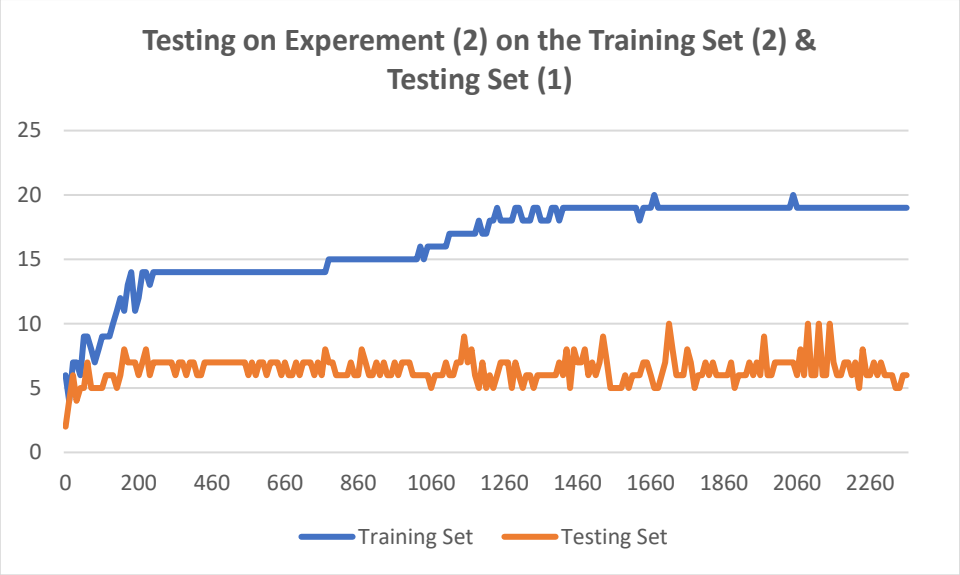


Figure 7.8 A node & Constraint conformance GNP-RL Testing Accuracy (Experiment (2)) with the number of correctly dropped Tiles*

When testing the best individual on the training set Figure 1.1, it gave 30/30 correctly dropped tiles (100%) of accuracy, and on the testing set (1) Figure 1.3, the results are as shown in Table 7.2 which gave 23/30 correctly dropped tiles and that provide a 76% of accuracy in the testing phase. This result was better than the best result that was produced by Li in 2018 [1], for the 2018 results, the test was random (randomly place initial positions of tiles/robot), and the averaged highest score was 19.3/30 = 64.33% of accuracy. To make the comparison fair, we created a group of random environments under the same conditions used in 2018 [1] in the random elementary positions of the tiles, and we tested the best chromosomes on them, and the results were as in Table 7.3 which was 24/30 (80%) of success on the testing set (2) Figure 1.4.

Best individual Test Accuracy on the Testing Set (1) Figure 1.3

Generation number 1770 in the test environment	
Env	Number of correctly dropped tiles
1	2
2	2
3	3
4	3
5	1
6	3
7	2
8	2
9	2
10	3
Total	23/30 → 76%

Table 7.2 Best individual test results on the testing set (1) from generation #1400

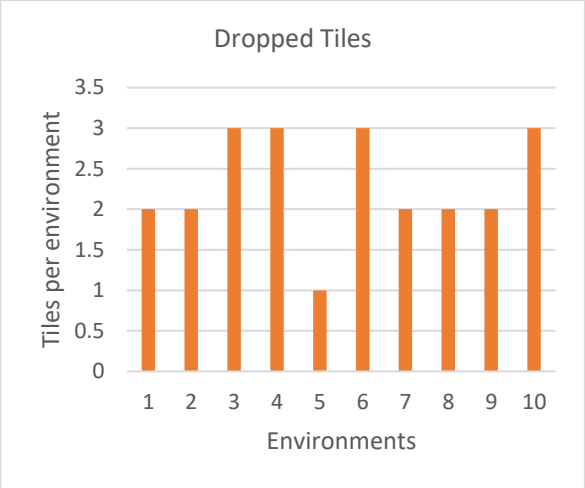


Figure 7.9 number of correctly dropped tiles for each environment for the best chromosome in the testing set (1)

Best individual Test Accuracy on the Testing Set (2) Figure 1.4 (Random Tiles initial positions)

Generation number 1770 in the test environment	
Env	Number of correctly dropped tiles
1	2
2	1
3	3
4	3
5	2
6	3
7	3
8	1
9	3
10	3
Total	24/30 → 80%

Table 7.3 Best individual test results on the testing set (2) from generation #1400

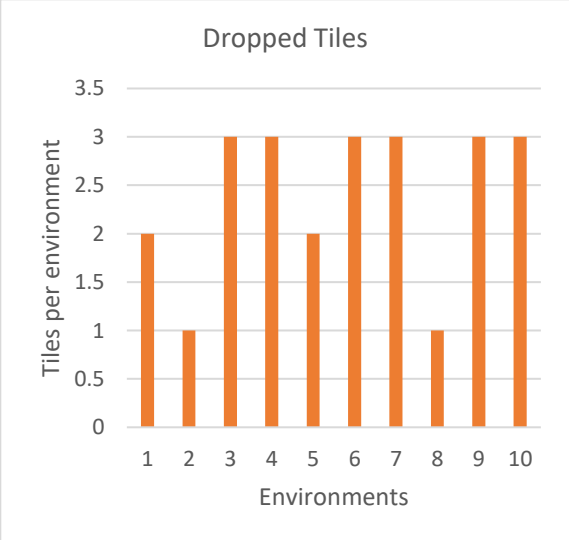


Figure 7.10 number of correctly dropped tiles for each environment for the best chromosome in the testing set (2)

We figure out that the GNP algorithm can implement an acceptable graph and the reinforcement learning algorithm can train the agents on this graph to choose the best sub node from each node, but after each generation, a crossover and mutation are executed in the same graph. So, the sub-nodes, the nodes, and their connection are changed within these operations. In this point, we decided to pick the best chromosome and train the agent on it more with the RL without applying the crossover and mutation on it, to check if the RL can help raise the performance of the chromosome, and the result we got as shown in Figure 7.11. Training the agents more using Reinforcement Learning did not increase the chromosome performance. Instead, the results fluctuate between 5 tiles (which is the lowest result) and the highest value obtained by the chromosome before (which is 30 for the training and 24 for the testing). So, this technique proved inefficient.

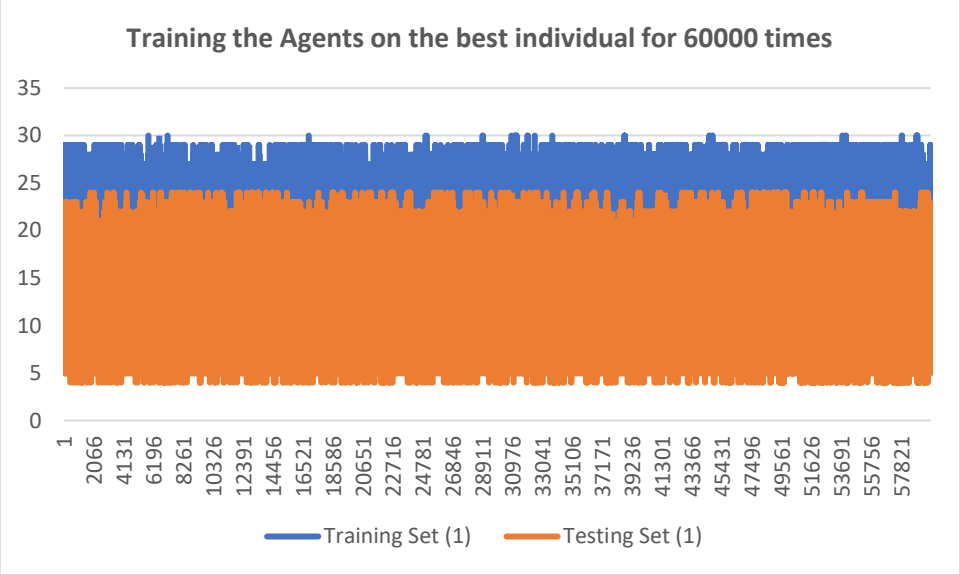


Figure 7.11 Training the Agents on the best individual for 60000 times

7.5 Summary

To make the direction determination more precise and to solve the problem of the agent trying to overcome the obstacles, we have introduced the optimal algorithm (A* algorithm) in the Judgment node as a way to determine the first point of the shortest path to the target. When adding this technique to the constraint conformance GNP-RL, the results achieved high performance of the algorithm in the training and testing phases. In training, we got the most top results in a shorter time, and in the testing phase, we got the best results discovered so far on the tile world problem. However, when we saw the agents working on the tile world problem in the testing phase, we noticed that the agents lost some tiles due to the agent fighting on the

tiles, mainly if one tile remained. This problem was the result of not taking into account the priority of tasks at execution between the agents (See this video https://youtu.be/jJQeM8e85p4). So, in the next chapter, we will address this problem.

Chapter 8 (Proposed Architecture #4) GNP-RL with Task Prioritization, Optimal Search and Constraint Conformance

8.1 Motivation

Cooperative work of agents increases the efficiency and performance of the system. Particularly when the agents cooperate to deliver the target to the desired place. In some cases, agents fight on one Tile, so that all agents attempt to push the Tile to the target location at the same time without direct contact between them. This leads to the agents wrapping around the target so that no one can push the Tile to the right place. This problem arises if one tile remains in the environment, or when one of the Tiles is the closest tile to two or more Agents. In these two cases, the agents dispute to work on this target, so all agents work on this tile at the same time because it is the closest and ignore the rest, leading to loss of the steps, time, and effort. To resolve this problem, we have added priority to the execution of tasks for agents, so that the agent is trained to prioritize the execution of tasks by promotional rewards. This method will be explained in detail in The algorithms section below.

8.2 Related work

The GNP-RL algorithm has been applied to the tile world problem in many researches from 2007 to today. Most of the research has determined the reward of the agent in the RL by (1 if the agent dropped the Tile into the Hole) such as Mabu in 2007, 2014 [9] [2], and Li in 2018 who set the equivalent values either by 1 for the Processing node or 0 for the Judgment node [1]. Since we have set the equivalence values either 1 or -1 in the Proposed Architecture #2 (see page 58), we will discuss in this chapter the use of scalable values of the reward from -10 to 10 to give each action its own value and to make the agents perform the tasks as a priority.

8.3 The algorithms

Apply the priority tasks in the Processing Node (Go Forward function)

```
Int Apply_Processing(ID)
1.      Switch(ID):
2.              Case(1):
3.                      If the agent pushes the nearest tile to the nearest hole
4.                              Reward ←4
5.                      ElseIf the agent pushes the nearest tile nearest to not the nearest hole
6.                              Reward ←2
7.                      ElseIf the agent pushes the nearest tile far away from the nearest hole
8.                              Reward ← -1
9.                      ElseIf the agent pushes a not nearest tile nearest to the hole
10.                             Reward ← 2
11.                     ElseIf the agent pushes a not nearest tile far away from the nearest hole
12.                             Reward ← -1
13.                     ElseIf the agent pushes the tile to the hole
14.                             Reward ← 10
15.                     ElseIf the agent pushes the tile to the trapped location
16.                             Reward ← -10
17.                     Go Forward if it is not trapped location;
18.             Case(2): Turn Left;
19.             Case(3): Turn Right;
20.             Case(4): stay;
21.     Return Reward;
```

Pseudo Code 8.1 Apply the priority tasks in the Processing Node (Go Forward function)

Prioritization technique combines the reward and punishment techniques with the optimal search to train the agent how to perform task prioritization. Applying the optimal search algorithm with the heuristic function on the problem before running the GNP-RL to order the tasks required for each agent allows for the sorting and distributing of the tasks to each of the agents without requiring any communications between them. Using the promotion rewards and punishments brings about the cooperative behavior between the agents that helps them to perform the tasks and solve the problem as a team. This is because each time the reward becomes higher, Q-value increases for the sub node, and thus to rise in probability of choosing the same sub node next time. The same is true with the effect of punishment. Each time the punishment is made higher (reward becomes lower), the Q-value decreases, diminishing the probability of choosing the same sub node again next time.

Population Size	Crossover	Mutation	Elite	Max # sub nodes
300	120	175	5	4
# Nodes	**# Judgment nodes**	**# Processing nodes**		**Tournament Size**
120	80	40		7
Crossover rate Pc	**Mutation rate Pm**	**ε**	**γ**	**α**
0.1	0.01	0.9 → 0.1	0.9	0.9
Delay time	1 → Judgment node 5 → Processing node		**Agent Steps**	60 for each agent

Table 8.1 GNP-RL with Task Prioritization, Optimal Search and Constraint Conformance Parameters

8.3.1 Individual (chromosome) structure

The same chromosome structure in Figure 6.2 Constraint conformance Chromosome Structure is used here in this implementation.

8.3.2 Initialize the first population

The same initial population in constraint conformance GNP-RL page 62.

8.3.3 Evaluate the chromosome

The priority tasks execution (own tile priority technique) is applied in the evaluation stage. This method has implemented within two parts. The first part is applied before the algorithm starts; A* algorithm should be applied to the environment to decide which tile and which hole is the nearest to each agent. So, each agent has its own tile and own hole depending on the shortest paths that have been discovered by A* algorithm. After that, to train the agent how to follow the own Tile and Holes first, the Reward in the (Equation 3.1 Update Q-value) has different values depending on which tiles and any Holes the agent has done during the current move. The Rewards are described in Pseudo Code 8.1 and these points:

a. If the agent pushes the nearest tile to the nearest hole, the algorithm will give reward 4.
b. If the agent pushes the nearest tile nearest to (not the nearest hole), the algorithm will give reward 2.
c. If the agent pushes the nearest tile far away from the closest hole, the algorithm will give a reward -1.
d. If the agent pushes a (not nearest tile) nearest to the hole, the algorithm will give reward 2.
e. If the agent pushes a (not nearest tile) far away from the nearest hole, the algorithm will give reward -1.
f. If the agent pushes the tile to the hole, the algorithm will give reward 10.
g. If the agent pushes the tile to the trapped location, the algorithm will give a reward -10.

When the agent pushes the closest Tile to the nearest Hole, that means the agent work on his own Tile and Hole, in this action; the agent will get reward 4. But if the agent pushes the nearest Tile closer to another Hole (This hole is not the Hole that was chosen by A* as the closest one) the agent will get reward 2 because it follows the priority in the Tile but not in the Hole. In case if the agent pushes any Tile closer to the nearest Hole, it will get reward 2, due to its following in the Hole priority but not for the Tile. For any actions that the agent pushes any Tile far away from the nearest Hole, it will get punished by -1.

To add the priority technique to the constraint conformance technique the last two steps (f & g) have been added to the previous points which they are, the agent will get reward 10 if it dropped any Tile into any Hole, and it will get punishment -10 if it tries to push the Tile to a trapped location.

8.3.4 Evolutionary Operators

The same crossover and mutation operations that have been used with constraint conformance GNP-RL (see page 63) is used in this implementation.

8.4 Empirical testing and analysis:

After implementing the task prioritisation (priority tasks execution) technique to the (A* node & Constraint Conformance GNP-RL algorithm) on the training set (1) Figure 1.1, the training results were excellent with experiment (1), the algorithm was able to reach the top performance after only 120 generations, and that saves a lot of time as shown in Figure 8.2. In the testing stage, it clearly to notice that the results are significantly increased until getting the best individual. After that, there was a fluctuation and decreasing in the results due to the inability of the algorithm to learn more, and this is what we observed in previous implementations. The best chromosome was appear in the generation number 860, which was able to get 100% of accuracy when testing it in training set (1) and (26/30) 86.66% of accuracy in the testing set (1), and 29/30 (96.66%) int Testing Set (2). This result is the best result that appears until now. On the other hand, in the experiment (2) in training the results could not reach the top until after 600 generations, and for the testing, the algorithm was able to get 100% of accuracy in the Training Set (2) after 2400 generations. That also proves that the easier the training environment, the easier it is to learn.

Experiment (1) – Training Set (1) [2] **Figure 1.1**

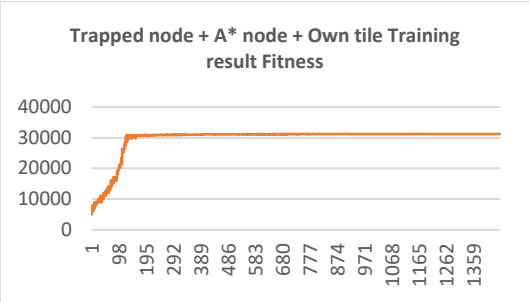


Figure 8.1 Constraint conformance + A node + task prioritisation training result on Training Set (1) by Fitness*

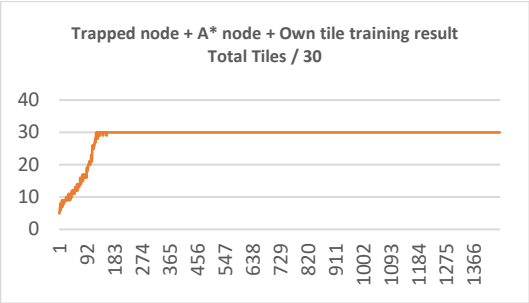


Figure 8.2 Constraint conformance + A node + task prioritisation training result on Training Set (1) by Number of correctly dropped Tiles*

Experiment (2) – Training Set (2)Figure 1.2

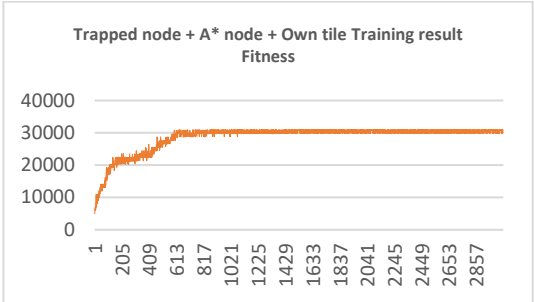


Figure 8.3 Constraint conformance + A node + task prioritisation training result on Training Set (2) by Fitness*

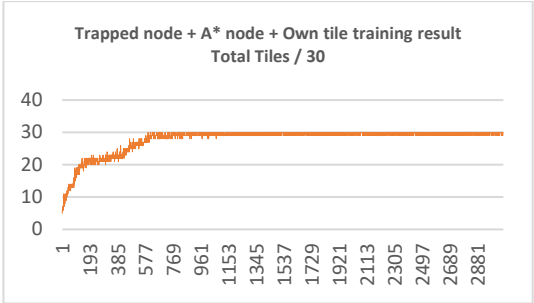


Figure 8.4 Constraint conformance + A node + task prioritisation training result on Training Set (2) by Number of correctly Dropped Tiles*

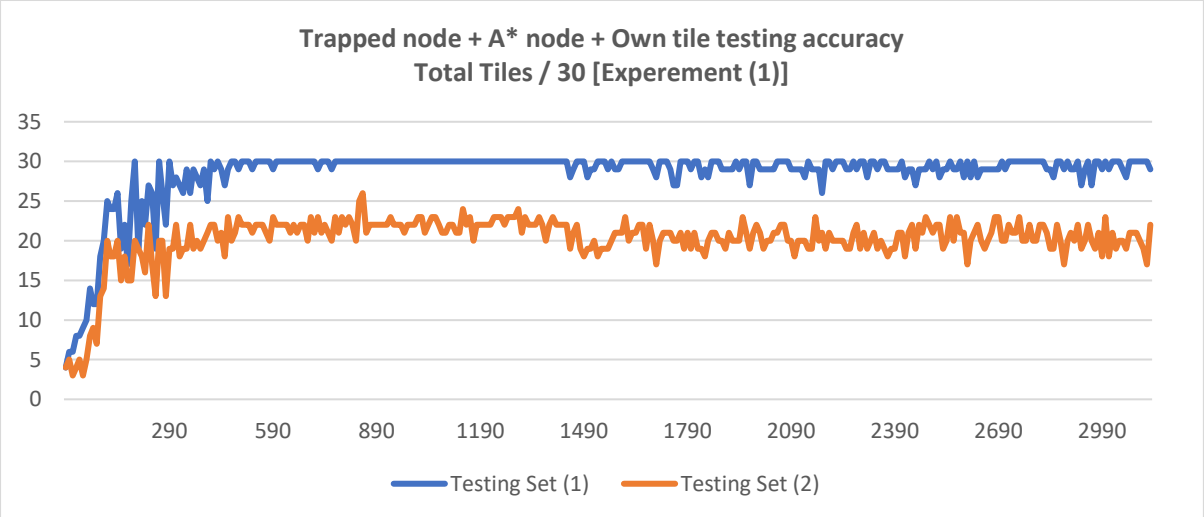


Figure 8.5 Constraint conformance + A node + Task prioritisation testing results on the experiment (1)*

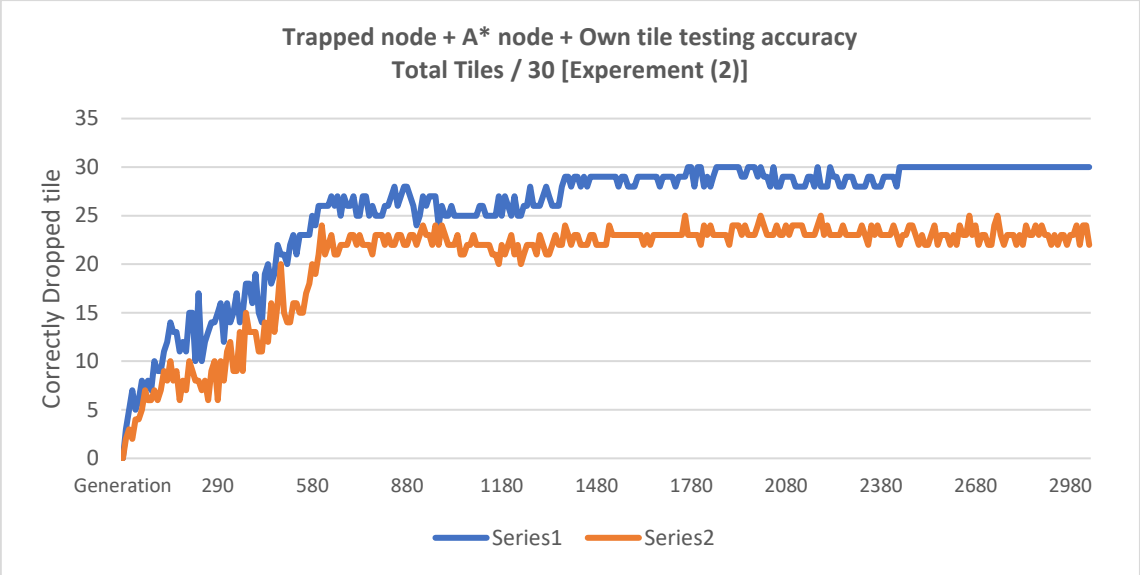


Figure 8.6 Constraint conformance + A node + task prioritisation testing results on the experiment (2)*

Best individual Test accuracy on the Testing Set (1) Figure 1.3 [2]

Generation number 860 in the test set (1)	
Env	Number of correctly Dropped Tiles
0	3
1	3
2	2
3	3
4	3
5	3
6	3
7	1
8	2
9	3
Total	26/30 → 86.66%

Table 8.2 Best Individual Test Result on the Testing Set (1) from generation #860

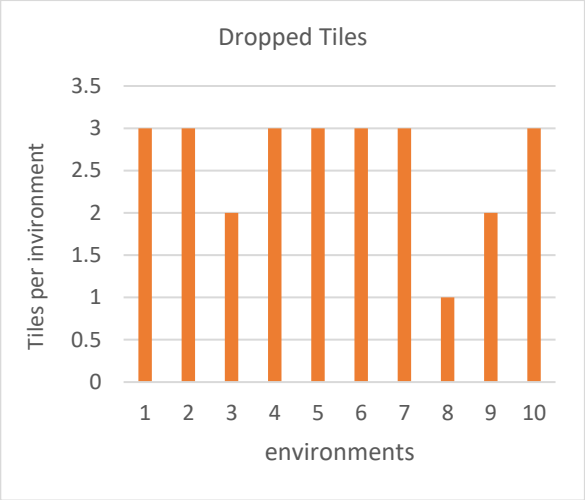


Figure 8.7 Best Individual Test Results on the Testing Set (1)

Best individual Test accuracy on the Testing Set (2) Figure 1.4 (Random Tiles initial positions)

Generation number 860 in the test set (2)	
env	tiles
0	3
1	2
2	3
3	3
4	3
5	3
6	3
7	3
8	3
9	3
Total	29/30 → 96.66%

Table 8.3 Best Individual Test Result on the Testing Set (2) from generation #860

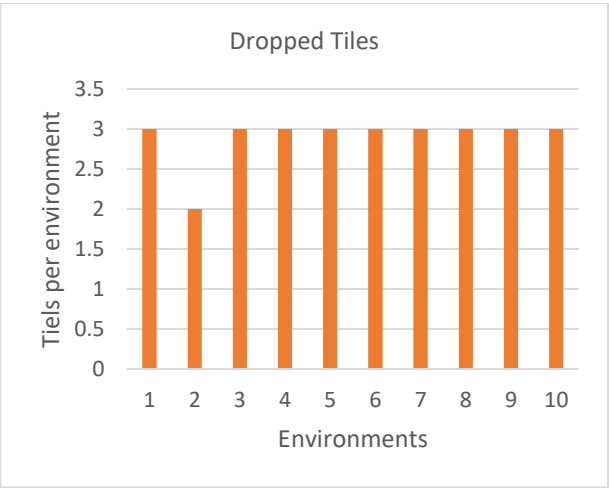


Figure 8.8 Best Individual Test Results on the Testing Set (2)

8.5 Summary

In this chapter, agents were trained to prioritize tasks to reduce their inconsistencies. To do this, we have changed the values of rewards and penalties when the agent moves to update the Q-values depending on the priority of the actions. When adding this technique to the (A* node & constraint conformance GNP-RL) algorithm, we have accomplished the best results achieved so far in applying the GNP-RL algorithm to the tile world problem(100%) on testing the training set (See this video https://youtu.be/QoFgPTMzwfs) and (86.66%) on the testing set (1) (See this video https://youtu.be/LRjvcfV4EyE) and (96.66%) when testing on testing set (2) (See this video https://youtu.be/I-1hTRHWUvY). In order to ensure the efficacy of the algorithm, and that the successful results were not happening randomly, four experiments have been implemented using this algorithm and the average of them has been shown in the next chapter Summary.

Chapter 9 Summary and Conclusions

9.1 Comparison of the Different Variants of the GNP-RL algorithm

Parameters	(Review) GNP-RL [9]	(Review) VSGNP-RL [2]	(Review) GNP with Rule accumulation [1]	(Proposed Architecture #1) GNP-RL with Elitism and Gene Diversification	(Proposed Architecture #2) GNP-RL Constraint conformance	(Proposed Architecture #3) GNP-RL Constraint conformance + Optimal Search	(Proposed Architecture #4) GNP-RL Constraint conformance + Optimal Search + Task prioritisation
By	S. Mabu	S. Mabu	Xianneng Li	Us	Us	Us	Us
In	2007	2014	2018	2019	2019	2019	2019
Chromosomes (Individuals)	300						
Genes (nodes)	120	120	60 5 from each type	120			
Node Types	8 Judgment nodes 4 Processing nodes						
Judgment Nodes	80						
Processing Nodes	40						
Max Subnodes	4						
Subprograms	-	3	-				
Crossover	120						
Mutation	175			170	175		
Elite	5			10	5		
Crossover Rate	0.1		0.9	0.1			
Mutation Rate	0.01	Pm 0.01 Pmtr 0.001 Pmtaff1 0.002 Pmtaff2 0.01	0.01				
Tournament size	7		2	7			
Fitness Function	# correctly dropped tiles	Fitness (1)	Fitness (3)	Fitness (2)			
Agent Steps	60 for each agent for 1 env		500 for the simulation	60 for each agent for 1 env			
Delay time	1 →Judgment						

Parameters	(Review) GNP-RL [9]	(Review) VSGNP-RL [2]	(Review) GNP with Rule accumulation [1]	(Proposed Architecture #1) GNP-RL with Elitism and Gene Diversification	(Proposed Architecture #2) GNP-RL Constraint conformance	(Proposed Architecture #3) GNP-RL Constraint conformance + Optimal Search	(Proposed Architecture #4) GNP-RL Constraint conformance + Optimal Search + Task prioritisation
	5 → Processing						
Q value Equation	Q value Equation Equation 3.1						
γ , ά	0.9						
ε	0.1			0.9 → 0.1 (- 0.1 after each 100 generations)			
Reward	1 if the agent pushes the tile into the hole.		0 → J 1 → P	1 if the agent pushes the tile into the hole.	1 if the agent pushes the tile to the hole. -1 if the agent tries to push the tile to trapped location.		(*)
Training Set	Training Set (1)	Training Set (1)	Training Set (1)		Experiment (1) → Training Set (1) Experiment (2) → Training Set (2)		
Testing Set	-	Testing Set (1)	1- randomly place initial position of tiles/agents. 2- randomly place initial position of tiles/agent, obstacles, holes.	-	Testing Set (1)	Testing Set (1) & Testing Set (2)	
Best Training Result	25/30 (83.3%)	30/30 (100%)	-	26/30 (86.66%)	28/30 (93.33%)	30/30 (100%)	30/30 (100%)
Best Testing Results on Training Set (1)	19/30 (63.33%)	29/30 (96.6%)	24.1/30 (80.3%)	-	25/30 (83.33%)	30/30 (100%)	30/30 (100%)
Best Testing Result on Testing Set (1)	9/30 (30%)	13/30 (43.3%)	-	-	17/30 (56.6%)	23/30 (76%)	26/30 (86.66%)
Best Testing Result on Testing Set (2)	-	-	19/30 (63%)	-	-	24/30 (80%)	29/30 (96.66%)

Table 9.1 Comparing the Algorithms

(*)

a. If the agent pushes the nearest tile to the closest hole, give reward 4.
b. If the agent pushes the nearest tile nearest to (not the nearest hole) give reward 2.
c. If the agent pushes the nearest tile far away from the nearest hole give reward -1.
d. If the agent pushes a (not nearest tile) nearest to the hole give reward 2.
e. If the agent pushes a (not nearest tile) far away from the nearest hole give reward -1.
f. If the agent pushes the tile to the hole, give reward 10.
g. If the agent pushes the tile to the trapped location give reward -10.

Table 9.1 compares 7 algorithms 3 of them are reviewed and 4 are proposed on this thesis.
Reviewed algorithms:

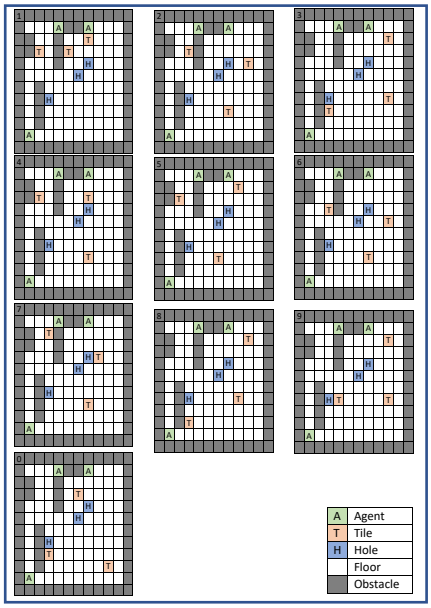
1- GNP-RL, which was in 2007 by Mabu with a training result 25/30 (83.3%), because the algorithm failed in the training stage, we have not started the testing phase.
2- VSGNP-RL which was in 2014 by Mabu with a training result 30/30 (100%), and testing results:
    a. 29/30 (96.6%) when testing on the training set.
    b. 13/30 (43.3%) when testing on the testing set (1).
3- GNP with Rule accumulation which was in 2018 by Li with a testing result:
    a. 24.1/30 (80.3%) when testing on the training set.
    b. 19/30 (63%) when testing on the testing set (2).

Proposed Algorithms:

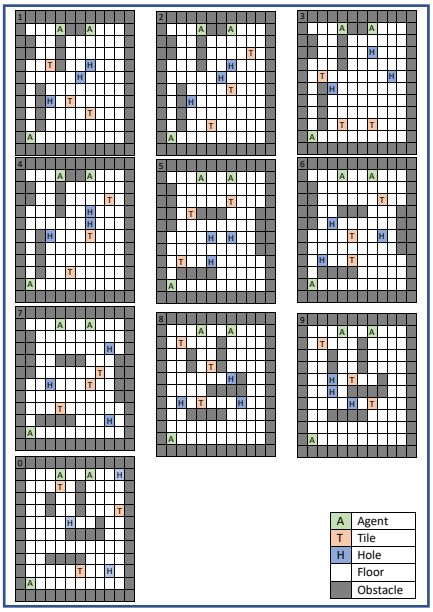
4- (Proposed Architecture #1) GNP-RL with Elitism and Gene Diversification which was addressed in this thesis with a training result 26/30 (86.66%).
5- (Proposed Architecture #2) GNP-RL with Constraint conformance which was addressed in this thesis with training results 28/30 (93.33%), and a testing result:
    a. 25/30 (83.33%) when testing on the testing set (1).
    b. 17/30 (56.6%) when testing on the testing set (2).
6- (Proposed Architecture #3) GNP-RL Constraint conformance + Optimal Search which was addressed by this thesis with a training result 30/30 (100%) and testing result:
    a. 23/30 (76%) when testing on the testing set (1).
    b. 24/30 (80%) when testing on the testing set (2).
7- (Proposed Architecture #4) GNP-RL Constraint conformance + Optimal Search + Task prioritisation which was addressed by this thesis with a training result 30/30 (100%), and testing results:
    a. 26/30 (86.66%) when testing on the testing set (1).
    b. 29/30 (96.66%) when testing on the testing set (2).

Training and Testing Sets

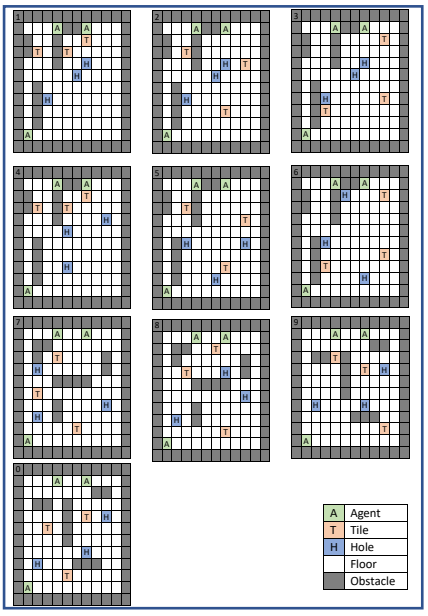
Training Set (1) [2]



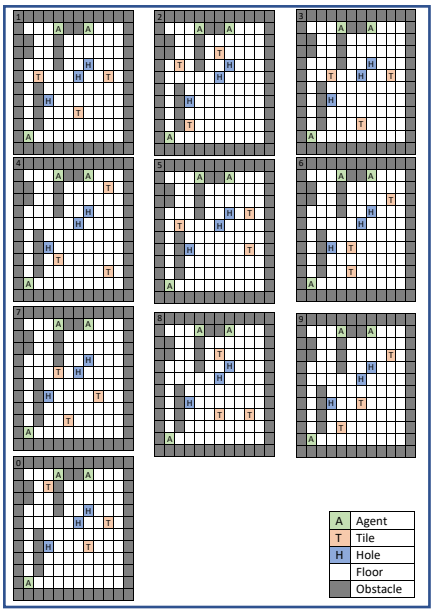
Testing Set (1) [2]



Training Set (2)



Testing Set (2)



Fitness Functions

$$\boldsymbol{Fitness}\ (\mathbf{1}) = \sum_{env=1}^{ENV}[(100\ \times D_{tile}) + (20\ \times\ D_{distance}) + (T_{remain})] \quad [2]$$

Where:

D_{tile}→ number of correctly dropped Tiles.

$D_{distance}$ → how many steps the agent pushes the tile closer to the hole.

T_{remain} → the remain steps for the agents if they finish dropping all the tiles to the hole.

$$\boldsymbol{Fitness}\ (\mathbf{2}) = \sum_{env\ =0}^{ENV} [(1000\ \times D_{tile}) + (D_{distance}) + (T_{remain})]$$

Where:

D_{tile}→ number of correctly dropped Tiles.

$D_{distance}$ → 2 if the agent pushes the tile closer to the hole< and -1 if the agent pushes the tile far away from the hole.

T_{remain} → the remain steps for the agents if they finish dropping all the tiles to the hole.

$$\boldsymbol{Fitness}\ (\mathbf{3}) = \ (100\ \times PT) + (3(ST - ST_{used}) + \left(20\left[\sum_{t\ \in Tile} D(t) - d(t)\right]\right) \quad [1]$$

Where:

PT→ number of correctly dropped Tiles.

ST → the total steps.

ST_{used} → the number of using steps.

$D(t)$ → the distance from the tile to the nearest hole before pushing the tile.

$d(t)$ → the distance from the tile to the nearest hole after pushing the tile.

Reinforcement Learning Equation

$$Qip = Qip + \left(\alpha * (Reward + (\gamma * Qjq) - Qip)\right) \quad [9]$$

Where:

Qip → the Q value for the visited sub node.

Qjq → the Q-value for the chosen sub-node in the next node.

α → a learning rate it could be any value from 0 to 1.

γ → a discount rate it is a value from 0 to 1.

$Reward$ → equal to 1 for each time a tile is pushed into a hole (This value changed from algorithm to another).

9.2 Summary of Performance Analysis

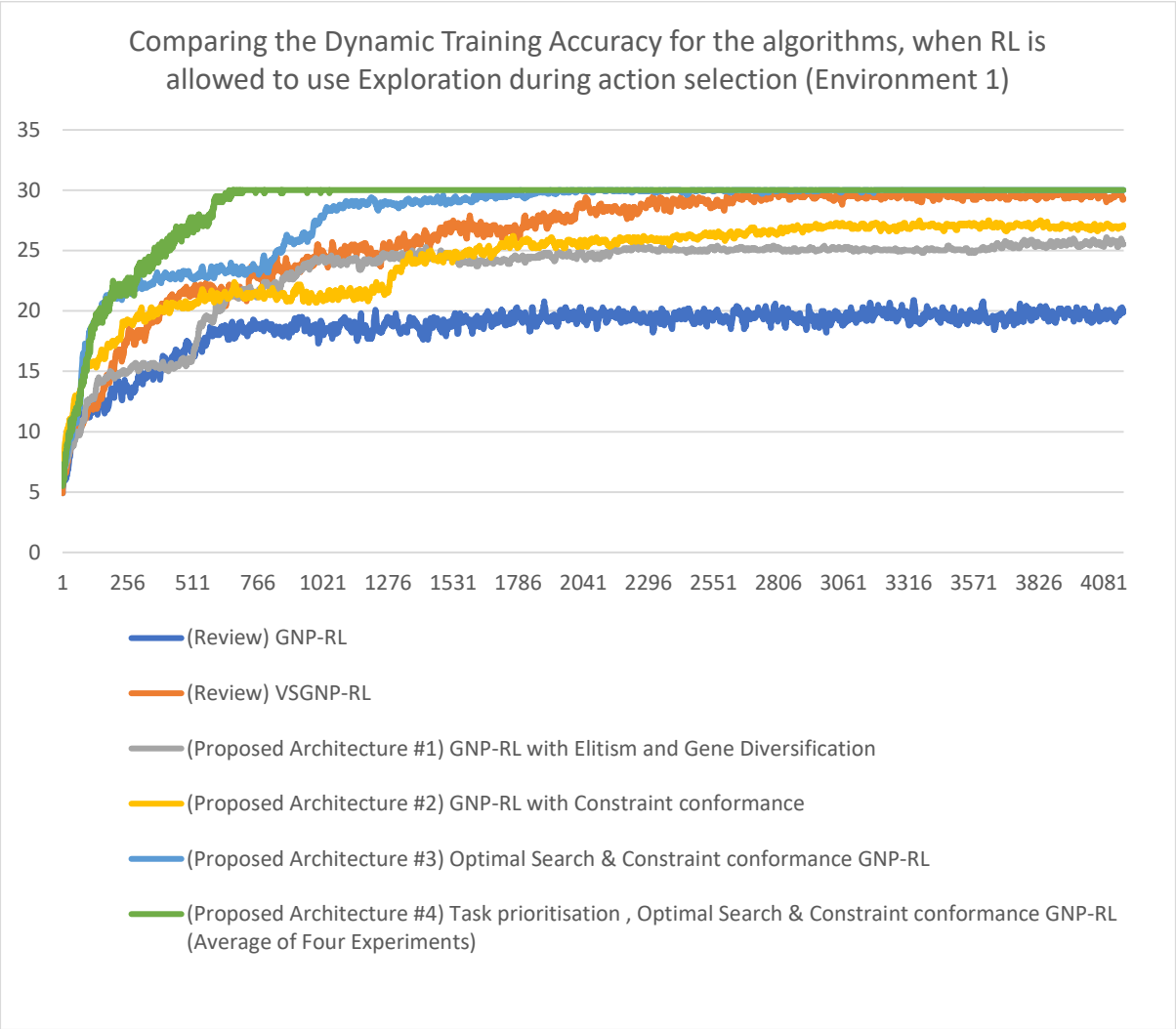


Figure 9.1 Compare the Dynamic training accuracy for the Algorithms by taking the average for each 10 generations

Figure 9.1 shows the dynamic training accuracy for the algorithms when running the RL on the GNP graph by calculating the number of correctly dropped tiles for each generation. The results illustrate the differences between the algorithms, the three algorithms that able to reach the top (30/30) are VSGNP-RL, proposed architecture #3, and proposed architecture #4. It's clear to notice that proposed architecture #4 is the algorithm that reaches the top earlier than the other.

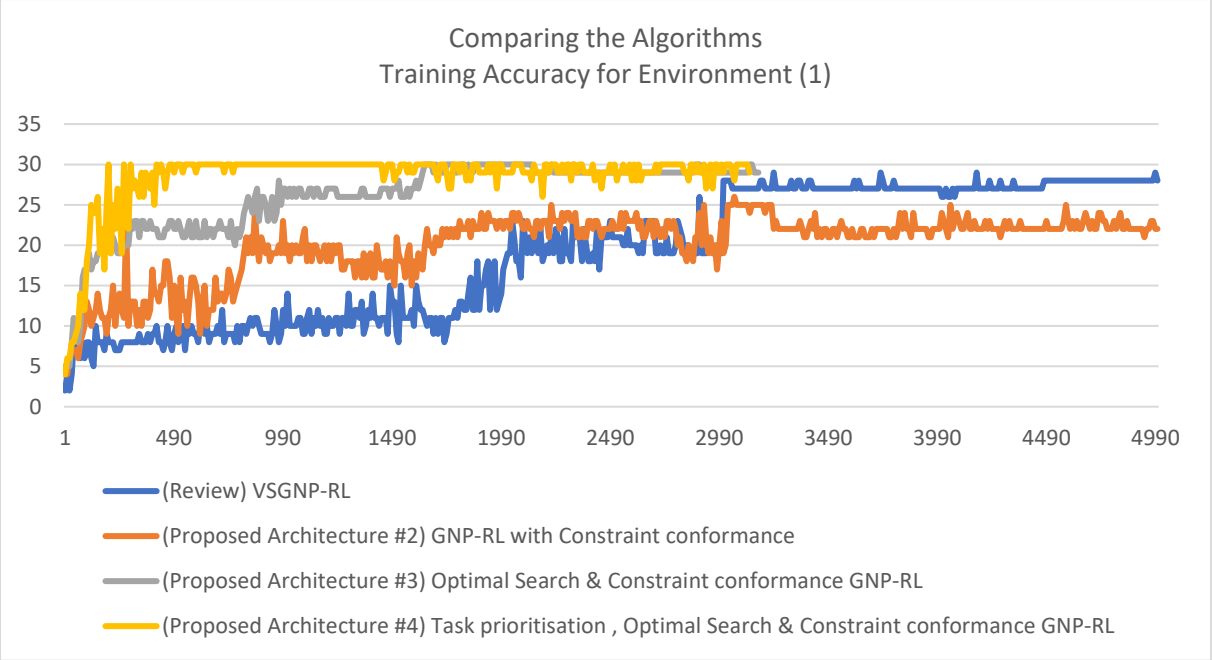


Figure 9.2 Comparing the algorithms – training accuracy for the training set (1)

Figure 9.2 shows the results for the algorithms when testing them on the training set (1) by choosing the sub node with the maximum Q-value. The results have been tested after each 10 generations. From the results, we have figured the overfitted for the algorithms. So, after a number of generations the algorithm results stopped increased; instead they decreased, in this stage, the algorithm has been stopped. Proposed Architecture #4 had the best results and the fastest algorithm that reach the top (30/30) 100% of accuracy. Proposed Architecture #3 also could reach 100% but it needed more time for that. The other algorithms could not achieve 100% of training accuracy.

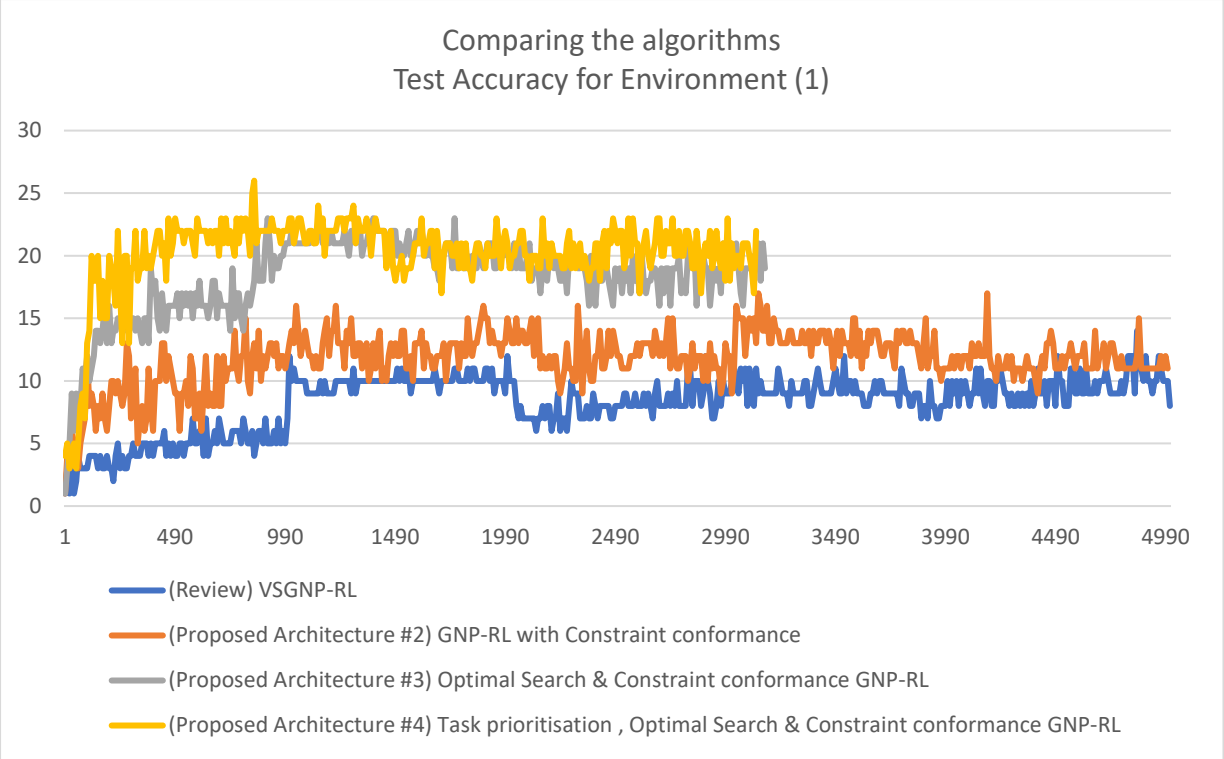


Figure 9.3 Comparing the algorithms' test accuracy for the testing environment (1)

Figure 9.3 shows the test accuracy for the algorithms when testing them on the testing set (1) after each 10 generations. From the figure, Proposed Architecture #4 was the algorithm that gave the best test accuracy on the testing environment (1).

This thesis covered some of the GNP-RL algorithms and their implementations on applying them on the Tile world problem (GNP-RL, VSGNP-RL GNP-RL with Elitism and Gene Diversification, Constraint conformance GNP-RL, Optimal Search & Constraint conformance GNP-RL, Task prioritisation & Optimal Search & Constraint conformance GNP-RL).

9.2.1 GNP-RL

This algorithm invented by Mabu in 2007 [9] which combine the genetic network programming with reinforcement learning Sarsa by dividing each node to several sub nodes, and then make the RL train the agent on the graph to choose the best sub node depending on the maximum Q-value. This technique allows the chromosome to contain more nodes to solve the complex problem while keeping the chromosome structure simple.

This algorithm proved superior to the GNP by obtaining higher results. But in spite of that, this algorithm is still unable to solve the tile problem effectively.

9.2.2 VSGNP-RL

In 2014 Mabu [2] produced the VSGNP-RL technique, which divides the chromosome into some subprograms. He implanted this method as a solution to break up the problem into smaller tasks so that each subprogram performs a task to make the agent able to move between subprograms and to solve tasks step by step.

This method achieved a higher result than the basic GNP-RL, but it could not effectively solve the tile world problem. And this method has proved its limitation in two issues; the first one is that when the algorithm is allowed to change the size of the subprograms in the individual, one of the subprograms become very large and contains the majority of the nodes and the other becomes too small; thus, ineffective. The second point is that in some cases when the agent starts working in a subprogram, it hangs inside it and can't move to another subprogram, which makes the agent unable to work correctly.

9.2.3 (Proposed Architecture #1) GNP-RL with Elitism and Gene Diversification

Using the diversity in the population has been discussed in the genetic algorithms, but it has not been covered for the GNP-RL till now. So, we decided to apply diversity in two axes. The first in the selection of the elite and the second when the chromosomes crossover. In the first axis, when selecting the best five individuals (elite) in the population, the algorithm also

selects the most diverse chromosome to the best five and passes them to the new population; thus, the algorithm ensures diversity in the next sample. The second axis is that at the crossover operation, the algorithm selects the first parent based on tournament selection, and the second parent chooses as the most diverse individual from the parent1 in the population. In this way, we guarantee that the nodes will be different in the parents, and this ensures that the children get new features.

These two mechanisms make the algorithm work faster and get the results in early generations, and improve the results slightly, but they also could not solve the tile world problem clearly.

9.2.4 (Proposed Architecture #2) GNP-RL with Constraint conformance

One of the noticed problems when testing the chromosome on the tile world problem is that the agent's loss some tiles due to pushing them to a trapped location, thus prevent them from being pushed again to anyway. To solve this issue, we came with a technique that trains the agent how to act before pushing the tile to a trapped position. This method works on reward and penalizes the agents for training them to avoid pushing the tiles to unsafe locations and to find another path on the graph to solve that.

This method proved it effectively in the training and testing results 28/30 (93.33%) & 17/30 (56.66%) respectively.

9.2.5 (Proposed Architecture #3) Optimal Search & Constraint conformance GNP-RL

With the previous algorithms, the agent did not take into account the obstacles when determining the direction of the target and this was observed when the agent tries to overcome the obstacles to reach the goal and thus lose a lot of steps without actually achieving the goal. Optimal search algorithm (A*) has been used in the Judgment node to solve this problem. The judgment nodes with (ID = 5,6,7, and 8) have been modified to use A* algorithm to find the shortest path to the goal and detect the direction to the first point in this shortest path as an answer for the judgment node.

When adding this procedure to the constraint conformance GNP-RL, the algorithm has improved significantly, with 30/30 (100%) and 23/30(76%) in the training and testing results respectively.

9.2.6 (Proposed Architecture #4) Task prioritisation, Optimal Search & Constraint conformance GNP-RL

Until now, agents work on the nearest target at the moment but without taking into consideration the priority of the implementation of the tasks and this is what made the agents in some cases conflicting on the target, especially if there is one goal is the closest to all agents and so the agents lose their steps without any achievement. Therefore, we have been working on adding priority to the agents. By changing the reward and punishment values for each agent, we have been able to train agents to prioritize by assigning tasks to each agent before starting the algorithm and then rewarding the agent when the same priority is implementing.

This technique has achieved the best results on the tile world problem using GNP-RL so far in training and testing results with 30/30 (100%) and 26/30 (86.66%) respectively.

9.2.7 Running Time

Although the added techniques to the GNP-RL increased the training time especially the Optimal Search algorithm, the results achieved by the new algorithm and the speed of learning made the final total time to reach the best results is much lower than in the previous algorithms. Table 9.2 shows this:

Algorithm	(Review) GNP-RL [9]	(Review) VSGNP-RL [2]	(Proposed Architecture #1) GNP-RL with Elitism and Gene Diversification	(Proposed Architecture #2) GNP-RL with Constraint Conformance	(Proposed) GNP-RL with Optimal search	(Proposed Architecture #3) GNP-RL with Optimal search and Constraint Conformance	(Proposed Architecture #4) GNP-RL with Constraint Conformance, Optimal Search and Tasks Priorities
The average time for each individual	0.554	0.823	0.906	0.748	2.628	2.628	2.696
# individuals	300	300	300	300	300	300	300

Algorithm	(Review) GNP-RL [9]	(Review) VSGNP-RL [2]	(Proposed Architecture #1) GNP-RL with Elitism and Gene Diversification	(Proposed Architecture #2) GNP-RL with Constraint Conformance	(Proposed) GNP-RL with Optimal search	(Proposed Architecture #3) GNP-RL with Optimal search and Constraint Conformance	(Proposed Architecture #4) GNP-RL with Constraint Conformance, Optimal Search and Tasks Priorities
# generations to reach the highest results	5000	2700	5000	5000	5000	1700	600
Total time	831000	670680	1359000	1122000	3664500	1340280	485280
The highest result on the testing stage (testing set (1))	9/30 (30%)	13/30 (43.3%)	-	17/30 (56.6%)	-	23/30 (76%)	26/30 (86.66%)

Table 9.2 Comparing the training time for each algorithm

9.3 Representation of One of the Best Chromosomes for the Tile World Problem

When testing one of the best chromosomes from Architecture #4 on the testing set (1) with a result (26/30), the transitions of the agents have been recorded, and the steps were as Figure 9.4, Figure 9.5, and Figure 9.6 for agent1, agent2, and agent3 respectively (See video https://youtu.be/LRjvcfV4EyE):

Follow the first agent transitions in the 10 environments with the node id:

Agent 1 Env 1 58 38 48 61 3 48 61 3 48 67 47 75 48 67 47 75 48 67 47 75 48 67 47 75 48 61 3 48 67 97 14 7 32 3 48 13 22 38 53 21 31 38 48 112 80 49 22 38 48 61 3 48 13 22 38 48 67 73 99 38 48 112 59 8 36 75 48 13 22 38 48 67 57 52 99 38 48 112 59 8 36 75 48 13 22 38 48 67 47 75 48 67 47 75 48 67 47 75 48 67 47 75 48 67 47 52 99 38 48 13 22 38 48 13 22 38

Agent 1 Env 2 58 38 48 61 3 48 61 3 48 67 47 25 60 48 67 47 25 60 48 67 47 25 60 48 61 3 48 67 47 55 7 32 3 48 13 22 38 48 67 57 25 60 48 67 73 99 38 48 112 59 8 36 75 48 13 22 38 48 67 57 52 99 38 48 112 59 8 36 75 48 13 22 38 48 67 47 75 48 67 47 55 7 32 3 48 61 3

Agent 1 Env 3 58 38 48 61 3 48 61 3 48 67 47 55 7 32 3 48 61 3 48 13 22 95 60 48 67 47 55 7 32 3 48 67 97 14 7 32 3 48 61 3 48 67 47 52 99 38 48 13 22 38 48 13 22 95 60 48 67 97 14 7 32 3 48 61 3 48 13 22 38 48 67 97 71 71 14 7 32 3 48 67 73 99 38 48 13 22 17 97 71 14 7 32 3 37 67 73 99 38 48 13 22 38 48 13 22 17 73 99 73 99 38 48 61 3 48 61 3 48 67 57 25 60 48 67 73 99 73 99 38 48 112 59 8 36 75 48 61 3 48 67 47 55 7 32 3 48 67 97 14 7 32 3 48 61 3 48 67 47 75 48 67 47 75 48 67 47 75 48 67 47 52 99 38 48 13 22 38 48 13 22 38 48 112 59 46 38 48 61 3 48 67 47 25 60 48 61 3 48 67 47 25 60 48 67 47 25 60

Agent 1 Env 4 58 38 48 61 3 48 61 3 48 67 47 25 60 48 67 47 25 60 48 67 47 25 60 48 61 3 48 67 73 99 11 68 17 97 14 7 32 3 48 13 22 38 48 67 47 55 7 32 3 48 13 22 115 38 48 61 3 48 67 97 14 7 32 3 48 67 73 99 38 48 112 59 68 38 48 112 59 68 115 38 48 13 22 38 48 13 22 38 48 13 22 38 48 67 47 52 99 38 48 13 22 38 48 67 57 52 99 38 48 13 22 38 48 61 3 48 67 73 99 38 48 112 80 49 22 38 48 13 22 38 48 67 57 52 99 38 48 112 59 8 36 75 48 13 22 38 48 61 3 48 13 22 38 48 67 47 55 7 32 3 48

Agent 1 Env 5 58 38 48 61 3 48 112 59 8 36 75 48 112 59 68 38 48 112 59 8 36 75 48 13 22 38 48 13 22 38 48 61 3 48 67 47 75 48 67 47 55 7 32 3 48 67 47 55 7 32 3 48 67 97 14 7 32 3 48 61 3 37 67 97 14 7 32 3 48 61 3 48 67 47 52 99 38 48 13 22 95 60 48 13 22 95 60 48 67 47 75 48 67 47 75

Agent 1 Env 6 58 38 48 13 22 95 51 32 3 48 13 22 95 60 48 67 73 99 11 46 38 48 13 22 95 51 32 3 48 61 3 48 112 59 8 36 75 48 13 22 95 60 48 112 59 8 36 75 48 67 47 52 99 38 48 112 59 8 36 75 48 13 22 38 48 61 3 48 67 47 25 60 48 13 22 38 48 67 47 75 48 67 47 75 48 112 59 67 73 99 38 48 13 22 17 47

Agent 1 Env 7 58 38 48 112 59 8 36 75 48 67 47 55 7 32 3 48 67 57 55 7 32 3 48 67 47 55 7 32 3 48 13 22 38 48 67 73 99 38 48 112 59 8 36 75 48 13 22 38 53 103 57 52 99 38 48 112 59 67 47 75 48 112 59 8 36 75 48 13 22 38 48 67 47 75 48 67 47 75 48 67 47 55 7 32 3 48 61 3 48 67 57 75 48 67 47 75 48 67 47 75 48 67 47 75

Agent 1 Env 8 58 38 48 112 59 8 36 75 48 13 22 17 57 75 48 67 57 75 48 67 47 75 48 67 47 55 7 32 3 48 61 3 48 67 47 75 48 112 59 8 36 75 48 13 22 95 60 48 67 47 75 48 61 3 48 67 47 52 99 11 68 38 48 13 22 95 60 48 67 47 52 99 11 68 38 48 13 22 95 60 48 67 47 52 99 11 68 38 48 13 22 95 60 48 67 47 52 99 11 68 38 48 13 22 95 60 48 67 47 52 99 11 68 38 48 13 22 95 60 48 67 47 52 99 11 68 38 48 13 22 95 60 48 67 47 52 99 11 68 38 48 13 22 95 60 48 67 47 52 99 11 68 38 48 13 22 95 60 48 67 47 52 99 11 68 38 48 13 22 95 60 48 67 47 52 99 11 68 38 48 13 22 95 60 48 67 47 52 99 11 68 38 48 13 22 95 60 48 67 47 52 99 11 68 38 48 13 22 95 60 48 67 47

Agent 1 Env 9 58 38 48 61 3 48 67 97 14 7 32 3 48 61 3 48 67 47 75 48 67 47 75 48 67 47 55 7 32 3 48 61 3 48 67 47 25 60 48 67 57 25 60 48 67 57 25 60 53 21 31 38 48 13 22 38 48 13 22 95 60 48 67 47 75 48 13 22 95 60 48 67 97 14 7 32 3 48 67 97 14 7 32 3 48 61 3 48 67 47 55 7 32 3 48 61 3 48 67 47 25 60 48 13 22 38 48 67 47 52 99 38 48 112 59 8 36 75 48 13 22 95 60 48 61 3 48 67 47 52 99 38 48 112 59 8 36 75 48 13 22 95 60 48 61 3 48 67 47 52 99 38 48 112 59 8 36 75 48 13 22 95 60 48 61 3 48 67 47 52 99 38 48 112 59 8 36 75 48 13 22 95 60 48 61 3 48 67 47 52 99 38 48 112 59 8 36 75 48 13 22 95 60 48

Agent 1 Env 10 58 38 48 112 80 49 22 95 104 13 22 38 48 67 97 14 7 32 3 48 61 3 48 13 22 38 48 67 97 14 7 32 3 48 67 97 14 7 32 3 48 67 97 14 7 32 3 48 61 3 48 67 97 75 48 67 97 14 7 32 3 48 61 3 48 67 47 52 99 38 48 13 22 38 48 13 22 95 60

Figure 9.4 Transitions of agent 1 on the 10 environments when running on the graph in Figure 9.7 Figure 9.7

Follow the second agent transitions in the 10 environments with the node id:

Agent 2 Env 1 58 95 51 32 3 48 61 3 48 13 22 38 48 67 47 75 48 61 3 48 67 97 75 48 67 97 14 7 32 3 48 67 97 14 7 32 3 48 61 3 48 67 97 14 7 32 3 48 61 3 48 67 47 52 99 38 48 13 22 38 48 13 22 38 48 61 3 48 13 22 38 48 13 22 38 48 67 73 99 38 48 112 59 8 36 75 48 13 22 38 48 13 22 17 47 75 48 67 47 75 48 67 47 52 99 38 48 13 22 38 48 67 47 52 99

Agent 2 Env 2 58 95 51 32 3 48 61 3 48 67 47 52 99 38 48 13 22 38 48 61 3 37 67 73 99 38 48 112 59 8 36 75 48 13 22 38 48 67 57 52 99 38 48 112 59 8 36 75 48 13 22 38 48 67 47 75 48 67 47 75 48 67 47 75 48 67 97 14 7 32 3 48 61 3 48 67 47 52 99 38 48 112 59 8 36 75

Agent 2 Env 3 58 95 51 32 3 48 61 3 48 67 47 55 7 32 3 48 61 3 48 67 47 75 48 112 59 8 36 75 48 67 97 14 7 32 3 48 67 57 25 60 53 103 57 52 99 38 48 112 59 8 36 75 48 13 22 38 48 67 57 25 60 48 67 73 99 38 48 112 59 68 38 48 112 59 8 36 75 48 67 73 99 38 48 112 59 8 36 75 48 13 22 95 60 53 103 47 52 99 11 68 38 48 13 22 95 60 48 67 47 52 99 11 68 17 73 99 73 99 38 48 61 3 48 67 47 52 99 38 48 13 22 95 60 48 67 57 52 99 11 68 38 48 13 22 95 60 48 67 47 52 99 11 68 38 48 13 22 95 60 48 67 57 25 60 48 67 73 99 38 48 112 59 8 36 75 48 13 22 38 48 67 47 52 99 38 48 13 22 38 48 13 22 95 60 48 67 47 75 48 67 47 55 7 32 3 48 61 3 48 67 47 75 48 67 47 55 7 32

Agent 2 Env 4 58 95 104 112 59 8 36 75 48 112 59 8 36 75 48 67 57 52 99 38 48 13 22 38 48 67 57 52 99 38 48 112 59 8 36 75 48 13 22 38 48 67 47 75 48 67 47 52 99 38 48 13 22 38 48 67 47 75 48 61 3 48 67 57 52 99 38 48 13 22 38 48 67 73 99 38 48 112 59 8 36 75 48 13 22 38 48 67 47 75 48 67 47 52 99 38 48 13 22 38 48 67 57 52 99 38 48 112 59 8 36 75 48 13 22 38 53 104 67 47 75 48 67 47 75 48 67 47 75 48 67 47 75 48 67 47 52 99 38 48 13 22 38 48 13 22 38 48 112 59 8 36 75 48 13 22 38 48 67 47 75

Agent 2 Env 5 58 38 48 112 59 8 36 75 48 112 59 8 36 75 48 13 22 38 48 61 3 48 67 47 75 48 67 47 75 48 13 22 38 48 67 47 55 7 32 3 48 67 97 14 7 32 3 48 61 3 48 13 22 38 48 67 73 99 38 48 13 22 38 48 67 57 52 99 38 48 112 59 8 36 75 48 13 22 17 47 52 99 73 99 11 8 36 75 48 13 22 38 48 67

Agent 2 Env 6 58 38 48 61 3 48 67 47 52 99 38 48 67 47 75 48 61 3 48 67 73 99 38 48 112 59 8 36 75 48 13 22 38 48 67 57 75 48 67 47 75 48 13 22 38 48 67 47 75 48 61 3 37 67 57 52 99 38 48 112 59 8 36 75 48 13 22 38 48 67 47 52 99 38 48 13 22 38 48 67 47 52 99 38 48 13 22 38 48 67 47 75 48 67 47 75

Agent 2 Env 7 58 38 48 13 22 95 60 48 67 73 99 11 68 38 48 13 22 95 60 48 112 59 8 36 75 48 67 47 25 104 13 22 115 38 48 67 97 71 71 14 7 32 3 48 61 3 48 67 47 25 60 48 67 97 14 7 32 3 48 61 3 48 13 22 38 48 67 73 99 38 48 13 22 38 53 21 31 38 48 112 59 8 36 75 48 13 22 17 57 25 60 48 13 22 38 48 67 47 52 99 38 48 61

Agent 2 Env 8 58 38 48 61 3 48 67 73 99 11 68 38 48 13 22 95 60 48 61 3 48 67 57 55 7 32 3 48 112 59 8 36 75 48 13 22 95 60 48 61 3 48 67 73 99 11 68 38 48 13 22 95 60 53 21 31 38 48 13 22 95 60 53 21 31 38 48 13 22 95 60 53 21 31 38 48 13 22 95 60 53 21 31 38 48 13 22 95 60 53 21 31 38 48 13 22 95 60 53 21 31 38 48 13 22 95 60 53 21 31 38 48 13 22 95 60 53 21 31 38 48 13 22 95 60 53 21 31 38 48 13 22 95 60 53 21 31 38 48 13 22 95 60 53 21 31 38 48 13 22 95 60 53 21 31 38 48 13 22 95 60 53 21 31 38 48 13 22 95 60 53 21 31 38 48 13 22 95 60 53 21 31 38 48 13 22 95 60 53 21 31 38 48 13 22 95

Agent 2 Env 9 58 38 48 112 59 8 36 75 48 112 59 8 36 75 48 13 22 38 48 61 3 48 67 47 52 99 38 48 112 59 8 36 75 48 13 22 95 60 48 61 3 48 13 22 95 60 48 67 47 75 53 104 67 47 75 48 67 47 55 7 32 3 48 67 97 14 7 32 3 48 61 3 48 67 47 75 48 67 47 55 7 32 3 48 67 47 55 7 32 3 48 13 22 38 48 112 59 8 36 75 48 13 22 38 48 13 22 95 60 48 61 3 48 67 47 25 60

Agent 2 Env 10 58 38 48 112 59 8 36 75 48 13 22 38 48 67 97 14 7 32 3 48 61 3 48 67 97 75 48 13 22 38 48 61 3 48 67 97 14 7 32 3 48 61 3 48 67 47 75 48 67 47 75 48 61 3 48 67 47 52 99 38 48 13 22 38 48 67 47 25 60 48 61 3 48 67 47 55 7 32

Figure 9.5 Transitions of agent 2 on the 10 environments when running on the graph in Figure 9.7

Follow the third agent transitions in the 10 environments with the node id:

Agent 3 Env 1 58 95 104 13 22 38 48 67 47 25 60 48 61 3 48 67 47 25 60 48 67 47 25 60 48 13 22 38 48 67 97 14 7 32 3 48 61 3 37 67 47 75 48 67 47 75 48 67 47 52 99 38 48 13 22 38 48 13 22 95 60 48 67 47 55 7 32 3 48 67 47 55 7 32 3 48 112 59 8 36 75 48 13 22 17 57 52 99 38 48 13 22 17 57 52 99 73 99 38 48 61 3 48 67 47 52 99 38 48 13 22 17 47 52 99

Agent 3 Env 2 58 95 60 48 67 47 75 48 67 47 75 48 67 47 75 48 67 47 75 48 13 22 38 48 67 47 55 7 32 3 48 67 57 55 7 32 3 48 67 73 99 38 48 13 22 95 60 48 13 22 115 38 48 67 73 99 38 48 67 97 14 7 32 3 48 61 3 48 67 97 14 7 32 3 48 61 3 48 67 47 75 48 67 47 75 48

Agent 3 Env 3 58 95 104 13 22 38 48 67 47 55 7 32 3 48 61 3 48 13 22 38 48 67 97 14 7 32 3 48 61 3 48 67 47 75 48 67 47 75 48 67 47 55 7 32 3 48 61 3 48 67 47 75 48 13 22 65 57 52 99 67 73 99 38 48 112 59 8 36 75 48 67 57 25 60 48 67 73 99 73 99 38 48 112 101 13 22 38 48 13 22 38 48 67 73 99 38 48 112 59 8 36 75 48 13 22 17 47 52 99 38 48 13 22 95 60 48 67 57 52 99 11 68 38 48 13 22 95 60 48 67 57 52 99 11 68 38 48 13 22 95 60 48 67 57 52 99 11 68 38 48 13 22 95 60 48 67 57 52 99 11 68 38 48 13 22 95 60 48 67 47 25 60 48 67 57 25 60 48 67 73 99 38 48 13 22 38 48 13 22 38 48 67 47 52 99 38 48 13 22 38 48 13 22 95 60 48 67 47 55 7 32 3 48 61 3

Agent 3 Env 4 58 95 60 48 67 47 75 48 67 47 75 48 67 47 75 48 67 47 75 48 13 22 38 48 67 47 55 7 32 3 48 67 57 55 7 32 3 48 67 73 99 11 68 38 48 13 22 95 104 67 73 99 11 68 38 48 112 59 68 38 48 112 59 8 36 75 48 13 22 38 48 67 57 52 99 38 48 112 59 8 36 75 48 67 73 99 38 48 67 97 14 7 32 3 48 13 22 38 53 21 31 38 48 112 59 8 36 75 48 13 22 38 48 67 57 52 99 38 48 112 59 8 36 75 48 13 22 38 48 61 3 48 67 57 52 99 38 48 112 59 8 36 75 48 13 22 38 48 67 47 75 48 67 47 75 48 67 47 55 7

Agent 3 Env 5 58 95 60 48 67 47 75 48 13 22 38 48 67 47 55 7 32 3 48 67 47 75 48 61 3 48 67 47 25 60 48 67 57 25 60 48 61 3 48 67 73 99 38 48 13 22 38 53 21 31 17 97 14 7 32 3 48 13 22 38 53 103 57 52 99 38 48 112 59 8 36 75 48 13 22 38 48 67 47 52 99 38 48 13 22 95 60 48 67 47 52 99 11 68

Agent 3 Env 6 58 95 60 48 67 47 55 7 32 3 48 67 57 55 7 32 3 48 67 57 55 7 32 3 48 13 22 38 48 67 97 14 7 32 3 48 61 3 48 13 22 38 48 67 97 14 7 32 3 48 61 3 37 67 97 14 7 32 3 48 61 3 48 67 47 52 99 38 48 13 22 38 48 13 22 38 48 67 97 75 48 61 3 48 67 73 99 38 48 67 57 52 99 38 48

Agent 3 Env 7 58 95 60 48 67 47 75 48 67 47 75 48 67 47 75 48 13 22 95 60 48 67 47 55 7 32 3 48 67 97 14 7 32 3 48 67 97 14 7 32 3 48 67 97 14 7 32 3 48 67 47 75 48 67 47 75 48 67 47 52 99 38 48 67 47 75 48 67 47 75 53 104 67 47 75 53 104 67 47 75 48 67 47 75 48 67 47 52 99 38 48 13 22 95 60 48 13 22 38 48 112 59 8

Agent 3 Env 8 58 95 60 48 67 47 52 99 11 68 38 48 13 22 95 60 48 67 47 52 99 11 68 38 48 13 22 95 60 48 13 22 38 48 67 47 25 60

Agent 3 Env 9 58 95 60 48 67 47 25 60 48 67 47 25 60 48 13 22 38 48 61 3 48 67 57 52 99 38 48 112 59 8 36 75 48 13 22 38 48 67 47 25 60 48 67 47 25 60 48 67 47 25 60 48 67 47 25 60 48 67 47 52 99 38 48 13 22 38 53 103 47 52 99 38 48 112 59 8 36 75 48 13 22 95 60 48 67 97 14 7 32 3 48 61 3 48 67 47 25 60 48 67 47 25 60 48 67 47 25 60 48 67 47 25 60 48 13 22 95 60 48 67 47 52 99 11 68 17 57 25 104 13 22 95 60 48 67 47 52 99 11 68 38 48 13 22 95 60 48 67 47 52 99 11 68 38 48 13 22 95 60 48 67 47 52 99 11 68 17 57 25 104 13 22 95 60 48 67 47 52 99 11 68 38 48 13 22 95 60 48 67 47 52 99 11 68 38 48 13 22 95 60 48 67 47 52 99 11 68 17 57 25 104 13 22 95 60 48 67 47 52 99 11 68 17 57 25 104 13 22 95 60 48 67 47 52 99 11 68 17 57 25 104 13 22 95 60 48

Agent 3 Env 10 58 95 60 48 13 22 38 48 67 97 14 7 32 3 48 61 3 48 13 22 95 60 48 67 47 75 48 67 47 75 48 67 47 75 48 67 47 75 48 67 97 14 7 32 3 48 61 3 48 67 47 75 48 67 47 75 48 61 3 48 67 47 52 99 38 48 112 59 8 36 75 48 13 22 95 60 48 61

Figure 9.6 Transitions of agent 3 on the 10 environments when running on the graph in Figure 9.7

	Node Type
0	Start Node
1	Judgment Node
2	Processing Node

Table 9.3 Node types

	Judgment Node Functions	*Judgment Node Connections*
1	Judge Forward (JF)	Floor = 0, Obstacle = 1, Tile = 2, Agent = 3, Hole = 4
2	Judge Backword (JB)	Floor = 0, Obstacle = 1, Tile = 2, Agent = 3, Hole = 4
3	Judge Left (JL)	Floor = 0, Obstacle = 1, Tile = 2, Agent = 3, Hole = 4
4	Judge Right (JR)	Floor = 0, Obstacle = 1, Tile = 2, Agent = 3, Hole = 4
5	Direction to the nearest Tile (DT)	Forward = 0, Left = 1, Right = 2, backword = 3, nothing = 4
6	Direction to the nearest Hole (DH)	Forward = 0, Left = 1, Right = 2, backword = 3, nothing = 4
7	Direction to the nearest Hole from the nearest Tile (TH)	Forward = 0, Left = 1, Right = 2, backword = 3, nothing = 4
8	Direction to the second nearest tile (DST)	Forward = 0, Left = 1, Right = 2, backword = 3, nothing = 4

Table 9.4 Judgment Node Functions

	Processing Node Functions	*Processing Node Connections*
1	Move Forward (MF)	Without trapping =0, with trapping = 1
2	Turn Right (TR)	One connection to the Next node
3	Turn Left (TL)	One connection to the Next node
4	Stay (ST)	One connection to the Next node

Table 9.5 Processing Node Functions

The properties for each node in the tested chromosome (note: the subnode with the maximum Q-value is shown):

id 0	NT 0	id 10	NT 1	id 20	NT 1	id 30	NT 1
Fun	0	Fun	1	Fun	3	Fun	6
58		93 27 108 101 21		38 95 65 17 115		47 97 57 73 91	
id 1	NT 1	id 11	NT 1	id 21	NT 2	id 31	NT 1
Fun	3	Fun	5	Fun	3	Fun	3
38 95 65 17 115		93 27 108 101 21		31		38 95 65 17 115	
id 2	NT 1	id 12	NT 2	id 22	NT 1	id 32	NT 2
Fun	5	Fun	4	Fun	5	Fun	3
67 61 13 112 69		36		38 95 65 17 115		3	
id 3	NT 2	id 13	NT 2	id 23	NT 1	id 33	NT 2
Fun	1	Fun	2	Fun	6	Fun	1
48 37		22		75 14 79 71 57		32 29	
id 4	NT 1	id 14	NT 2	id 24	NT 2	id 34	NT 1
Fun	5	Fun	2	Fun	3	Fun	3
67 61 13 112 69		7		3		38 95 65 17 115	
id 5	NT 1	id 15	NT 2	id 25	NT 1	id 35	NT 2
Fun	3	Fun	3	Fun	5	Fun	3
38 95 65 17 115		3		60 113 104 51 74		34	
id 6	NT 2	id 16	NT 1	id 26	NT 1	id 36	NT 2
Fun	4	Fun	3	Fun	2	Fun	2
75		38 95 65 17 115		59 58 80 101 83		75	
id 7	NT 2	id 17	NT 1	id 27	NT 1	id 37	NT 1
Fun	1	Fun	6	Fun	2	Fun	5
32 29		47 97 57 73 91		59 58 80 101 83		67 46 68 8 28	
id 8	NT 2	id 18	NT 1	id 28	NT 1	id 38	NT 2
Fun	4	Fun	5	Fun	3	Fun	1
36		60 113 104 51 74		38 95 65 17 115		48 53	
id 9	NT 2	id 19	NT 2	id 29	NT 1	id 39	NT 1
Fun	3	Fun	3	Fun	5	Fun	6
3		31		67 61 13 112 69		47 97 57 73 91	

id 40	NT 1	id 50	NT 1	id 60	NT 2	id 70	NT 1
Fun	2	Fun	3	Fun	1	Fun	5
59 58 80 101 83		38 95 65 17 115		48 53		60 113 104 51 74	
id 41	NT 2	id 51	NT 2	id 61	NT 2	id 71	NT 1
Fun	3	Fun	1	Fun	3	Fun	6
91		32 29		3		75 14 75 71 57	
id 42	NT 2	id 52	NT 2	id 62	NT 2	id 72	NT 1
Fun	4	Fun	3	Fun	3	Fun	6
75		99		3		87 23 50 57 94	
id 43	NT 2	id 53	NT 1	id 63	NT 1	id 73	NT 2
Fun	4	Fun	5	Fun	3	Fun	3
75		104 84 103 21 32		38 95 65 17 115		99	
id 44	NT 2	id 54	NT 1	id 64	NT 1	id 74	NT 1
Fun	2	Fun	6	Fun	2	Fun	6
7		47 97 57 73 91		59 58 80 101 83		87 23 50 57 94	
id 45	NT 1	id 55	NT 2	id 65	NT 1	id 75	NT 2
Fun	1	Fun	2	Fun	6	Fun	1
38 11 67 73 92		7		47 97 57 73 91		48 53	
id 46	NT 1	id 56	NT 1	id 66	NT 2	id 76	NT 2
Fun	3	Fun	6	Fun	3	Fun	3
38 95 65 17 115		47 97 57 73 91		31		3	
id 47	NT 1	id 57	NT 1	id 67	NT 1	id 77	NT 2
Fun	7	Fun	7	Fun	6	Fun	1
75 55 52 25 16		75 55 52 25 16		47 97 57 73 91		32 29	
id 48	NT 1	id 58	NT 1	id 68	NT 1	id 78	NT 1
Fun	5	Fun	3	Fun	3	Fun	5
67 61 13 112 69		38 95 65 17 115		38 95 65 17 115		77 98 60 70 50	
id 49	NT 2	id 59	NT 1	id 69	NT 1	id 79	NT 2
Fun	2	Fun	5	Fun	1	Fun	3
22		67 46 68 8 28		38 11 67 73 92		99	

id 80	NT 1	id 90	NT 1	id 100	NT 1	id 110	NT 1
Fun	5	Fun	3	Fun	6	Fun	6
16 52 2 49 89		38 95 65 17 115		47 97 57 73 91		87 23 50 57 94	
id 81	NT 2	id 91	NT 1	id 101	NT 1	id 111	NT 1
Fun	1	Fun	6	Fun	5	Fun	7
32 29		47 97 57 73 91		67 61 13 112 69		75 55 52 25 16	
id 82	NT 2	id 92	NT 1	id 102	NT 1	id 112	NT 1
Fun	3	Fun	2	Fun	6	Fun	2
20		56 58 80 101 83		47 97 57 73 91		59 58 80 101 83	
id 83	NT 2	id 93	NT 1	id 103	NT 1	id 113	NT 1
Fun	2	Fun	4	Fun	6	Fun	3
7		46 79 87 89 118		47 97 57 73 91		38 95 65 17 115	
id 84	NT 1	id 94	NT 1	id 104	NT 1	id 114	NT 1
Fun	1	Fun	2	Fun	5	Fun	1
38 11 67 73 92		59 58 80 101 83		67 61 13 112 69		38 11 67 73 92	
id 85	NT 2	id 95	NT 1	id 105	NT 1	id 115	NT 1
Fun	2	Fun	5	Fun	3	Fun	1
7		60 113 104 51 74		38 95 65 17 115		38 11 67 73 92	
id 86	NT 1	id 96	NT 1	id 106	NT 1	id 116	NT 1
Fun	5	Fun	6	Fun	5	Fun	6
60 113 104 51 74		47 97 57 73 91		67 61 13 112 69		47 97 57 73 91	
id 87	NT 1	id 97	NT 1	id 107	NT 1	id 117	NT 1
Fun	5	Fun	6	Fun	3	Fun	6
67 46 68 8 28		75 14 79 71 57		38 95 65 17 115		47 97 57 73 91	
id 88	NT 2	id 98	NT 1	id 108	NT 1	id 118	NT 1
Fun	4	Fun	5	Fun	5	Fun	6
36		67 61 13 112 69		67 46 68 8 28		47 97 57 73 91	
id 89	NT 2	id 99	NT 1	id 109	NT 1	id 119	NT 1
Fun	3	Fun	1	Fun	5	Fun	5
31		38 11 67 73 92		67 61 13 112 69		67 61 13 112 69	

Note:

The chosen sub node is the sub node with the maximum Q-value.

In the chromosome representation here only the maximum sub node is showed.

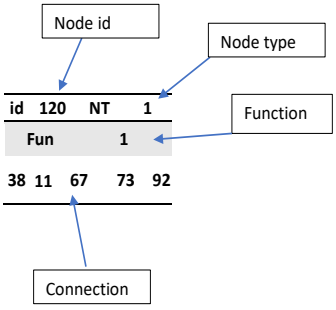


Figure 9.7 A chromosome representation (with the best sub node) for one of the best chromosomes

Figure 9.4, Figure 9.5, and Figure 9.6 show the transitions for each agent in the chromosome that represented in Figure 9.7

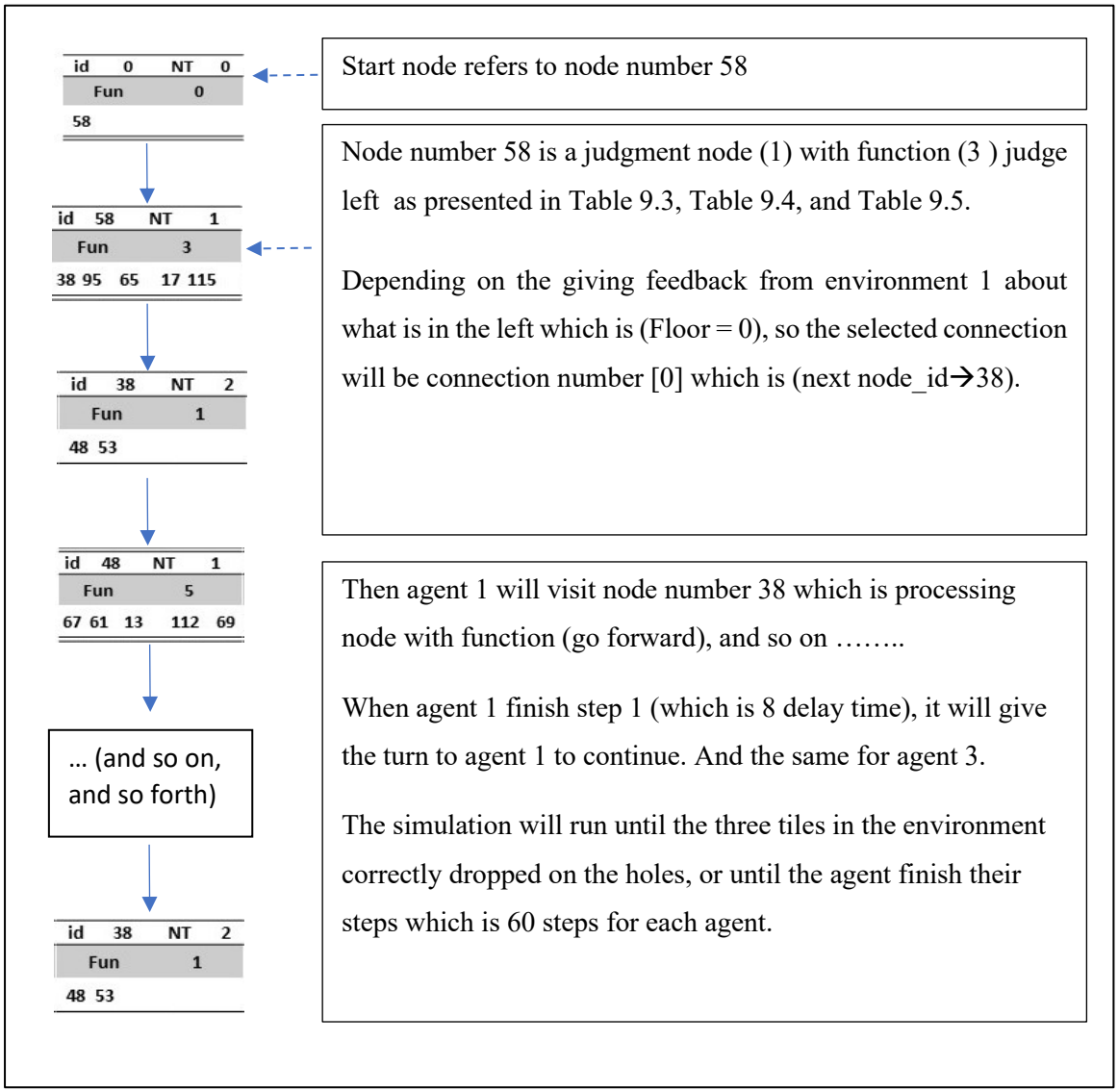


Figure 9.8 Example for following the agent's transitions

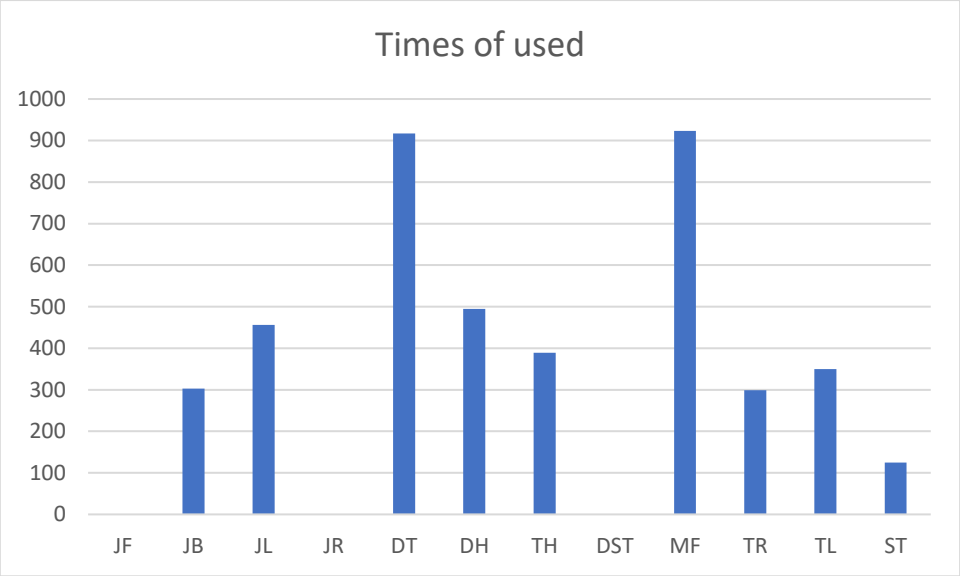


Figure 9.9 Times of using for each function type when testing a chromosome from Architecture #4 on the testing set (1) with the result (26/30)

From Figure 9.9, it has been noticed that the agents have not used (JF, JR, DST). (Judgment Forward, Judgment Right, and the direction to the second nearest tile). It seems that the (judgment Backward, and Judgment Left) are enough for the agent to give decisions. And (direction to the second nearest tile) is not important, the agents used the (direction to the nearest tile more than the other judgment nodes). (The direction to the nearest tile and Move forward) are the most using nodes in the chromosome.

9.4 Conclusion

In conclusion, through developed techniques on the GNP-RL; Constraint Conformance, optimal search and tasks priorities, we have achieved the highest results to date on the tile world problem. The previous best result was in December 2018 (19/30 = 63.33% [11]). After using the techniques presented in this study, we were able to achieve 100% training accuracy and 86.66% test accuracy in Testing Environment (1) [2] and 96.66% test accuracy in Testing Environment (2, proposed environment). These results were implemented on the complex benchmark which is tile world problem. The difficulty of this problem is that the training set is sparse and the agents work together to put all the tiles in the holes within the shortest possible time. The agents cannot pull the tiles and all they can do is push or turn right or left. This makes the agent sometimes push the tiles to a place where it cannot be pushed again anymore to anywhere, which makes it lose the tiles. We implemented the restriction of constraints GNP-RL technique to train the agent on how to prevent pushing the tile into a trapped location. We used the optimal search algorithm inside GNP-RL to give an accurate direction for the objects in the grid world. We added the task prioritization to GNP-RL in order to influence the way agents work to solve the problem more efficiently (e.g. prevent any collisions or waste time and efforts). The proposed training strategies were employed on two experiments for each variant of the GNP algorithm; each one on a different type of training set. Training is stopped based on the performance of the algorithm on the validation/test set, in order to avoid overfitting (as we have observed). When we combined all the three proposed techniques and the training strategies on the GNP-RL, we achieved the best results.

Generalised strategy for solving real world problems

Here we summarize the main points that need to be considered when applying the proposed three techniques on real world problems with multi-agents and dynamic environment.

To apply the three proposed mechanisms, the following general approach applies:

Constraint conformance:

Every problem imposes some conditions or limitations on the agent's behavior. For example, agent actions should: prevent collisions between agents, avoid some hazardous paths or walls, or avoid putting objects on unwanted places, etc. To use the constraint conformance mechanism on any real-world problem, firstly, identify all conflicting situations in the problem domain and incorporate the appropriate functions (into the library of functions for which GNP-RL selects from) for detecting those conflicting situations. Secondly, train the agents how to

avoid all the unwanted cases by punishing them each time their actions produce one of those unwanted cases. The punishment is imposed by way of penalties when updating the Q-values in RL.

Task prioritization:

In order to influence the agents to perform task prioritization, motivational/promotional rewards should be given to the agents each time they finish a task. Each task should be given a specific reward that is proportional to its importance/priority. The greater the importance is, the greater the reward must be. By using this technique, the agents will try to perform the most importance task first, to get more rewards, before solving tasks of lesser importance.

Optimal Search:

Using the optimal search component alone without the help of the other proposed mechanisms will not solve the problem itself. That means, to solve the problem, the GNP-RL needs to derive a meta-level reasoning strategy that combines selectively together a number of functions from the given library of functions, specific to the problem domain. When asking about the shortest path using one of the optimal search algorithms like A*, A* in this proposed framework will not tell the agent a complete path (where to go). Instead, the GNP-RL will detect the next action for the agent by considering many questions that are answered by A*, such as (where is the nearest Tile, where is the nearest hole, where is the nearest hole to the nearest tile, and where is the second nearest tile). So, A* can answer these questions by finding the shortest paths to each of those queries and gives the direction to the first waypoint along that path. It is important to note that A* does not give the ultimate next action for the agent. GNP cleverly combines the answers to those queries using multiple judgement nodes. Consequently, combining the optimal search together with the constraint conformance and task prioritization components with the GNP-RL ultimately achieves a meta-level reasoning strategy. In our proposed framework, A* can be found at the lower level of decision-making, while GNP-RL is at the top of the hierarchy of decision-making.

Other real-world examples to demonstrate the proposed strategies

Robot Football:

One of the real-world problems that could benefit from the proposed algorithm is the Robot Football. It involves two opposing teams, with each group having attacking/defending players and a goalkeeper. To apply the three techniques on this problem, firstly: impose constraint conformance by punishing the agents when unwanted actions occur (e.g. pushing the ball to a player from the other group, pushing the ball outside the playing field, kicking the ball into their own team goal, touching the ball with the hand, and so on…). Secondly: apply the optimal search by finding the shortest paths to multiple source and target destinations (path to the ball, to the nearest player from the same team, to the nearest player with the ball, and so on…). Thirdly: apply the task prioritization by giving the highest reward (10) when the player scores a goal. Another reward when the player kicks the ball to another player that scores a goal (this time the reward is lower than the first one by one point (9)). Moreover, a reward for the player that prevents the ball from entering their own goal post (8) … (etc.).

Automated Storage System (Multi-Agent):

Business owners are working on automating storage systems to save money and time. The difficulty of this problem is how can the agent store the object in the right place with the perfect size without wasting any free space between the objects. Moreover, it should reuse the same space once it becomes free. To utilize the proposed method on this problem, firstly: implement the constraint conformance by punishing the agents when it makes unwanted decisions (e.g. putting a small object in a big place, trying to put a big object in a smaller destination space, putting an object in a wrong place, colliding with another agent, loosing the object, … and so on). Secondly: using the optimal search to compute the shortest paths to appropriate target destinations (path to the nearest object that needs to be moved, to the nearest fitting destination for the object, and so on…). Thirdly: implementing task prioritization by giving the best reward when the agent puts the nearest object to the nearest fitting destination that is free (10), a lower reward when the agent puts the nearest object to the nearest destination that is a little bigger (9), and so on and so forth.

9.5 Future work

For the future work, the Task prioritisation & Optimal Search & constraint conformance GNP-RL algorithm could be applied to another benchmark. Also, A* technique could be

exchanged with other optimal search algorithms such as (LPA*, D*, …) to save more time. The effectiveness of the Elitism and Gene Diversification could be checked by implementing it on another problem. The reward and penalization technique could be used with another variant of GNP-RL to prove its benefits.

References

[1] X. Li, M. Yang and S. Wu, “Niching genetic network programming with rule accumulation for decision making: An evolutionary rule-based approach,” *Expert Systems With Applications,* vol. 114, p. 374–387, 2018.

[2] S. Mabu, K. Hirasawa, M. Obayashi and T. Kuremoto, “A variable size mechanism of distributed graph programs and its performance evaluation in agent control problems,” *Expert Systems with Applications,* vol. 41, p. 1663–1671, 2014.

[3] J. H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor: MIT Press, 1975.

[4] J. R. Koza, Genetic Programming On the Programming of Computers by Means of Natural Selection, Cambridge: MIT Press, 1992.

[5] J. R. Koza, Genetic Programming On the Programming of Computers by Means of Natural Selection, Cambridge: MIT Press, 1994.

[6] V. Ciesielski, D. Mawhinney and P. Wilson, Genetic Programming for Robot Soccer, Berlin, Heidelberg: Springer, 2002.

[7] B. Li, S. Mabu and K. Hirasawa, “Genetic Network Programming with Automatic Program Generation for Agent Control,” *Journal of Evolutionary Computational Society,* vol. 1, no. 1, pp. 43 - 53, 2010.

[8] H. KATAGIRI, K. HIRASAWA and J. HU, “Network Structure Oriented Evolutionary Model: Genetic Network Programming,” *Trans.Of the Society Of Instrument and Control Engineers,* vol. 38, no. 5, p. 485/494, 2002.

[9] S. Mabu, K. Hirasawa and J. Hu, “A Graph-Based Evolutionary Algorithm: Genetic Network Programming (GNP) and Its Extension Using Reinforcement Learning,” *Evolutionary Computation,* vol. 15(3), pp. 369-398, 2007.

[10] S. Mabu , H. Hatakeyama, M. Thu Thu, K. Hirasawa and J. Hu, “Genetic Network Programming with Reinforcement Learning and Its Application to Making Mobile Robot Behaviour,” *IEEJ Transactions on Electronics Information and Systems ,* vol. 126, no. (8), pp. 1009-1015, 2006.

[11] Y. Yang, Z. He, S. Mabu and K. Hirasawa, “A Cooperative Coevolutionary Stock Trading Model Using Genetic Network Programming-Sarsa,” *Journal of Advanced Computational Intelligence and Intelligent Informatics,* vol. 16, no. 5, pp. 581 - 590, 2012.

[12] S. Mabu, S. Gotoh and T. Obayashi, “A Structural Optimal Method of Genetic Network Programming for Enhamcing Generalization Ability,” *International Journal of Engineering Innovation and Management,* vol. 6, no. 2, pp. 13-18, 2016.

[13] S. Mabu, K. Hirasawa and J. Hu, “Genetic Network Programming with Reinforcement Learning and Its Performance Evaluation,” in *enetic and Evolutionary Computation – GECCO 2004. GECCO*, Springer, Berlin, Heidelberg, 2004.

[14] M. E. Pollack and M. Ringuette, “Introducing the Tileworld: Experimentally Evaluating Agent Architectures,” *AAAI,* p. 183–189, 1990.

[15] D. Whitley, “A Genetic Algorithm Tutorial,” *Statistics and Computing ,* vol. 4, pp. 65-85, 1994.

[16] H. Kargupta, J. R. Gattiker and K. Buescher, “Credit Card Fraud Detection: An Application Of The Gene Expression Messy Genetic Algorithm,” *Knowledge Discovery & Data Minimg,* 1996.

[17] W. Min and Y. Shuyuan, “A hybrid genetic algorithm-based edge detection method for SAR image,” *IEEE International Radar Conference,* 2005.

[18] Y. Sun, C. F. Babbs and E. Delp, “A Comparison of Feature Selection Methods for the Detection of Breast Cancers in Mammograms: Adaptive Sequential Floating Search vs. Genetic Algorithm,” *IEEE Engineering in Medicine and Biology 27th Annual Conference Engineering in Medicine and Biology Society,* 2005.

[19] A. Nelsona, G. J. Barlow and L. Doitsidis, “Fitness functions in evolutionary robotics: A survey and analysis,” *Robotics and Autonomous Systems,* vol. 57, p. 345–370, 2009.

[20] S. S. Juneja , P. Saraswat , K. Singh, J. Sharma, R. Majumdar and S. Chowdhary, “Travelling Salesman Problem Optimization Using Genetic Algorithm,” in *Amity International Conference on Artificial Intelligence (AICAI)*, Dubai, United Arab Emirates, United Arab Emirates, 2019.

[21] H. Mühlenbein and D. Schlierkamp-Voosen, “Predictive Models for the Breeder Genetic Algorithm I. Continuous Parameter Optimization,” *Evolutionary Computation,* vol. 1, no. 1, pp. 25-49, 1993.

[22] T. Blickle and L. Thiele, “A Comparison of Selection Schemes used in Evolutionary Algorithms,” Computer Engineering and Communication Networks Lab, Switzerland, 1996.

[23] J. D. SCHAFFER, “An adaptive crossover distribution mechanism for genetic algorithms,” in *2nd Int'1 Conf. on Genetic Algorithms (ICGA'87)*, 1987.

[24] D. Fogel and J. Atmar, “Comparing genetic operators with gaussian mutations in simulated evolutionary processes using linear systems,” *Biological,* vol. 63, p. 111–114, 1990.

[25] P. LARRANAGA, C. KUIJPERS, R. MURGA, I. INZA and S. DIZDAREVIC, “Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators,” *Kluwer Academic Publishers,* vol. 13, p. 129–170, 1999.

[26] K. Deep and M. Thakur, “A new mutation operator for real coded genetic algorithms,” *Applied Mathematics and Computation,* vol. 193, p. 211–230, 2007.

[27] J.-Y. Potvina, P. Sorianoa and M. Vale, “Generating trading rules on the stock markets with genetic programming,” *Computers & Operations Research,* vol. 31, p. 1033 – 1047, 2004.

[28] M. L. Raymer, W. F. Punch, E. D. Goodman and L. A. Ku, “Genetic programming for improved data miningApplication to the biochemistry of protein interactions,” *Proc. 1st Annu. Conf. Genetic Program,* pp. 375-380, 1996.

[29] H. Guo and A. K. Nandi, “Breast cancer diagnosis using genetic programming generated feature,” *Pattern Recog,* vol. 5, no. 980-987, p. 39, 2006.

[30] C. Estbanez, J. M. Valls and R. Aler, “GPPE: A method to generate ad-hoc feature extractors for prediction in financial domains,” *Appl. Intell,* vol. 29, no. 3, pp. 174-185, 2008.

[31] M. T. Ahvanooey, Q. Li, M. Wu and S. Wang, “A Survey of Genetic Programming and Its Applications,” *KSII TRANSACTIONS ON INTERNET AND INFORMATION SYSTEMS,* vol. 13, no. 4, pp. 1765-1794, Apr. 2019.

[32] P. G. Espejo, S. Ventura and F. Herrera, “A Survey on the Application of Genetic Programming to Classification,” *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART C: APPLICATIONS AND REVIEWS,* vol. 40, no. 2, pp. 121-144, 2010.

[33] H. Katagiri, K. Hirasawa and J. Hu, “Genetic Network Programming - Application to Intelligent Agents -,” *IEEE,* pp. 3829-3834, 2000.

[34] K. Hirasawa, M. Okubo, H. Katagiri, J. Hu and J. Murata, “Comparison between Genetic Network Programming(GNP) and Genetic Programming(GP),” *IEEE,* pp. 1276-1282, 2001.

[35] S. Mabu, K. Hirasawa, J. Hu and J. Murata, “Online Learning of Genetic Network Programming (GNP),” *IEEE,* pp. 321-326, 2002.

[36] S. Mabu, K. Hirasawa, J. Hu and J. Murata, “Online Learning of Genetic Network Programming and its Application to Prisoner's Dilemma Game,” *IEEJ Transactions on Electronics Information and Systems,* vol. 123, no. 3, pp. 535-543, 2003.

[37] S. MABU, Y. CHEN, S. ETO, K. SHIMADA and K. HIRASAWA, “Genetic Network Programming with Intron-like Nodes,” *Electronics and Communications in Japan,* vol. 93, no. 8, p. 1312–1319, 2010.

[38] S. SENDARI, “Waseda University Doctoral Dissertation,” in *Study on Robustness and Adaptability of Genetic Network Programming with Reinforcement Learning for Mobile Robot*, Japan, Graduate School of Information, Production and Systems Waseda University, 2013, p. 87.

[39] T. Murata and T. Nakamura, “Multi-Start Node Genetic Network Programming for Controlling Multiple Agents,” *IEEE International Conference on,* pp. 1-4244-0100-3, 2006.

[40] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, Cambridge, Massachusetts, London, England: MIT Press, 1998.

[41] J. L. Shapiro, “Diversity Loss in General Estimation of Distribution Algorithms,” *Parallel Problem Solving from Nature - PPSN IX,* vol. 10, no. 1, p. 92–101, 2006.

[42] X. Li, S. Mabo and K. Hirasawa, “Towards the maintenance of population diversity: A hybrid probabilistic model building genetic network programming,” *Trans. Japanese Soc. Evol. Comput.,* vol. 1, no. 1, pp. 89 - 101, 2010.

[43] P. h. Winston, Artificial Intelligence Third Edition, USA: Addison-wesley Publishing Company , 1993.

[44] H. Anton and C. Rorres, Elementary Linear Algebra (11th ed.), United States of America: Wiley, 2014.

[45] Michalewicz and Zbigniew, Genetic Algorithms + Data Structures = Evolution Programs, Berlin Heidelberg: Springer-Verlag, 1996.

[46] C. J. C. H. Watkins, Learning from delayed rewards, England: PhD Thesis, University of Cambridge, 1989.

[47] R. Sutton, Temporal credit assignment in reinforcement learning, Amherst, MA: PhD Thesis, University of Massachusetts, 1984.

[48] N. J. Nilsson, Problem-Solving Methods in Artificial Intelligence, New York: McGraw-Hill, 1971.

[49] P. E. Hart, N. J. Nilsson and B. Raphael, “A Formal Basic for the Heuristic Determination of Minimum Cost Paths,” *IEEE Transactions of Systems Science and Cybernetics,* vol. 4, no. 2, pp. 100 - 107, 1968.