Waseda University Doctoral Dissertation

A Study on the Memory Schemes for Genetic Network Programming

YE, Fengming

Graduate School of Information, Production and Systems

Waseda University

June, 2011

I would like to dedicate this thesis to my loving parents for their tremendous encouragement and steady support. Without them something and somebody would not be going well.

Especially, I am gratified by everything that have been done for me by Yuan, the true love throughout my lifetime, with the deepest feelings. You have settled softly and firmly in my soul, like the glory light removing the veiling of darkness.

Most sincerely, I am grateful to my dear grandparents who once presented me profuse loving memory that I am still cherishing and would never forget.

Finally, this thesis is dedicated to all those who believe in the richness of learning and knowledge.

Abstract

From 1960s, Evolutionary Algorithms (EA), which is an important subtopic of Artificial Intelligence (AI), has been studied a lot and great progresses have been made continuously to improve the existed algorithms or propose novel methods. For example, the studies on many classical methods such as Genetic Algorithm (GA), Genetic Programming (GP), Evolutionary Strategies (ES), etc. have made significant contribution to the research of EA.

In the past decade, a new evolutionary approach named Genetic Network Programming (GNP) was proposed and attracted more and more attention. GNP which is based on the idea of Genetic Algorithm, also can evolve itself and search in the solution domain of large scale and finally find the (approximate) optimal solutions. The unique character of GNP which make it very different from other methods of EA is the utilization of the data structure of directed graphs. Many research has demonstrated that GNP can deal with complex problems in the dynamical environments very efficiently and effectively due to its graph based structure. As a result, recently, GNP is being used in many different areas such as data mining, extracting trading rules of stock markets, elevator supervised control systems, etc. and GNP has obtained outstanding results in all the above fields. On the other hand, many research shows that classical EAs such as GA, usually fail to solve problems in dynamical environments. So, scholars devote themselves to the research on the enhancement of the architecture of EAs. For example, different memory schemes storing historical informations during evolution and reusing them later are designed for EAs to solve complex problems in dynamical environments.

So, the motivation of this research is designing memory schemes for GNP in order to improve its performance further in the dynamical environments. So, four different memory schemes are proposed: GNP with rules, GNP with reconstructed individuals, GNP with route nodes and adaptive mutation in SARSA learning of GNP. GNP with rules stores first-order information on GNP rules and uses them to generate new individuals. GNP with reconstructed individuals will stores the complete node transitions which can guide the agent with much more effectiveness and uses them to enhance the gene structures of the worst individuals. GNP with route nodes employs an indirect memory scheme which uses the stored information associated with current environments. The adaptive mutation using Q values to evaluate node branches adjusts the mutation rates and mutation directions for node branches and achieves the balance between exploration and exploitation. In order to measure the performance of the proposed architectures, the benchmark of tile-world was used as the simulation environments. The simulation results show some improvements brought by the memory schemes to conventional GNPs.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

In computer science, evolutionary computation whose essential concept comes of Darwin's Evolution Theory, is a subfield of artificial intelligence (more particularly computational intelligence). It uses iterative progress, such as growth or development in a population. In this population, a guided random search using parallel processing to achieve the desired end is executed. Such processes are often inspired by biological mechanisms of natural evolution.

As an effective way to solve optimization problems, evolutionary computation has been drawing attentions and endeavors for decades. A large number of studies on evolutionary computation techniques have been executed and many significant research achievements, such as Genetic Algorithm (GA) by J. Holland (1; 2), Genetic Programming (GP) by J. Koza(3; 4; 5; 6), Evolutionary Programming (EP) by L. Fogel(7; 8; 9) and Evolutionary Strategy (ES) by I. Rechenberg and H. Schwefel(10; 11; 12), have been obtained. The gene of GA is represented as a string structure which is mostly used to search the global optimal solution in a feasible searching space. Traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. Extended from GA, GP takes string as its encoding style in most cases, but pioneers in generating treelike programs instead of pure strings as the solutions to a given problem. The intensive optimization and search ability of GP appeals to researchers enormously, and multitude efforts has been dedicated towards its different aspects, such as solution representation(13; 14; 15), grammar(16; 17; 18), genetic

operations(19; 20; 21) and so on. The tree structure can be easily evaluated in a recursive way. Every node of the tree has an operator function and every terminal node has an operand, as a result, mathematical expressions are very easy to evolve and evaluate. For EP, Fogel used finite state machines as predictors and evolved them. The finite state machine is a model of behavior composed of a finite number of states, transitions between those states, and actions. ES is similar to GA while its genetic operators only include selection and mutation. The selection is only concerned about the fitness value rankings of the individuals (not the real fitness values). After mutation, only the offspring with high fitness rankings become the parents of next generation while the current parents are always disregarded.

In the last decade, a new graph-based evolutionary algorithm named Genetic Network Programming (GNP) is developed. It is devised to deal with problems in dynamic environments effectively and efficiently and it is found that GNP has some advantages over traditional evolutionary computation techniques especially in dynamical environments(22; 23; 24; 28). As the name suggests, GNP adopts directed graphs rather than trees as their phenotype, so, in some scholar communities, GNP is considered as a special variation of GP due to its graph based structure which brings at least two advantages: reusability of nodes, and implicit memory function. For one thing, GP has a tree structure which brings a problem that the size of tree is uncontrollable, if the problem complexity is unexpectedly high. However, in GNP, no restriction is imposed on the design of the node functions or genetic operations, while it is still immune to the bloat problem(18). For another, since the nodes in the graph might possibly be revisited during execution, GNP offers flexible transitions from node to node, and further benefits by creating potentially more sophisticated programs than the basic GP does, especially in the creation of sub-programs, loop and recurrence. That is to say, the reusability of nodes makes GNP's structure more compact than that of GP(26; 27). On the other hand, the node transition of GNP begins from a start node and transfers based on the judgments on the nodes and node connections, thus it can be said that, for example, agent's actions in the past are implicitly memorized in the network flow. The effectiveness of GNP has been demonstrated by previous research on various complex applications, such as stock market prediction(29), data mining (30), online auction(31), elevator control systems(32), and so forth. There is also fundamen-

tal research on the algorithm level, e.g., one furnishes GNP with the online learning ability (33; 34; 35; 36) as verified by the Tile-world(37) problem.

1.2 Related Works on EAs with Memory Schemes

Traditionally, the research on evolutionary algorithms (EAs), e.g., GA has focused on stationary optimization problems, like numerical optimization problems, whose environment conditions, design variables, fitness function, etc. remain fixed during the evolution process and the environment conditions are precisely given in advance. For these stationary problems, the aim of EAs is to quickly and precisely locate the optimal solution(s) in the search space of a large scale. However, the environments of many real-world problems are more dynamical and complicated, e.g., in financial markets, elevator group systems, etc. For these dynamical problems, the aim of EAs is to find successive behaviors for agents making judgments and taking proper actions for the current environment. So, the optimal solutions of these problems are no longer the locus in the solution space, but a series of action regulations for agents.

So, the traditional EAs such as GA usually fail to solve these dynamical problems. But, some researchers have introduced a kind of memory schemes, which stores historical informations on good solutions, and reuse them later, to enhance the performance of EAs in dynamical problems(38; 39; 40; 41).

The adoption of the memory schemes has proved to be able to enhance EA's performances in many applications, especially in dynamical environments, where the environments keep on changing during the evolution process. The basic principle of the memory schemes is to store the information, e.g. good solutions, from the current generation and reuse it in later generations. This useful information can be stored in two ways: by implicit memory mechanisms and by explicit memory mechanisms.

1.2.1 Implicit Memory Schemes

For the implicit memory schemes, EAs use genotype representations that contain redundant information. Here, the redundant representation stores good (partial) solutions to be reused later as memory. Typical examples of the implicit memory schemes are GAs based on diploidy or multiploidy representations. For example, Goldberg and

Smith first extended the simple haploid GA to a diploid GA with a tri-allelic dominance scheme(42). Thereafter, Ng and Wong developed a dominance scheme with four alleles for a diploidy-based GA(43). Lewis et al. further investigated an additive diploidy scheme, where a gene becomes 1 if the addition of all alleles exceeds a certain threshold, or 0 otherwise(44). All these different kinds of diploid GAs share the following characteristics:

- 1. The gene structures of GA individuals are represented by the diploid genotypes, each of which consists of two genotypic chromosomes.
- 2. The dominance scheme is used to map the two genotypic chromosomes to the haploid phenotype and the fitness of the individual is evaluated according to the haploid phenotype.
- 3. The genetic recombination, i.e., crossover and mutation affects on the two genotypic chromosomes of the diploid genotype.

Different approaches may organize different dominance schemes, i.e., different genotype-phenotype mapping mechanisms and the dominance schemes may be updated in different ways during the evolution.

In addition to multiploidy GAs, Dasgupta and McGregor proposed the structured GA which is a quite different implicit memory scheme(45). In structured GA, the individua has a multileveled structure. In this representation, high-level genes can regulate the activation of a set of low-level genes. The set of low-level genes can memorize good (partial) solutions at the current stage, which can be reactivated by high-level genes later.

In summary, the implicit schemes contain redundant information in the genotype of individuals. The redundant information may not affect the fitness evaluation, but it can be remained in the gene structures so as to maintain the population diversity. Meanwhile, in structured GA, genetic recombinations among different levels have different effects, which make the search range flexible and wide.

1.2.2 Explicit Memory Schemes

While the implicit memory schemes depend on the redundant representation, where the useful information is stored during the evolution, the explicit memory scheme makes

use of valuable knowledge in an extra storage space, where the useful information from the current generation can be explicitly stored and reused in later generations. For example, Louis and Xu used a memory to store the best individuals during a run for the open shop rescheduling problem(46). Whenever a change occurs, GA is restarted from a population with partial $(5\% - 10\%)$ individuals retrieved from the memory corresponding to the previous run, while the rest is initialized randomly. And instead of storing the best solutions only, the approach of storing good individuals and their corresponding environment information has been also proposed. For example, Ramsey and Grefenstette studied GA for a robot control problem, where good candidate solutions are stored in a permanent memory together with the information about the current robots environment(47). When reusing the good individuals, the similarity of the current environment to the recorded environment information is measured and the associated best individuals will be selected and reused. The memory scheme recording only the best solutions is called *directed memory scheme* and the one recording both solutions and their associated environment informations is called *indirected memory scheme* or *associative memory scheme*.

Usually, the memory size is fixed. So, when the memory is full with the past knowledge, the update mechanism should be designed to maintain the memory. A common way is to replace some solutions by the better ones. When the environment information is also stored, the similarity of the current environment to the recorded environment information will be measured. Then, the associated solutions with the highest similarity will be compared and the solution with higher fitness will replace the one with worse fitness.

1.3 Contents of this Research

1.3.1 Motivation and Objective

All the aforementioned research showed significant improvements when GA is equipped the memory schemes. So, it is expected that GNP will be enhanced if it also utilizes a well designed memory scheme. Originally, GNP is devised to solve dynamical problems effectively. Considering the profit of GA using the memory scheme, we think GNP will be also enhanced by employing the memory scheme in dynamical problems.

The main objective of the research is to develop a memory scheme to enhance the architecture of standard GNP and improve its performance when solving dynamical problems.

In GNP, the route of GNP transitions which consists of a series of successive GNP transitions from node to node corresponds to the agent's actions. However, it is generally found that not all of the GNP nodes and connections but only a part of them is included in the route and others are not used by agents at all. Thus, one of the points of the research is that we can extract and accumulate the information on the node and connection transitions that are carried out by agents from each individual and reuse the accumulated information later. Another point is that we can design a precise criterion to evaluate the merits of all the branches of nodes over the population and make use of the information to guide the evolution process.

1.3.2 Research Topics

This thesis includes four topics to be studied based on the aforementioned motivation and objective.

In Chapter 2, a memory scheme name GNP with Rules (GNP-R) is proposed. The memory scheme stores the informations on the node branches with their importance values which are evaluated according to the fitness values and reuse them to construct a fixed number of new individuals to replace the worst individuals in each generation. The construction of new individuals utilizes a certain probabilistic policy that the rules with higher importance values have higher probability to be used. The proposed method is evaluated in the tile-world problems where the agents have sensors with the limited sight range in a maze. The program should control the agents to conduct a series of actions to accomplish their missions. The tile-world problem is very difficult for agents. Firstly, the agents have only the sight of very limited range, so they cannot see the environment situations in advance. Secondly, during the task execution of the agent, the actions of itself and other agents will change the environment dynamically. As a result, it is an excellent benchmark problem for the agent control approaches.

In Chapter 3, another memory scheme named GNP with Reconstructed Individuals (GNP-RI) is studied. The memory stores the informations of the best solutions (instead of node branches in GNP-R) and reuse them to modify the gene structures of the worst

individuals. The best solutions containing the whole node transitions of agents imply the successive series of actions taken by the best agents. So, the worst individuals can learn experiences from their elite peers from the memory scheme. The proposed approach is also evaluated in tile-world problems and compared with standard GNP.

In Chapter 4, a more effective memory scheme name GNP with Route Nodes (GNP-RN) is designed for GNP. The memory also stores the informations of the best solutions which are reused by all the individuals during the evolution. Each individual has some extra route nodes which mark an reference to the memory. When the agent transfers to the route node, it will refer to the memory and select one solution to use. When using the best solutions, the current environment information is also considered by the agent and only the rules which satisfy the environment condition will be used by the agents. The architecture of GNP-RN is evaluated in different kinds of tile-world problems and compared with GNP-RI and standard GNP.

In Chapter 5, SARSA learning is used to evaluate the node branches (instead of importance values in GNP-R) and the memory stores a table containing the Q values of the node branches. And the adaptive mutation which dynamically configures the mutation rate and flexibly guides the mutation direction is applied depending on the Q table information storing the memory. The adaptive mutation in SARSA learning of GNP (GNP-SLAM) is also evaluated in different kinds of tile-world problems and compared with GNP-RI and standard GNP.

Chapter 2

GNP with Rules

2.1 Introduction

In artificial intelligence, an agent is used for intelligent actors which observe and act upon an environment. A rational agent is an entity that is capable of perception, action and goal directed behavior. And the node transition of GNP which begins from a start node and transfers based on the judgments on the nodes and connections, is just the behavior regulation to guide the agent's action upon the environment and it can be said that the agent's actions in the past are implicitly memorized in the network flow of GNP.

In other words, in GNP, the route of GNP transitions which consists of a series of successive GNP transitions from node to node corresponds to the agent's actions. However, it is generally found that not all of the GNP nodes and connections but only a part of them is included in the route and others are not used by agents at all. Thus, one of the points of the proposed method is that we can extract and accumulate the information on the nodes and connections that are carried out by agents and make use of the accumulated information to guide the evolution process. In this sense, the proposed method could strengthen the exploitation ability during evolution in order to obtain better performances than conventional GNP does.

2.2 Motivation of GNP with Rules

Conventional GNP has a feature that some of the nodes and connections of GNP may not be used by agents during its transition. For example, Figure 2.1 shows such a

case that the agent follows the route 1-2-3-5-4. In this route, firstly after starting from the start node, the agent executes the processing on Node 1 and transfer to Node 2. Then, after making the judgement on Node 2, it makes a decision to move to Node 3, etc. Finally, the agent ends with Node 4 and the task is finished. So, we can see the connections from node 2 to node 6, from node 3 to node 4, from node 5 to node 6 and from node 6 to node 1 are not used. Since only a part of the nodes and connections is used, we concentrate our attention on the used part of GNP because the unused part is considered unimportant.

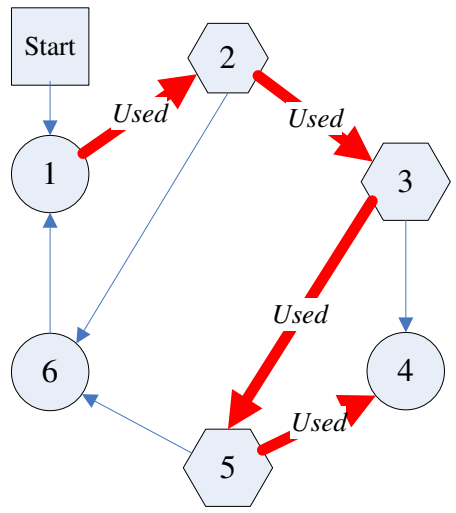


Figure 2.1: An example of GNP route

For further explanation, some definitions are given as follows.

Definition 1 (GNP route).

GNP route is a path on which the agent travels in a GNP individual. It consists of all the nodes and connections that the agent passes by.

Definition 2 (GNP rule).

GNP rule is a GNP connection contained on a GNP route. It can be denoted as $r(i(\alpha), j)$, which represents the connection from node i to node j via the α^{th} branch of node i.

As shown in Figure 3.1, in this example, the GNP route is $1-2-3-5-4$ consisting of all the used nodes and connections. This GNP route contains rules: $r(1(1), 2)$,

$r(2(2), 3)$, $r(3(1), 5)$ and $r(5(1), 4)$.

Obviously, the nodes and connections which are excluded from the GNP route are not used at all. Hence, an individual's fitness value is calculated only by the combination of its GNP rules. So, GNP rules could be considered essential to obtain good fitness values. The aim of the proposed method is to collect the information of GNP rules and make use of them to guide the evolution process and finally to get better individuals.

2.3 Algorithm of GNP with Rules

We consider the GNP rules of better individuals are better than those of worse individuals. In other words, good individuals could be obtained by the combination of good GNP rules. This approach strengthens the ability of exploitation. So, we make use of good GNP rules in evolution process. The proposed method consists of 4 steps:

- Step 1: Rule extraction to obtain all the GNP rules and their importance value in each individual in every generation and store them in the rule pool;
- Step 2: Rule selection to choose GNP rules for their use from the rule pools;
- Step 3: Individual construction to generate some new individuals by using the selected GNP rules;
- Step 4: Individual replacement to renew worse individuals by the constructed ones for the next population of the evolution process.

Figure 2.2 shows the outline of the algorithm of GNP-R. We are going to explain each step in detail in the followings.

2.3.1 Rule Extraction

In each generation, we record all the GNP rules of each individual and the number of individuals which include rule $r(i(\alpha), j)$ denoted by $c(i(\alpha), j)$. For example, if there are m individuals that include rule $r(i(\alpha), j)$, then $c(i(\alpha), j) = m$.

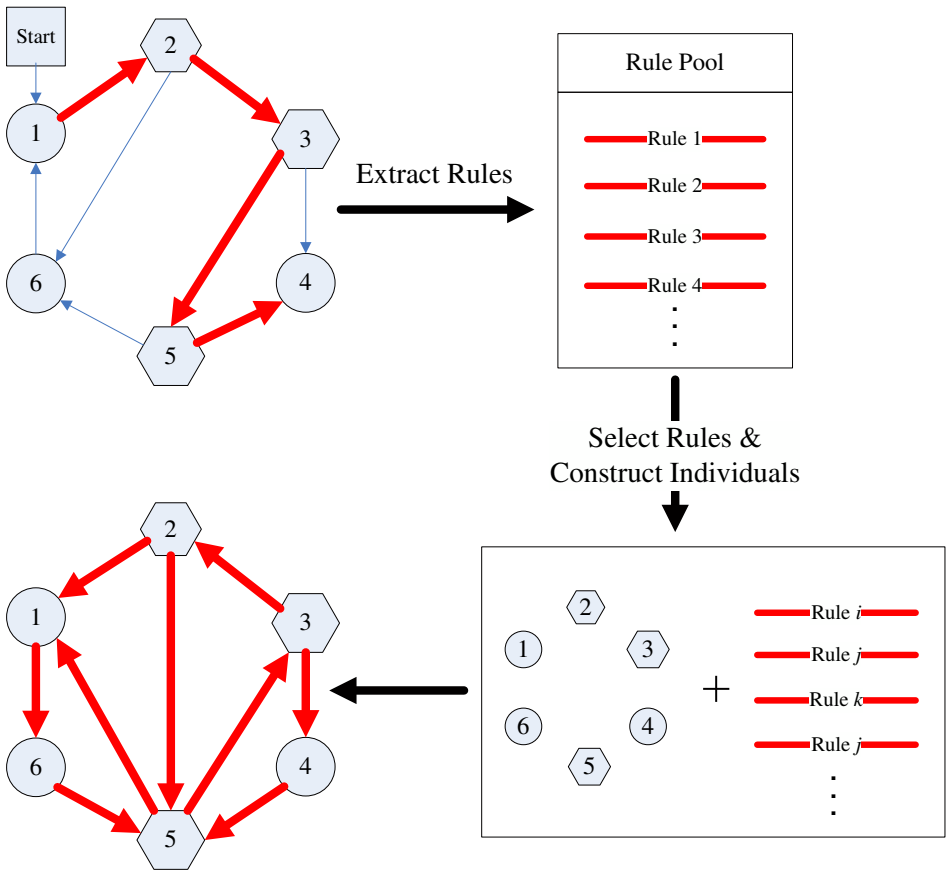


Figure 2.2: Outline of GNP with rules

And we evaluate the rules with "importance value" denoted by $v(i(\alpha), j)$. Importance value is the criterion of rule selection and it is calculated as follows:

$$v^n(i(\alpha), j)) = \frac{1}{c^n(i(\alpha), j)} \sum_{m \in IND} f_m^n \tag{2.1}$$

where, $v^n(i(\alpha), j)$ is the importance value of rule $r(i(\alpha), j)$ in the n^{th} generation, $c^n(i(\alpha), j)$ is the number of individuals which include rule $r(i(\alpha), j)$ in the n^{th} generation, f_m^n is the fitness value of the m^{th} individual in the n^{th} generation, and IND is the set of suffixes of individuals which include rule $r(i(\alpha), j)$.

The rule pool is updated generation by generation. If $v^{n-1}(i(\alpha), j)) < v^n(i(\alpha), j))$, the importance value of $r(i(\alpha), j))$ is updated by the higher one.

2.3.2 Rule Selection

The greedy policy is to choose the rule with the highest importance value. But, instead, the Boltzmann probability policy is used when selecting the rules such as $r(i(\alpha), j)$.

In the n^{th} generation, the probability of selecting rule $r(i(\alpha), j)$ is denoted as $P^n_{r(i(\alpha),j)}$ and it is calculated as follows:

$$P^n_{r(i(\alpha),j)} = \frac{e^{v^n(i(\alpha),j)/T}}{\sum_{j \in N} e^{v^n(i(\alpha),j)/T}} \tag{2.2}$$

where, N is the set of suffixes of nodes and T is the temperature parameters.

2.3.3 Individual Construction

After the step 2: rule selection, we construct new individuals by using the selected rules. To construct a new individual, we first generate an individual which contains only nodes, but no connections. Then, we select rules and add connections of the selected rules to the new individual. For example, if rule chain $r(i(\alpha), j)$ is selected for individual construction, the α^{th} branch of node i connects node j. If a node branch is not stored in the rule pool, an individual is selected by running a tournament selection and the branch will be connected to the same node as the one that is connected from the corresponding branch in the selected individual.

To construct how many new individuals depends on the problems. In our simulations, 20% individuals of the population size are constructed.

2.3.4 Individual Replacement

As is shown in Figure 2.3, after the individual construction, we replace the worst individuals of the population by constructed ones to obtain a new population. In our simulations the worst 20% individuals are replaced. Then the rest 80% individuals will undergo genetic operations.

Figure 2.4 shows the flow chart of the algorithm of GNP with rules.

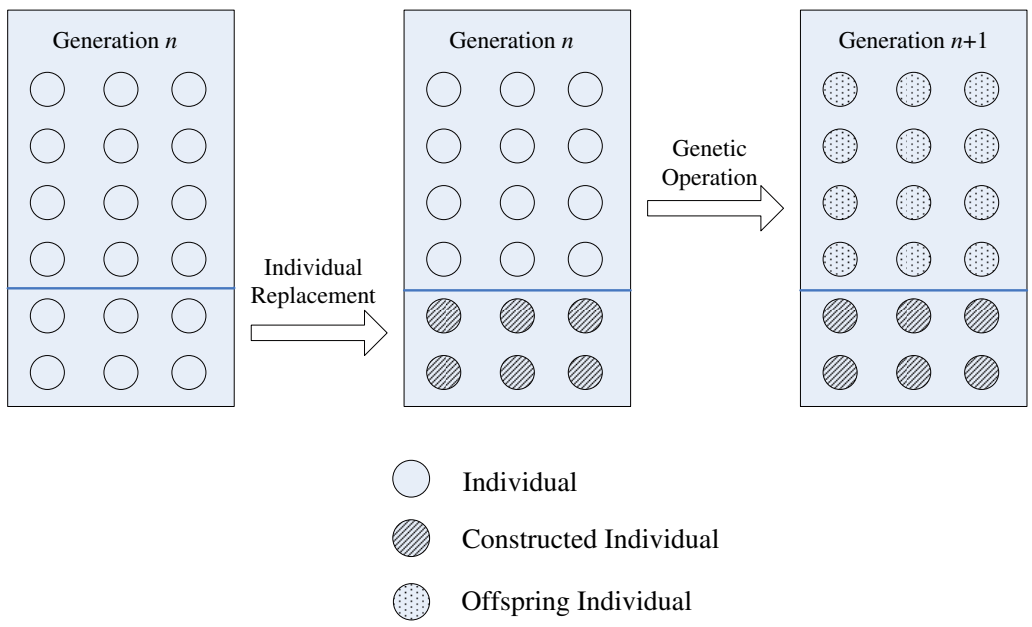


Figure 2.3: Individual replacement

2.4 Comparison between GNP with Rules and GNP

The evolution process of conventional GNP is similar to GA's. Essentially, the most significant difference between GNP with rules and conventional GNP is that GNP with rules replaces the worst part of individuals with new ones reconstructed by GNP rules in every generation. And the used GNP rules are mostly extracted from excellent individuals. Hence in GNP with rules, accumulated information is used to strengthen the ability of exploitation.

On the other hand, sometimes, in some generations, the genetic operators of GNP don't take effect. Because if the mutation and crossover parts of parents are never included in the GNP routes, the changes made by genetic operators do not influence the calculation of fitness values. This kind of situation occurs frequently especially when the individual has many nodes and connections. However, GNP with rules deletes the cases caught by such situations. In every generation, the population is preprocessed and the worst part of individuals are replaced by new individuals reconstructed by GNP rules before genetic operations. This replacement reduces the probability of occurrence of such situations.

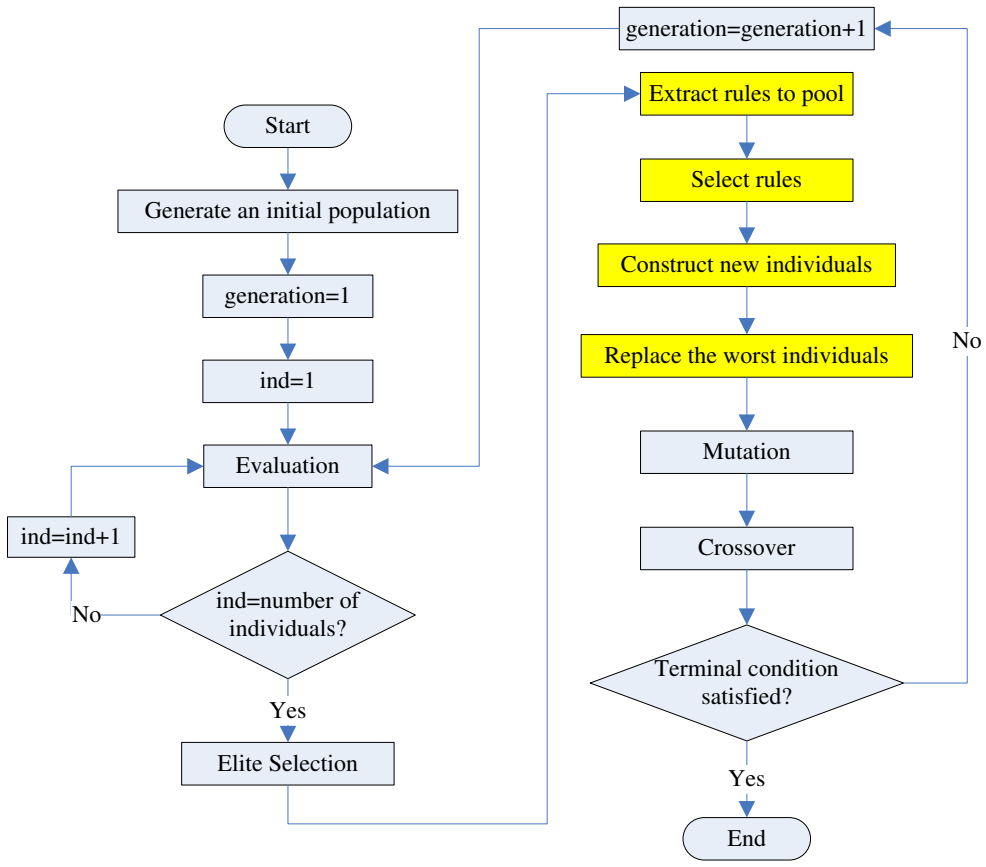


Figure 2.4: Flow chart of GNP with rules

2.5 Simulations

2.5.1 Tile-world

Since GNP is an agent-based algorithm, the effectiveness of GNP is evaluated based on the behaviors of the agents. In many research, tile-world problem has been adopted to study various GNP-related algorithms. Therefore, our experiments are still on the tile-world so as to compare the proposed method with standard GNP.

Tile-world is a two-dimensional world that contains five types of objects, namely, floor, obstacle, tile, hole and agent(37). The objective is to make the agents drop as many tiles into the holes as possible within a certain amount of time. Agents are able to move one grid per time step, and are able to push a tile to its neighboring grids.

The move or push should fail if the target grid contains an obstacle, a hole, or another agent. If a tile is dropped into a hole, they both will vanish and the grid will turn into a normal floor. Agents have some sensors and action abilities which should be predefined and they are required to use sensors and take actions properly according to their situations. Since the given sensors and simple actions are not enough to achieve tasks, agents must make smart combinations of judgments and processings. So, the tile-world is a reasonable benchmark to evaluate agent oriented systems, especially the GNP-based agent oriented systems.

2.5.2 Experimental Environments

We run 2 simulations to evaluate the performance of GNP with rules in the training and testing phase. In simulation 1, we trained GNP for the agents in 10 different worlds. Each world has 3 agents, 3 tiles and 3 holes. The positions of holes, obstacles and agents are the same in the 10 worlds. However, the positions of tiles are different from each other. Figure 2.5 shows the environments for training.

In simulation 2, after training, we tested the trained GNP in 8 new different environments, where the positions of tiles, holes and obstacles are totally different. Figure 2.6 shows the environments for testing.

2.5.3 Programming Configuration

In our program, there are 8 kinds of Judgment Nodes. Agents can find out what exists in front of each agent, in the same way, right, left, and back of each agent. Agents can also find out the rough direction from the agents to the place where the nearest tile is, where the second nearest tile is and where the nearest hole is. Furthermore, they can find out the rough direction from the nearest tile of the agent to the nearest hole. These different judgements help agents to make a decision in the following step.

And there are 4 kinds of Processing Nodes: to go forward, to turn left, to turn right and to stay. Once the agent takes an action, it consumes one step. In our program, totally, there are 60 allowable steps.

Each individual contains 60 nodes including 40 Judgement Nodes (5 for each kind of Judgement Nodes) and 20 Processing Nodes (5 for each kind of Processing Nodes). Each Judgement Node has 5 branches and each Processing Node has only one branch.

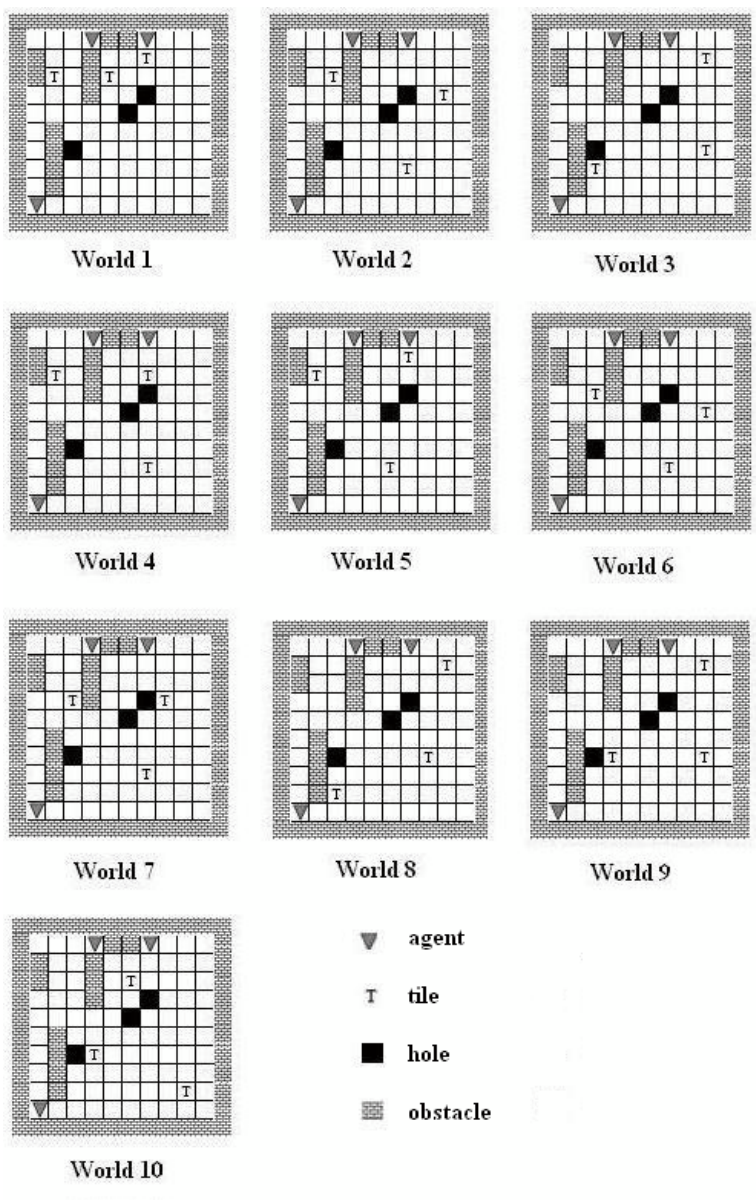


Figure 2.5: Training environments

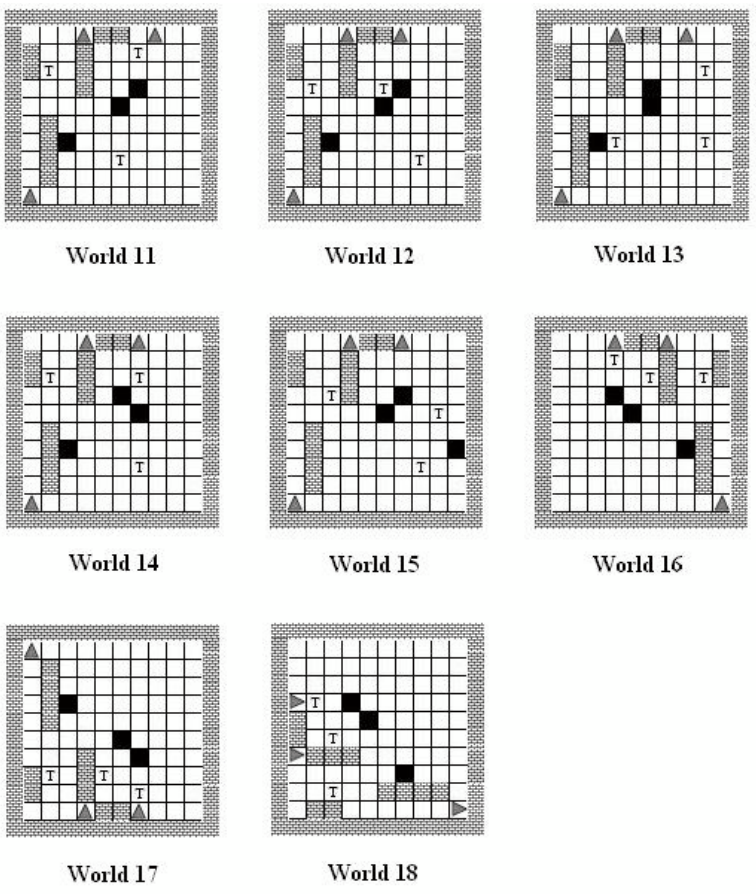


Figure 2.6: Testing environments

We used different populations of 30 individuals, 50 individuals, 100 individuals and 200 individuals in the experiments, with $crossover\ rate$ = 0.1, $mutation\ rate$ = 0.01. And all cases of different populations are carried out for 10 rounds. Table 2.1 shows other parameter configurations.

Table 2.1: Parameter Configuration

Individuals	Mutation size	Crossover size
31	20	10
51	30	20
101	60	40
201	120	80
Elite size		1
Mutation rate		0.01
Crossover rate		0.1
Generations		500
Temperature parameters T		20

The fitness is calculated by accumulating the scores obtained from each tile-world. The score function is closely related to the objective of the tile-world problem, represented by

$$Score = 100 \cdot DT + 20 \cdot \sum_{p=1}^{P} d(p) + (M_t - U_t), \tag{2.3}$$

where, DT is the number of tiles dropped into the holes, p is the ID of the relatively nearest tile-hole pair at every time step in the trials, P is the maximum number of the relatively nearest tile-hole pairs, $d(p)$ is the decrease of the distances between the tiles and holes in the pairs, M_t is the maximum time step, and U_t is the used time step.

Then, the fitness function is defined by

$$Fitness = \sum_{w=1}^{W} Score(w), \tag{2.4}$$

where, w is the ID of the tile-world, W is the maximum number of the training tile-worlds, and $Score(w)$ is the score obtained in the wth tile-world.

2.5.4 Simulation Results

2.5.4.1 Simulation 1

We compare GNP with rules and standard GNP. Figure 2.7 shows the average training results of different populations in 10 rounds. After 500 generations, in the case of 30, 50, 100 and 200 individuals, the average fitness value of standard GNP is 2943.8, 3154.6, 3432.9 and 3651.9, respectively, while the GNP with rules obtained the value of 3117.2, 3319.4, 3603.4 and 3844.4, respectively.

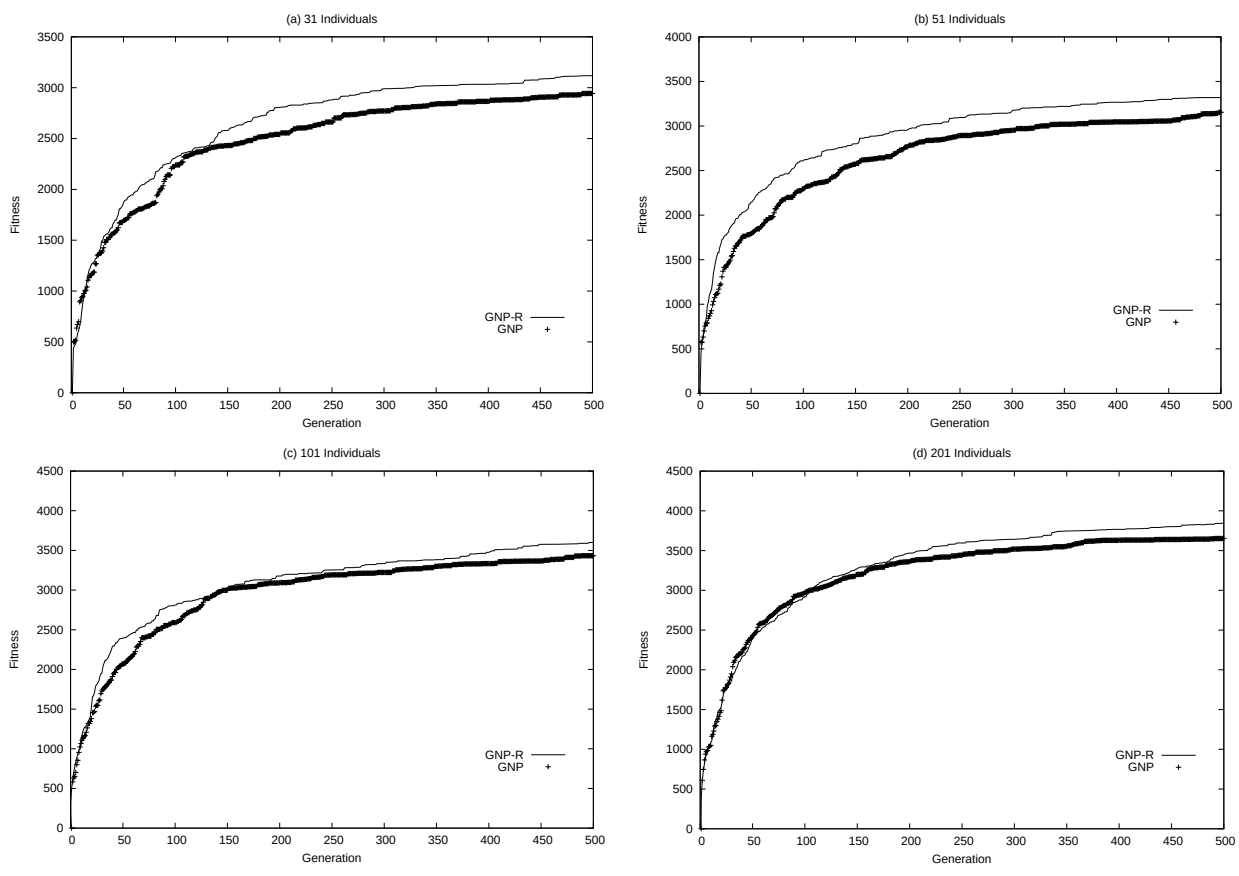


Figure 2.7: Training results

We can see that the proposed method obtained better performances than the conventional GNP did in the training process. Its advantage is more remarkable especially when the population size is smaller. Because population size is large, the agent can

search in a wide range and finally find the optimal solutions. In this case, the effects brought by the extracted rules are not very remarkable. But, when the population size is small, the agent failed to find the optimal solutions due to a very limited search range. So, the extracted rules can make worst individuals learn experiences from the memory and construction of the individuals gains new recombinations in gene structures.

2.5.4.2 Simulation 2

After simulation 1, we tested the trained agents in 8 new environments showed in Figure 2.6. Table 2.2 and Table 2.3 show the testing results of different populations of conventional GNP and the proposed method. The testing results are the average best fitness obtained for each test set over ten runs.

Table 2.2: Testing results of the proposed method

Population Size	31	51	101	201
World 11	880	1060	1460	1820
World 12	520	1100	1460	1520
World 13	60	100	100	220
World 14	1280	1340	1160	1580
World 15	-60	-80	0	20
World 16	780	700	560	660
World 17	240	220	480	420
World 18	1880	1360	880	840
Average	723	725	763	885

We can see that GNP with rules can strengthen the search ability of GNP. However, in the testing phase, when the trained agents deal with the new environments, although GNP with rules obtained better results than GNP, either standard GNP or GNP with rules can solve the new problems perfectly. Actually, it is very difficult for trained agents to deal with such situations and this drawback in the testing phase needs more hard research work in the future.

Table 2.3: Testing results of the conventional GNP

Population Size	31	51	101	201
World 11	1220	1640	1400	2880
World 12	1260	1000	1620	1260
World 13	-120	-180	-80	240
World 14	1520	1200	1400	1660
World 15	0	0	-200	-210
World 16	320	540	140	240
World 17	120	360	440	500
World 18	640	600	640	480
Average	620	645	670	881

2.6 Summary

This thesis proposed the method of GNP with rules to obtain better training results, faster convergence rate and better testing results. The proposed method strengthens the ability of exploitation and as a result, efficiently enhances the conventional GNP, especially when the population size is small.

In order to improve GNP with rules, some future work should be done in the future. We could extend GNP rules to even longer transitions of connections. As this extension considers more about the sequence of node connections, it could bring more exploitation ability in evolution process contributing to gain faster convergence rate, better training and testing results. And we could improve the method of GNP rules by considering not only the fitness value, but also other elements, e.g. rule occurrence frequency. That is to say, a more appropriate and scientific criterion to evaluate rules should be considered.

And the studies on strengthening the ability of solving complex problems in new environments in the testing phase is attractive and needed.

Chapter 3

GNP with Reconstructed Individuals

3.1 Introduction

GNP with rules adopts a memory scheme storing the node branches used by the agents and their importance values. In this chapter, another memory scheme named GNP with reconstructed individuals (GNP-RI) which stores the whole node transitions used by the agents, i.e., the GNP routes, is studied. In the previous chapter, the GNP rule is a connection from node to node indicating only one judgement or action for the agents. But, the complete node transition consists of successive judgements and actions of the agents which can be considered as the series of regulations guiding the agents to solve concrete problems. So, GNP-RI could be considered as an extension of GNP with rules.

In GNP-RI, the GNP routes of the best individuals are stored in the memory and they are used to reconstruct the worst individuals. So, the worst individuals can learn more knowledge from the elite ones. This approach mimics the maturing phenomenon in nature where bad individuals can become smarter after receiving a good education. In this sense, the proposed method could strengthen the exploitation ability during the evolution. Also in this chapter, GNP-RI is evaluated in tile-world problems.

3.2 Motivation of GNP-RI

As mentioned above, only the part of the nodes and connections are used during the agent's execution of the task in GNP, so the solution of GNP for the concrete problem

is represented only by the used nodes and connections instead of the whole individuals. As a result, in the case of the explicit memory, GNP has an advantage in the smaller size of memory, because it only needs to store the part of nodes and connections of the good solutions. So, it is natural to design an elegant explicit memory for standard GNP to improve its performances. In this chapter, an explicit memory scheme for GNP named GNP with reconstructed individuals (GNP-RI) is proposed.

GNP-RI is very similar to the aforementioned explicit memory of GA which stores the previous best solutions. In traditional GA, fitness is calculated by the string or vector of the individual. In that case, the best solutions is just the best individuals. While in GNP, an individual's fitness value is calculated only by the GNP route, so the GNP route can be considered the best solution instead of the whole gene structure of the individual.

The aim to propose GNP-RI is to collect the information of GNP routes of the best individuals in each generation and make use of them to guide the evolution process and finally to get better individuals.

3.3 Algorithm of GNP-RI

The aim of GNP-RI is to enhance the GNP population by reconstructing the worst individuals. The information on GNP routes of the best individuals is considered as the excellent model from which the bad individuals should learn and imitate. The reconstruction of bad individuals is executed before genetic operations as the education process instead of reproducing or selecting individuals as parents directly to the next generation without any reconstruction.

In nature, individuals grow up and become more suitable to the environments by learning from good examples. To incorporate this phenomenon into GNP, the reconstruction of the bad individuals is adopted, which is inspired by social interaction of knowledge. Currently, such kind of inspiration has been used in many studies. For example, Particle Swarm Optimization (PSO) which works based on social adaptation of knowledge introduces the concept of ”social” and ”cognition” by which the individuals share information and their individually learned knowledge each other(48; 49). The reconstruction can be considered similar to the ”social” concept in PSO that the bad individuals is reconstructed by the information of elites. There are also many research

indicating that such kind of enhancement for the population in the genetic algorithm can achieve better performance in applications(50)(51). For example, in (50), Juang proposed a hybrid method combining GA with PSO. In the hybrid method, in every generation the worst half of the population is enhanced by PSO and the better half reproduces offspring using GA's genetic operators.

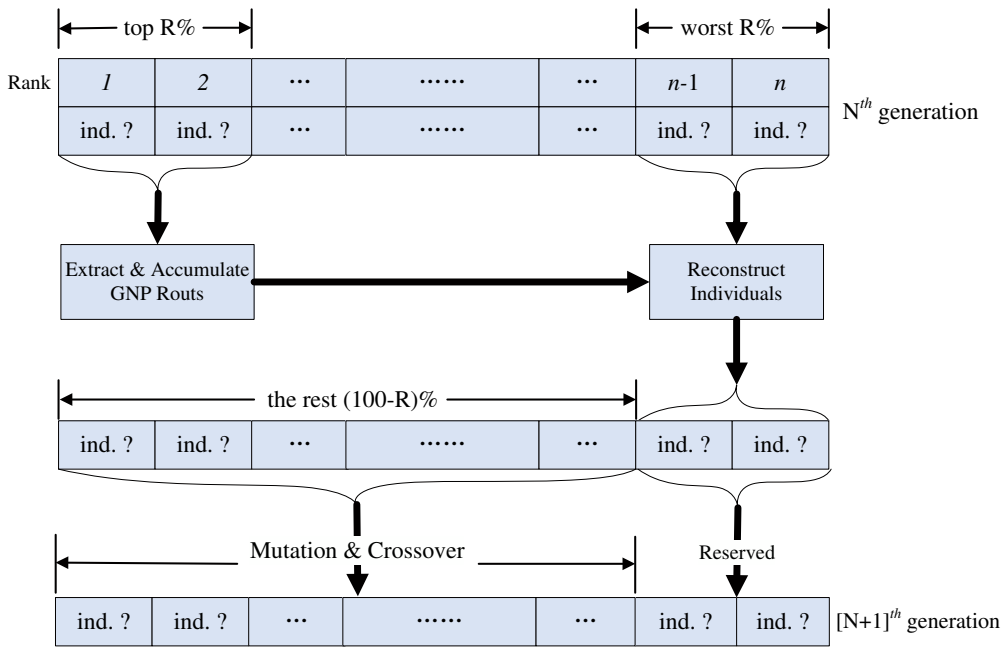


Figure 3.1: Flow of GNP-RI

For clarity, the flow of GNP-RI algorithm is illustrated in Figure 3.1. In each generation, after the fitness values of all the individuals are calculated, the top $R\%$ best-performing ones are regarded as elites. All the GNP routes of the elites are extracted and accumulated, then before undergoing genetic operations, the accumulated information is used to reconstruct the worst $R\%$ individuals. So, after reconstruction, the genes of the worst $R\%$ individuals are modified but the population size does not change.

In concrete, the GNP route of each of the best $R\%$ individuals is coded in a string structure, where the node number and the connection index of nodes are coded bit by

bit as follows:

$$|Node\_No.|Branch\_No.|Node\_No.|Branch\_No.|.....| \tag{3.1}$$

Figure 3.2 shows the string code of the GNP route which indicates a transition from Node I to Node J via the b_i^{th} branch, from Node J to Node K via the b_j^{th} branch, etc.

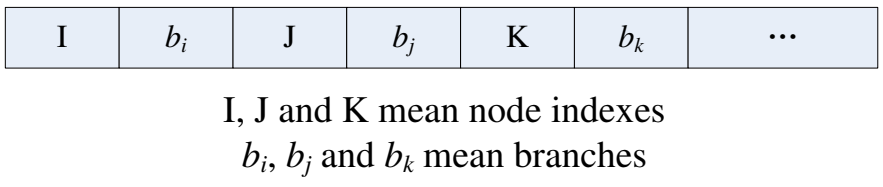


Figure 3.2: GNP Route Coding

For example, if we use the GNP route shown in Figure 3.2 to reconstruct individual r, the corresponding connections of individual r will be modified according to the route information. Consequently, in individual r, the b_ith branch of Node I will connect to Node J and the b_jth branch of Node J will connect to Node K, etc. The gene of individual r is modified by every accumulated GNP routes one by one. The modification sequence starts from the routes with lower fitness values to the ones with higher fitness values. At the beginning, the route with the lowest fitness value will modifies individual r and then the one with the second lowest will modifies individual r, etc. And the route with the highest fitness value will come at last. This sequence ensures that the GNP routes with higher fitness values have the priority to modify the gene of individuals over the ones with lower fitness values. For example, now GNP routes R_m and R_n have a disagreement on the lth branch of Node L. R_m who has the lower fitness value makes the lth branch of Node L connect to Node P, then, R_n who has the higher fitness value makes the lth branch of Node L connect to Node Q. R_m makes the modification first because it has the lower fitness value, but R_n overrides R_m's modification and at last the lth branch of Node L becomes connected to Node Q. These research indicate that the agents can improve themselves by learning form the elite peers and the historical information.

Figure 3.3 shows an example of how to use a GNP route to reconstruct the individual. Here, we use the route 1, 1, 4, 2, 3 which means node 1 connects to node 4 via its

1st branch, and node 4 connects to node 3 via its 2nd branch, to reconstruct an individual. After reconstruction, in the new individual, the 1st branch of node 1 connects to node 4 which once was connected to node 2 and the 2nd branch of node 4 connects to node 3.

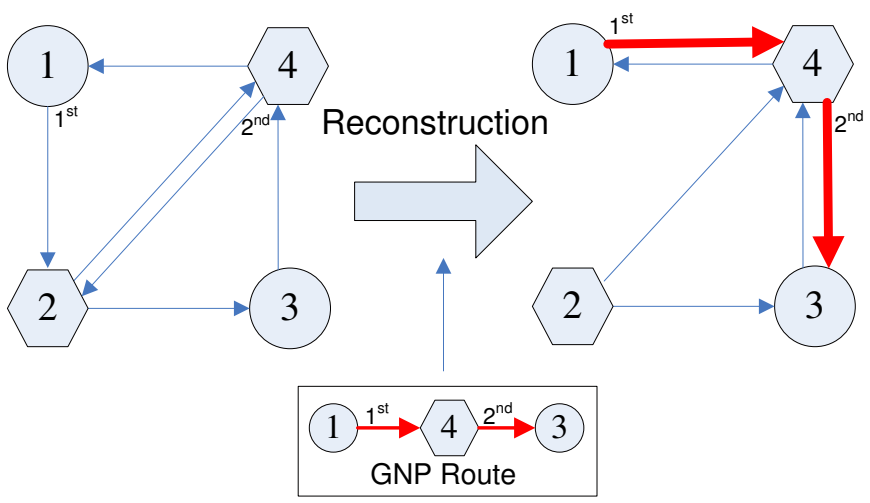


Figure 3.3: An example of individual reconstruction

During the evolution process, in each generation, we first evaluate all the individuals. The $R\%$ individuals with the smallest fitness values will be reconstructed, and as a result, their genes are changed. After the reconstruction, the rest $100 - R\%$ of the population undergoes the mutation and crossover. So, in GNP-RI, reconstruction can be considered as a new genetic operator and it produces the next generation combining with mutation and crossover.

3.4 Comparison between GNP-RI and GNP with rules

Essentially, the most significant difference between GNP-RI and GNP with rules is that GNP-RI modifies the gene structures of the worst part of individuals by using GNP routes in every generation. And the used GNP routes are extracted from excellent individuals. While GNP with rules replace the worst individuals with new individuals constructed by GNP rules. GNP rules are connections between nodes which indicates

one step of judgement or action for the agents. But, GNP routes are the whole transition of nodes and connections which indicates the complete regulations to execute the task for the agents. Both approaches make use of the accumulated information strengthening the exploitation ability.

3.5 Simulations

3.5.1 Experimental Environments

The performance of GNP-RI is evaluated in the benchmark of title-world problems using 3 simulations. In simulation 1, we trained GNP-RI and GNP for the agents in 10 different worlds. Each world has 3 agents, 3 tiles and 3 holes. The positions of holes, obstacles and agents are the same in the 10 worlds. However, the positions of tiles are different from each other. The training environments are shown in Figure 2.5.

In simulation 2, after training, we tested the trained agents in 8 new different environments, where the positions of tiles, holes and obstacles are totally different. The tesing environments are shown in Figure 2.6.

As some research has demonstrated the significant superiority of GNP over some classic evolutionary algorithms such as GA an EP in the bench mark problem of tile-world, so in this paper, we only compare the performance of GNP-RI and GNP.

In simulation 3, we studied the best $R\%$ of reconstructed individuals for GNP-RI. The parameter $R\%$ is very important to the architecture because it controls the degrees of exploitation and exploration during the evolution which generate significant effects on the performance of agents.

3.5.2 Programming Configuration

In our program, there are 8 kinds of Judgment Nodes: J-forward, J-left, J-right, J-backward, J-near-tile, J-near-hole, J-near-tile-to-hole and J-second-near-tile. The first 4 kinds of nodes represent the judgement of what is in front of the agent, what is at the left of the agent, what is at the right of the agent, and what is at the back of the agent, respectively. Each agent has a sensor, which can help the agent to identify which range the target objects locate in. So, the last 4 kinds of nodes represent the judgement of

where the nearest tile is, where the nearest hole is, where the nearest tile's nearest hole is, and where is the second nearest tile is, respectively.

And there are 4 kinds of Processing Nodes: to go forward, to turn left, to turn right and to stay. Once the agent takes an action, it consumes one step. In our program, totally, there are 60 allowable steps.

Each individual contains 60 nodes including 40 Judgement Nodes (5 for each kind of Judgement Nodes) and 20 Processing Nodes (5 for each kind of Processing Nodes). Each Judgement Node has 5 branches and each Processing Node has only one branch.

We used the population of 31, 121 and 201 individuals in the experiments for GNP and GNP-RI, with $crossover\ rate$ = 0.1, $mutation\ rate$ = 0.01. And all cases of different populations are carried out for 30 random rounds. Table 3.1 shows the details about parameter configurations.

Table 3.1: Parameter Configuration

GNP			
Population size	Mutation size	Crossover size	Elite size
31	20	10	1
121	72	48	1
201	120	80	1
Mutation rate		0.01	
Crossover rate		0.1	
Number of generations		1000	
GNP-RI			
Population size	Mutation size	Crossover size	Elite size
31	16	8	1
121	58	38	1
201	96	64	1
Mutation rate		0.01	
Crossover rate		0.1	
$R\%$		20%	
Number of generations		1000	

The fitness is calculated by accumulating the scores obtained from each tile-world. The score function is closely related to the objective of the tile-world problem, repre-

sented by

$$Score = 100 \cdot DT + 20 \cdot \sum_{p=1}^{P} d(p) + (M_t - U_t), \tag{3.2}$$

where, DT is the number of tiles dropped into the holes, p is the ID of the relatively nearest tile-hole pair at every time step in the trials, P is the maximum number of the relatively nearest tile-hole pairs, $d(p)$ is the decrease of the distances between the tiles and holes in the pairs, M_t is the maximum time step, and U_t is the used time step.

Then, the fitness function is defined by

$$Fitness = \sum_{w=1}^{W} Score(w), \tag{3.3}$$

where, w is the ID of the tile-world, W is the maximum number of the training tile-worlds, and $Score(w)$ is the score obtained in the wth tile-world.

3.5.3 Simulation Results

3.5.3.1 Simulation 1

Figure 3.4 shows the average best fitness curve of training results of GNP-RI and GNP with population of 31, 121 and 201 individuals over 30 random rounds. We can see that when the population size is small (31 individuals) both GNP-RI and GNP made premature convergence and their performances are almost the same due to the low diversity of the population. When the population size becomes larger, GNP-RI shows an increasing superiority over GNP. Table 3.2 shows the average fitness of the best individuals result of GNP-RI and GNP at the last generation and the results of t-test which demonstrate that there are significant differences between the training results of GNP-RI and GNP.

Table 3.2: Average of the best individual fitness results at the last generation

Population size	31	121	201
GNP-RI	3063.1	3876.1	4363.5
GNP	2957.8	3653.6	3994.6
t-test (p-value)	0.0175	0.0133	0.0096

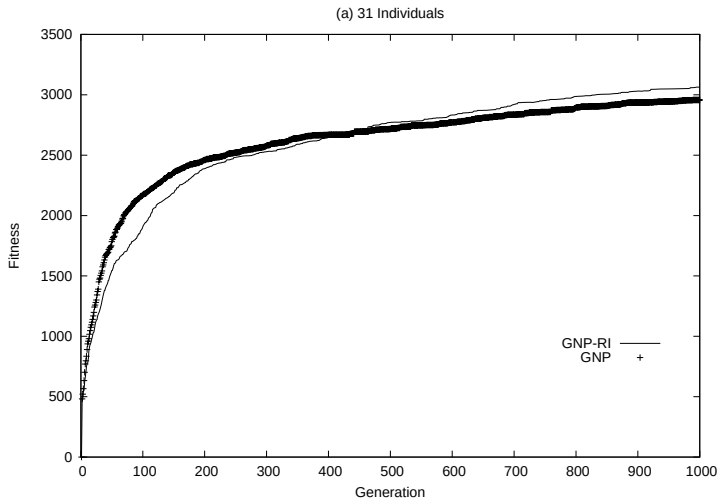


Figure 3.4: Training results

3.5.3.2 Simulation 2

Table 3.3 and Table 3.4 shows the average best fitness of GNP-RI and GNP over 30 random rounds in the 8 new worlds in the testing case, respectively. We can see that in new environments, where the agents have never been trained before, GNP-RI can obtain better testing results than GNP. GNP-RI performed better than the conventional GNP in most of the worlds. The experiment is a stochastic process and the simulation result varies between different random runs. So, it is possible that there exist the cases when GNP-RI is worse than GNP. I have run the program for many random rounds (30 rounds) in order to show that in most cases, the proposed method can obtain better performance than GNP, but due to the stochastic characteristic of the problem, we cannot ensure the proposed method is sure to be better than GNP every time. The testing results indicate that not only GNP-RI can perform better than GNP in the trained environments, but also GNP-RI has more generalization ability than GNP in the new environments. But, the testing results are not as good as the training results which means the generality of agents in different environments are not obtained yet.

Table 3.3: Testing results of GNP-RI

Population size	31	121	201
World 11	36.7	162.3	203.3
World 12	24.3	133.7	196.7
World 13	11.7	128.7	245.0
World 14	-10.3	102.3	184.7
World 15	64.3	152.3	227.3
World 16	77.3	-54.7	172.3
World 17	46.7	201.7	248.7
World 18	12.7	144.3	196.3
Average	32.925	121.325	209.2875

3.5.3.3 Simulation 3

In this simulation, we trained agents using GNP-RI with different $R\%$ settings in 10 tile-worlds. The 10 environments are shown in Figure 3.5 where the tile positions are different and the initial positions of the agents are the same. World 1-6 have the same

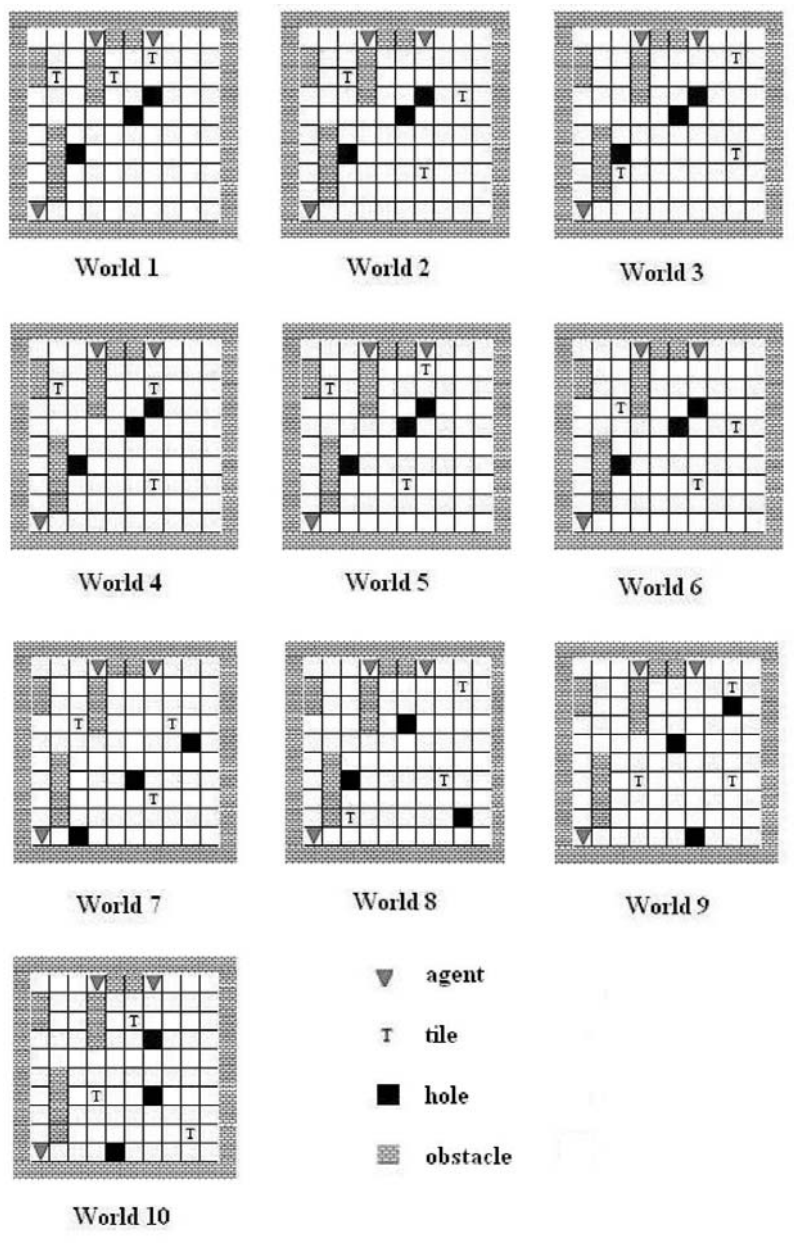


Figure 3.5: Environment set for simulation 3

Table 3.4: Testing results of GNP

Population size	31	121	201
World 11	46.0	106.7	154.7
World 12	-16.7	99.3	215.3
World 13	20.0	93.7	198.3
World 14	24.3	-59.3	166.7
World 15	-18.7	88.7	237.3
World 16	84.7	115.3	-56.7
World 17	-9.3	126.7	226.3
World 18	24.3	73.7	174.3
Average	19.325	80.6	164.525

distribution of obstacles and holes and World 7-10 have the same obstacle distribution, but the hole positions are different. So, the last 4 worlds are more complicated. The purpose of this simulation is to find the optimal $R\%$ configuration in general cases.

Figure 3.6 shows the average of the best fitness curves of GNP-RI with the population of 201 in 500 generations over 30 random rounds using different $R\%$ configurations. The result suggests that GNP-RI obtains an excellent performance when $R\%$ is set at small values (0.1, 0.15 and 0.2). If $R\%$ is set at too large values, a great number of individuals will be reconstructed resulting in a massive loss in the population diversity.

3.6 Summary

We proposed a method of GNP with Reconstructed Individuals (GNP-RI) which shows a significant improvement of the performance of GNP. The proposed method modifies the gene structures of the worst individuals before undergoing genetic operations in every generation by using the information on the routes of elite GNPs in order to enhance the performance of the conventional GNP. The enhanced reconstruction makes the bad individuals learn from the elite individuals before they reproduce offspring. The simulation results in the tile-world problem shows the superiority of the performance of GNP-RI over that of the conventional GNP both in the training and testing phase.

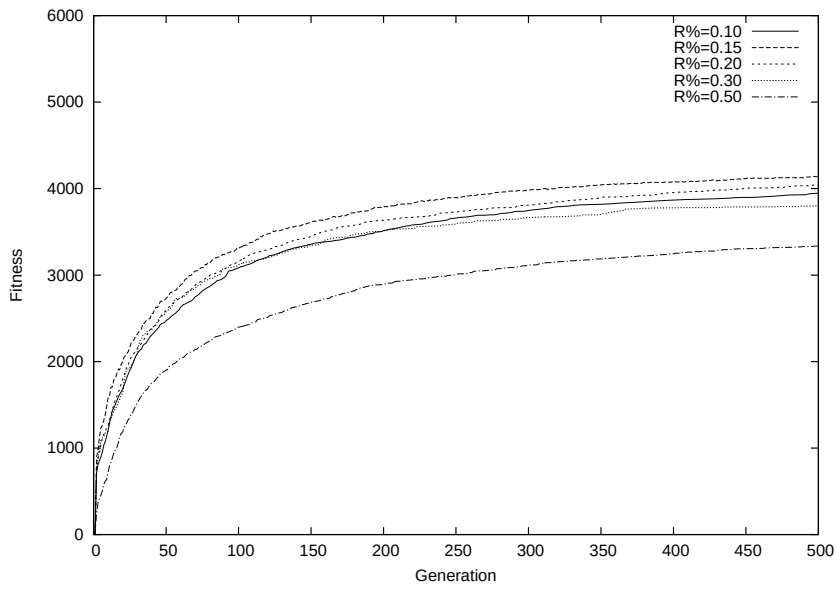


Figure 3.6: Average of the best fitness curves of GNP-RI using different reconstruction sizes

Furthermore, the simulation results show that GNP-RI can obtain much more remarkable superiority over GNP in the case of large population than that in the case of small population. While the opposite results are obtained by GNP with rules. Because much more useful route informations is extracted and used during the evolution in the case of large population. As a result, GNP-RI can perform much better than GNP when population size is large. While GNP with rules extracts and uses first-order rule information (length 1) which is less sufficient and effective than route information. Since GNP itself can evolve to achieve good results when population size is large, the less effective first-order rule information cannot enhance GNP too much.

Chapter 4

GNP with Route Nodes

4.1 Introduction

In Artificial Intelligence, an agent is used for intelligent actors which observe and act upon an environment. A rational agent is an entity that is capable of doing perception, action and goal directed behavior. The aforementioned GNP-RI employs an explicit memory scheme storing the GNP routes of the best solutions in each generation. Because the node transition of GNP which begins from a start node and transfers based on the judgments on the nodes and connections, is just the behaviors regulation to guide the agent's action in the environment. In other words, the route of GNP transitions which consists of a series of successive GNP transitions from node to node in each individual of GNP corresponds to the agent's behaviors. So, the stored GNP routes can be considered as the best solutions.

The memory of GNP-RI reuses the best solutions in the way that the gene structures of the worst individuals are modified by the stored GNP routes. In this chapter, a new explicit memory scheme for GNP named GNP with route nodes (GNP-RN) is proposed. GNP-RN also stores the GNP routes of the best individuals, but reuse them in a different way which is much more flexible and efficient. In order to verify the effectiveness of the proposed architecture we report the experimental results using the tile-world.

4.2 Motivation of GNP-RN

It is mentioned before that GNP-RI uses the stored best solutions to modify the gene structures of the worst individuals. But this reusing mechanism has the following two disadvantages: firstly, after reconstruction, the gene structures of the worst individuals become more similar to the elite ones' which means a loss in population diversity. Secondly, the worst individuals learn experiences from the elite ones without considering their own situation but other ones' experiences may not suitable for them at all. So, GNP-RN is designed to overcome the above drawbacks which let individuals learn information from the memory more efficiently.

4.3 Mechanism of GNP-RN

4.3.1 New Nodes: Route Nodes

In GNP-RN, the memory also stores the GNP routes of the best individuals and the memory is updated to ensure the elitism of the recorded GNP routes in each generation. The encoding of the GNP routes and the organization of the memory are similar to GNP-RI. But in GNP-RN, a fixed number of *route nodes* is added into each GNP individual which is connected from other nodes and has only one output to the next node. When the agent transfers to the route node, it will refer to the memory and retrieve some useful information from the recorded GNP routes.

4.3.2 Procedure of the GNP-RN

In the initial generation, an empty route pool without any route in it is built and N empty route nodes are assigned to each individual, which means when the agent reaches these empty route nodes, it will take no actions at all. After evaluating GNP, the GNP routes of the top $R\%$ individuals are extracted and accumulated in the route pool. In the following generation, new GNP routes will be extracted and the route pool will be updated. The whole procedure of the memory maintenance consists of 2 steps:

- Step 1: GNP route extraction;
- Step 2: Route pool update;

In the first step, after evaluating all the individuals, the GNP route of each individual is extracted and an important value is assigned to each route, which equals to its fitness value. In the second step, according to the fitness values, the top $R\%$ routes are mixed with the routes accumulated in the route pool. So, the route pool contains $2 \times R\%$ routes and in order to maintain the size of the pool, only the better half of the routes will remain in the route pool. This procedure ensures that the only the historical best solutions are stored in the memory. Figure 4.1 shows the flow of the GNP-RN architecture.

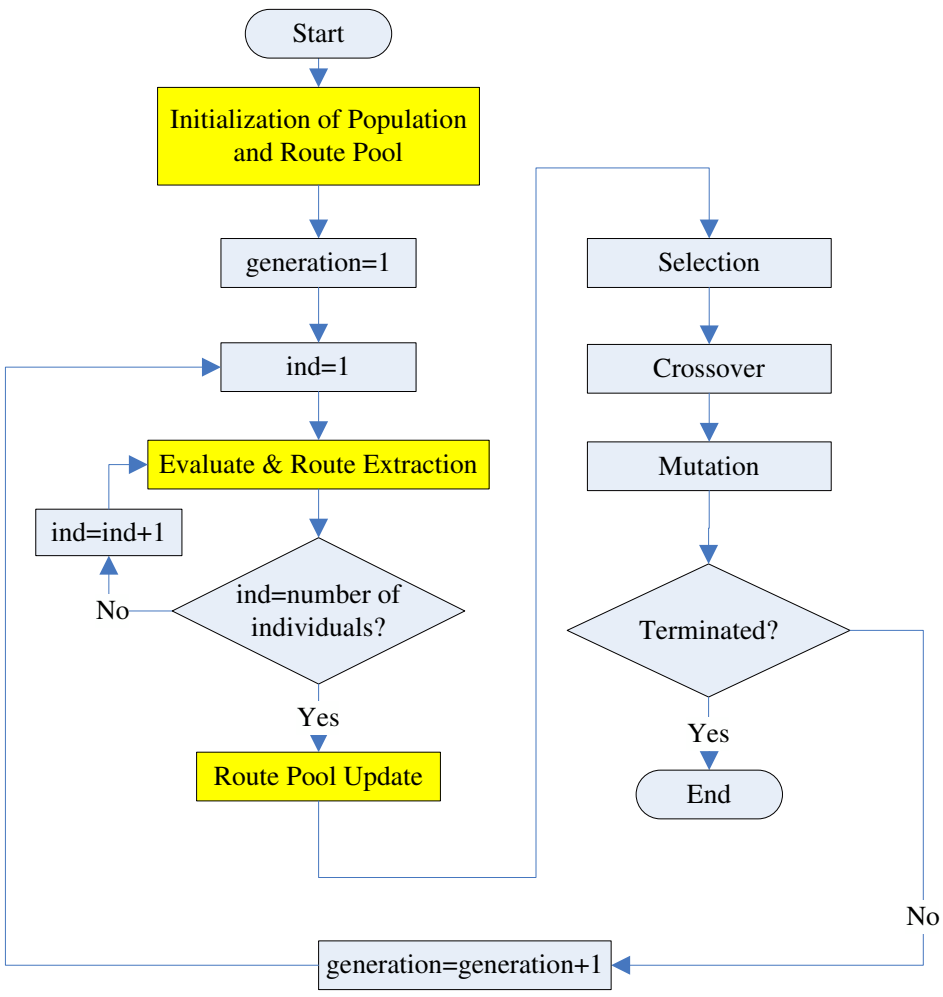


Figure 4.1: GNP-RN architecture

When the agent reaches the route node, the memory runs a tournament selection to select a better recorded GNP route, and the agent directly follows the judgements and processings on the selected GNP route. If the agent meets a judgement node on the route, it will make the judgement of the node using the current condition in the environment. If the judgement result satisfies the condition indicated by the route, the agent will move to the next node on the route, otherwise, it will directly jump to the judgment node right after the next processing node on the route. Let's make the maze problem as an example. An agent is located in a maze with obstacles and paths distributed in it. The agent should make judgements in many situations, such as whether an obstacle or a path is in front of it. When the agent transfers to a route node and refer to a certain recorded route in the memory, it will consider the judgements and processings on this route. For example, if the judgement and processing nodes on the route indicate a task regulation like **IF** ***front is path,*** **THEN** ***move forward.***, the agent will judge whether there is a path in front of it in the current environment. If there is a path, the agent will move forward. Otherwise, it will make the next judgment on the route. Figure 4.2 illustrates how the recorded GNP routes are used by GNP individuals in the proposed memory schemes. In this figure, when the agent reaches the route node, it first judges the current situation on judgement node 2 and if the judgement result doesn't satisfy the route condition, then the agent directly jumps to judgement node 3 instead of taking an action on processing node 1.

4.3.3 Discussion

GNP route is a path on which the agent transfers in a GNP individual. It consists of all the nodes and connections that the agent passed by. It actually contains the series of regulations that the agent should follow in the form of "Judge & Process". So, the individuals can learn experiences from the route of the best individuals recorded in the memory. Unlike the aforementioned EAs with memory, GNP-RN stores only GNP routes that are the useful part of the elite individuals in the memory. GNP-RN storing GNP routes in the memory can be naturally considered as the associative memory scheme which not only stores the best solutions, but also their environment information, when considering the unique characteristics of GNP that the transitions

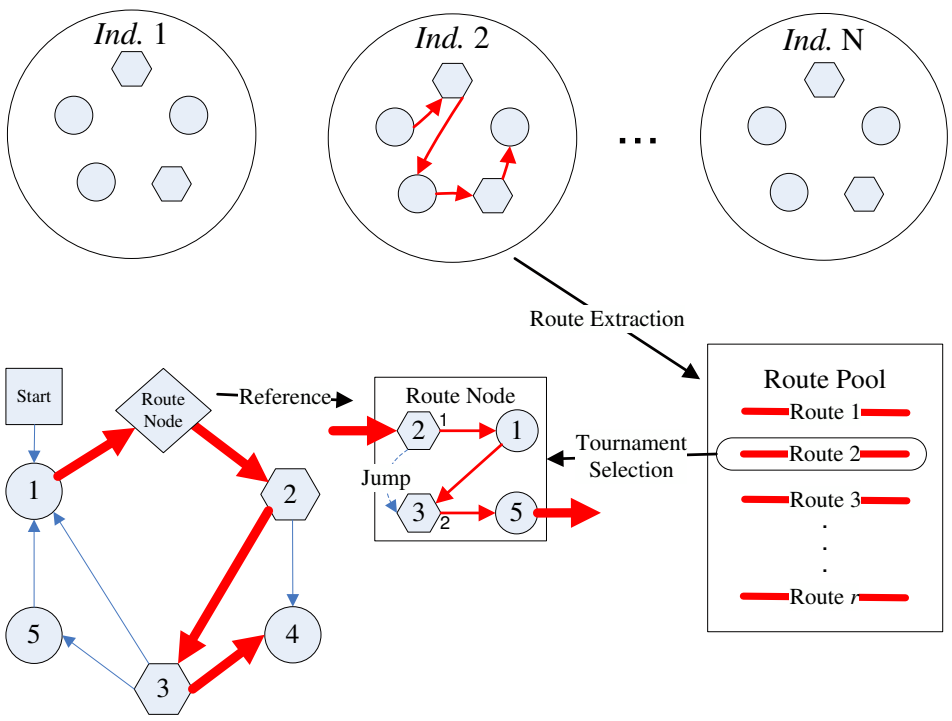


Figure 4.2: The Procedure of the Proposed Memory Scheme

of judgement nodes and processing nodes can memorize the past environment information and its associated judgement results and its associated actions. Consequently, when the agent refers to the memory, it will use the good experiences accumulated in the past. This characteristic can make reuse of the past better knowledge more flexibly and efficiently than the direct memory scheme.

4.4 Simulations

GNP-RI can be also considered as an explicit memory scheme for GNP. The memory organization of GNP-RI is very similar to GNP-RN and the main difference is in the mechanisms of reusing memory information. Therefore in the simulation part, GNP-RI and standard GNP are compared with the proposed method GNP-RN.

4.4.1 Experimental Environments

We still use the tile-world to demonstrate the effectiveness of the proposed architecture. Totally 3 simulations are conducted. In simulation 1, we trained the agents in 10 tile-worlds. Each environment is a 2D space which contains 3 agents, 3 tiles and 3 holes. And the performances of GNP-RN, GNP-RI and GNP are compared. In simulation 2, we trained the agents in another 6 tile-worlds which are much more complicated for the agents and the performances of GNP-RN, GNP-RI and GNP are evaluated. In simulation 3, we tested the trained agents in simulation 1 using 9 different tile-worlds from the training.

As some research has demonstrated the significant superiority of GNP over some classical evolutionary algorithms such as GA, GP and EP (33), so in this simulation, we only compared the performance of GNP-RN, GNP-RI and standard GNP to demonstrate whether the memory scheme can enhance the performance of GNP or not.

4.4.2 Programming Configuration

In our program, there are 8 kinds of Judgment Nodes: J-forward, J-left, J-right, J-backward, J-near-tile, J-near-hole, J-near-tile-to-hole and J-second-near-tile. The first 4 kinds of nodes represent the judgement of what is in front of the agent, what is at the left of the agent, what is at the right of the agent, and what is at the back of the agent, respectively. Each agent has a sensor, which can help the agent to identify which range the target objects are located in. Then, the last 4 kinds of nodes represent the judgement on where the nearest tile is, where the nearest hole is, where the nearest tile's nearest hole is, and where the second nearest tile is, respectively.

And there are 4 kinds of Processing Nodes: to go forward, to turn left, to turn right and to stay. Once the agent takes an action, it consumes one step. In our program, totally, there are 60 allowable steps.

Each individual contains 60 nodes including 40 Judgement Nodes (5 for each kind of Judgement Nodes) and 20 Processing Nodes (5 for each kind of Processing Nodes). Each Judgement Node has 5 branches and each Processing Node has only one branch. And for GNP-RN, each individuals has 5 route nodes, and in each generation the GNP routes of the best 20% individuals are extracted.

We used the population of 201 individuals in the experiments and the crossover and mutation rate are predefined as P_c = 0.1 and P_m = 0.01. For GNP-RI, the reconstructed size $R\%$ = 15% which is considered as the optimal setting for GNP-RI in the previous chapter. All the predefined settings of parameters make these methods to achieve the best result. And all the simulations are carried out for 30 random rounds for average calculation. Table 4.1 shows the details about parameter configurations in the simulations.

Table 4.1: Parameter Configuration

Parameter	Value
Population Size	201
- Elite	1
- Crossover	80
- Mutation	120
Generation	500
Crossover Rate P_c	0.1
Mutation Rate P_m	0.01
Node	
- Judgement Node	40
- Processing Node	20
- Start Node	1
- Route Node (for GNP-RN only)	5
Memory Size	
- GNP-RN	40 routes
- GNP-RI	15% of the population

The fitness is calculated by accumulating the scores obtained from each tile-world. The score function is closely related to the objective of the tile-world problem, represented by

$$Score = 100 \cdot DT + 20 \cdot \sum_{p=1}^{P} d(p) + (M_t - U_t), \tag{4.1}$$

where, DT is the number of tiles dropped into the holes, p is the ID of the relatively nearest tile-hole pair at every time step in the trials, P is the maximum number of the relatively nearest tile-hole pairs, $d(p)$ is the decrease of the distance between the tile

and hole in the relatively nearest pairs, M_t is the maximum time step, and U_t is the used time step.

Then, the fitness function is defined by

$$Fitness = \sum_{w=1}^{W} Score(w), \tag{4.2}$$

where, w is the ID of the tile-world, W is the maximum number of the training tile-worlds, and $Score(w)$ is the score obtained in the wth tile-world.

4.4.3 Simulation Results

4.4.3.1 Simulation 1

Figure 4.3 illustrates the 10 tile-worlds of the simulation environments. The tile positions are different and the initial positions of the agents are the same. World 1-6 have the same distribution of obstacles and holes and World 7-10 have the same obstacle distribution, but the hole positions are different. So, the last 4 worlds are more complicated. Figure 4.4 shows the averaged best fitness curves over 30 random rounds in the training of GNP-RN, GNP-RI and GNP, which shows that GNP-RN obtained a better result than GNP-RI and GNP-RI performed better than GNP. The average of the best fitness values of GNP-RN, GNP-RI and GNP are 4272.0, 4142.6 and 3632.1, respectively in the last generation.

4.4.3.2 Simulation 2

In this simulation, we trained GNP-RN, GNP-RI and GNP in another 6 tile-worlds. Figure 4.5 shows the experimental environments used in this simulation. We can see the distributions of obstacles, tiles and hole are different from each other, which means the environments are more complicated than the ones in simulation 1. So, it is difficult for agents to accomplish their tasks. Figure 4.6 shows the average best fitness curves over 30 random rounds in the training of GNP-RN, GNP-RI and GNP in this simulation. The average of the best fitness values of GNP-RN, GNP-RI and GNP are 1953.5, 1757.6 and 1570.4, respectively in the last generation. We can find that agents failed to drop all the tiles into the holes in all environments. The performance of GNP is poor because the agent can drop only 2 tiles on average over tile-worlds according

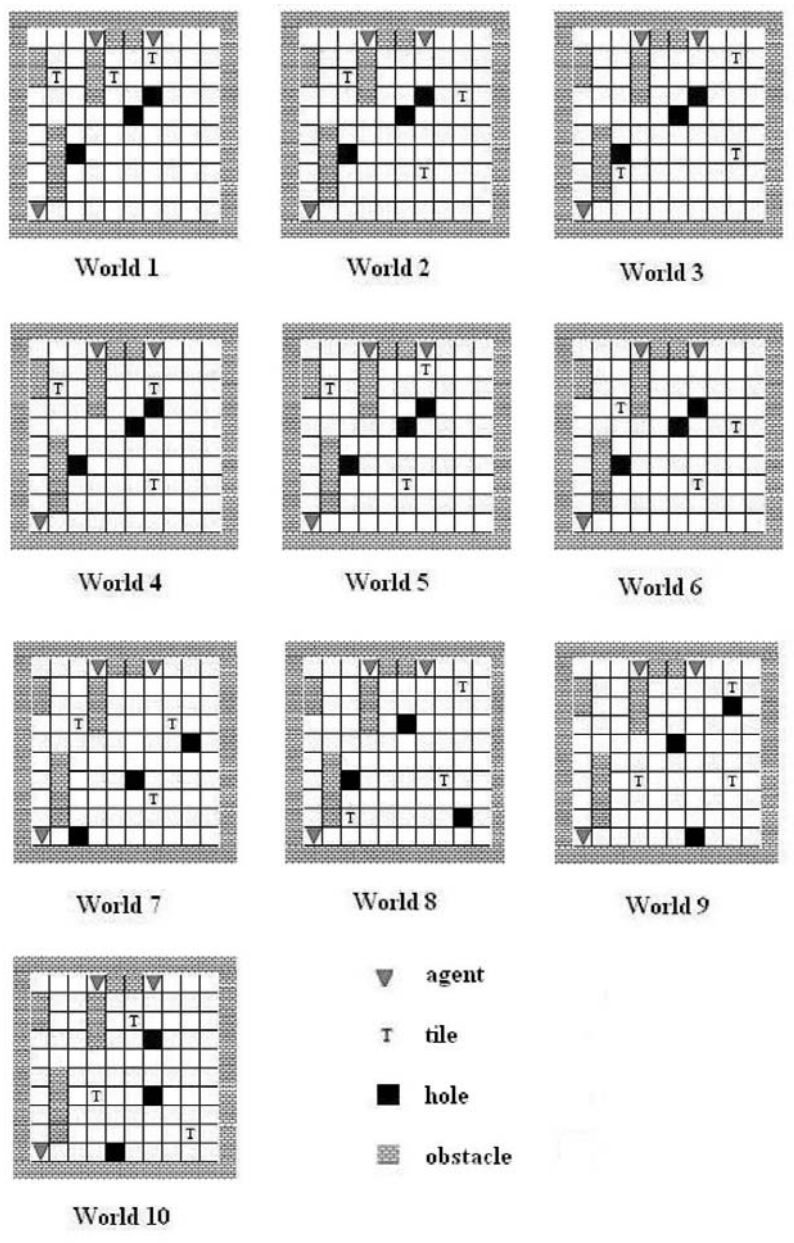


Figure 4.3: Training environments in simulation 1

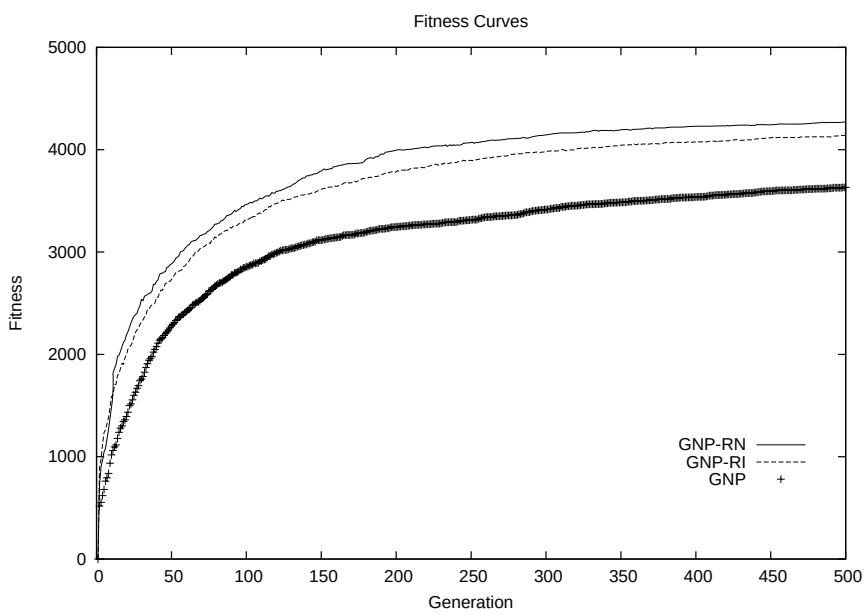


Figure 4.4: Averaged best fitness curves over 30 random rounds in simulation 1

to the simulation result. GNP-RI performed better than GNP. It obtained the average fitness of 293 over tile-worlds (failed to drop all the tiles in the 6 worlds). However, GNP-RN achieved the average fitness of 325 over tile-worlds (drop 3 tiles in most worlds) which means that it has a wider generality in more complicated environments. Therefore GNP-RN performed better than GNP-RI and GNP when dealing with more complicated problems.

The results of simulation 1 and 2 demonstrate that GNP-RN and GNP-RI can obtain better training results than standard GNP. It is natural that better performances can be gained when GNP is equipped with memory schemes because the past experiences are used during the evolution. The reason why GNP-RN performed better than GNP-RI coms form the following points:

- 1. In GNP-RI, the memory only records the information on the best solutions of the current generation, but in GNP-RN, the memory records the information on the best solutions of the whole history of the population, which means that the agent in GNP-RN can learn better knowledge than GNP-RI.
- 2. In GNP-RI, the GNP routes stored in the memory are used to reconstruct only

the gene structures of the worst individuals. So, only a part of the population are benefited by the memorized information, while in GNP-RN, all the individuals are guided by the stored information on the route nodes.

- 3. When the memorized GNP routes in GNP-RN are used, the agent will consider whether the rules contained in the GNP route satisfy the situation of the current environment. But in GNP-RI, no such consideration is adopted by the agent.

- 4. In GNP-RI, the accumulated GNP routes will modify the gene structures of the worst individual, which decreases the population diversity. On the other hand, the individuals in GNP-RN can learn the past experiences and meanwhile, their own gene structures are not destroyed, so the population diversity is maintained.

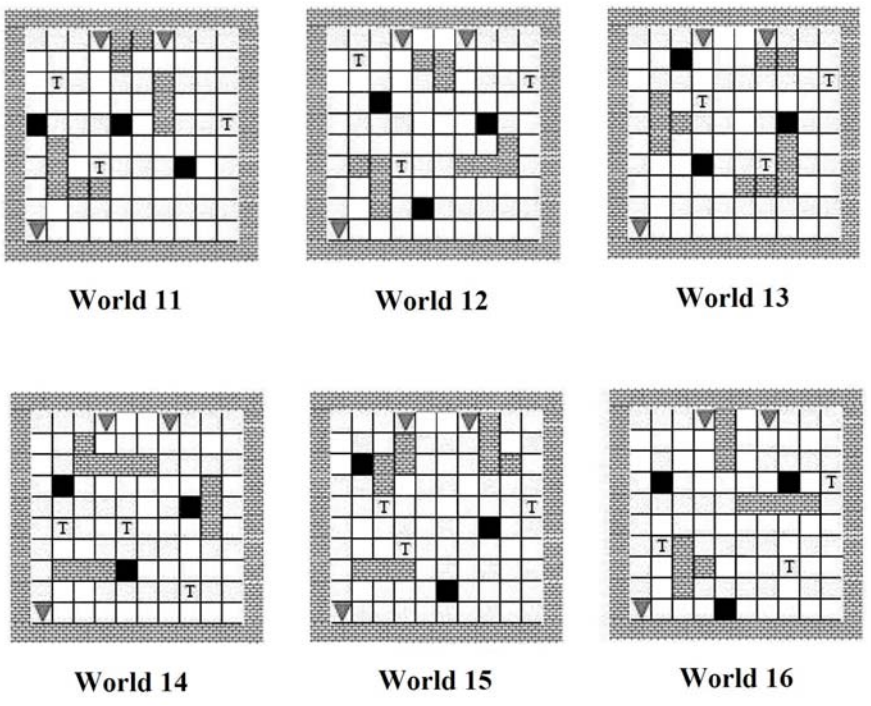


Figure 4.5: Training environments in simulation 2

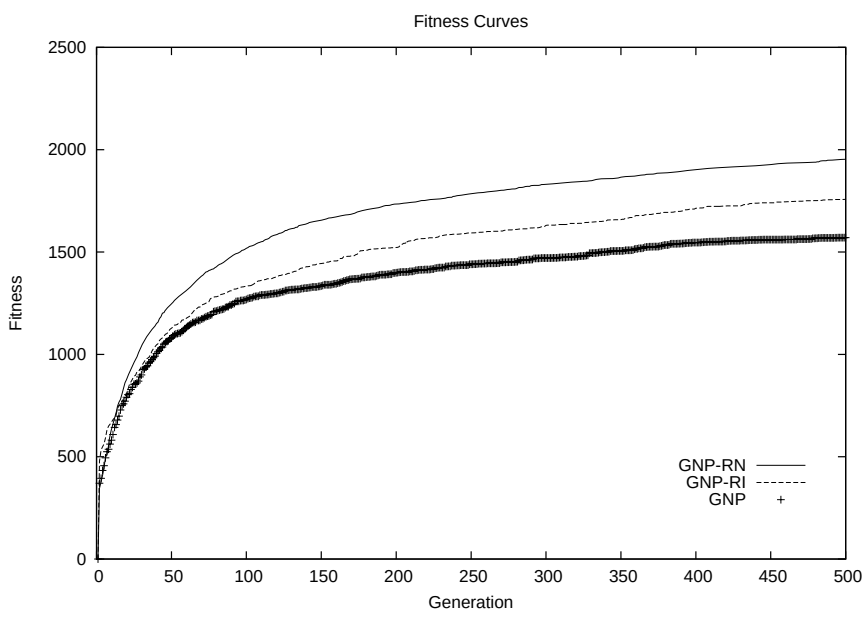


Figure 4.6: Averaged best fitness curves over 30 random rounds in simulation 2

4.4.3.3 Simulation 3

Although the main objective of this research is to study the search ability of GNP in the solution space, the generalization ability should be checked. To this end, after the training in simulation 1, we tested GNP-RN, GNP-RI and GNP in 9 different worlds to compare their performances in untrained environments. Figure 4.7 shows the experimental environments. World 17-20 have the same obstacle distribution to the ones in simulation 1, but tile and hole positions are different. World 21-25 have different obstacle, tile and hole distributions. So, the new testing environment is very difficult for agents to achieve the task. Table 4.2 shows the average of the testing results over 30 random rounds using the best trained individual in the cases of GNP-RN, GNP-RI and GNP. We can see that in new unexperienced environments, in most cases, the trained GNP-RN can obtain better testing results than GNP-RI and GNP-RI performed better than standard GNP in most of the environments. However, we can still see that GNP-RN, GNP-RI and GNP failed to perform well in the testing worlds. Although the agents are trained to fit the training environments, they can gain some general knowledge from the training phase. But, when experiencing a totally new environment, the

learned general knowledge is not enough for guiding the agents in the new environments. So, there should be more work to improve the testing results.

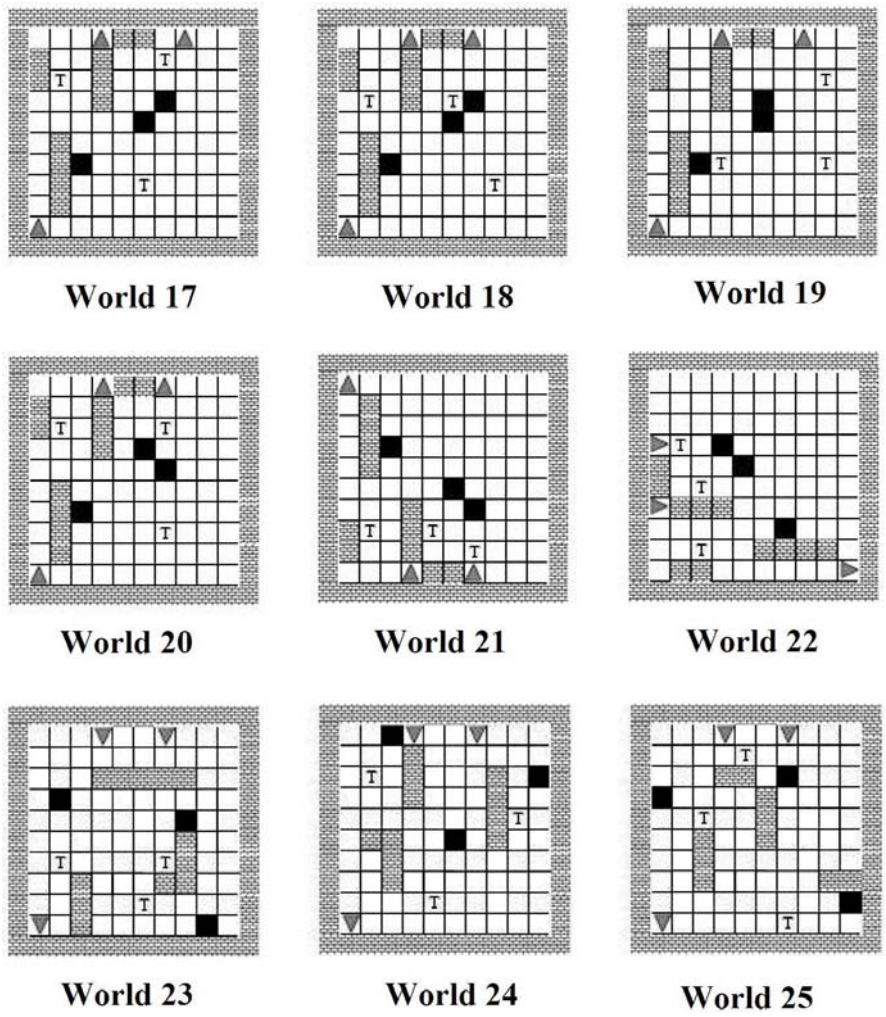


Figure 4.7: Testing environments in simulation 2

4.5 Summary

The proposed explicit memory scheme for GNP: GNP-RN is also inspired by the studies of the memory schemes which enhanced traditional EAs. It is a novel approach designed to solve the problems in dynamic environments effectively and efficiently.

Table 4.2: Testing results of GNP-RN, GNP-RI and GNP

No.	GNP-RN	GNP-RI	GNP
World 17	168.3	133.3	106.7
World 18	171.0	156.7	173.3
World 19	184.3	121.3	113.3
World 20	204.7	188.7	161.3
World 21	103.0	89.3	24.7
World 22	48.7	68.7	16.0
World 23	138.3	146.7	103.3
World 24	102.3	75.3	87.3
World 25	157.7	41.3	36.0
Average	142.0	113.5	91.3

The performance of GNP is enhanced by the proposed explicit memory scheme which stores the best solutions represented by the GNP routes. The agents can utilize the knowledge from the memory when dealing with the dynamic environments. The simulation results show that the proposed architecture can obtain better results than GNP with reconstructed individuals (GNP-RI) and conventional GNP in normal and complex environments demonstrating the effectiveness of the memory scheme. However, there are still some to be improved in the further research. Although the stored GNP routes in the memory contain some information of the environments, a better mechanism is still needed to bring much more generalization ability to GNP in different new environments.

Chapter 5

Adaptive Mutation in SARSA Learning of Genetic Network Programming

5.1 Introduction

We have introduced 3 memory schemes for GNP in the previous chapters: GNP with rules, GNP-RI and GNP-RN which are inspired by the research in traditional EA with explicit memory scheme. These three schemes focus on storing information of best solutions which are represented in the form of GNP routes. In GNP with rules, the memory stores the rules on the GNP routes and their importance values and reuses them to construct new individuals. In GNP-RI and GNP-RN, the memory stores the whole GNP routes of the best individuals, while they employ different mechanisms to reuse the stored informations.

In this chapter, a new architecture named adaptive mutation in SARSA learning of GNP (GNP-SLAM) is studied, which uses SARSA learning(52) to evaluate the branches of nodes and records the information of the evaluation during the evolution. According to the stored learning information on each branch of node, an adaptive mutation which determines the flexible and proper mutation rates for every branch and its mutation direction is adopted instead of the common uniform mutation with a fixed mutation rate and random mutation directions.

GNP-SLAM records the Q values measured by SARSA learning of each branch of node instead of the direct information of the best solutions and the information

affects on the mutation phase instead of changing the individuals. So, the GNP-SLAM can cooperate with the direct memory scheme, e.g., GNP-RI, to balance the degrees between exploitation and exploration. The performance of GNP-SLAM is evaluated in tile-world problems in the simulations.

5.2 Motivation

In GNP-RI, in every generation, before the genetic operators, i.e., crossover and mutation, are conducted, the GNP routes of the elite individuals with the best fitness values are extracted to reconstruct the worst individuals. And then, the rest part of the population is recombined by genetic operators. But, apparently, this approach, i.e., the worst individuals use the elites' gene information to imitate the better individuals, will result in the loss in balance between exploitation and exploration. One simple method to handle this consequence and improve the performance of GNP-RI is to raise the mutation rate in order to bring more population diversity during the evolution. However, it is very difficult to find such an appropriate mutation rate by setting the rate at a constant and reckless value. So, a more scientific and reasonable method is to seek a flexible mechanism to guide the adaptive mutation according to a certain quantum model during the evolution.

Back to the conventional GNP, the connections between nodes are traditionally treated uniformly by the genetic operations. Not only selection is conducted based on the fitness values of the individuals, but also crossover and mutation are performed at constant crossover rate and mutation rate, which means different branches have the same chance to change. It is the same case in GNP with rules that each used node branch is assigned an importance value which is equal to the fitness value. So, the branches in the same individual shares the same importance value. However, it has been noticed that even the high fitness individuals might possess some logically inappropriate branches, i.e., the nodes that they point to are incorrect, which might cause severe consequences. If the inappropriate branches are not used, they temporarily do not influence the performance of the individuals. However, in other circumstances where these branches are used, the performance could become undesirable, thus jeopardizing the generalization ability of the obtained solution. Besides, in the training

phase, there exists a possibility of these branches being passed generation to generation, which could make genetic weaknesses spread over the population.

To remedy this situation, in this research, we propose a SARSA learning model(52) to measure the utilities of different branches, i.e., the Q values. The general idea is that the branches and nodes are defined as states and actions as in reinforcement learning(53; 54), where different runs of the individuals are viewed as different trials. This way, the fitness values and observable rewards could be utilized to update the Q values of the branches that appear during the execution. As the evolution proceeds, the giant number of trials provides the learning model with a plenty of experiences and knowledge to approximate the true utilities of the branches. Meanwhile, the obtained Q values are applied to the mutation operation, herein we call it adaptive mutation, on the premise that the low Q value indicates a possibly inappropriate branch. The mutation rates of different branches are adjusted based on the Q values at the end of each generation, where the branch with low Q value will be mutated at the probability above the average. On the other hand, when a branch is being mutated, the node to which it potentially points is also decided by the probability model based on the Q values. As a result, the inappropriate branches have a larger chance to mutate to better ones, and genetic weaknesses could be gradually reduced, even partially eliminated as the evolution goes on.

5.3 Architecture of GNP-SLAM

5.3.1 Outline

Many research has been done in term of applying reinforcement in GP(55; 56). The states and actions in reinforcement learning can be represented by the nodes and the transitions in the tree structures of GP. The agents should make a decision to choose the next node to move. The consequent of the move will get rewards or punishments to update the Q values of agents. The case of GNP is very similar to that of GP. In GNP, since a branch is connected to the node where it comes from and the node to which it points, a Q value actually shows the evaluation about that a particular branch points to a particular node. In GNP-SLAM, the genetic operators will be conducted and specifically, the traditional uniform mutation is replaced with an adaptive mutation, where

the branches with low Q values have higher mutation rates. Moreover, when deciding the potential node for a mutated branch to connect, the corresponding Q values are also considered in a way that the branches with lower Q values are less likely to be connected to the potential node. The framework of GNP-SLAM is illustrated in Figure 5.1.

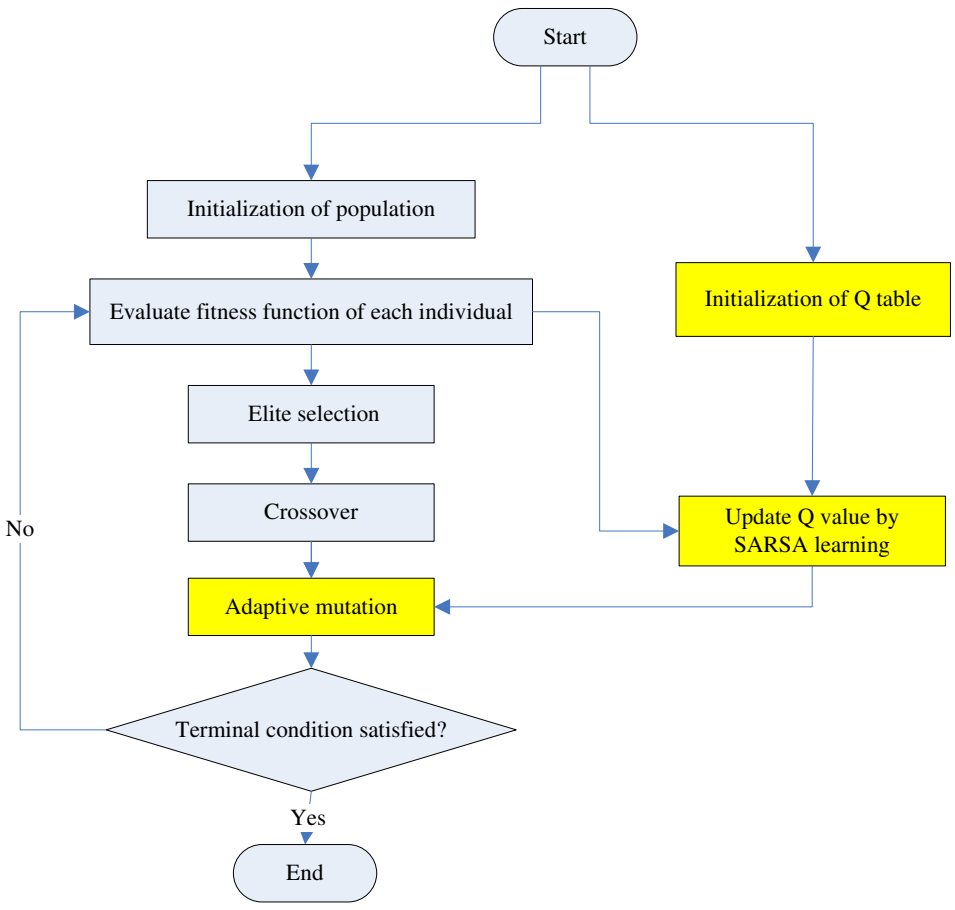


Figure 5.1: Framework of the proposed method

As a reinforcement learning approach, SARSA learning aims to learn a state-action policy in a Markov decision process. The reason we adopt reinforcement learning is that it is a kind of unsupervised learning technique, and GNP are mostly applied to unsupervised learning problems. Moreover, we adopt SARSA learning over Q learning because our main target is to locate the inappropriate branches. In another word, we

require the evaluation of the branches to be objective. As an off-policy approach, the value update function in Q learning always seeks the potentially highest rewarding actions instead of the true ones taken in a trial. Although it works well for finding the best rewarding policy, the greediness of Q learning will show the negative aspects for a branch in GNP. SARSA learning, in contrast, is an on-policy approach, and updates the Q values based on the true experience, by which we are able to learn the average utilities of the branches.

5.3.2 Definitions

For further explanation, some definitions are given as follows.

Trial: A trial refers to the process for an agent to execute a task being supervised by GNP. For instance, if GNP is supervising an agent in the maze problem, a trial is defined by the agent's behaviors from the moment when the agent enters the maze to the moment when it reaches the destination or when time is out. Note that if a GNP individual is used to supervise more than one agent or more than one maze, then we have multiple trials for one individual.

Route: A route refers to the sequence of nodes and branches occurring in a trial. It starts with the branch of the start node, and ends at the last node visited.

State: A state refers to a branch of a node. Since the number of the branches of each node is predefined, there is a fixed number of branches in total, i.e., the number of states is finite.

Action: An action refers to a node. The number of nodes in GNP is also predefined, so there is a finite number of actions.

Figure 5.2 shows an example of these definitions. In the phenotype representation of GNP, we use bold lines to mark the nodes and branches visited during a trial, then we record these information in the route, and finally turn it into a sequence of states and actions. In Figure 5.2, N_i denotes the ith node, B_i^j denotes the jth branch of the ith node. s_i^j is the state corresponding to B_i^j, and a_i is the action corresponding to N_i.

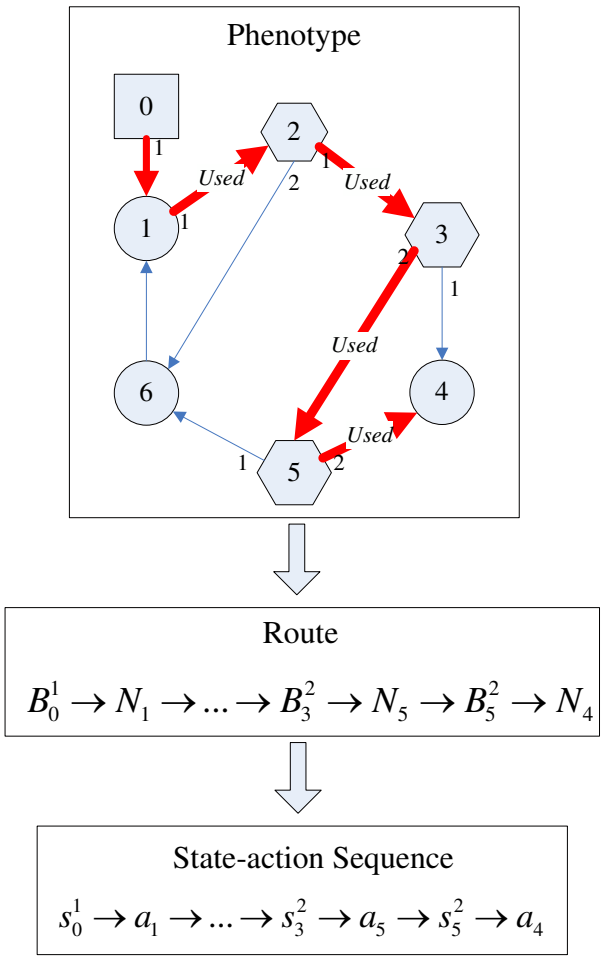


Figure 5.2: Route, State and Action

5.3.3 SARSA Learning Model

Based on the above definitions, a trial could be substituted by a route, and a route could be further represented by a sequence of states and actions as shown in Figure 5.2. This way, we build a bridge between the dynamic execution of GNP and its static structure, which makes it possible to utilize SARSA learning to study the structure of GNP. Note that we do not have to consider the action selection policy which takes a significant part in the conventional reinforcement learning, because the possible actions for each state are already generated by GNP, and the real action taken at each state could be obtained from the route simply.

The proposed learning approach mainly includes four steps, summarized as follows:

- 1. Establish a Q table that contains all the possible state-action pairs at the beginning of the evolution. Initialize all the Q values as 0.

- 2. After each trial, obtain a route, a score and some instant rewards. The score could be the fitness or a part of the fitness, which is application specific. The rewards are given to some actions for them to be encouraged or punished.

- 3. Use the score and rewards to update the Q value for each state-action pair in the route with the following update equation, following a backwards order.

$$Q(s,a) = Q(s,a) + \alpha \cdot (r + \gamma \cdot Q(s',a') - Q(s,a)), \tag{5.1}$$

  where, $Q(s,a)$ is the Q value of the current state-action pair, $Q(s',a')$ is the Q value of the next state-action pair. r is the reward if the current state is not the terminal state, otherwise would be the score assigned to this trial. α denotes the learning rate, while γ denotes the discount factor.

- 4. For different trials, repeat step 2 and step 3 to update the Q table iteratively until the end of the evolution.

Note that the SARSA learning approach described here is a little different from the traditional one. Since different trials of GNP may end at different states, the learning process is considered as reinforcement learning without an explicit terminal state(57), where the Q values has been proved to converge.

This approach works well for finding the inappropriate branches of the individuals. Still take the maze problem for example. Assume node N_1 judges the object in front of the agent, and its branches indicate the possible objects, among which branch e means obstacle, and also assume node N_2 encodes the function that makes the agent move forward. Therefore, if branch e of N_1 is connected to N_2, the control sequence could

be translated as $if\ facing\ an\ obstacle$, $then\ move\ forward$, which is apparently illegal and should fail every time it is executed. In the proposed SARSA learning approach, if we give a proper negative reward value as a punishment to such failure actions by updating the Q values iteratively during a great number of trials, the Q values of the aforementioned illegal actions have a large chance to be negative, or a very small positive value. Since the trials containing illegal actions are not able to get a high score, but a lot of punishments in most cases, we earn a confidence to think that the branches with low-Q-values are inappropriate.

5.3.4 Adaptive mutation

In GNP or other evolutionary algorithms, one of the ways to explore in the solution search space is by mutation. Traditionally, the mutation rate is a predefined constant value so that every genetic unit mutates at the same probability, which we herein call uniform mutation. There is no bias in uniform mutation, due to the fact that the only thing that concerns us is the fitness, i.e., we neither care about nor know the differences between the micro structures of an individual. After SARSA learning, however, we are indeed able to locate a number of branches which have extremely low Q values, so it is no longer necessary to do mutation uniformly. Instead, if we know that a branch is dangerously unreliable, we are supposed to decrease the possibility of it appearing in the gene strings, which is the reason we propose the adaptive mutation approach.

The basic idea is that we define a threshold T to determine whether a branch-node pair is viewed as normal or not in advance. If the Q value of the branch-node pair passes the threshold, we still adopt the predefined mutation rate to perform mutation, otherwise a monotonically decreasing function is utilized to calculate the probability for the corresponding branch-node pair to mutate. Of course, the newly calculated mutation rate is always higher than the predefined one. In addition, we also adjust the probabilities of the nodes the mutated branch will point to, according to the Q value based monotonically increasing function. These two steps ensure that the branches with lower Q values have less frequency of occurrence in the population. To be specific, the proposed adaptive mutation approach is summed up as follows:

- 1. Calculate the average Q value for each branch with the following equation:

$$Q_{avg}(s) = \frac{1}{n}\sum_{n=1}^{N} Q(s, a_n), \tag{5.2}$$

where, s is the current state, i.e., a branch. $Q_{avg}(s)$ is the average Q value over all the possible actions at state s. Actually, a_n is the action that has occurred at state s before, so if an action has not appeared so far at state s, its count is not considered. N is the total number of actions occurred at state s.

- 2. Multiply the average Q value by scalar t to obtain the threshold for the corresponding branch:

$$T(s) = t \cdot Q_{avg}(s), \tag{5.3}$$

where, $T(s)$ is the threshold for the given branch s, and t is a scalar.

- 3. Compare the current Q value of the branch-node pair, i.e., (s, a) with its threshold. If the Q value fails to pass the threshold, calculate the nonuniform mutation rate as follows:

$$P_{am}(s, a) = P_m + (1 - P_m) \cdot \sigma(Q(s, a)), \tag{5.4}$$

and,

$$\sigma(y) = \frac{1}{1 + e^{k \cdot y}}, \tag{5.5}$$

where, $P_{am}(s, a)$ is the adaptive mutation rate of branch s and action a, P_m is the predefined mutation rate and k is a possitive coefficient. $Q(s, a)$ is the Q value of the current state-action pair (s, a), i.e., the Q value of the current branch and node. $\sigma(y)$ is a logistic function which guarantees the decreasing monotonicity. The property of the logistic function perfectly satisfies our requirement: it shows a linear decrease when y is near 0, but becomes saturated when y is negatively or positively large enough. When the Q values are too small, their corresponding branches will be assigned to very high mutation rates anyway.

- 4. Perform mutation branch by branch. If a branch is considered as inappropriate, let it mutate at probability of $P_{am}(s,a)$, otherwise at probability P_m. If a branch is determined to to be mutated, calculate the probabilities of the nodes that the branch will potentially point to as follows:

$$P_{tb}(s,a) = \frac{\sigma(-Q(s,a))}{\sum_{a \in A(s)} \sigma(-Q(s,a))}, \tag{5.6}$$

  where, $P_{tb}(s,a)$ is the probability that node a is selected as the next node to connect at branch s. $A(s)$ is the set of all the possible actions for the current state s. Note that Eq. (6) also employs the logistic function for the aforementioned reasons.

In this step, not only the mutation rate is determined by the Q values as demonstrated in the above step, but also to which node the current branch is to be mutated is guided by the probabilistic model. The guiding mutation mechanism is somehow similar to the Estimation of Distribution Algorithm(EDA)(58) and Population Based Incremental Learning (PHIL)(59; 60), which analyzes the distribution and linkage information of each bit from sampling elite individuals and build a probabilistic model to generate new individuals. The proposed method in this reserach also utilizes the Q value information to build the probabilistic model and guides the mutation to generate new individuals.

The whole idea is inspired by the evolution of human race. The development of our intelligence relies not only on the natural selection, but also on the self-learning and self-enhancement through our entire life. Besides, we accumulate the knowledge by recording it in a variety of media in order to enlighten the future generations. In GNP-SLAM, the SARSA learning model mimics the self-learning and knowledge accumulation of human beings, and the adaptive mutation is a way to utilize the knowledge. We believe GNP combined with learning algorithms has a better chance to evolve towards the correct direction.

Usually, reinforcement learning incrementally change the program using the current information of state and reward, i.e., online learning. In (33), Mabu also combines

SARSA learning with GNP, but in a quite different form. In his method, each GNP node has several functions, but in standard GNP, each node has only one function. So, Mabu considers each node as a state, and the plural functions inside it are considered as the optional actions. The agent will select the function according to a ϵ-greedy policy. That is, the node function with the maximum Q value will be selected with the probability of 1-ϵ or a random one is selected with the probability of ϵ. So, we can see that Mabu's method uses the online learning ability of SARSA learning to immediately make decisions on selecting the next action during the agent task execution. However, the method proposed in this reserach uses SARSA learning in a different way. Here, the node structure is the same as that of standard GNP, i.e., each node has only one function. So, each branch of the node is consider as a state and each node is considered as an action. The SARSA learning maintains a Q value table and the table information is used in the mutation phase. The branch with a smaller Q value will have a higher mutation rate and the Q table also makes the branch to mutate to a node with a larger Q value more frequently. So, we can see that instead of using the reward information immediately, in this research, SARSA learning accumulates the information and make use of them in the mutation phase after the task execution.

Besides, SARSA learning is conducted during the task execution of agents and adaptive mutation is conducted at the genetic recombination stage. So GNP-SLAM can be easily cooperate with other direct memory scheme, e.g., GNP-RI. In that case, GNP-RI strengthens the exploitation ability in search and adaptive mutation also can adjust the mutation rate for each branch to balance the exploitation and exploration. Figure 5.3 illustrates the framework of adaptive mutation combining individual reconstruction. In each generation, after all the fitness values of all the individuals are calculated, the top $R\%$ of the best-performing ones are regarded as elites. All the GNP routes of the elites are used to reconstruct the worst $R\%$ of individuals. Then, the rest $(100 - R)\%$ individuals minus one elite individual are generated by genetic operators for the next generation.

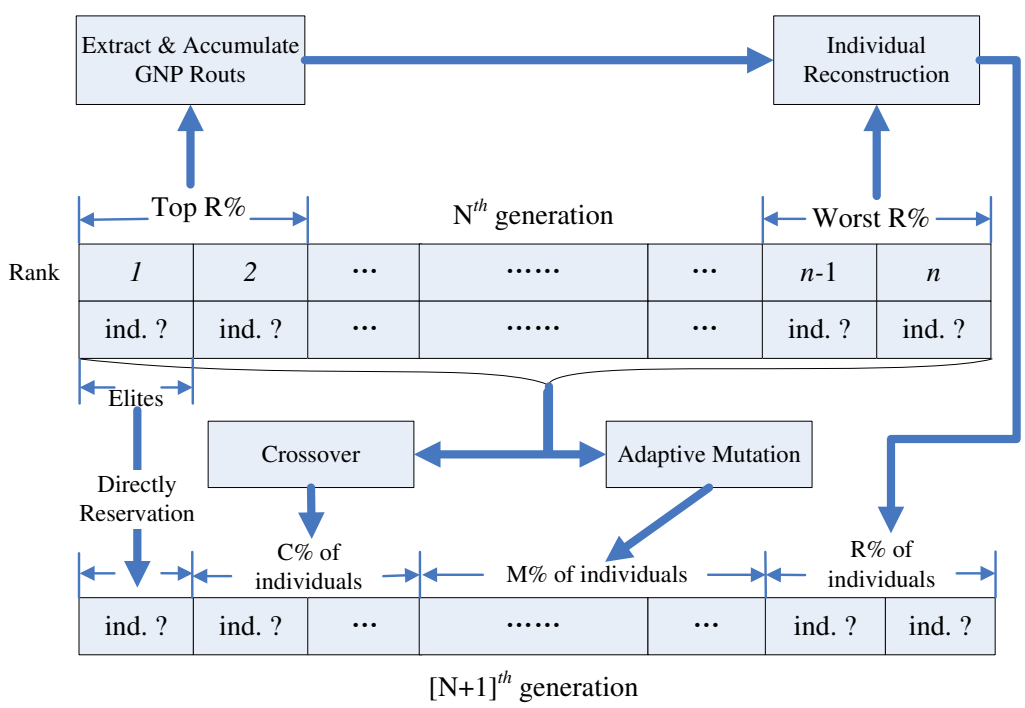


Figure 5.3: Framework of adaptive mutation combing individual reconstruction

5.4 Simulations

5.4.1 Experimental Environments

The simulations utilized the excellent benchmark for agent control problems: tile-world. To demonstrate the effectiveness of the proposed architecture, we conducted 4 simulations. In simulation 1, we make different experimental trials changing the learning rate α in Equation 5.1 and t parameter in Equation 5.3 for GNP-SLAM. The purpose of this simulation is to find the optimal configuration of α and t setting so as to use them in the later simulations. The experimental environments are 10 tile-worlds. Each world is a 2D space which contains 3 agents, 3 tiles and 3 holes. GNP-SLAM with different α and t settings are trained in the environments. In simulation 2, we trained the agents in the same environments as the ones in simulation 1 and compared the performances of adaptive mutation combining individuals reconstruction (GNP-RISLAM), GNP-RI, GNP-SLAM and GNP. The performance of GNP-SLAM is eval-

uated. Furthermore, by comparing the performance of GNP-RISLAM and GNP-RI shows whether the performance of GNP-RI is enhanced by employing adaptive mutation. The parameters α and t are configured as the optimal one obtained in simulation 1 and the individuals reconstruction size $R\%$ of GNP-RI is also set as the optimal one obtained in simulation 2 of Chapter 2. In simulation 3, we trained the agents in another 6 tile-worlds which are much more complicated for the agents and the performances of GNP-RISLAM, GNP-RI, GNP-SLAM and GNP are evaluated. In simulation 4, we tested the trained agents in simulation 2 using 9 different tile-worlds from the training.

As some research has demonstrated the significant superiority of GNP over some classical evolutionary algorithms such as GA, GP and EP (33), so in this simulation, we only compared the performance of GNP-RISLAM, GNP-RI, GNP-SLAM and standard GNP.

5.4.2 Programming Configuration

In our program, there are 8 kinds of Judgment Nodes: J-forward, J-left, J-right, J-backward, J-near-tile, J-near-hole, J-near-tile-to-hole and J-second-near-tile and 4 kinds of Processing Nodes: to go forward, to turn left, to turn right and to stay. Once the agent takes an action, it consumes one step. In our program, totally, there are 60 allowable steps. Each individual contains 60 nodes including 1 start node, 40 Judgement Nodes (5 for each kind of Judgement Nodes) and 20 Processing Nodes (5 for each kind of Processing Nodes). Each Judgement Node has 5 branches and each Processing Node has only one branch. The detail settings of node functions are given in Table 5.1.

We used the population of 201 individuals in the experiments for GNP-RISLAM, GNP-RI, GNP-SLAM and GNP. The crossover and mutation rate are predefined as P_c = 0.1 and P_m = 0.01 which are empirical settings for GNP. For SARSA learning phase, a negative reward r_1 = -10 is given to each failure action as a punishment. Also, two positive rewards r_2 = 7 and r_3 = 20 are given as reinforcement when the agent successfully pushes a tile forward, or drops it into a hole, respectively. The details of the specifications of parameter settings could be found in Table 5.2. And all simulations are carried out for 30 random rounds.

The fitness is calculated by accumulating the scores obtained from each tile-world. The score function is closely related to the objective of the tile-world problem, repre-

Table 5.1: Node Functions

Function ID	Description
J1	Identify the grid in front of the agent
J2	Identify the grid behind the agent
J3	Identify the left grid of the agent
J4	Identify the right grid of the agent
J5	Judge the direction of the closest tile
J6	Judge the direction of the second closest tile
J7	Judge the direction of the closest hole
J8	Judge the direction of the closest hole to the closest tile
P1	Move one grid forward
P2	Turn left
P3	Turn right
P4	Stay still

sented by

$$Score = 100 \cdot DT + 20 \cdot \sum_{p=1}^{P} d(p) + (M_t - U_t), \tag{5.7}$$

where, DT is the number of tiles dropped into the holes, p is the ID of the relatively nearest tile-hole pair at every time step in the trials, P is the maximum number of the relatively nearest tile-hole pairs, $d(p)$ is the decrease of the distances between the tiles and holes in the pairs, M_t is the maximum time step, and U_t is the used time step.

Then, the fitness function is defined by

$$Fitness = \sum_{w=1}^{W} Score(w), \tag{5.8}$$

where, w is the ID of the tile-world, W is the maximum number of the training tile-worlds, and $Score(w)$ is the score obtained in the wth tile-world.

5.4.3 Simulation Results

5.4.3.1 Simulation 1

In this simulation, we studied the best learning rate α for GNP-SLAM and the effects of different settings of parameter t.

Table 5.2: Parameter Configuration

Parameter	Value
Population Size	201
-GNP	
– Elite	1
– Crossover	80
– Mutation	120
-GNP-RI and GNP-RISLAM	
– Elite	1
– Reconstruction	$\mathrm{R}\% \cdot 201$
– Crossover	$(1-\mathrm{R}\%) \cdot 80$
– Mutation	$(1-\mathrm{R}\%) \cdot 120$
– $R\%$	0.15
Generation	500
Crossover Rate P_c	0.1
Mutation Rate P_m	0.01
Node	
- Judgement Node	40
- Processing Node	20
- Start Node	1
Discount Factor γ	0.8
t, k	0.2, 1

We use the same environments of simulation 1 in Chapter 4. Figure 4.3 illustrates the 10 tile-worlds of the simulation environments. The tile positions are different and the initial positions of the agents are the same. World 1-6 have the same distribution of obstacles and holes and World 7-10 have the same obstacle distribution, but the hole positions are different. So, the last 4 worlds are more complicated.

Figure 5.4 shows the average of the best fitness values of GNP-SLAM with different learning rates in the last generation. It seems that when α is set at 0.6, the best result is obtained. So, in simulation 2 and 3, α is set at 0.6 based on the experimental results of Simulation 1.

In GNP-SLAM, the parameter t is very important to determine the degree of exploitation and exploration. If t is large, the node branch will undergo the adaptive mutation with a higher probability. That means more exploitation ability is brought to

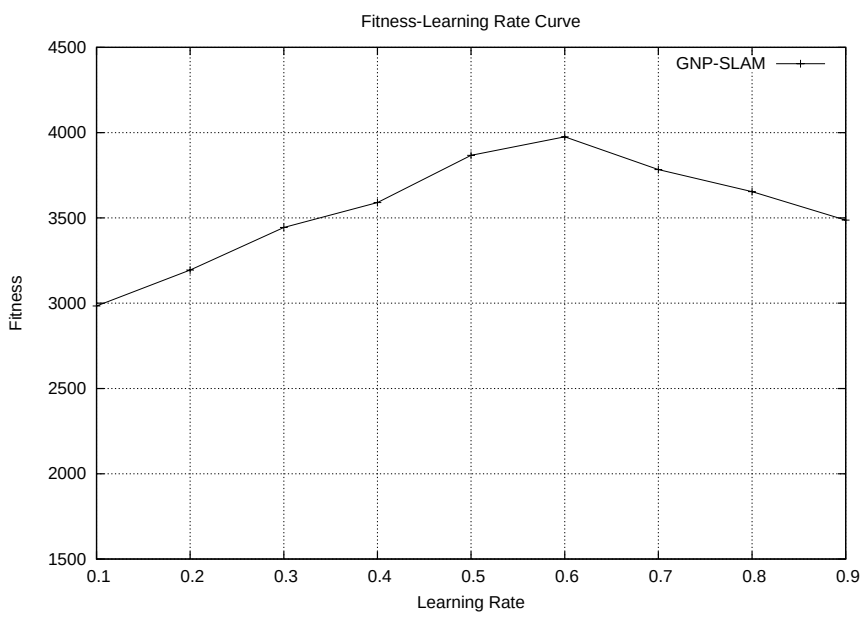


Figure 5.4: Fitness of GNP-SLAM using different learning rates

the program and vice versa. So, we studied the performance of GNP-SLAM with different settings of parameter t. When $t = 0$, the threshold will also be 0 leading that all the branches are mutated with the conventional mutation rate P_m and no adaptive mutation rate is used. If t is set at a very large number, the mutation of too many branches will be guided by the Q value information, which heavily strengthens the degree of exploitation. Figure 5.5 shows the average of the best fitness values of GNP-SLAM in the last generation with different t settings. And Figure 5.6 shows the average best fitness curves of GNP-SLAM with different t settings comparing with standard GNP. We can find that when t is set at around 0.2, the best result will be obtained. When t is set as a large value, e.g., 1.0, the performance of GNP-SLAM becomes even worse than that of standard GNP. Please notice that when $t = 0$, still a good result is obtained which means that only using Q value information to guide the mutation direction of node branches can bring significant improvement to GNP.

So, in simulation 2 and 3, $t = 0.2$ is used based on the above study.

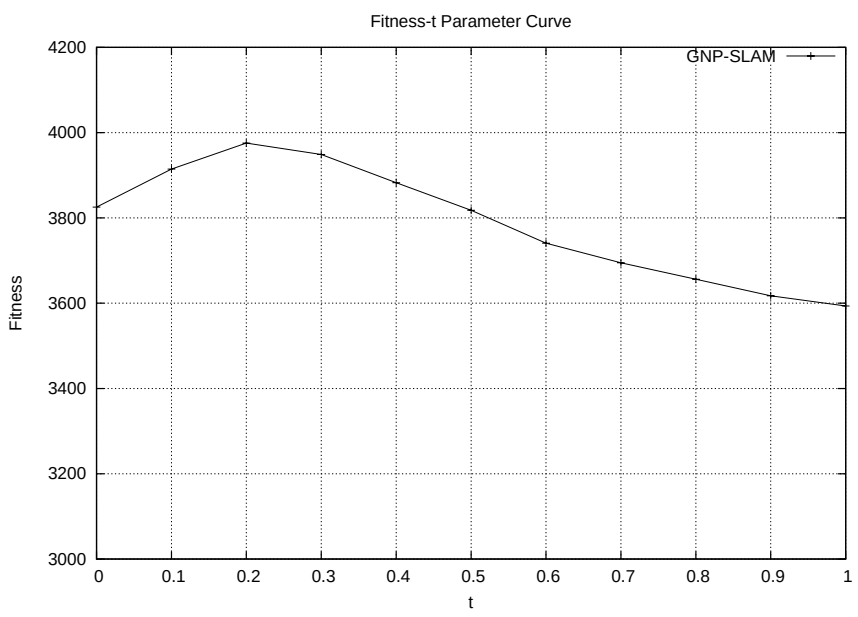


Figure 5.5: Fitness of GNP-SLAM with different t settings

5.4.3.2 Simulation 2

In this simulation, we trained agents in 10 tile-worlds. The experimental environments are the same as the ones in simulation 1 which are shown in Figure 4.3. Figure 5.7 shows the average best fitness curves over 30 random rounds in the training of GNP-RISLAM, GNP-RI, GNP-SLAM and GNP, which shows that GNP-RISLAM obtained the best results among 4 methods. Table 5.3 shows the p-values of t-test for the data shown in Figure 5.7. among the 4 methods. The test result shows that there are significant differences between GNP-RISLAM and standard GNP. However, GNP-RI and GNP-SLAM also perform better than standard GNP. The result suggests both GNP-RI and GNP-SLAM can enhance the architecture of GNP and the combination of these two approaches can make the performance of it even better.

An example of an elite individual from one of the early generations is given in Figure 5.8 to demonstrate the effectiveness of individual reconstruction. This example shows the part of the GNP route of the elite individual which control the agent to execute the task in World 1 in Figure 4.3. Please notice the node function indexes such as J_1, J_3, P_1, etc. as shown in Table 5.1, are used instead of the node numbers for

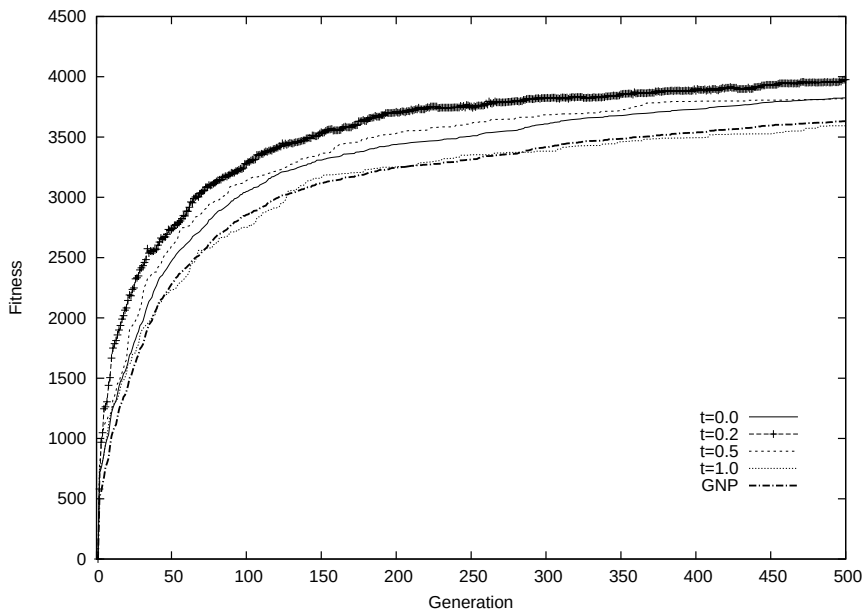


Figure 5.6: Average of the best fitness curves of GNP-SLAM with different t settings

Table 5.3: The t-test result for the average of the best fitness results over 30 random rounds in the last generation in simulation 2

t-test (p-value)	GNP-RISLAM	GNP-RI	GNP-SLAM
GNP-RI	5.70×10^{-5}	-	-
GNP-SLAM	2.34×10^{-5}	4.97×10^{-7}	-
GNP	3.68×10^{-7}	8.42×10^{-5}	7.10×10^{-6}

better understanding. We can see at first the agent makes several judgments to identify what is in front of it, at its left, at its right, etc. and according to the judgment results, it turns right which is described in the function of P_3. When moving in the world, the agent is looking for the tiles, for example, in J_5, the agent identify the direction of the closet tile and then move forward which is described in the function of P_1. In Figure 5.8, the solid branches come from the GNP routes used by individual reconstruction in the previous generation. So, we can see that individual reconstruction can bring worst individuals the experiences of the best individuals and make improvements in the fitness evaluation. The individual shown in Fig. 17 is very typical because it is one of the worst individuals (ranked 192) in the previous generation, but after being enhanced

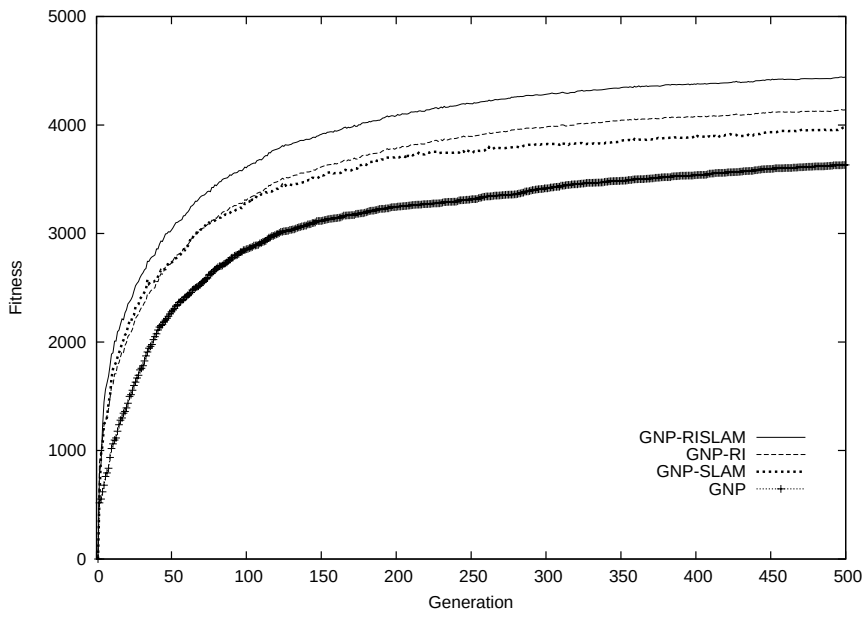


Figure 5.7: Average best fitness curves over 30 random rounds in simulation 2

by the proposed method, it becomes the elite individual in the next generation.

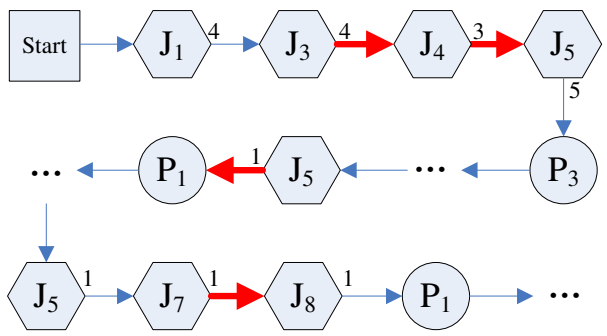


Figure 5.8: A typical example of an elite individual

Another example of the worst individual from one of the early generations is given in Figure 5.9 to show the details of adaptive mutation of the branches. In this individual, 6 branches were changed by the adaptive mutation. The solid branches show that the branches were changed to connect to other nodes with higher Q values. For example, the second branch of Node 7 which originally connected to Node 28 with the

Q value of 2.5 was adjusted to connect to Node 42 with the higher Q value of 24.7. And the mutation of other nodes was also guided by the Q information based mutation.

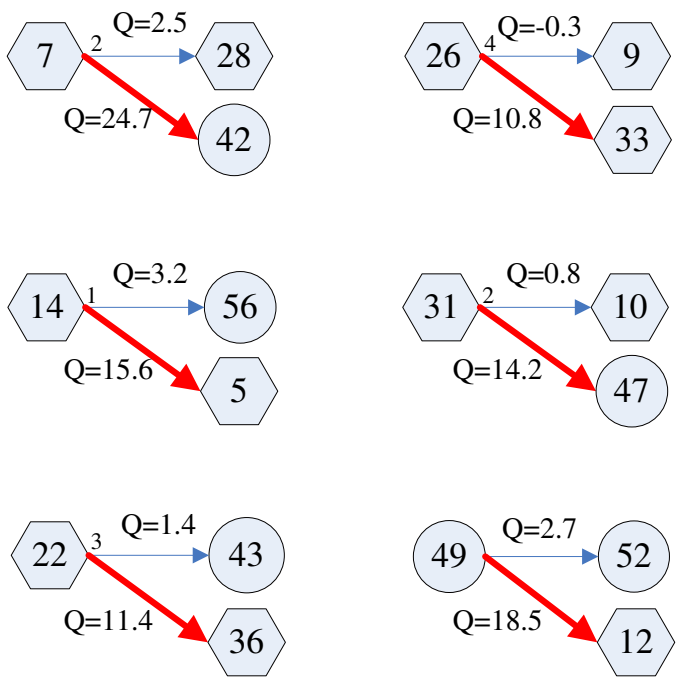


Figure 5.9: An example showing the mutation of a worst individual

5.4.3.3 Simulation 3

In this simulation, we trained GNP-RISLAM, GNP-RI, GNP-SLAM and GNP in another 6 tile-worlds which are the same environments as the ones in simulation 2 in Chapter 4 shown in Figure 4.5. We can see the distributions of obstacles, tiles and holes are different from each other which means the environments are more complex than the ones in simulation 2. So, it is difficult for agents to accomplish their tasks in all environments. Figure 5.10 shows the average best fitness curves over 30 random rounds in the training of the four architectures. The experimental result also shows that GNP-RISLAM obtained the best results among 4 methods. We can find that in more complicated environments, GNP-SLAM can perform better than GNP-RI which is different from Simulation 2. To discuss this interesting phenomenon, let's review the architectures of GNP-SLAM and GNP-RI. GNP-SLAM maintains a Q table and

utilizes it to update the mutation rate and guide the mutation direction. The Q table contains the Q value information of all the individuals during the whole evolution process. So, the evaluation criterion for the adaptive mutation of the branches and adaptive selection of the nodes are reasonable. On the other hand, GNP-RI utilizes the elites' experience to reconstruct the worst individuals which will significantly enhance the whole population. However, individual reconstruction causes the loss in the population diversity leading the worse performance of GNP-RI than GNP-SLAM in more complicated dynamic environments. Table 5.4 shows the p-values of t-test for the data shown in Fig 5.10. Comparing the 4 methods demonstrates that there are great differences between GNP-RISLAM and GNP.

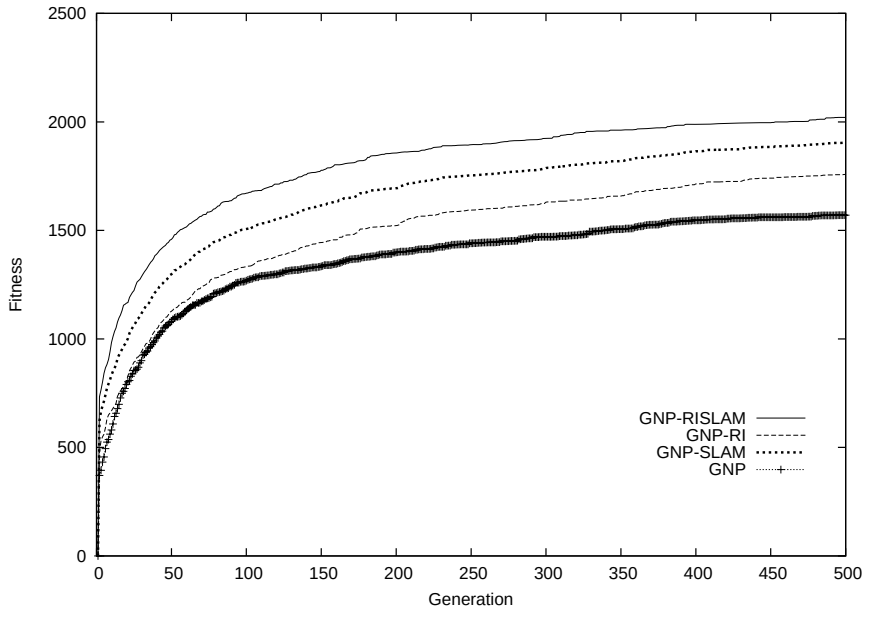


Figure 5.10: Average best fitness curves over 30 random rounds in simulation 3

5.4.3.4 Simulation 4

Although the main objective of this research is to strengthen the search ability of GNP in the solution space, the generalization ability should be checked. To this end, after the training in simulation 2, we tested GNP-RISLAM, GNP-RI, GNP-SLAM and GNP in 9 totally different worlds to compare their performances in untrained environments.

Table 5.4: The t-test result for the average of the best fitness results over 30 random rounds in the last generation in simulation 3

t-test (p-value)	GNP-RISLAM	GNP-SLAM	GNP-RI
GNP-SLAM	1.47×10^{-5}	-	-
GNP-RI	7.64×10^{-5}	4.20×10^{-3}	-
GNP	4.09×10^{-4}	2.51×10^{-5}	2.84×10^{-5}

The purpose of Simulation 2 and Simulation 3 is to check the performances of all the methods in normal and complex problems, respectively. And we found GNP-SLAM has more adaptiveness than GNP-RI in more complex problems. The worlds in Simulation 3 are very difficult for agents to execute tasks because the locations of obstacles, tiles and holes are totally different. So, we can see all these methods failed to solve the complex problems in Simulation 3 perfectly. This means that the agents trained in these environments didn't achieve the optimal solutions. That is why we used the training results in Simulation 2 instead of the ones in Simulation 3 for the testing phase. The experimental environments in this simulation are the same as the ones in simulation 3 of Chapter 4 shown in Figure 4.7. World 17-20 have the same obstacle distributions as the ones in simulation 1, but tile and hole positions are different. World 21-25 have different obstacle, tile and hole distributions. So, the new testing environment is very difficult for agents to deal with. Table 5.5 shows the average of the best testing results of the 4 architectures. We can see that in new unexperienced environments, the 4 methods failed to drop all the tiles in most cases. However, the trained GNP-RISLAM still performs an overall superior score comparing with the other 3 architectures. Furthermore GNP-SLAM and GNP-RI also can obtain better testing results than conventional GNP.

5.4.4 Analysis and Discussion

According the simulation results, we can confirm the following several points: First, it has been noticed that GNP-RI performs better than standard GNP because the worst individuals learn experiences from the best ones in GNP-RI, which is an exploitation of the knowledge obtained. Since the goal of GNP is to search for the global optimum in the solution space, appropriate exploitation means as if making a short cut to the

Table 5.5: Testing results of GNP-RISLAM, GNP-SLAM, GNP-RI and GNP with population of 201

No.	GNP-RISLAM	GNP-SLAM	GNP-RI	GNP
World 17	152.0	182.7	133.3	106.7
World 18	232.7	103.3	156.7	173.3
World 19	186.7	148.0	121.3	113.3
World 20	264.7	80.7	188.7	161.3
World 21	181.3	157.3	89.3	24.7
World 22	77.3	94.0	68.7	16.0
World 23	108.0	128.7	146.7	103.3
World 24	175.3	61.3	75.3	87.3
World 25	146.7	109.3	41.3	36.0
Average	169.4	118.4	113.5	91.3

destination, and eventually accelerates the search. Second, GNP-SLAM also performs better than GNP because it employs SARSA learning to evaluate the branches, then use the learned Q values to guide the mutation. It is well known that a significant role of mutation is to jump out of the local minimum, but the traditional uniform mutation is totally stochastic and unpredictable. With SARSA learning based mutations, however, the low-Q-value branches are assigned to higher mutation rates, so the inappropriate branches might probably appear less frequently than usual, which earns a better opportunity of avoiding local minimum. Third, as mentioned above, the behavior of worst individuals' learning from the best ones is a kind of exploitation of the social knowledge. While GNP-SLAM utilizes the past knowledge to guide the mutation. So, it is very natural to consider to combine GNP-RI and GNP-SLAM to employ both of their advantages. GNP-RISLAM maintains the mutation probability distribution based on Q values. Generally speaking, the smaller the Q value is, the less likely the corresponding node will be chosen. Therefore, it adds the bias into the candidate nodes instead of the uniform distribution in GNP and GNP-RI. Provided most of the Q values stand for the true utilities of the branches, the occurrence of the inappropriate ones will be gradually reduced generation by generation. Moreover, the probability distribution offers a more flexible way to determine the candidate nodes than GNP and GNP-RI. To sum up, GNP-SLAM is capable of gradually and smoothly reducing the genetic weakness through evolution and GNP-RI can be enhanced combining adaptive mutation.

5.5 Summary

This research introduces an approach to improve Genetic Network Programming (GNP) named adaptive mutation in SARSA learning of GNP(GNP). GNP-SLAM possesses two innovative features. Firstly, it integrates SARSA learning into GNPs evolutionary framework, thus making it possible to study the structure of GNP. Secondly, based on the learned knowledge, it replaces the traditional uniform mutation with adaptive mutation, which aims to gradually reduce the genetic weakness through evolution. The experiments conducted on the tile-world problem reveal several advantages of GNP-SALM. For one thing, GNP-SLAM is able to find a better solution in a limited training period compared with other GNP. For another, it manifests a better performance in the testing tile-worlds, which accounts for a relatively reliable adaptiveness and robustness. Furthermore, the adaptive mutation can easily combine with other explicit memory schemes and enhance the performance. The simulation results of adaptive mutation combing individual reconstruction have confirmed that point. Admittedly, we could see from the testing result that different memory schemes for GNP are still imperfect, and leave several issues to be concerned with as future work. The most important one is the insufficient inductive learning ability. Although GNP-SLAM works better than the conventional GNP in the test, it does not show desirable performances. Therefore, the inductive learning of GNP-SLAM still requires a complete and utter focus.

Chapter 6

Conclusions

In this research, some studies on the memory schemes for Genetic Network Programming are done and 4 memory schemes are designed in term of multiple objectives. Inspired by the research of other scholars in the evolutionary community on memory schemes for traditional evolutionary algorithms (EAs), the aim of the proposed memory schemes is to enhance the performance of GNP for dealing with dynamical problems and balance the exploitation and exploration degrees.

The general concepts of the proposed memory schemes are recording the historical informations of the population in the memory and reusing them later to guide the evolution process. Four different schemes are studied.

In the GNP with rules, the GNP rules and their importance values evaluated by individual fitness values are stored. The reuse of the memory is in the way that the worst individuals are replaced by the ones constructed by the selected GNP rules in each generation.

Another scheme, GNP with reconstructed individuals (GNP-RI) extends GNP with rules to a new form which stores the best solutions represented by GNP routes that indicate the successive regulations of judgements and actions for the agents. The gene structures of the worst individuals are modified by the stored GNP routes in each generation. So, the worst individuals can learn experiences from the elite ones during the evolution.

Then, GNP with route nodes (GNP-RN) further strengthens the learning ability of GNP-RI. In GNP-RN, each individual can learn knowledge from the memory. Furthermore, when the agents reuse the recorded regulations, they simultaneously consider

the current environment conditions and make decisions on whether or not adopting the regulations.

All the above three memory schemes strengthen the exploitation ability of GNP while the adaptive mutation in SARSA learning of GNP (GNP-SLAM) can balance the exploration and exploitation degrees of evolution. GNP-SLAM evaluates the branches of nodes with Q values updated by SARSA learning and uses the Q information to adjust the mutation rate flexibly and guide the mutation direction. GNP-SLAM can easily combine other GNP memory schemes to improve the algorithms.

The simulations compare the proposed architectures and the standard GNP. The results show the superiorities of these architectures over GNP. Furthermore, these memory schemes are compared with each other in different environments.

But there is still a lot of work to do to improve. From the simulation results on testing phases, it is found that GNP and its memory schemes mostly failed to solve the dynamical problems in new environments perfectly.

References

[1] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975. 1

[2] D. E. Goldberg, *Genetic algorithm in search optimization and machine learning*, Reading, MA: Addison-Wesley, 1989. 1

[3] J. R. Koza, *Genetic Programming, on the programming of computers by means of natural selection*, Cambridge, MA: MIT Press, 1992. 1

[4] J. R. Koza, *Genetic Programming II, Automatic Discovery of Reusable Programs*, Cambridge, MA: MIT Press, 1994. 1

[5] J. R. Koza, *Genetic Programming III, Darwinian Invention and Problem Solving*, Cambridge, MA: MIT Press, 1999. 1

[6] J. R. Koza, *Genetic Programming IV, Routine Human-Competitive Machine Intelligence*, Kluwer Academic Pub., 2003. 1

[7] L. J. Fogel, A. J. Owens and M. J. Walsh, *Artificial Intelligence through Simulated Evolution*, John Wiley, 1966. 1

[8] L. J. Fogel, *Intelligence through Simulated Evolution : Forty Years of Evolutionary Programming*, John Wiley, 1999. 1

[9] D. B. Fogel, ”An introduction to simulated evolutionary optimization”, *IEEE Trans. Neural Networks*, vol. 5, no.1, pp. 3-14, 1994. 1

[10] I. Rechenberg, *Evolutionsstrategie C Optimierung Technischer Systeme nach Prinzipien der biologischen Evolution* (PhD thesis), 1971. 1

[11] H. P. Schwefel, *Numerische Optimierung von Computer-Modellen* (PhD thesis), 1974. 1

[12] H. G. Beyer and H. P. Schwefel, "Evolution Strategies: A Comprehensive Introduction", *Journal Natural Computing*, Vol. 1, No. 1, pp. 3-52, 2002. 1

[13] M. Brameier and W. Banzhaf, "A Comparison of Linear Genetic Programming and Neural Networks in Medical Data Mining", *IEEE Trans. on Evolutionary Computation*, Vol.5, No.2, pp.17-26, 2000. 1

[14] J. F. Miller and P. Thomoson, "Cartesian Genetic Programming", in *Proc. of the 3rd European Conference on Genetic Programming*, pp.121-132, 2000. 1

[15] R. Poli, "Parallel Distributed Genetic Programming", Technical Report CSRP-96-15, School of Computer Science, The University of Birmingham, 1996. 1

[16] C. Ryan and J. J. Collins, "Grammatical Evolution: Evolving Programs for an Arbitrary Language", in *Proc. of the First European Workshop on Genetic Programming*, pp.83-95, 1998. 1

[17] C. Ferreira, "Gene Expression Programming: A New Adaptive Algorithm for Solving Problems", *Complex Systems*, Vol.13, No.2, pp.87-129, 2001. 1

[18] P. A. Whigham, "Inductive Bias and Genetic Programming", in *Proc. of the First International Conference on Genetic Algorithms on Engineering Systems: Innovations and Applications*, pp.83-95, 1998. 1, 2

[19] R. I. Mckay, "Fitness Sharing in Genetic Programming", in *Proc. of the Genetic and Evolutionary Computation Conference*, pp.435-442, 2000. 2

[20] J. Page, R. Poli and W. B. Langdon, "Smooth Uniform Crossover with Smooth Point Mutation in Genetic Programming: A Preliminary Study", in *Proc. of the Second European Workshop on Genetic Programming*, pp.39-48, 1999. 2

[21] P. Nordin, F. D. Francone and W. Banzhaf, "Explicitly Defined Introns and Destructive Crossover in Genetic Programming", *Advances in Genetic Programming II*, pp. 111-134, Cambridge, CA: MIT Press, 1996. 2

[22] H. Katagiri, K. Hirasawa and J. Hu, ”Genetic network programming: application to intelligent agents”, in *Proc. of IEEE International Conference on Systems, Man and Cybernetics*, pp. 3829-3834, 2000. 2

[23] S. Mabu, K. Hirasawa and J. Hu, ”Genetic network programming with learning and evolution for adapting to dynamical environments”, in *Proc. of the 2003 Congress on Evolutionary Computation*, pp.69-76, 2003. 2

[24] H. Katagiri, K. Hirasawa, J. Hu, J. Murata and M. Kosaka, ”Network Structure Oriented Evolutionary Model: Genetic Network Programming”, *Trans. of the Society of Instrument and Control Engineers*, Vol.38, No.5, pp.485-494, 2002. 2

[25] W. B. Langdon, ”Quadratic Bloat in Genetic Programming”, in *Proc. of the Genetic and Evolutionary Computation Conference*, pp.451-458, 2000.

[26] H. Katagiri, K. Hirasawa, J. Hu and J. Murata, ”Network Structure Oriented Evolutionary Model: Genetic Network Programming and Its Comparison with Genetic Programming”, in *2001 Genetic and Evolutionary Computation Conference, Late Breaking Papers*, pp. 219-226, 2001. 2

[27] K. Hirasawa, M. Okubo, H. Katagiri, J. Hu and J. Murata, ”Comparison between genetic network programming (GNP) and genetic programming (GP)”, in Proc. of *Genetic and Evolutionary Computation Conference*, pp.1276-1282, 2001. 2

[28] H. Katagari, K. Hirasawa, J. Hu and J. Murata, ”Genetic Network Programming and Its Application to the Multiagent Systems”, *Trans. IEE Japan*, Vol. 122-C, No. 12, pp. 214-2156, 2002. 2

[29] Y. Chen, S. Mabu, K. Shimada and K. Hirasawa, ”Trading Rules on Stock Markets Using Genetic Network Programming with Sarsa Learning”, *Journal of Advanced Computational Intelligence and Intelligent Informatics*, Vol.12, No.5, pp.467-478, 2008. 2

[30] H. Y. Zhou, W. Wei, K. Shimada, S. Mabu and K. Hirasawa, ”Time Related Association Rules Mining with Attributes Accumulation Mechanism and its Application to Traffic Prediction”, *Journal of Advanced Computational Intelligence and Intelligent Informatics*, Vol.12, No.4, pp.383-392, 2008. 2

[31] C. Yue, S. Mabu, Y. Wang and K. Hirasawa, ”Multiple Round English Auction Agent based on Genetic Network Programming”, *IEEJ Trans. on Electrical and Electronic Engineering*, Vol.5, No.4, pp.450-458, 2010. 2

[32] K. Hirasawa, T. Eguchi, J. Zhou, L. Yu and S. Markon, ”A Double-Deck Elevator Group Supervisory Control System using Genetic Network Programming”, *IEEE Trans. on System, Man and Cybernetics, Part C*, Vol.38, No.4, pp.535-550, 2008. 2

[33] 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, No.3, pp.369-398, MIT press, 2007. 3, 40, 58, 61

[34] S. Mabu, K. Hirasawa, and J. Hu, ”Genetic network programming with learning and evolution for adapting to dynamical environments”, in *Proc. of 2003 Congress on Evolutionary Computation*, pp. 69-76, 2003. 3

[35] S. Mabu, K. Hirasawa, J. Hu and J. Murata, ”Online Learning of Genetic Network Programming and its Application to Prisoner’s Dilemma Game”, *Trans. of IEE Japan*, Vol. 123-C, no. 3, pp. 535-543, 2003. 3

[36] S. Mabu, K. Hirasawa and J. Hu, ”Genetic network programming with Reinforcement Learning and its Performance Evaluation”, in *Proc. Part II of 2004 Genetic and Evolutionary Computation*, pp. 710-711, 2004. 3

[37] M. E. Pollack and M. Ringuette: ”Introducing the tile-world: experimentally evaluating agent architectures”, in *Proc. of the conference of the American Association for Artificial Intelligence*, pp. 183-189, AAAI Press, 1990. 3, 14

[38] N. Mori, H. Kita and Y. Nishikawa, ”Adaptation to changing environments by means of the memory based thermodynamical genetic algorithm”, in *Proc. of the 7th International Conference on Genetic Algorithms*, pp. 299-306, 1997. 3

[39] J. Branke, ”Memory enhanced evolutionary algorithms for changing optimization problems”, in *Proc. of the 1999 Congress on Evolutionary Computation*, pp. 1875–1882, 1999. 3

[40] A. Simoes and E. Costa, ”An immune system-based genetic algorithm to deal with dynamic environments: Diversity and memory”, in *Proc. of the 6th International Conference on Neural Networks and Genetic Algorithms*, pp. 168-174, 2003. 3

[41] S. Yang, ”Memory-based immigrants for genetic algorithms in dynamic environments”, in *Proc. of the 2005 Genetic Evolutinary Computation Conference*, pp. 1115-1122, 2005. 3

[42] D. E. Goldberg and R. E. Smith, ”Nonstationary function optimization using genetic algorithms with dominance and diploidy”, in *Proc. of the 2nd International Conference on Genetic Algorithms*, pp. 59-68, 1987. 4

[43] K. P. Ng and K. C. Wong, ”A new diploid scheme and dominance change mechanism for non-stationary function optimisation”, in *Proc. of the 6th International Conference on Genetic Algorithms*, pp. 159-166, 1995. 4

[44] J. Lewis, E. Hart and G. Ritchie, ”A comparison of dominance mechanisms and simple mutation on non-stationary problems”, in *Proc. of the 4th International Conference on Parallel Problem Solving From Nature*, pp. 139-48, 1998. 4

[45] D. Dasgupta and D. McGregor, ”Nonstationary function optimization using the structured genetic algorithm”, in *Proc. of the 2nd International Conference on Parallel Problem Solving From Nature*, pp. 145-154, 1992. 4

[46] S. J. Louis and Z. Xu, ”Genetic algorithms for open shop scheduling and rescheduling”, in *Proc. of the 11th ISCA International Conference on Computation and Their Application*, pp. 99-102, 1996. 5

[47] C. L. Ramsey and J. J. Grefenstette, ”Case-based initialization of genetic algorithms”, in *Proc. of the 5th International Conference on Genetic Algorithms*, pp. 84-91, 1993. 5

[48] R. Eberchart and J. Kennedy, ”A new optimizer using particle swarm theory”, in *Proc. Int. Sym. Micro Machine and Human Science*, pp. 39-43, 1995. 23

[49] J. Kennedy and R. Eberhart, "Particle swarm optimization", in *Proc. of the IEEE International Conference on Neural Networks*, pp. 1942-1948, 1995. 23

[50] C. Juang, "A Hybrid of Genetic Algorithm and Particle Swarm Optimization for Recurrent Network Design", *IEEE Transaction on Systems, Man, and Cybernetics, Part B*, Vol. 34, No. 2, pp. 997-1006, 2004. 24

[51] F. Grimaccia, M. Mussetta and R. E. Zich, "Genetical Swarm Optimization: Self-Adaptive Hybrid Evolutionary Algorithm for Electromagnetics", *IEEE Transactions on Antennas and Propagation, Part 1*, Vol. 55, No. 3, pp. 781-785, 2007. 24

[52] G. A. Rummery and M. Niranjan, *On-Line Q-Learning Using Connectionist Systems*, Technical Note, 1994. 49, 51

[53] R. S. Sutton, "Learning to predict by the method of temporal differences", *Machine Learning*, Springer, 1988. 51

[54] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Cambridge, MA: MIT Press, 1998. 51

[55] H. Iba, "Multi-agent reinforcement learning with genetic programming", in *Proc. of the Third Annual Conference of Genetic Programming*, pp. 162-172, 1998. 51

[56] H. Iba, "Evolutionary learning of communicating agents", *Information Science (USA)*, Vol. 108, No. 1-4, pp. 1811-205, 1998. 51

[57] M. Riedmiller, "Reinforcement Learning Without an Explicit Terminal State", in *Proc. of the Internationl Joint Conference on Neural Networks*, pp.1998–2003, 1998. 55

[58] P. Larranaga and J. Lozano, *Estimation of distribution algorithms: A new tool for evolutionary computation*, Kluwer Academic Publishers, 2002. 58

[59] S. Baluja, "Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning", Technical Report CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, PA, 1994. 58

[60] S. Baluja and R. Caruana, "Removing the genetics from the standard genetic algorithm", in *Proc. 12th International Conference on Machine Learning*, pp. 38-46, 1995. 58

Appendix A

Genetic Network Programming

Genetic Network Programming (GNP) is an extended approach of GA and GP to deal with dynamical environments efficiently. GA with a string structure is able to search the global optimal solutions of the problems, which is mainly applied to numerical optimization problems. So GA usually fails to solve complicated problem in dynamical environments where the the agents should handle varying situations. On the other hand, GP proposed later expands the expression ability of GA by using tree structures. This structural change of solutions brought significant progresses and made GP applicable to more fields and problems. But, it is generally said that GP is sometimes difficult to search for an optimal solution because the searching space of solutions becomes tremendous due to the difficulty to control the size of the tree. When the problem is complex, the tree size may bloat excessively. Genetic Network Programming (GNP) with a directed graph structure, is proposed to overcome the disadvantages of GP. The graph based structure of GNP has more general representation ability than that of trees, and the inherently equipped functions in it. Each node of GNP executes the judgment or processing operation for the agents, and the transition rule of GNP expresses the behavior sequences by transiting those nodes. GNP aims to be more applicable to many problems by separating the judgment nodes and processing nodes structurally so that the network can be easily evolved. Hence, it is the unique directed graph structure that brings GNP several advantages over GA and GP when dealing with problems in dynamical environments.

The gene of GNP is directed graph structure where several nodes are connected by directed branches like Parallel Algorithm Discovery and Orchestration (PADO) and Evolutionary Programming (EP). The fundamental differences between GNP and PADO or EP are as follows. PADO is originally designed to construct the static programs as GP, which can be seen from the fact that PADO has external memories. In PADO, the process boots from the initial boot node and terminates at the terminal node using the explicit indexed memory. So after the processing, it must return to the initial boot node again and update the indexed memory. EP is a Markov Decision Process, therefore, every input and output for all states have to be prepared in the structure of EP, so EP is likely to expand its size for complicated problems.

On the other hand, GNP boots from the start node and never returns to it, during the execution a series of node transitions generate the solutions of GNP. Therefore, these node transitions act like an implicit memory function in GNP. In other words, GNP is a new evolutionary method to construct generalized discrete event systems by combining program modules. GNP aims to be more applicable to many problems by separating the judgment nodes and processing nodes structurally so that the network can be easily evolved. Hence it is the unique directed graph structure that brings GNP several advantages over GA and GP when dealing with problems in dynamic environments.

A.1 Directed Graph Structure of GNP

An individual of GNP contains a fixed number of nodes which are classified into 2 categories in terms of their functions. These nodes are named Judgment Node and Processing Node, respectively. The numbers of 2 kinds of nodes are both fixed. Judgment Node judges the current state on the environments, and according to the judgement result, the agent selects the following node. In the concrete, Judgment Node has multiple branches connecting to different nodes, respectively, and after judgement, a decision should be made to select one branch and move to the next node. Accordingly, if there are a lot of judgement results, the number of branches increase, the network structure become complicated. Processing Node takes some actions (or implements some functions) and changes the current state according to some regulations. Different processing nodes take different actions. Therefore, Processing Node has only one

branch connecting to the following node. If there are a lot of different kinds of actions, the number of processing nodes also increase, and the network structure become complicated. The number of each kinds of nodes, concrete functions of nodes, states and regulations are defined by GNP designer and stored in its own dictionary. Figure A.1 shows a simple example of an conventional GNP's individual. We can see that in the example, the GNP individual has totally 5 nodes including 1 start node, 2 judgement nodes and 2 processing nodes. And each judgement node has 2 branches connecting to 2 nodes, respectively.

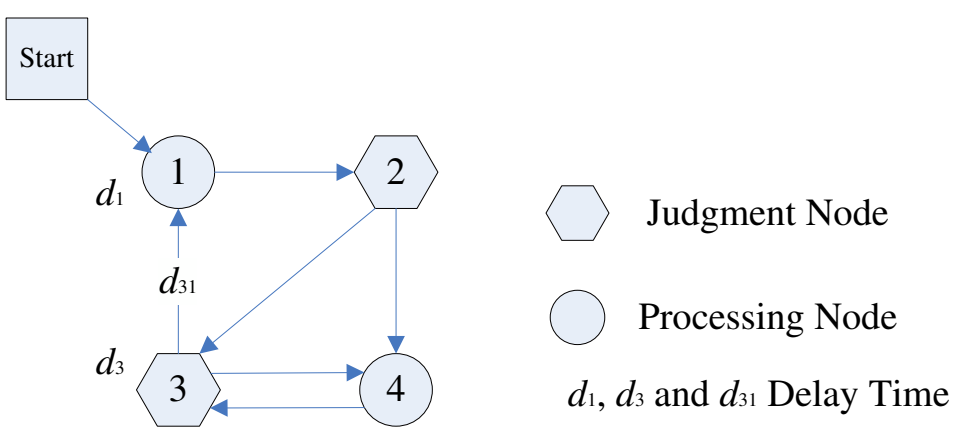


Figure A.1: The directed graph structure of GNP

Figure A.2 shows the genotype expression of the conventional GNP which provides the chromosomes encoded into bit-strings. In the conventional GNP, usually, each individual contains a start node, a set of m Judgment Nodes and n Processing Nodes. An matrix is used to express the directed graph structure of GNP. Each row of the matrix represents the information of a node. The first column NT_i in the node part represents the node type $\{0 : Start\ Node, 1 : Judgment\ Node, 2 : Processing\ Node\}$. For example, if NT_i = 1, it means Node i is a Judgment Node. The second column ID_i represents the node functions like judgment functions and processing functions. The third column d_i stores the delay time on the node. On the other hand, the connection part of the matrix represents the connected nodes from node i: C_{i1}, C_{i2}, ... and its connection delay time: d_{i1}, d_{i2}, ..., respectively. GNP has two kinds of time delays: one spends on judgment node or processing node, and the other one spends on node

transition. In Figure 2.1 d_1 and d_3 is the required time of executing the process and judgement of node 1 and node 3, respectively. And d_{31} is the one spent on the transition from node 3 via its 1st branch. They have been introduced to model GNP like human brain needs time for thinking. The structure of GNP can become more realistic by setting two kinds of time delays. Since GNP has a directed network structure, it is very likely that loops are formed in GNP individuals. Delay time is accumulated during the transit from node to node. Once the accumulated delay time exceeds a certain criterion, it could be considered as an infinite circle. Hence, the delay time could be used as a controller to prevent the agents from being trapped in the infinite circle.

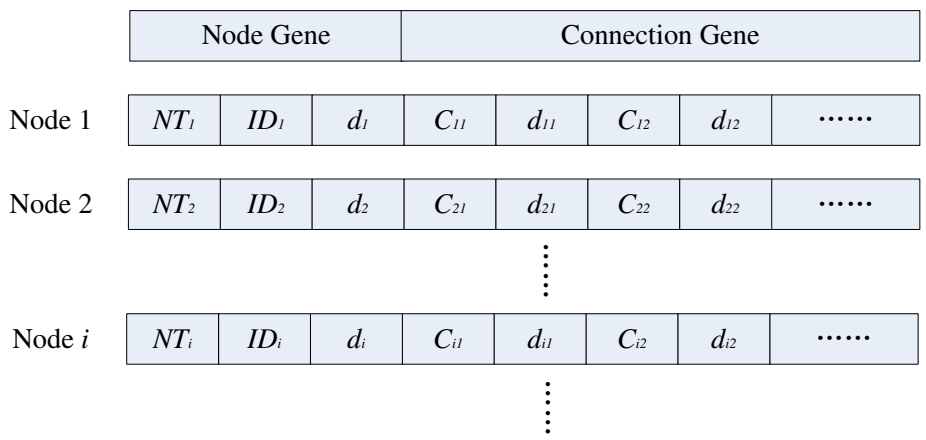
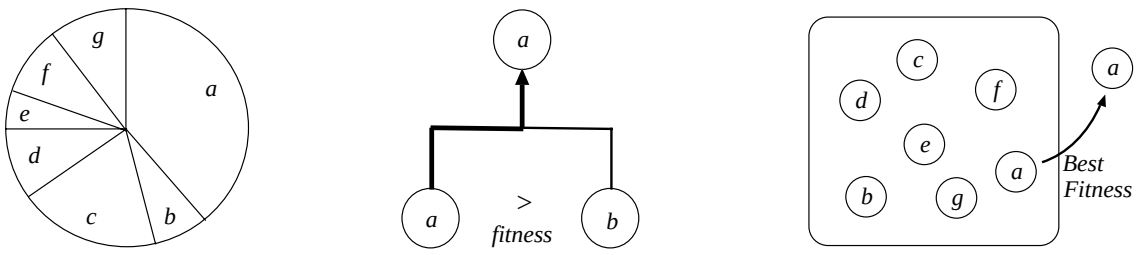


Figure A.2: The genotype expression of GNP

Once GNP is booted up, the execution starts from the start node, then the next node to be executed is determined according to the connection from the current activated node. If the activated node is judgement node, the next node is determined by the judgement results. When processing node is executed, the next node is uniquely determined by the single connection from Processing nodes.

A.2 Genetic Operators of GNP

Genetic variation is a necessity for the process of evolution. Genetic operators are the process to maintain genetic diversity, and they are found in the natural world. GNP



(a) Roulette Selection (b) Tournament Selection (c) Elite Selection

Figure A.3: Different kinds of selections of GNP

also have its own genetic operators which are similar to that of GA: selection, mutation and crossover. Selection is to select an individual randomly or according to a certain probability policy. Mutation is to change the gene of the selected individual. And crossover is to exchange the corresponding part of 2 selected parent individuals and obtain 2 offspring having new genes.

A.2.1 Selection

Figure A.3 shows examples of different kinds of selections in GNP.

As the roulette selection of GNP, the selection is carried out by the probability in proportion to the relative value of the fitness of the individual. Individuals with higher fitness have higher probabilities to be selected and vice versa. As a result, the roulette selection has not been used in conventional research because it could not be applied to the case where lower fitness is dominant.

In tournament selection, after comparing with N individuals selected from the population randomly, the individual having the highest fitness is selected among them. N is the tournament size and $N = 2$ is generally used. The tournament selection is mainly used in conventional GNP research, because the tournament selection is available in the case where lower fitness is dominant.

After comparing with the fitness of all individuals of the population, elite selection moves M individuals having higher fitness to the next generation. M is the number of

elite individual, and the solution converges quickly if M is high. In this thesis, elite selection is used and M=1.

A.2.2 Mutation

Mutation operator affects only one individual. All the gene information of each node are changed randomly by mutation rate of P_m, and one offspring is generated. There are 2 kinds of mutations: mutation of connections and mutation of nodes.

In mutation of connections, the connection between nodes is modified. The mutation refers to the change of genetic information C_{ij} of each node in GNP by the predefined probability P_{mc}, and decides the connection for the mutation. Therefore the genetic information C_{ij} is modified in the range of the node number randomly. The example of the mutation is shown in Figure A.4. In the case of just carrying out this mutation, the kinds of nodes having the same node number are not modified because NT_i and ID_i are not modified.

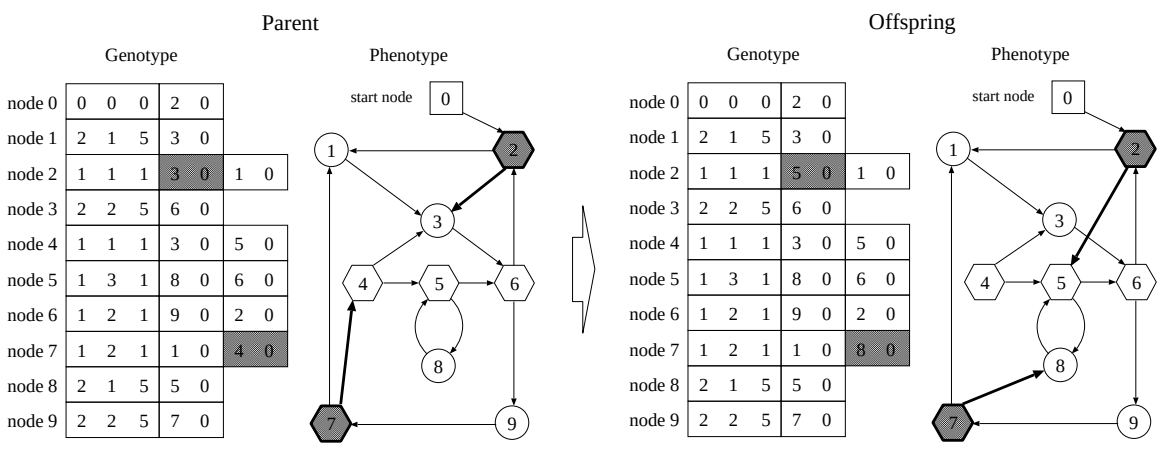


Figure A.4: Mutation of connections

In mutation of nodes, the kinds of node are changed. The mutation refers to the change of genetic information NT_i and ID_i of each node in GNP by the predefined probability P_{mn}, and decides the node and the genetic information for the mutation. Therefore, the genetic information on the NT_i and ID_i is modified randomly. If the

number of the connections are increased by modifying the node function (e.g., modifying the $Processing\ node$ to the $Judgement\ node$), added connections are defined randomly. On the other hand, if the number of the connections are decreased, they are deleted. The example of the mutation is shown in Figure A.5.

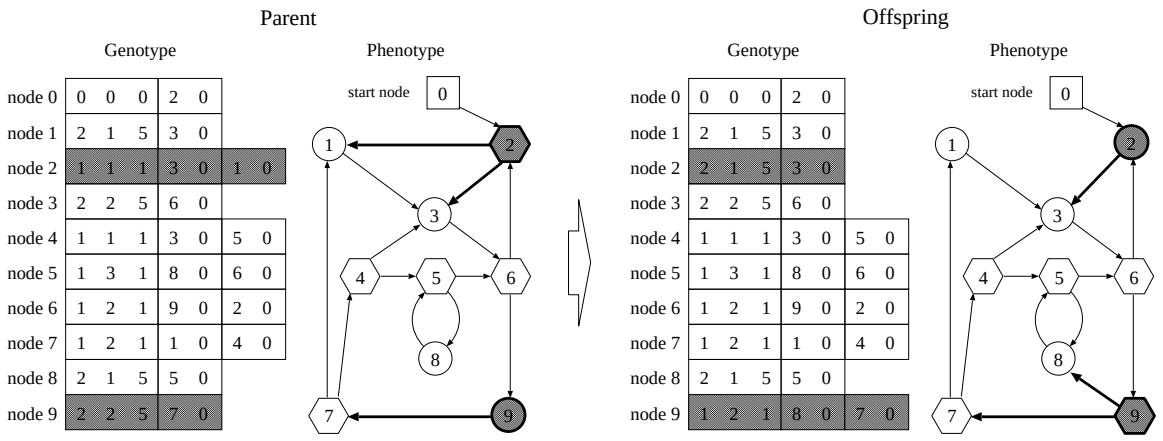


Figure A.5: Mutation of nodes

We can combine these 2 kinds of mutations or just use one of them in some problems. In this thesis, we use mutation of connections only.

A.2.3 Crossover

Crossover is executed between two parents and produces two offspring. The connections of the uniformly selected corresponding nodes in two parents are swapped with each other by crossover rate of P_c, and two offspring are generated. In GNP, there are 3 kinds of crossover: one point crossover, several points crossover and uniform crossover.

One point crossover selects one node as the crossover point randomly, the whole genetic information separated by its node is exchanged. The example of one point crossover is shown in Figure A.6 where the performance of the generated offspring individual is influenced by the position of the crossover point.

Several points crossover selects several nodes (generally two) as the crossover point randomly, the whole genetic information separated by their nodes is exchanged. The

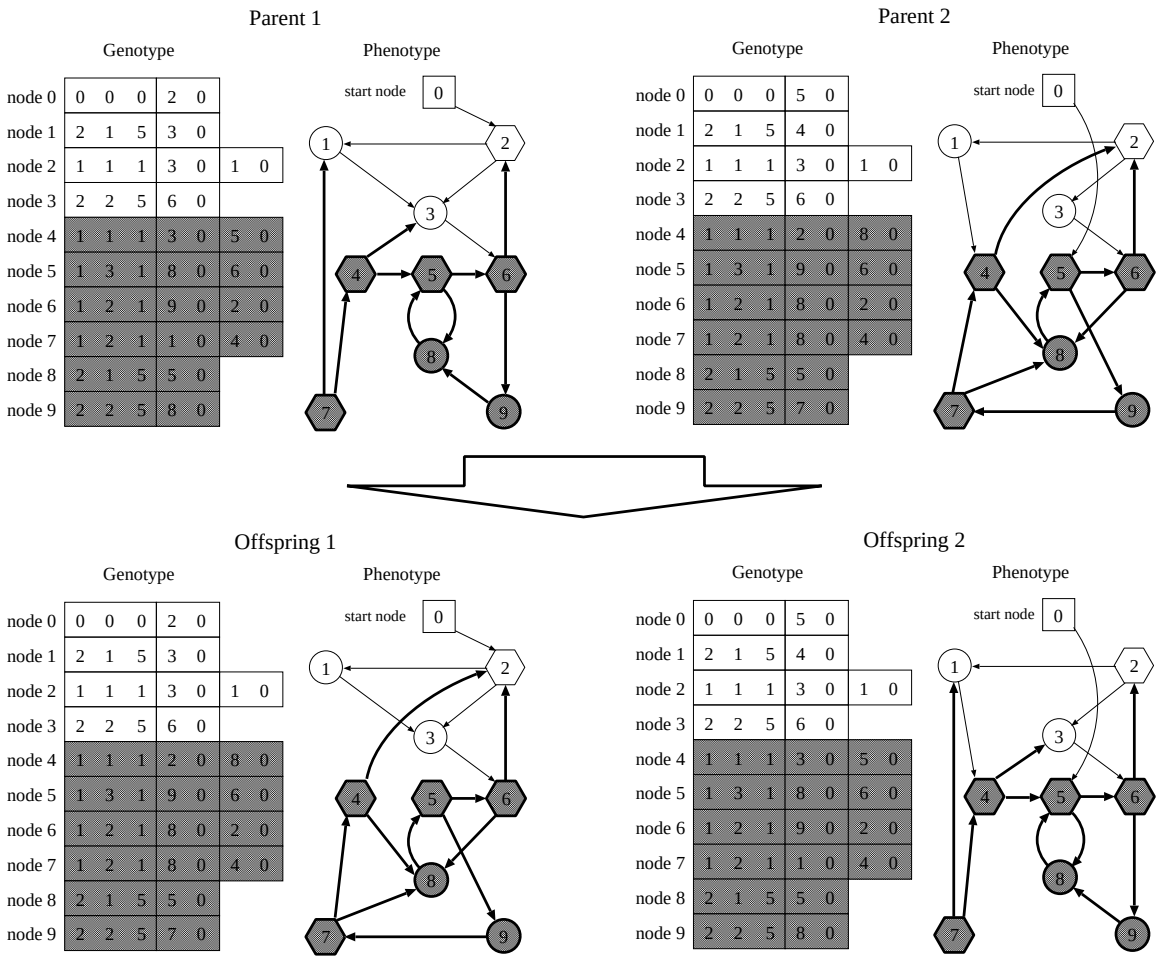


Figure A.6: One point crossover

example of several points crossover is shown in Figure A.7. Exchanging more small sub networks is available by dividing the network into more blocks.

In uniform crossover, the crossover nodes in the offspring is decided by the predefined crossover probability P_c for each node of the parent individual, the whole genetic information corresponding to the crossover node is exchanged between the parents. The example of uniform crossover is shown in Figure A.8.

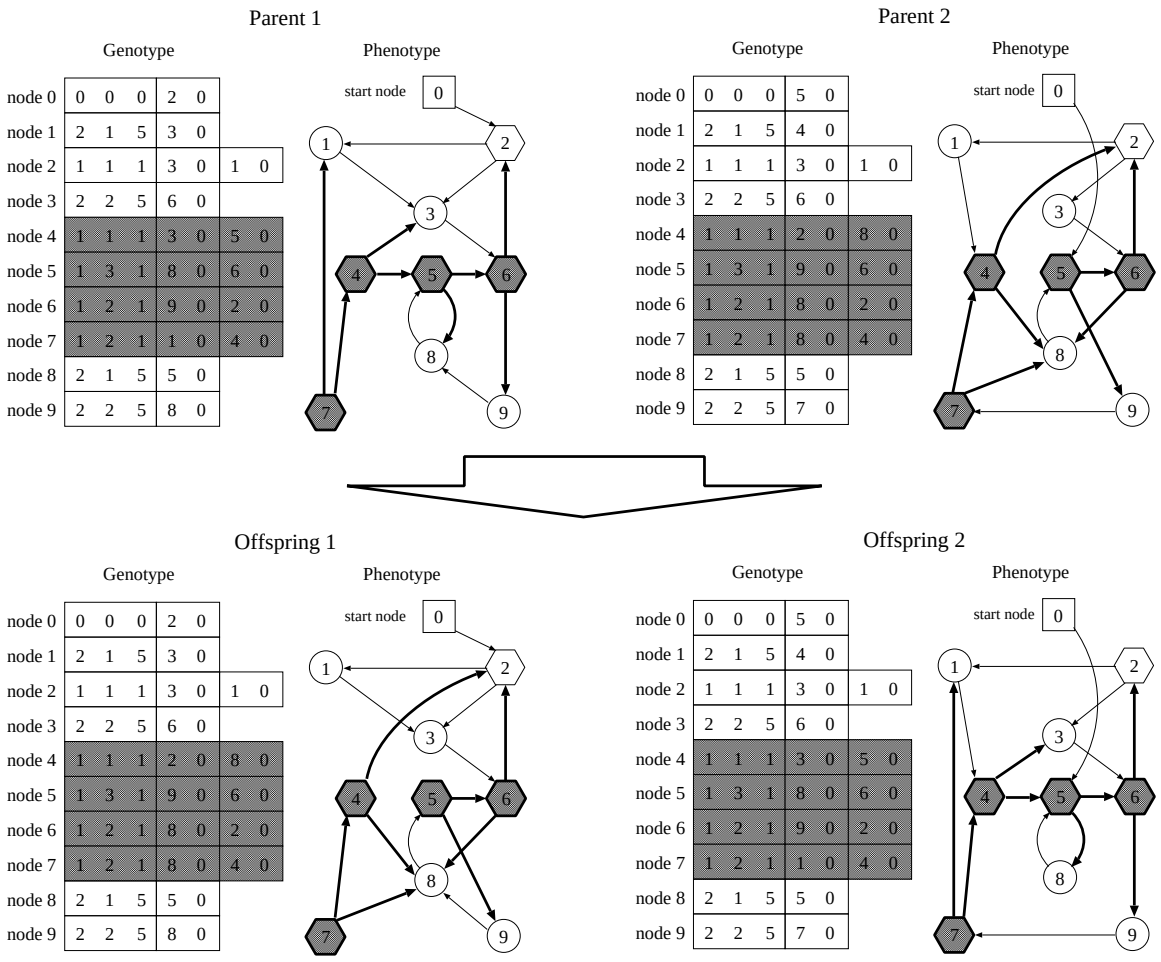


Figure A.7: Several points crossover

A.3 Evolutionary Algorithm of GNP

Basically, GNP finds adaptive solutions by carrying out the evolution in the population. Figure A.9 shows the flow of GNP algorithm, and we can see this process is similar to GA's algorithm. The individuals moving to the next generation are selected with the fitness. After all, the result to be executed by the program could be obtained, and the fitness of solution is calculated for each individual.

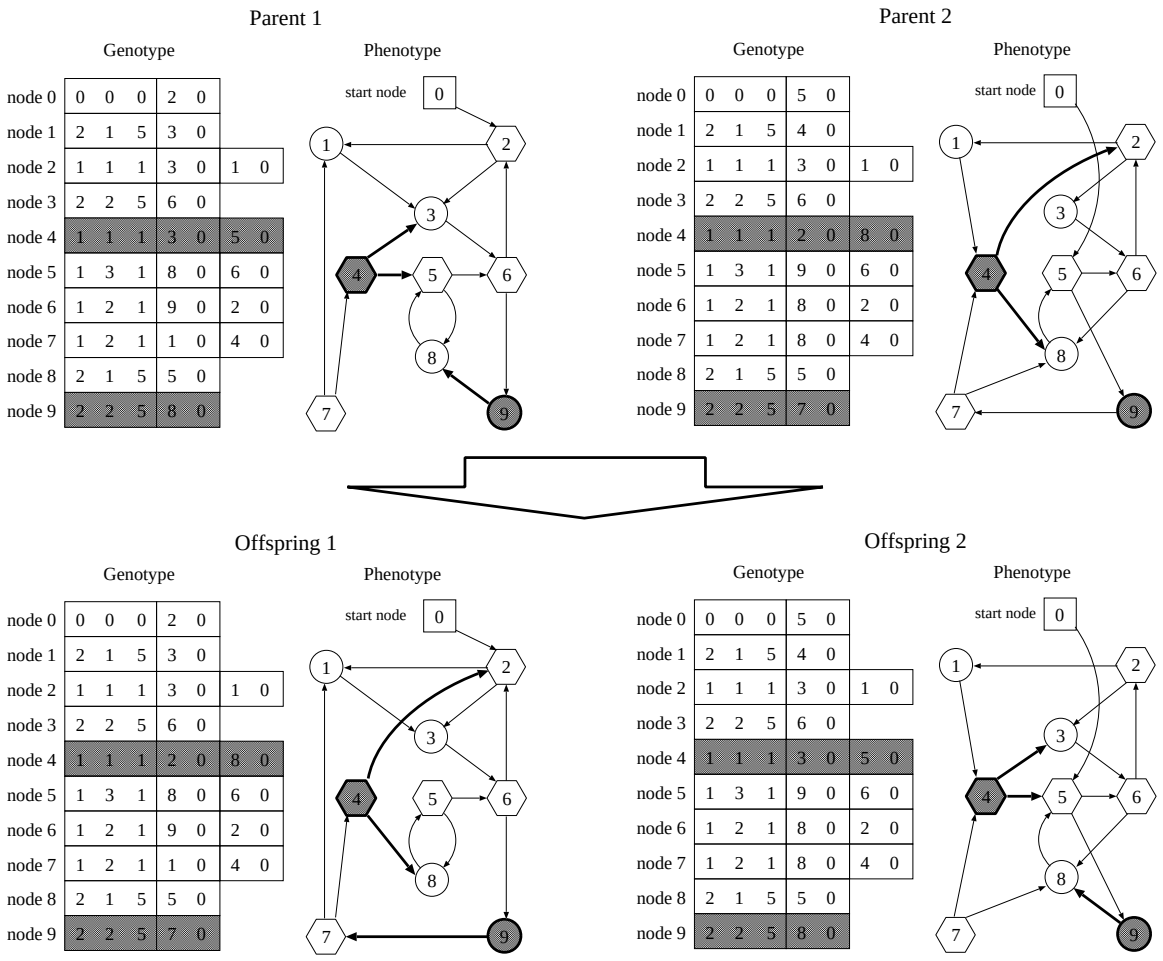


Figure A.8: Uniform crossover

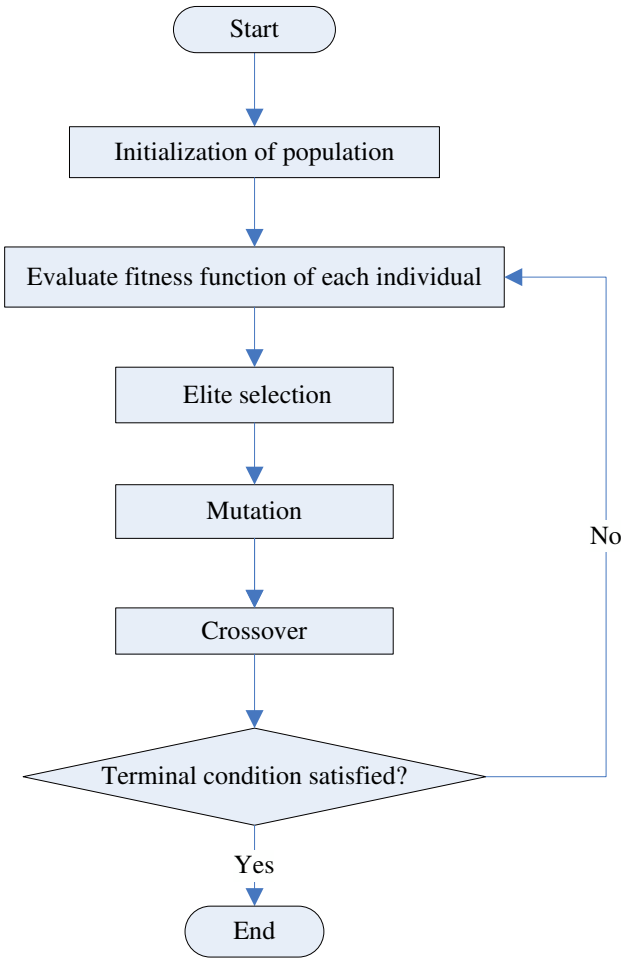


Figure A.9: Flow chart of GNP

Acknowledgements

From the formative stages of this thesis to the final draft, I owe an immense debt of gratitude to my supervisor, Prof. Kotaro Hirasawa. His sage advice, insightful criticisms, and patient encouragement aided the writing of this thesis in innumerable ways.

I would also like to acknowledge Dr. Osamu Yoshie and Dr. Shigeru Fujimura. Their sound suggestion and careful guidance were unique and invaluable.

And thanks Dr. Shigo Mabu for all the indispensable assistance he has offered.

To each of the colleagues in our laboratory, I extend my deepest appreciation. They are all smart, inspired, aggressive and enthusiastic people and they are always willing to offer me assistance during the past five years.

Research Achievements

Journal Paper

(1) F. Ye, S. Mabu, L. Wang, S. Eto and K. Hirasawa, “Genetic Network Programming with Reconstructed Individuals”, SICE Journal of Control, Measurement, and System Integration (SICE JCMSI), Vol. 3, No.2, pp. 121-129, 2010.

(2) F. Ye, L. Yu, S. Mabu, K. Shimada and K. Hirasawa, “Genetic Network Programming with Rules”, Journal of Advanced Computational Intelligence and Intelligent Informatics, Vol. 13, No. 1, pp. 16-24, 2009.

International Conference Paper

(1) F. Ye, S. Mabu and K. Hirasawa, " A Memory Scheme for Genetic Network Programming with Adaptive Mutation", ACM Genetic and Evolutionary Computation Conference 2011 (GECCO2011), Dublin, Ireland (accepted).

(2) F. Ye, S. Mabu, L. Wang and K. Hirasawa, "Genetic Network Programming with Route Nodes", TENCON2010, pp. 1404-1409, Fukuoka, 2010/11.

(3) F. Ye, S. Mabu, L. Wang and K. Hirasawa, "Genetic Network Programming with New Genetic Operators", 2010 IEEE International Conference on Systems, Man, and Cybernetics, pp. 3346-3353, Istanbul, Turkey, 2010/10.

(4) L. Wang, S. Mabu, F. Ye and K. Hirasawa, "Generalized Rule Accumulation Based On Genetic Network Programming Considering Different Population Size and Rule Length", SICE International Annual Conference 2010, pp.2631-2636, Taipei, Taiwan, 2010/8.

(5) F. Ye, S. Mabu, L. Wang and K. Hirasawa, "Genetic Network Programming

with General Individual Reconstruction", ICROS-SICE International Joint Conference 2009, Japan, pp. 3474-3479, Fukuoka International Congress Center, 2009/8.

(6) L. Wang, S. Mabu, F. Ye and K. Hirasawa, "Rule Accumulation Method with Modified Fitness Function based on Genetic Network Programming", ICROS-SICE International Joint Conference 2009, pp. 1000-1005, Fukuoka International Congress Center, Japan, 2009/8.

(7) F. Ye, S. Mabu, L. Wang, S. Eto and K. Hirasawa, "Genetic Network Programming with Reconstructed Individuals", IEEE Congress on Evolutionary Computation 2009, pp. 854-859, Trondheim, 2009/5.

(8) L. Wang, S. Mabu, F. Ye and K. Hirasawa, "Genetic Network Programming with Rule Accumulation Considering Jugdment Order", IEEE Congress on Evolutionary Computation 2009, pp. 3176-3182, Trondheim, 2009/5.

(9) F. Ye, S. Mabu, K. Shimada and K. Hirasawa, "Genetic Network Programming with Rule Chains", SICE International Annual Conference 2008, pp. 1220-1225, Tokyo, 2008/8.

(10) S. Yu, F. Ye, H. Wang, S. Mabu, K. Shimada and K. Hirasawa, "A Global Routing Strategy in Dynamic Traffic Environments with A Combination of Q Value-based Dynamic Programming and Boltzmann Distribution", SICE International Annual Conference 2008, pp. 623-627, Tokyo, 2008/8.

(11) S. Yu, H. Wang, F. Ye, S. Mabu, K. Shimada and K. Hirasawa, "A Q Value-based Dynamic Programming Algorithm with Boltzmann Distribution for Optimizing the Global Traffic Routing Strategy", SICE International Annual Conference 2008, pp. 619-622, Tokyo, 2008/8.

(12) L. Yu, J. Zhou, F. Ye, S. Mabu, K. Shimada and K. Hirasawa, "Double-deck Elevator System using Genetic Network Programming with Genetic Operators based on Pheromone Informatio", ACM Genetic and Evolutionary Computation Conference 2008 (GECCO2008), Late Breaking Papers, pp. 2239-2244, Atlanta, 2008/7.

(13) F. Ye, S. Mabu, K. Shimada and K. Hirasawa, "Genetic Network Programming with Rules", IEEE Congress on Evolutionary Computation 2008, pp. 413-418, Hongkong, 2008/6.