Waseda University Doctoral Dissertation

Genetic Network Programming Based Rule Accumulation for Agent Control

Lutao WANG

Graduate School of Information, Production and Systems

Waseda University

January 2013

Abstract

Multi-agent control is a hot but challenging research topic which covers many research fields, such as evolutionary computation, machine learning, neural networks, etc.. Various approaches have been proposed to guide agents' actions in different environments. Evolutionary Computation (EC) is often chosen to control agents' behaviors since it can generate the best control policy through evolution. As a powerful machine learning approach, Reinforcement Learning (RL) is competent for agent control since it enables agents to directly interact with environments and get rewards through "trial and error"s. It is fast and efficient in dealing with some simple problems. However, its state-action pairs may become exponentially large in complex environments, which is computationally intractable. Neural Networks (NNs) could also be used to guide agents' actions since it can map between the input of the environment information and the output of control signals. However, in some high dynamic environments which are uncertain and changing all the time, NN could not work.

Genetic Network Programming is a newly developed EC method which chooses the directed graph structure as its chromosome. High expression ability of the graph structure, reusability of nodes and realization of partially observable processes enable GNP to deal with many complex problems in dynamic environments. One of the disadvantages of GNP is that its gene structure may become too complicated after generations of the training. In the testing phase, it might not be able to adapt to the new environment easily and its generalization ability is not good. This is because the implicit memory function of GNP can not memorize enough information of the environment, which is incompetent in dealing with the agent control problems in high dynamic environments. Therefore, explicit memory should be associated with GNP in order to explore its full potential.

Various research has revealed that memory schemes could enhance EC in dynamic environments. This is because storing the useful historical information into the memory could improve the search ability of EC. Inspired by

this idea, a GNP-based explicit memory scheme named "Genetic Network Programming with Rule Accumulation" is proposed in this thesis. Focusing on this issue, it is studied in the following chapters of this thesis how to create action rules and use them for agent control, how to improve the performance in Non-Markov environments, how to prune the bad rules to improve the quality of the rule pool, how to build a rule pool adapting to the environment changes and how to create more general rules for agent control in dynamic environments. The organization of this thesis is as follows.

Chapter 1 describes the research background, problems to be solved and outline of the thesis. Some classical methods in the domain of evolutionary computation and reinforcement learning are reviewed.

Chapter 2 designs the general framework of GNP-RA, which contains two stages, the training stage and the testing stage. In the training stage, the node transitions of GNP are recorded as rules and stored into the rule pool generation by generation. In the testing stage, all the rules in the rule pool are used to determine agents' actions through a unique matching degree calculation. The very different point of GNP-RA from the basic GNP is that GNP-RA uses a great number of rules to determine agents' actions. However, GNP could use only one rule corresponding to its node transition. Therefore, the generalization ability of GNP-RA is better than that of GNP. Moreover, GNP-RA could make use of the previous experiences in the form of rules to determine agents' current action, which means that GNP-RA could learn from agents' past behaviors. This also helps the current agent to take correct actions and improve its performance. Simulations on the tile-world demonstrate that GNP-RA could achieve higher fitness values and better generalization ability.

Chapter 3 aims to solve the perceptual aliasing problem and improve the performance for multi-agent control in non-Markov environments. The perceptual aliasing problem refers to that different situations seem identical to agents, but different optimal actions are required. In order to solve this problem, a new rule-based model, "GNP with multi-order rule accumulation (GNP-MRA)" is proposed in this chapter. Each multi-order rule records not only the current environment information and agent's actions, but also the previous environment information and agent's actions, which helps agents to distinguish the aliasing situations and take proper actions. Simulation results prove the effectiveness of GNP-MRA, and reveal that the higher the

rule order is, the more information it can record, and the more easily agents can distinguish different aliasing situations. Therefore, multi-order rules are more efficient for agent control in non-Markov environments.

Chapter 4 focuses on how to improve the quality of the rule pool. Two improvements are made in order to realize this. One of them is that during the rule generation, reinforcement learning is combined with evolution in order to create more efficient rules. The obtained knowledge during the running of the program could be used to select the proper processing for judgments, i.e., better rules. The second approach is that after the rule generation, a unique rule pruning method using bad individuals is proposed. The basic idea is that good individuals are used as rule generators and bad individuals are used as monitors to filter the generated bad rules. Four pruning methods are proposed and their performances are discussed and compared. After pruning the bad rules, the good rules could stand out and contribute to better performances. Simulation results demonstrate the efficiency and effectiveness of the enhanced rule-based model.

Chapter 5 is devoted to improve the adaptability of GNP-RA depending on the environment changes. GNP-RA has poor adaptability to the dynamic environments since it always uses the old rules in the rule pool for agent control. Generally speaking, the environment keeps changing all the time, while the rules in the rule pool remain the same. Therefore, the old rules in the rule pool become incompetent to guide agents' actions in the new environments. In order to solve this problem, Sarsa-learning is used as a tool to update the old rules to cope with the inexperienced situations in the new environments. The basic idea is that when evolution ends, the elite individual of GNP-RA still execute Sarsa-learning to update the Q table. With the changes of the Q table, the node transitions could be changed in accordance with the environment, bringing some new rules. These rules are used to update the rule pool, so that the rule pool could adapt to the changing environments.

Chapter 6 tries to improve the generalization ability of GNP-RA by pruning the harmful nodes. In order to realize this, "Credit GNP"is proposed in this chapter. Firstly, Credit GNP has a unique structure, where each node has an additional "credit branch"which can be used to skip the harmful nodes. This gene structure has more exploration ability than the conventional GNP-RA. Secondly, Credit GNP combines evolution and rein-

forcement learning, i.e., off-line evolution and on-line learning to prune the harmful nodes. Which node to prune and how many nodes to prune are determined automatically considering different environments. Thirdly, Credit GNP could select the really useful nodes and prune the harmful ones dynamically and flexibly considering different situations. Therefore, Credit GNP could determine the optimal size of the program along with the changing environments. Simulation results demonstrated that Credit GNP could generate not only more compact programs, but also more general rules. The generalization ability of GNP-RA was improved by Credit GNP.

Chapter 7 makes conclusions of this thesis by describing the achievements of the proposed methods based on the simulation results.

Acknowledgements

I would like to offer my great gratitude to my supervisor: Professor Kotaro Hirasawa for his invaluable guidance, care and encouragement. He is always my best supervisor whom I will never forget.

I would like to give my sincere thanks to Professor Takayuki Furuzuki, Professor Osamu Yoshie and Professor Shigeru Fujimura for their helpful comments and good suggestions in organizing this paper.

I would like to thank Waseda University and China Scholarship Council (CSC) for giving me the opportunity and support to study in Japan.

I would like to show my appreciation to my friends for their sincere help and priceless friendship.

I owe my deepest gratitude to my family, especially my beloved grandma for their steady support, great encouragement and endless love.

Without you, I would't have finished this thesis. I thank all of you from the bottom of my heart.

Contents

List of Figures

List of Tables

1

Introduction

How to create efficient action rules for multi-agent control in dynamic and uncertain environments has been a hot topic for many decades. It covers Artificial Intelligence (AI), Machine Learning (ML) and many other research fields. In this chapter, the classical methods in the related domains of this thesis are briefly reviewed, including evolutionary computation, reinforcement learning and the memory schemes.

1.1 Evolutionary Computation

In the field of Artificial Intelligence (AI), Evolutionary Computation (EC) is an important branch whose essential concept is from Darwin's Theory of Evolution. It uses computer programs to imitate the biological evolutionary mechanisms such as selection, crossover and mutation and to get the optimal solutions. EC is often chosen as a tool to guide agents' actions since it can generate the best control policy through evolution(1, 2). A large number of research has been done on evolutionary computation and many classical evolutionary algorithms (EAs) have made great contributions to this field, such as: Genetic Algorithm (GA)(3, 4), Genetic Programming (GP)(5, 6, 7, 8), Evolutionary Programming (EP)(9, 10) and Evolution Strategy (ES)(11, 12). In 1975, a standard genetic algorithm (GA) based on bit representation, one-point crossover, bit-flip mutation, and roulette wheel selection was proposed and widely applied during the 1980's. The gene of GA is represented as a string structure composed of 0s and 1s, which is good at searching the global optimal solutions in a feasible searching space. However, the expression ability of the string structure is a bit limited. GP extends GA's expression ability by using tree structures where each non-terminal node has an operator function and each terminal node has an operand. GP was mainly used to solve relatively simple problems because it is very computationally intensive and

would easily cause the exponential growth of the genes. During the evolution, the increase of the depth of GP causes the enlargement of the structure, resulting in the large occupation of memory and increase in calculation time, known as the "bloating problem"(13, 14). EP evolves its program by using finite state machines as predictors which is a model of behaviors composed of a finite number of states transitions using the states and actions. EP is a useful optimization method especially when other techniques such as gradient descent or direct, analytical discovery are not possible. ES uses a natural problem-dependent representation and only mutation and selection are chosen as genetic operators. It is typically applied to numerical optimization problems because it is fast and is competent for dealing with real-valued optimization problems. Its speciality is the self-adaptation of mutation step sizes. Many optimization methods such as Particle Swarm Optimization (PSO) (15, 16) and Ant Colony Optimization (ACO)(17, 18) could also be used for multi-agent control problems(19, 20, 21).

Genetic Network Programming (GNP)(22, 23, 24, 25) is a new evolutionary computation method which adopts directed graphs as its gene structure. The advantages of GNP are:

- *Compact Gene Structure.* GNP programs are composed of a number of nodes which execute basic judgment and processing functions. These nodes are connected by directed links and thus the gene structure is very compact.
- *Reusability of Nodes.* The nodes in the gene structure of GNP could be revisited many times which greatly decreases the memory cost, therefore, GNP never causes the bloating problem.
- *Partially Observable Process.* The judgment nodes can judge any partial information and processing nodes can generate any action based on the judgments. GNP doesn't require the complete information of the environment to take an action, thus, the partially observable process could be easily realized by GNP.

However, the conventional GNP also has its shortcomings such as:

- *Over-fitting problem*(26, 27). The nodes of GNP and their connections make the gene structure very complicated after generations of the training. Such complex gene structure may over-fit the training instances easily. In other words, the generalization ability of conventional GNP is not good.
- *Weak implicit memory.* Although the redundant nodes in the chromosome enable GNP to have some implicit memory function, they are not enough to store enough information in evolution. On the other hand, too many redundant nodes could aggravate the over-fitting problem of GNP. These shortcomings should be overcome in order to explore GNP to its full potential.

Conventional GNP pays more attention to the final solution, that is, the optimal gene structure obtained in the last generation. However, the experiences generated during the evolutionary process are ignored and not taken into account. The subgraphs of GNP and good transition routes in the evolutionary process which are useful for directing agent's actions should also be preserved and used to guide agents' actions.

Currently, a lot of research has been done to optimize GNP. In the theoretical aspect, GNP with individual reconstruction (28) proposed a method to replace the worst GNP individuals by using the extracted rule chains. GNP with subroutines(29) introduced subroutines into GNP which serve as good building blocks and improved its performance. Also, GNP could be combined with other techniques, such as: reinforcement learning(22), ACO(30), etc.. In the application aspect, GNP has been applied to many real world problems such as: elevator supervisory control systems(31), stock market prediction(32), traffic volume forecasting(33), network intrusion detection(34), and class association rule mining (35) in data mining field.

1.2 Reinforcement Learning(RL)

RL is an important branch of machine learning. It learns how to act given an observation of the environment. Each action has some impacts to the environment, and the environment provides the feedback in the form of rewards that guides the learning algorithm. RL is often selected as the learning strategy for multi-agent systems(36, 37), since it enables agents to directly interact with the unknown environment and get rewards through "trial-and-error"s. RL is very fast and efficient in learning some simple problems. However, when the problem become complicated, its state-action pairs (Q table) will become exponentially large which is computationally intractable(38). As a result, the solution space becomes too large and the obtained rewards are not enough, which degrades its performance. How to control the Q-table becomes an important issue in RL.

Among RL, the most widely used techniques are Q-learning(39) and Sarsa-learning(40). However, Q-learning is an off-line policy(41) since the selection of the maximum Q-value at each state is necessary. Sarsa-learning is an on-line approach(42) because it updates its Q values based on the really obtained rewards during the task execution. Therefore, in this thesis, Sarsa-learning is chosen as the learning strategy for different purposes.

Various research(22, 43, 44, 45, 46) revealed that combining EC and RL could bring better solutions. This is because RL could build a bridge between the static chromosome of EC and the dynamic execution of its program. The obtained knowledge during the programming running could be utilized to create more reasonable gene structures.

Therefore, in real-world applications, RL is often integrated with EC to enhance its search ability. In this thesis, RL is introduced into the GNP framework to create more efficient rules in chapter 4, improve the adaptability of the rule pool in chapter 5 and improve the robustness of the rule-based model in chapter 6.

1.3 Inspiration

1.3.1 Memory schemes enhance evolutionary computation

The research on discovering historical information on good examples and reusing them later has been conducted for many years, which reveals that using past experiences can benefit the current decision making. Typical examples could be said of Case-base Reasoning (CBR)(47, 48, 49) and Memory-based Reasoning (MBR)(50) in cognitive science. For example, CBR could solve a new problem by retrieving relevant cases stored in the memory. After the problem is successfully solved, it is stored into the memory as a new case, which helps to solve the similar problems in the future.

In the domain of EC, using memory schemes which stores historical information on good solutions and reuse them later, could enhance the performance of EAs in dynamic problems(51, 52, 53, 54). Generally speaking, there are two types of memory schemes, implicit memory scheme and explicit memory scheme.

1.3.2 Implicit memory schemes

In implicit memory schemes, EAs use genotype representations that are larger than necessary. The redundant gene segments store some good (partial) solutions to be used in future. For example, Goldberg and Smith extended the simple haploid GA to a diploid GA with a tri-allelic dominance scheme(55). Thereafter, Ng and Wong developed a dominance scheme with four alleles for a diploidy-based GA(56). 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(57). In these methods, 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, which make the search range flexible and wide. GNP also has an implicit memory function(22), since there exist many redundant nodes in its chromosome. The nodes which are not used in the current environment might be used in future environments. However, the implicit memory function of GNP is not satisfactory since it is not sufficient enough to store the environment information in complex problems. On the other hand, if too many redundant nodes exist in its gene structure, the generalization ability of GNP would

decrease and the over-fitting problem could happen easily. Therefore, how to store the useful information into the explicit memory and how to use it properly are essential to improve the performance of GNP.

1.3.3 Explicit memory schemes

In explicit memory schemes, useful information of good solutions is stored in the outside memory. The knowledge from the current generation can be explicitly memorized and reused in future generations. In (58), the best individuals during a run are stored into a memory for the open shop rescheduling problem. Whenever a change happens, the population retrieves partial (5%-10%) individuals from the memory corresponding to the previous run, and initializes the rest randomly. In a robot control problem(59), the best individuals are stored in the memory together with its environment information. In the new environment, the similarity of the current environment to the recorded environment information is measured, and the best individual with the most similar associated environment information is selected and reused. A case-based GA(60) records and uses its prior experiences to tackle the current problem. But, these cases could not be used directly as decision rules. GP with reusable programs(6) automatically discovers the subsets of GP and reuses them. However, the reusable programs are mainly used to discompose complex problems into detachable subproblems, which could not be used directly. The memory scheme storing merely the best solutions is called *directed memory scheme* and the one storing both solutions and their associated environment information is called *indirected memory scheme* or *associative memory scheme*.

1.4 Contents of this thesis

Inspired by the above research, a GNP-based explicit memory scheme name "GNP with Rule Accumulation(GNP-RA)" is proposed in this thesis. The purpose is to improve performance for multi-agent control in dynamic environments. The unique points of GNP-RA are:

- Rule-based model. Different from the traditional memory schemes, GNP-RA does not store the best individuals themselves directly. Instead, it extracts a large number of rules from the best individuals, and stores the rules into the rule pool as memory. These rules are very simple and general, which enables GNP-RA to avoid the over-fitting problem and improve the generalization ability.

• Accumulation mechanism. Rules extracted from the best individuals are stored into the rule pool generation by generation, and the rule pool serves as an experience set. Past experiences of the best solutions could be used to determine the current action, which could improve the accuracy of guiding agents' actions.

• Unique matching calculation. GNP-RA does not use the best individual directly, instead, it uses all the rules in the rule pool to make decisions through a unique matching degree calculation. Rules contains the partial information of agents' environments and the action under such environments, therefore, they don't need to be associated with the best solutions and could be used directly to determine agents' actions. This is also quite different from the conventional memory schemes.

1.4.1 Research topics

This thesis concerns on how to create actions rules from GNP, how to build an efficient and robust rule pool with good adaptability to the environment changes, and how to guide agents' actions using the rules in the rule pool. There are 7 chapters in the thesis.

In Chapter 2, a unique rule-based model is proposed and its framework is designed. The rule of GNP-RA is defined, and how to extract rules from the node transitions of GNP and how to make use of these rules for multi-agent control is described in details. The performance of GNP is compared with the conventional GNP in the tile-world problem.

In Chapter 3, a new rule-based model named "GNP with multi-order rule accumulation (GNP-MRA)" is proposed. The purpose is to improve the performance in non-Markov environments and solve the perceptual aliasing problem. Each multi-order rule contains not only the current environment information and agent's action, but also the past environment information and agent's actions. The past information serves as some additional information which helps agents to distinguish different aliasing situations. Rules of different orders are extracted and how the rule order affects the proposed model is studied. Two matching methods, completely matching and partially matching are designed to make use of the multi-order rules, and their performance are evaluated.

In Chapter 4, in order to improve the quality of the rule pool, reinforcement learning is introduced into evolution to generate more efficient rules. Besides, a unique rule pruning approach using bad individuals is proposed. The idea is to use good individuals as rule generators and bad individuals as rule monitors to filter the generated bad rules. Four pruning methods are designed and their performances are compared and analyzed.

In Chapter 5, GNP with updating rule accumulation (GNP-URA) is designed to improve the adaptability of GNP-RA. The general idea is, after evolution, the best individuals still execute Sarsa-learning to generate some new rules corresponding to some new environments. The new rules are added into the rule pool to update the old rules, therefore, the rule pool could adapt to the changing environment gradually. Sarsa-learning is used in both the off-line evolution phase and on-line updating phase. In the off-line phase, Sarsa-learning helps to create better rules, while in the on-line phase, it is used to realize the rule pool updating.

In Chapter 6, "Credit GNP-based Rule Accumulation(CGNP-RA)" is proposed in order to improve the generalization ability of GNP-RA. Credit GNP is proposed in this chapter which has an additional credit branch to prune the harmful nodes. Which node to prune and how many nodes to prune are determined automatically by reinforcement learning. Credit GNP could generate not only more compact programs, but also more general rules. Therefore, CGNP-RA can guide agents' actions more efficiently in dynamic environments and exhibits better generalization ability.

In Chapter 7, some conclusions are made based on the previous chapters.

2

Genetic Network Programming with Rule Accumulation

2.1 Introduction

2.1.1 Problem to be solved: over-fitting

In this chapter, a GNP-based rule accumulation method (GNP-RA) is proposed. The main purpose is to solve the over-fitting problem and improve performance for multi-agent control.

GNP is a newly developed evolutionary algorithm as an extension of GA and GP. Directed graph structure, reusability of nodes and partial observable processes enables GNP to deal with complex problems in dynamic environments efficiently and effectively. GNP evolves the population and uses the best individual in the last generation to guide agents' actions. However, the gene structure of GNP is relatively complex which contains many nodes and connections. Such complex structure could over-fit the training data easily and its generalization ability is not good. In this chapter, it is tried to discover the useful information in the best individuals in the form of rules, which are more simple and general. Each rule contains only partial information of agents' environment, which enables agents to execute partial observable processes; A large of rules could be used to determine agents' actions. This helps agents to choose the correct action. How to create actions rules and use them to guide agents' actions is to be studied in this chapter.

2.1.2 Contents of the proposed method

The main point of this chapter are as follows:

• The definition of the rule in GNP-RA is unique, which is different from those of the conventional EC methods and the class association rule(61) in data mining.

• How to generate rules from the node transitions of GNP, and how to store the rules are described in this chapter.

• A unique rule matching method is designed to make use of the accumulated rules in the rule pool to determine agents' actions, which is different from the conventional decision making mechanism.

The organization of this chapter are as follows. In section 2.2, the general framework of GNP-RA and its novelties are described by comparing it with GNP. In section 2.3, the basic concepts of GNP are briefly reviewed in terms of its gene structure, genetic operators and flowchart. Section 2.4 shows the algorithms of GNP-RA in details. Section 2.5 introduces the simulation environment and analyzes the simulation results, and Section 2.6 is devoted to summary.

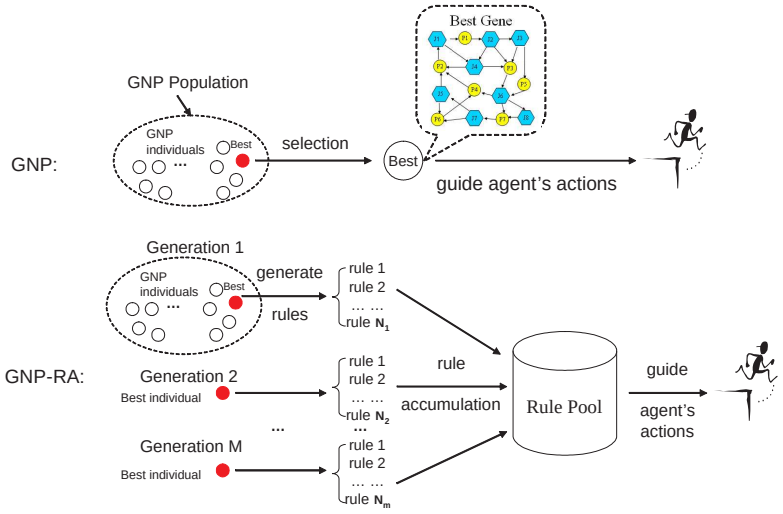

Figure 2.1: Comparison of GNP and GNP-RA

2.2 Novelties of GNP-RA over GNP

Fig. 2.1 shows the general idea of GNP-RA compared with GNP. The major differences between them are as follows.

• In GNP, the best individual in each generation is regarded as the solution. While in GNP-RA, the rules generated from the best individual are regarded as solutions.

- The aim of evolution in GNP is to evolve the population to generate the best individuals. However, the purpose of GNP-RA is to generate a large number of rules and accumulate them into the rule pool, which represents the good experiences from agents' previous actions.
- The best individual of GNP in the last generation is used to guide agents' actions, while in GNP-RA, all the rules in the rule pool are used to determine agents' actions through a matching calculation.

2.3 Review of GNP

This section briefly reviews the basic concepts of GNP in terms of its gene structure, genetic operators and the flowchart.

2.3.1 Gene structure of GNP

Similar to other evolutionary computation methods, GNP inherits some features from Darwin's Theory of Evolution such as the concept of selection, crossover and mutation. Different from the bit string structure of GA or binary tree structure of GP, GNP chooses the directed graph structure as its gene. Fig. 2.2 shows the phenotype and genotype of GNP. There are three kinds of nodes in GNP: a start node, plural judgment nodes and processing nodes which are connected by the directed branches. The function of the start node is to choose the first node to be executed, i.e., the entry of the program. A judgment node has one function which judges the information from the environment. Each judgment node has multiple outputs which connect to the next node depending on different judgment results. A processing node enables the agent to take actions and change the environment, and there is only one connection, which selects the next node. At each time step, the current node indicates what the agent should do (judge or action) and selects the next node to visit.

The right part of Fig. 2.2 shows the gene structure of GNP from the genotype perspective. A two-dimensional array is used to store the gene of nodes. Each gene is the combination of the node information and connection information, namely, node gene and connection gene. In the node gene segment, ID_i is the identification number of node i. NT_i represents the node type of node i. NT_i =0 means it is a start node, NT_i =1 means it is a judgment node and NT_i =2 means it is a processing node. d_i is the time delay for judgment and processing, {d_i=1 for judgment; d_i=5 for processing and d_i=0 for connection in the simulations}. In the connection gene segment, C_{i1}, C_{i2} ...represent the nodes connected from node i firstly, secondly and so on and d_{i1} , d_{i2},

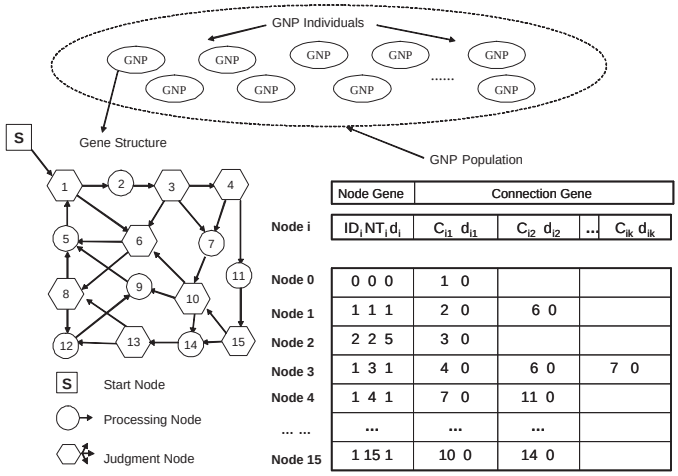

Figure 2.2: Phenotype and Genotype of GNP

... represent the time delays spent on the transition from node i to node C_{i1}, C_{i2}, ..., respectively. The connections of these nodes could be changed by genetic operators such as crossover and mutation which enable individuals to evolve.

2.3.2 Genetic operators

The GNP population is composed of a number of GNP individuals, which are the candidate solutions evolved generation by generation. Fitness is introduced to evaluate the individuals and discriminate good individuals from bad ones. The commonly used genetic operators are crossover, mutation and selection which enable GNP to generate better solutions. The evolutionary process of GNP is a balance of exploration and exploitation. The purpose of crossover and mutation is to explore the solution space by generating new individuals. The aim of selection is to preserve better individuals and eliminate worse ones, i.e., to exploit the obtained solutions.

2.3.2.1 Crossover

Crossover is executed between two parents and two offspring are generated after the execution. Two parents are selected using tournament selection, and each node of them is chosen as a crossover node with the probability of P_c . The parents exchange the gene segments of the crossover nodes, and generated new individuals become the new ones of the next generation. Fig. 2.3 shows a simple example of crossover.

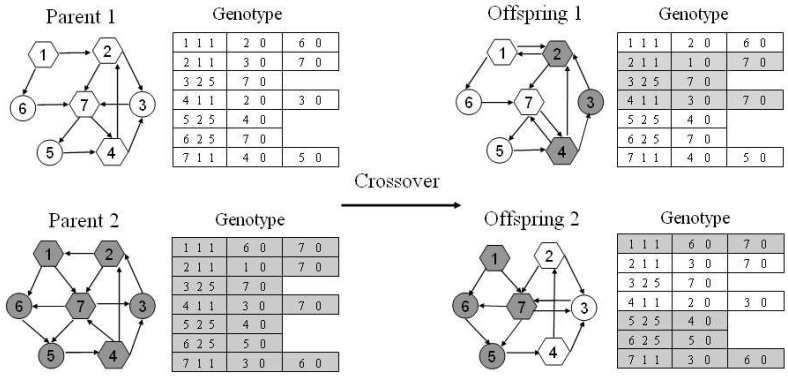

Figure 2.3: Crossover

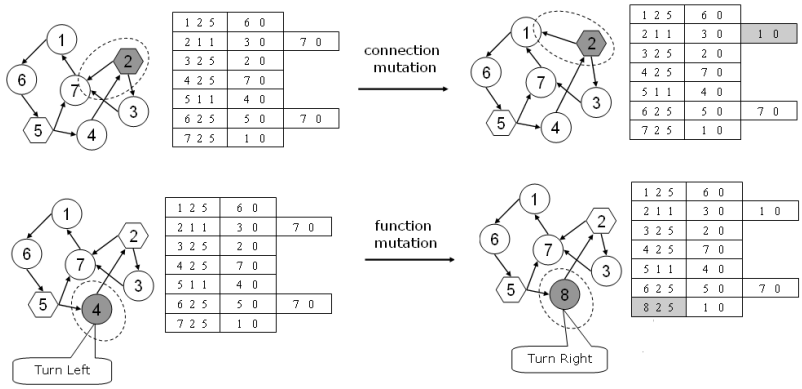

Figure 2.4: Mutation

2.3.2.2 Mutation

Mutation is performed in one individual and a new one is generated. Basically, there are two kinds of mutations: connection mutation and function mutation. Connection mutation refers to that the connections of an individual (parent) are randomly changed to other nodes with the probability of P_m. Function mutation refers to that the functions of a particular individual (parent) are changed to other ones randomly with the probability of P_m. Fig. 2.4 describes the two types of mutations.

2.3.2.3 Selection

At the end of each generation, the elite individuals with higher fitness values are selected and preserved, while the rest individuals are replaced by the new ones gen-

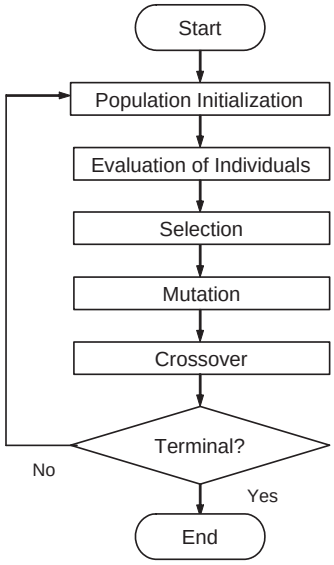

Figure 2.5: Flowchart of GNP

erated by crossover and mutation. There are virous kinds of selection methods, such as *Roulette Selection*, *Tournament Selection* and *Elite Selection*, etc. In the proposed method, *Elite Selection* and *Tournament Selection* are used for selecting the best individuals.

Fig. 2.5 shows the flowchart of GNP program.

2.4 GNP with Rule Accumulation (GNP-RA)

In GNP, the agents' judgments and actions correspond to a series of successive GNP transitions from node to node. Generally, it is noticed that some transition routes consisting of some judgment nodes with their results appear frequently in individuals with high fitness values. These judgment nodes with their judgment results and their succeeding processing node could be viewed as a good experience during the evolutionary process. They are picked up and regarded as "rules". Then, all the rules from the elite individuals are picked up and accumulated in the rule pool, which serves as an experience set. After that, the rule pool is used by classifying these rules, matching them with the situations of the environments and guiding agents' actions.

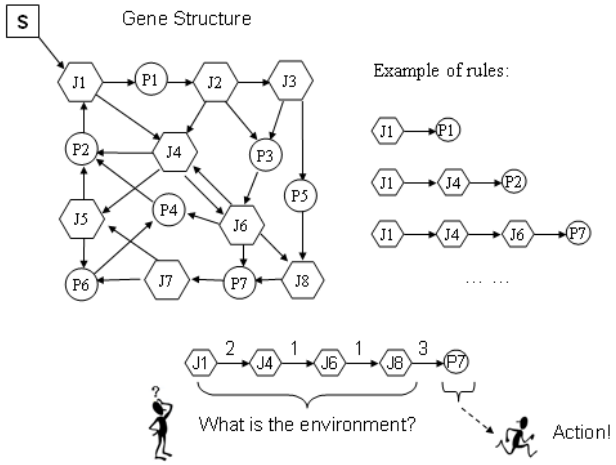

Figure 2.6: Example of the rules

2.4.1 Definition and examples of the rule of GNP-RA

The rule of GNP-RA is defined as a sequence of successive judgment nodes with their judgment results and the succeeding processing node. It starts with the first judgment node after the previous processing node and ends up with the next processing node. The judgment node chain reflects the information of the environment, which tell agents "what the current environment is like", and the processing node denotes what action to take under such an environment, i.e., "what to do in the current situation". Fig. 2.6 shows some examples of the rules of GNP-RA.

The rule of GNP-RA doesn't require the complete information of agents' environment. It contains only partial and necessary information of agents' environment, which is more general. Over-fitting problem could be relieved using such rules for decision making. Besides, the rules of GNP-RA are generated by the evolution of GNP, which are quite different from the class association rules(61) of data mining. The class association rules are generated by analyzing the correlations of different attributes in the database, which usually requires scanning millions of data. Therefore, rules of GNP-RA are easier to obtain compared with the class association rules.

2.4.2 Two stages of GNP-RA

There are two stages in the algorithm of GNP-RA, i.e., the training stage and the testing stage. In the training stage, rules are recorded from the node transitions of GNP and accumulated into the rule pool generation by generation. In the testing stage, the

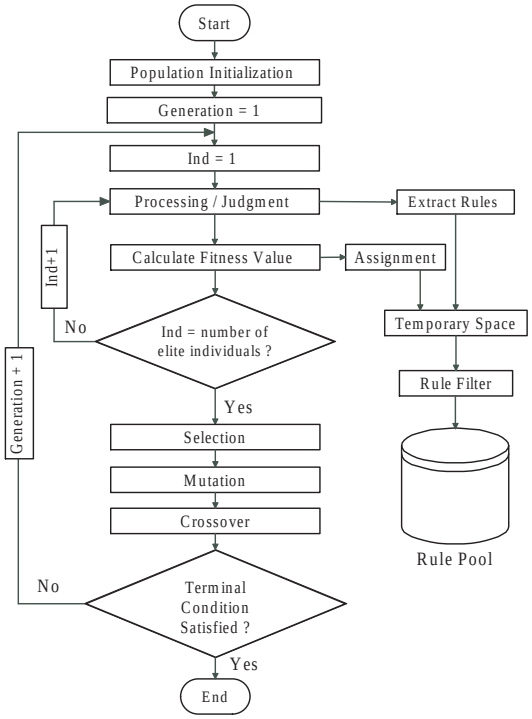

Figure 2.7: Flowchart of the training stage

accumulated rules are used to determine agents' actions through a unique matching calculation.

The training stage concerns how to store and accumulate rules into the rule pool. Fig. 2.7 shows the flowchart of the training stage, which contains two steps.

Step 1: Rule Storage. In the evolutionary period, during the running of GNP program, the node transitions of the best GNP individuals are recorded. The judgment nodes and their corresponding judgment results with the succeeding processing node are stored as a rule. For example, suppose a rule is like $J_1^2 -> J_4^1 -> J_6^1 -> J_8^3 -> P_7$, and the judgment result is {2:obstacle, 1:floor, 1:floor, 3:left }, and the processing node is {2:turn left}. The subscripts denote the judgment node indexes and the superscripts denote the corresponding judgment results. This rule is stored in the form of one-dimensional string, as described by Table 2.1.

Step 2: Strength calculation. After the rules are generated, a strength value is assigned to each rule, which represents its contribution to the fitness.

Fitness and frequency are used to evaluate the contribution of the rule. Eq. (2.1)

Table 2.1: An example of rule storage

Node	*J1*	*J2*	*J3*	*J4*	*J5*	*J6*	*J7*	*J8*	*P*	*Strength*
Value	2	0	0	1	0	1	0	3	2	150

shows the definition of strength.

$$str(r) = fit(r) + \mu * F(r) \tag{2.1}$$

where, *str(r)* is the strength of rule r; *fit(r)* is the fitness value of the best individual from which rule r is generated; *F(r)* is the frequency, i.e., the number of extracted times of rule r in each generation; μ is constant parameter.

The same rule could be extracted many times in different generations. The rule is updated by the following policy: if the strength of the rule is higher than that of the same old rule, the strength of the rule is updated to the higher strength. Otherwise the strength of the rule remains the same.

The testing stage concerns how to make use of the rules accumulated in the rule pool. Fig. 2.8 describes the flowchart of the testing stage, which is divided into 3 steps as follows.

Step 1: Rule pool division. After all the rules are accumulated in the rule pool, the rule pool is divided into several classes according to the final processing nodes. All the rules in the same class has the same processing node, therefore, each class represents a specific action.

Step 2: Class matching. The environment data d is matched with all the rules in each class in order to find the class whose situations are the most similar to the current situation. A completely matching method is adopted in this paper, i.e., a rule is matched if and only if all the judgment results are exactly the same as the environment data *d*. Average Strength (AS) is calculated as follows in order to measure the similarity of the current situation (environment data d) with all the situations recorded in the rules of this class.

$$AS_k = \frac{str_1 + str_2 + ... + str_{N_k}}{N_k}. \tag{2.2}$$

Where, AS_k represents the average strength of the environment data d in class k, str_1, str_2,...and str_{N_k} are the strengths of the matched rules in class k and N_k is the number of matched rules in class k.

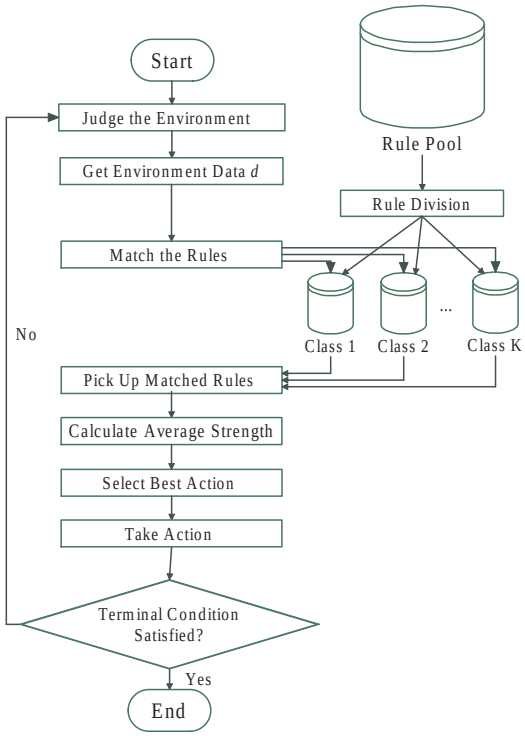

Figure 2.8: Flowchart of the testing stage

Step 3: Action taking. The class with the highest average strength k is selected, and its processing node is executed. The action is taken corresponding to the processing node, and the environment is changed by taking this action. After an action is taken as shown in Eq. (2.3), the environment is updated and a new environment data d is obtained, as a result, agents could take the next action.

$$k = arg \max_{k \in K} \{AS_k\}. \tag{2.3}$$

Where, K is the set of suffixes of classes.

Fig. 2.9 shows the process of rule matching and action taking.

2.5 Simulations

2.5.1 Simulation environment

As an excellent test bed for the agent-based systems, the tile-world problem(62) is chosen as the simulation environment. Fig. 2.10 shows an example of the tile-world

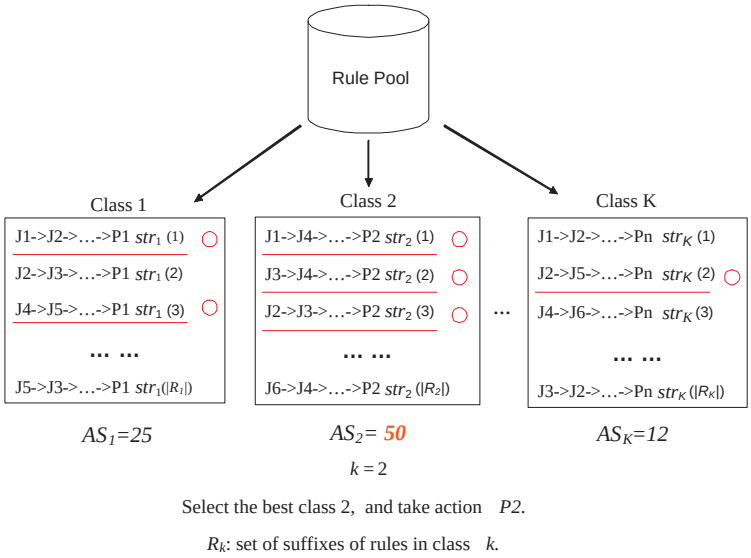

Figure 2.9: Process of rule matching and action taking

whose size is 12×12. Tile-world is a two-dimensional grid which contains five types of objects: tiles, holes, obstacles, floors and agents. Agents could move each time step and push a tile into its neighboring cell. The goal of agents is to push tiles into holes as many as and as quickly as possible without hitting obstacles or dropping themselves into holes. Once a tile is dropped into a hole, they disappear to form a floor. Tile-world is an ideal environment for multi-agent control problems. Therefore, the tile-world is selected as the simulation environment.

Action taking in the tile-world is a great challenge to the multi-agent control problems, since the world information is unpredictable beforehand and agents have limited sight. Agents should distinguish different objects in order to take a reasonable action. The environment is always changing and agents cannot get the full information of the environment, so that the cooperation between them becomes very difficult.

There are two types of tile-worlds, static tile-world and dynamic tile-world. In a static tile-world, when a tile is dropped into a hole, they disappear to form a floor. No new tile or new hole appear. In a dynamic tile-world, after a tile is dropped into a hole, they disappear to form a floor. After that, a new tile and a new hole appear at random positions. Compared with static tile-worlds, dynamic tile-worlds are more challengeable for agents since the appearance of new objects are unpredictable, and the environment keeps changing all the time. In this chapter, for simplicity, static tile-worlds are used to test the performance of the proposed method.

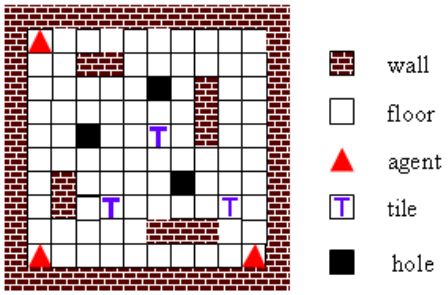

Figure 2.10: An example of the Tile-world

2.5.2 Node function of GNP-RA

In order to get the environment information and take actions, 8 kinds of judgment nodes(J1∼J8) and 4 kinds of processing nodes(P1∼P4) are set for the tile-world problem. J1 to J4 return {1: floor, 2: obstacle, 3: tile, 4: hole, 5: agent}, and J5 to J8 return the direction information {1: forward, 2: backward, 3: left, 4: right}. The functions of judgment and processing nodes are listed in Table 2.2.

2.5.3 Environment data *d*

Fig. 2.11 shows an example of the environment data d in the tile-world problem. Environment data d records the current environment information of the agent, which is represented by a string of integers. J1 to J8 are the judgment nodes which get the information from the environment, and the integers in the string represent the object information or direction information. For example, J1 is to get the information of "what is in front of the agent?", and there is a tile in the environment, whose number is 3. Therefore, the number of J1 is set at 3. The numbers of other judgment nodes are set in a similar way.

2.5.4 Simulation configuration

In the training stage, 300 individuals are used in the population, which keeps evolving for 500 generations. 60 nodes are set for GNP-RA, including 40 judgement nodes and 20 processing nodes. 10 different tile-worlds are used to train GNP individuals and generate rules. In the testing stage, 8 new tile-worlds are used to test the performance of the proposed method, where the locations of tiles and holes are different from the training instances. In each tile-world, the number of tiles, holes and agents is set at 3. Table 2.3 shows the parameter configuration for simulations.

Table 2.2: Node function of GNP-RA

Node	Information that GNP-RA judges
J1	The object in front of the agent
J2	The object at the back of the agent
J3	The object on the left of the agent
J4	The object on the right of the agent
J5	The direction of the nearest tile
J6	The direction of the nearest hole
J7	The direction of the nearest hole from the nearest tile
J8	The direction of the second nearest tile
P1	Move Forward
P2	Turn Left
P3	Turn Right
P4	Stay

Table 2.3: Parameter configuration for simulations

Population:	300	Generations:	500
Mutation:	175	Judgment nodes:	40
Crossover:	120	Processing nodes:	20
Elite Individual:	5	Crossover Rate P_c:	0.1, 0.2
Tournament Size:	6	Mutation Rate P_m:	0.01, 0.05
Number of Agents:	3		

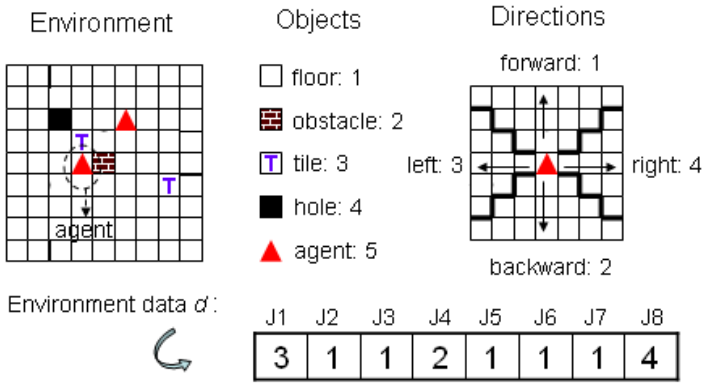

Figure 2.11: An example of the environment data d

2.5.5 Fitness function

Fitness function for the static tile-world is defined as follows:

$$Fitness = C_{tile} \times DroppedTile + C_{dist} \times \sum_{t=1}^{T} (InitialDist(t) - FinalDist(t)) + C_{stp} \times (TotalStep - UsedStep). \quad (2.4)$$

Where, *DroppedTile* is the number of tiles the agents have dropped into holes. *InitialDist(t)* represents the initial distance of the t^{th} tile from the nearest hole, while *FinalDist(t)* represents the final distance of the t^{th} tile from the nearest hole. T is the set of suffixes of tiles, and *TotalStep* and *UsedStep* are the number of steps which are set and used for the simulation. C_{tile}, C_{dist} and C_{stp} are assigned constants. In the situations, C_{tile} =100, C_{dist} =20, C_{stp} =10 and *TotalStep* =60 are used. Actually, *UsedStep* is the number of steps spent on the tile-worlds when all the tiles are dropped.

2.5.6 Results and analysis

Fig. 2.12 shows the training results with three pairs of crossover and mutation rates: (0.1, 0.01), (0.1, 0.05) and (0.2, 0.05). It is noticed that when crossover and mutation rates are larger, the fitness curve increases faster in earlier generations. This is because larger crossover and mutation rates enable GNP-RA to explore the solution space efficiently and generate better solutions faster. However, in later generations, their performances are not good because the exploitation ability is not satisfactory. Large crossover and mutation rates may ruin the optimal gene structure. (0.1, 0.01)

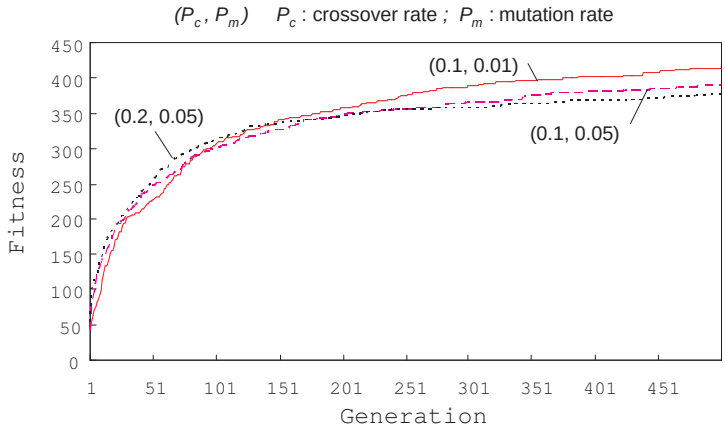

Figure 2.12: Training results of GNP-RA

enables GNP-MRA to generate the best program in the last generation because it can well balance the exploration and exploitation ability.

Fig. 2.13 shows the testing results of GNP and GNP-RA. It is noticed that in some tile-worlds, GNP can good results, while in other tile-worlds, its performance is not satisfactory. For example, in the testing tile-world 1 and 8, GNP even gets negative results which means that, agents can not drop the tile into the hole, but push the tiles even further from their original nearest holes. This is because GNP uses the best individual to determine agents' actions, whose gene structure is very complex (too many nodes and connections) after hundreds of generations. Such an individual can not be generalized well in different testing tile-worlds, since its complex structure can not fit into different situations which are quite different from the training environments.

On the other hand, GNP-RA could get relatively better and stabler results in the 8 testing tile-worlds. Furthermore, the average fitness of GNP-RA is higher that of GNP. This is because GNP-RA guide agents' actions using a large number of rules, which are very simple and general. This helps to solve the over-fitting problem efficiently. Besides, these rules are generated by all the best individuals from the previous generations, which represent the good experiences from agents' past behaviors. The explicit memory function of GNP-RA could make use of the past experiences to determine the current action, which could guide agents' actions more accurately. Fig. 2.14 shows the number of rules generated by GNP-RA in different generations, from which it is noticed that GNP-RA could accumulate a large number of rules as the generation goes on.

In the following simulation, 100 randomly generated tile-worlds with different distribution of the tiles and holes are used to test the performance of GNP and GNP-RA.

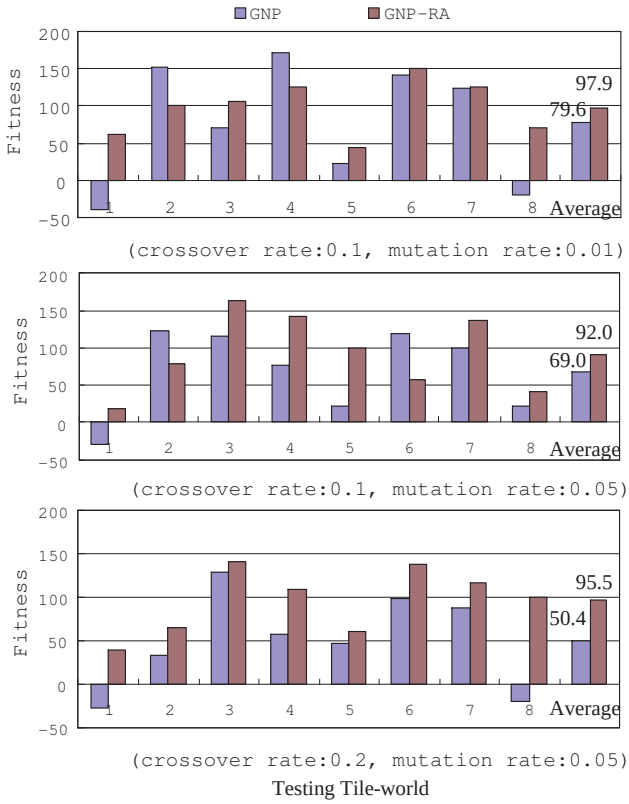

Figure 2.13: Testing results of GNP-RA

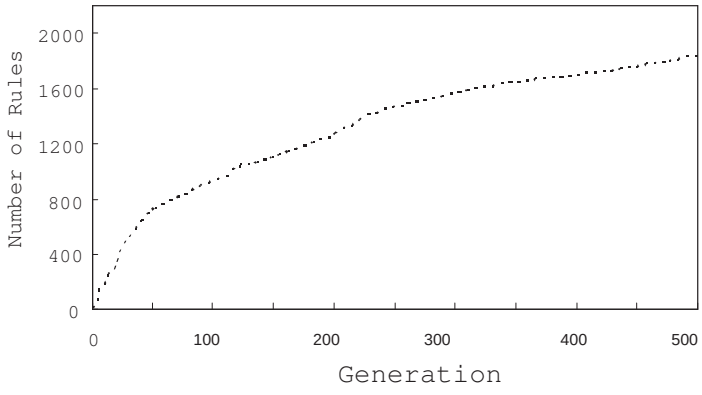

Figure 2.14: Number of rules generated by GNP-RA

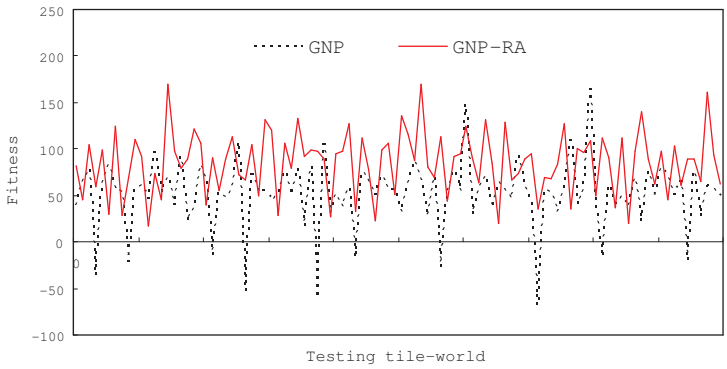

Figure 2.15: Random test of GNP-RA

Table 2.4: Average fitness and standard deviation of GNP and GNP-RA

matching methods	GNP	GNP-RA
Average fitness	61.4	80.7
Standard deviation	23.2	21.4
p-value(t-test)		2.76×10^{-8}

Fig. 2.15 shows the testing results of the two methods in different tile-worlds. Table 2.4 lists the average fitness values over the 100 testing instances. It is seen that GNP-RA outperforms GNP in most testing tile-worlds, and the average fitness of GNP-RA is higher than that of GNP. The p-value of the t-test in Table 2.4 suggests that improvement of GNP-RA over GNP is obvious and evident. The results of the random test also demonstrate the effectiveness of GNP-RA.

2.6 Summary

In this chapter, a GNP-based explicit memory scheme named GNP with Rule Accumulation was proposed. The purpose is to solve the over-fitting problem and improve the performance for multi-agent control. In order to realize this, in this chapter, the useful node transitions of GNP programs are defined as "rules" and used to determine agents' actions. The rules of GNP-RA are very simple, general and easy to get, which helps to solve the over-fitting problem. A large number of rules are extracted from the best individuals throughout the generations, which represent the good experiences

from the past behaviors. These rules could help agents to choose the correct actions and improve the performance.

In this explicit memory scheme, a two-stage framework is designed to build the rule-based model. In the training stage, the node transitions of the best individuals are accumulated into the rule pool generation by generation. In the testing stage, all the rules in the rule pool are used to determine agents' actions through a unique matching calculation. Simulations on the tile-world problem show that GNP-RA could achieve higher performance than GNP. Also, the generalization ability has been improved.

It is noticed that the testing results of GNP-RA is still much lower than that of training, which means that this explicit memory scheme still has a broad space to explore its full potential.

3

Genetic Network Programming with Multi-order Rule Accumulation

For many real-world agent control problems, their environments are non-Markov environments(63, 64, 65, 66). Multi-agent control in such environments is very difficult because the environment information is partially observable. Agents suffer from the *perceptual aliasing* problem(67) and couldn't take proper actions. Therefore, the co-operation between them becomes extremely hard. In order to solve this problem, this chapter proposes a rule-based model named "multi-order rule accumulation" to guide agent's actions in non-Markov environments. The advantages are, firstly, each multi-order rule memorizes the past environment information and agent's actions, which serves as the additional information to distinguish the aliasing situations, secondly, multi-order rules are very general, so that they are competent for guiding agents' actions in Partial Observable Markov Decision Process (POMDP), thirdly, multi-order rules are accumulated throughout the generations, which could cover many situations experienced in different generations. This also helps agents to take proper actions. The new rule-based model is tested using the tile-world to demonstrate its efficiency and effectiveness.

3.1 Introduction

3.1.1 Problem to be solved: perceptual aliasing

Multi-agent control problems in such non-Markov environments are difficult because agents couldn't get full information on their environments. Therefore, they cannot take proper actions and the cooperation between them becomes extremely hard. In such situations, agents suffer from the *perceptual aliasing problem*, which means that different situations seem *identical* to agents, but require *different* optimal actions. Fig. 3.1 shows an example of such a perceptual aliasing problem. In Fig. 3.1, the agent's task is to reach the goal as soon as possible, but the direction of the goal is unknown since it has limited sight. In position 1, the best action seems to turn right whereas in position 2, turn left is the best action. Agent becomes puzzled in such situations. However, if position 1 and position 2 could be distinguished, e.g., if the agent knows the additional environment information on position 1 and 2, or if it can remember which side of the environment(left or right) it enters, it can take proper actions and perceptual aliasing could be solved.

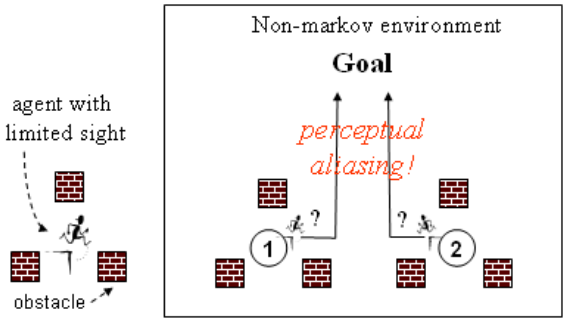

Figure 3.1: An example of perceptual aliasing

3.1.2 Motivation

Non-Markov environments could turn to Markov environments or get close to Markov environments if enough information could be obtained(68, 69). However, sometimes it is difficult. Since agents cannot rely on their current situations to determine the best action, we can give them a memory to store additional useful information together with the current situations to take correct actions. Recurrent Neural Network (RNN)(70) was used to solve POMDP because it can get feedbacks which serves as additional

information to distinguish the aliasing situations. A bit-register memory(71) was incorporated into an accuracy based classifier system and proved to be effective in solving simple non-Markov problems. Reinforcement Learning could make use of its past state-action pairs(72) to improve its performance when coping with POMDP. These research indicates that additional information is useful in solving POMDP in non-Markov environments.

3.1.3 Novelties of this chapter

In the previous chapter, a rule-based model named GNP-RA was proposed, which helps solving the over-fitting problem and improved the performance and generalization ability of GNP. However, the biggest problem of GNP-RA is that the rules are too simple and contain only the current information. Therefore, historical information on the environment and agents' actions is lost, so that GNP-RA cannot distinguish aliasing situations efficiently. In this paper, a more generalized method named *"GNP with Multi-order Rule Accumulation"(GNP-MRA)* is proposed for multi-agent control in non-Markov environments. The most important point of this paper in solving the non-Markov problem is that, *the history information of the environment and agent's behaviors are recorded in multi-order rules to distinguish different aliasing situations.* The novelties of GNP-MRA are as follows.

1) **Multi-order structure.** The multi-order rule of GNP-MRA is different from the rule of GNP-RA. Each multi-order rule considers not only the current environment information and agent's actions, but also the previous environment information and agent's actions, which serve as the past observations of agent's environments. This additional information helps to distinguish the aliasing situations and improve the performance in non-Markov environments. Rules of different order are extracted from the node transitions of GNP, and how the rule order affects GNP-MRA is studied and discussed in detail.

2) **Matching degree calculation.** In GNP-RA, a matching method using average strength was used to select the relevant rules for the agent control. However, matching of the multi-order rules is relatively more complicated. A novel matching method named "average matching degree calculation" is designed to select the similar multi-order rules to the current situation. Two matching methods, namely, *completely matching method* and *partially matching method*, are used to match the multi-order rules, and how they affect the performance of the rule-based model are studied.

The rest of this paper is organized as follows. Section 3.2 describes the flowchart and algorithm of the proposed method in details. Section 3.3 compares GNP-MRA with the

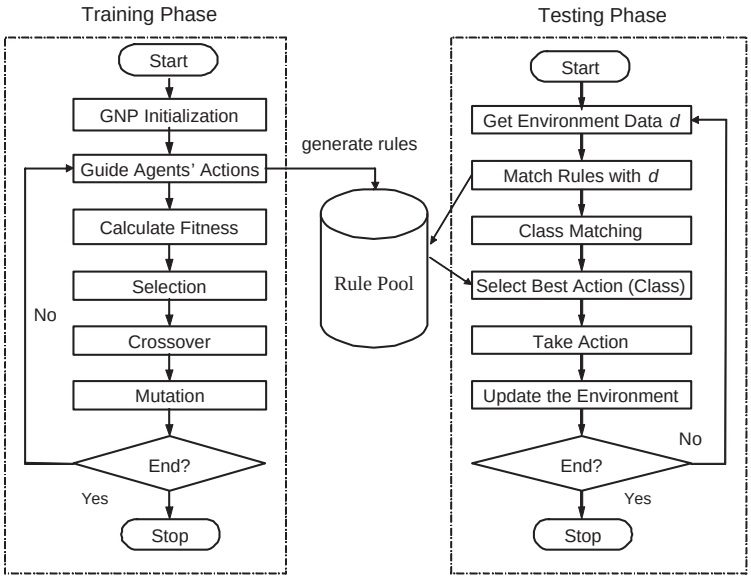

Figure 3.2: Flowchart of GNP-MRA.

conventional methods such as GP, GNP and previous GNP-RA method. Simulation results are studied and analyzed. Section 3.4 is devoted to summary.

3.2 GNP with multi-order rule accumulation (GNP-MRA)

In this section, a GNP-based multi-order rule accumulation method is proposed for agent control in non-Markov environments, and it is named as GNP-MRA. How to generate multi-order rules from the node transitions of GNP and how to guide agent's actions using these multi-order rules are explained in details.

3.2.1 Flowchart

Fig. 3.2 shows the flowchart of the proposed method, which has two phases, i.e., the training phase and testing phase. In the training phase, the GNP population is trained in order to increase the fitness and generate enough rules. In each generation, the node transitions of the best-fitted individual are recorded as multi-order rules and stored in the rule pool. When evolution terminates, it comes to the testing phase. Multi-order rules in the rule pool are used to guide agent's actions through a unique matching calculation.

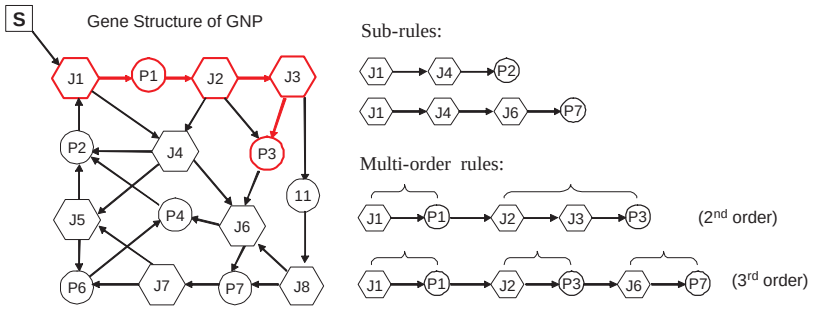

Figure 3.3: Examples of Rules in GNP-MRA

3.2.2 Definition and examples of multi-order rules

1) Sub-rule: A sub-rule is defined as a sequence of successive judgment nodes (with their judgment results) and the succeeding processing node in GNP. The rule of GNP-RA is actually the sub-rule of GNP-MRA. Fig. 3.3 shows an example of rules. $J1 \rightarrow J4 \rightarrow P2$ is a sub-rule, since judgment 1 and 4 are necessary for processing 2. Similarly, $J1 \rightarrow J4 \rightarrow J6 \rightarrow P7$ is another sub-rule.

2) Multi-order rule: A multi-order rule is defined as the sequence of N simple sub-rules, where the N^{th} sub-rule represents the current rule, and the previous *N-1* sub-rules correspond to the past rules in the previous *N-1* steps. Compared with the sub-rule, the multi-order rule contains more history information in the previous time steps. In Fig. 3.3, $J1 \rightarrow P1 \rightarrow J2 \rightarrow J3 \rightarrow P3$ is a second-order rule since it records two consecutive actions and $J1 \rightarrow P1 \rightarrow J2 \rightarrow P3 \rightarrow J6 \rightarrow P7$ could be regarded as a third-order rule.

3.2.3 Multi-order rule generation in the training phase

In the training phase, the GNP population is trained in order to increase the fitness and generate multi-order rules. The best individuals in each generation are selected as the multi-order rule generators. During the running of the GNP program, the visited judgment nodes (with judgment results) and the succeeding processing node are recorded as a multi-order rule. An N^{th} order rule contains not only the current environment information and agent's action, but also the environment information and agent's actions of the previous $N-1$ steps. Rules are stored in the memory in the form of one-dimensional string. For example, the second order rule in Fig. 3.3 could be stored as $J^2(1) \rightarrow P(1) \rightarrow J^1(2) \rightarrow J^2(3) \rightarrow P(3)$, where, superscripts denote the judgment results.

Each multi-order rule has a *strength* value which represents its importance. The fitness and frequency are used to evaluate the contribution of the rule. Eq. (3.1) shows the definition of strength.

$$str(r) = fit(r) + \mu * F(r) \tag{3.1}$$

where, *str(r)* is the strength of rule r; *fit(r)* is the fitness value of the best individual from which rule r is generated; *F(r)* is the frequency, i.e., the number of extracted times of rule r per each generation; μ is constant parameter.

The same multi-order rule could be extracted many times in different generations. The rule is updated by the following policy: if the strength of the rule is higher than that of the same old rule, the strength of the rule is updated to the higher strength. Otherwise the strength of the rule remains the same.

3.2.4 Multi-order matching in the testing phase

This phase concerns how to guide agent's actions using multi-order rules in the rule pool. In the previous GNP-RA method, the rules are matched by an average strength calculation. The complete matching method is adopted, i.e., a rule is regarded as "matched" if and only if all its judgment results are exactly the same as environment data *d*. There is no problem when the rule is short. However, for the multi-order rules of GNP-MRA which are usually longer than those rules of GNP-RA, the complete matching method is very difficult. Therefore, in this chapter, a unique matching calculation method named "average matching degree calculation" is designed for multi-order rule matching. The general idea is to find those multi-order rules whose judgment and action sequence is similar to agents' environment, and use them to guide the agent's actions in such an environment. There are 3 steps in the testing phase.

Step 1: Rule matching Before the rule matching, all the rules in the rule pool are categorized into $|K|$ classes according to the final processing nodes, so that each class represents a specific action. We adopt two approaches to match the accumulated multi-order rules with the environment: *completely matching* and *partially matching*.

Approach 1: completely matching. For the completely matching, firstly the multi-order rules whose judgments and actions in the past *N-1* sub-rules match with the past environment data are picked up. Then, the matching degree of the N^{th} sub-rule with the current environment data is calculated.

Approach 2: partially matching. For the partially matching, all the judgments and processings in the antecedent part of the rule are used to calculate the matching degree with the environment data. Their differences are, the completely matching uses

only the rules whose environments are completely the same as the environment data, while the partially matching uses all the rules which are similar to the environment data.

The matching degree of environment data d with rule r in class k is calculated as follows.

$$Match_k(d,r) = \frac{N_k(d,r)}{N_k(r)}, \tag{3.2}$$

where, $N_k(d,r)$ is the number of matched judgment and processing nodes with data d in the antecedent part of rule r in class k; $N_k(r)$ is the number of judgment and processing nodes in the antecedent part of rule r in class k.

Step 2: Class Matching. This step aims to calculate the similarity of each class to environment data d. A threshold value T_k is set for each class k to filter the lower matched rules. If the matching degree of a rule is higher than the threshold, it could be selected to calculate the average matching degree with environment data d.

$$T_k = mean_k + \lambda * std_k, \tag{3.3}$$

$$mean_k = \frac{1}{|R_k|} \sum_{r \in R_k} Match_k(d,r), \tag{3.4}$$

$$std_k = \sqrt{\frac{1}{|R_k|} \sum_{r \in R_k} (Match_k(d,r) - mean_k)^2}, \tag{3.5}$$

where, T_k is the threshold for class k; $mean_k$ and std_k are the mean value and standard deviation of the matching degrees of class k; R_k is the set of suffixes of rules in class k.

The average matching degree of environment data d with the rules in class k is calculated as follows.

$$m_k(d) = \frac{1}{|M_k|} \sum_{r \in M_k} Match_k(d,r) * str_k(r), \tag{3.6}$$

where, M_k is the set of suffixes of the selected rules in class k, and $str_k(r)$ is the strength of rule r in class k.

Step 3: Action Taking. Finally, the class which has the highest average matching degree is picked up and its corresponding action is taken. After an action is taken as shown in Eq. (3.7), the environment is updated and new environment data d is obtained, as a result, agents could take the next action.

$$k = arg \max_{k \in K} m_k(d), \tag{3.7}$$

where, K is the set of suffixes of classes (actions).

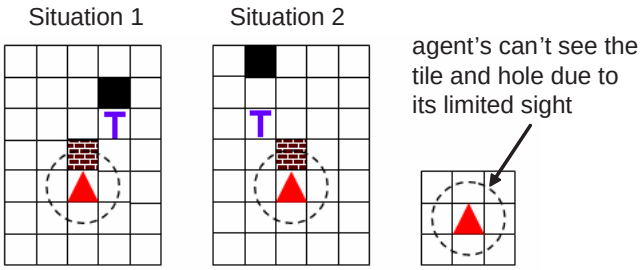

Figure 3.4: Aliasing situations in the Tile-world

3.3 Simulations

3.3.1 Simulation environment

The tile-world problem is also selected as the simulation environment of GNP-MRA. In POMDP, the environment is partially observable and agents can not get full information of their environments. Therefore, the cooperation between agents becomes very difficult. Tile-world is a typical POMDP because the world information is unpredictable beforehand and agents have limited sight. Besides, the combinations of the grid objects are almost infinite, and many aliasing situations exist in the tile-world. Fig. 3.4 shows a simple example of such aliasing situations in the tile-world problem. Suppose agents can only "see" the objects at its surrounding. In situation 1, the agent should turn right in order to avoid the obstacle and drop the tile. However, in situation 2, the agent should turn left. Situation 1 and situation 2 seem identical to the agent with limited sight, which requires different actions. This problem happens because the agent's current information is not enough for its next action. Therefore, the past information of the environment or agent's actions is needed to determine the next action.

3.3.2 Simulation conditions

In the simulation, the generation is set at 500 in order to accumulate enough multi-order rules. The population is set at 300, and the best 5 individuals with the highest fitnesses are selected as multi-order rule generators. 60 nodes are used in GNP including 40 judgment nodes and 20 processing nodes. As a classical method in EC for agent control, GP is selected for comparison. For GP, the function nodes are selected from the judgment nodes J1∼J8, and the terminal nodes are selected from the processing nodes P1∼P4. The function nodes have multiple arguments corresponding to different

judgment results and the terminal nodes have no argument. In order to avoid bloating, the maximum depth of GP is set at 4. Since each non-terminal node has five outputs, the total number of nodes in GP is 781. The best individual of GP is selected to guide agents' actions. Each simulation is carried out for 30 independent runs and the average result is calculated in order to get reliable results.

3.3.3 Results and analysis

Simulation results include the comparison of GNP-MRA with GNP-RA and the conventional methods such as GP and GNP using different matching methods.

In this simulation, two types of environments are selected, i.e., single-agent environment which has 1 agent and multi-agent environment which has 3 agents. Fig. 3.5 shows the training results of the best individuals averaged over 30 independent simulations(in the evolutionary period, the training results of GNP and GNP-MRA are the same). The performance of GP is relatively low because of the limited size of its gene. Tile-world is a difficult problem for GP since its expression ability is not enough. However, if we increase the depth of GP, the memory cost and calculation time would increase exponentially, and the program becomes hard to run. GNP-MRA(GNP) could reuse nodes to avoid bloating, so that it can create compact programs with a small number of nodes. Furthermore, GP execute its program from a root node to a certain terminal node, so agents' behaviors are guided only by the current information. GNP-MRA(GNP) can determine agents' actions by not only the current, but also the past information. Therefore, they can distinguish the aliasing situations easily and achieve better performance.

Fig. 3.6 shows the testing results of different methods. It can be seen that for GP and GNP, there are some negative values in some tile-worlds. This is because agents cannot drop the tiles and move them farther from their original nearest holes. For GNP-MRA, there exits no negative value. The average fitness value of GNP-MRA over the 8 testing tile-worlds is higher than that of GNP and GP. This is because GNP and GP use the best individual to control agents' actions, whose genes are very complicated, and this is also because in the testing phase, there is no genetic operations such as crossover and mutation, so that the gene structure of GNP and GP cannot be changed to adapt to different testing instances. However, GNP-MRA uses a large number of rules to guide agents' actions. These rules are very general and flexible, which could cover many experienced situations in different generations. Agents could take proper actions based on a large number of previous experiences. Therefore, this rule-based

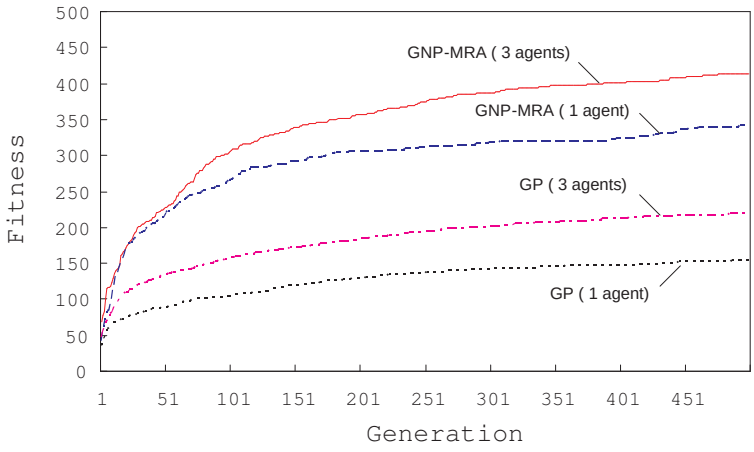

Figure 3.5: Training results of different methods

model could get relatively better results than the individual-based models, even if the testing instances are different.

It is discovered that the performance of three agents is better than one agent in both the training and testing. This is because pushing tiles into holes is a difficult task in a single-agent environment, because the world map is broad and the agent relies only on its own information. In the multi-agent environment, three agents share the information of the environment and take actions considering the other two agents. They could cooperate together to push tiles into holes, which is more efficient than a single agent. Therefore, the number of agents is set at 3 in later simulations.

Fig. 3.7 shows an example of the agents' traces within 60 time steps. Compared with GP and GNP, GNP-MRA could drop more tiles and achieve higher fitness using less steps. This also proves the effectiveness of the proposed method in non-Markov environments.

The parameter λ plays an important role in adjusting threshold T_k. Different values of λ are tried from 0 to 0.5 in our simulations. Fig. 3.8 shows the average fitness values using different λs. It is noticed that when λ is 0 and 0.5, the fitness is relatively low. This is because when λ is 0, T_k is a bit small, therefore, some unrelated rules which are not so similar to the current situation are picked up to determine agents' actions. This decreases the accuracy of decision making. On the other hand, if λ is 0.5, T_k becomes very high. As a result, many rules are filtered and not enough rules could be matched, which means not enough experiences could be obtained to guide agents' actions. 0.1

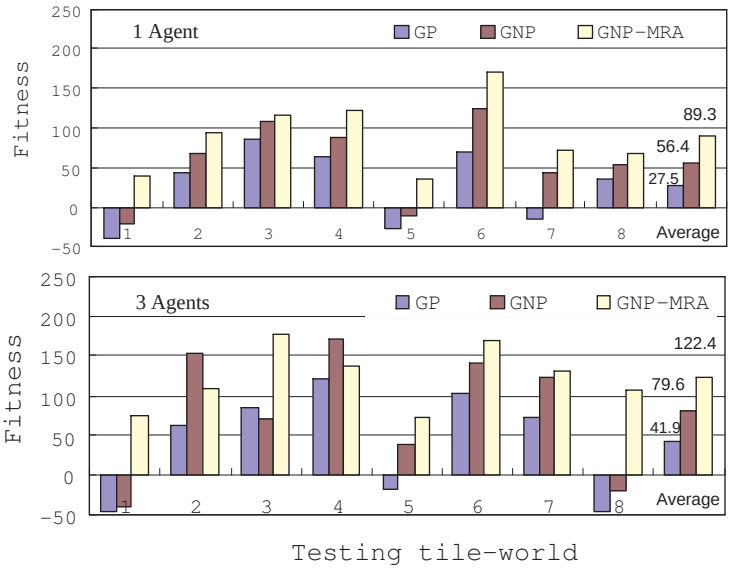

Figure 3.6: Testing results of different methods

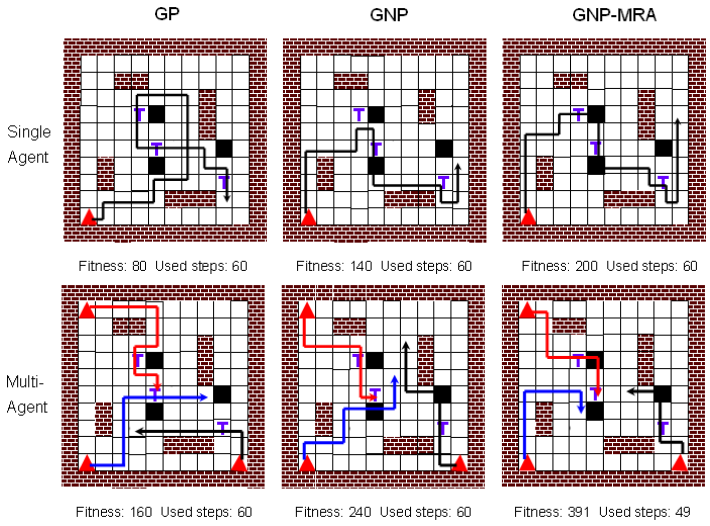

Figure 3.7: Agents' traces of different methods

seems to be the optimal λ because it can well balance the number of rules and the similarity of these rules to the current situation.

The following simulation compares GNP-MRA with the previous GNP-RA method, and the purpose it to find out how the rule order affects the proposed method. GNP-

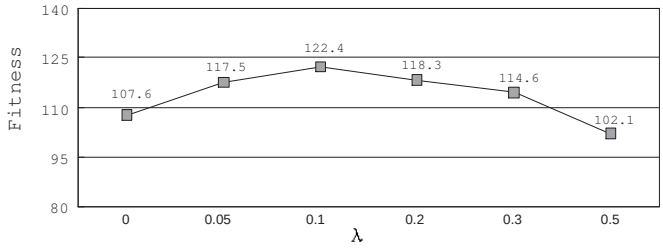

Figure 3.8: Testing results using different λs

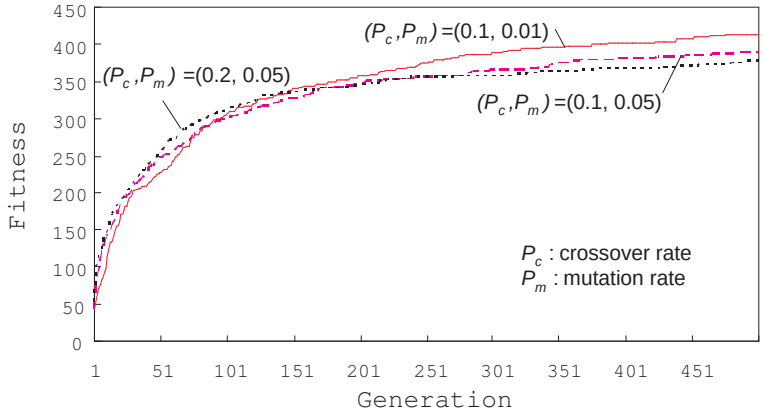

Figure 3.9: Training results with different crossover and mutation rates

RA is a special case of GNP-MRA when the rule order is one. For simplicity, the order of all the rules of GNP-MRA using the completely matching is set at two. In this simulation, the performance of GNP is also provided as a benchmark.

Fig. 3.9 shows the training results with three pairs of crossover and mutation rates: (0.1, 0.01), (0.1, 0.05) and (0.2, 0.05). It is noticed that when crossover and mutation rates are larger, the fitness curve increases faster in earlier generations. This is because larger crossover and mutation rates enable GNP-MRA to explore the solution space efficiently and generate better solutions faster. However, in later generations, their performances are not good because the exploitation ability is not satisfactory. Large crossover and mutation rates may ruin the optimal gene structure. (0.1, 0.01) enables GNP-MRA to generate the best program in the last generation because it can well balance the exploration and exploitation ability.

Fig. 3.10 shows the testing results of each tile-world and Table. 3.1 lists the average fitness values of the three methods with different pairs of crossover and mutation

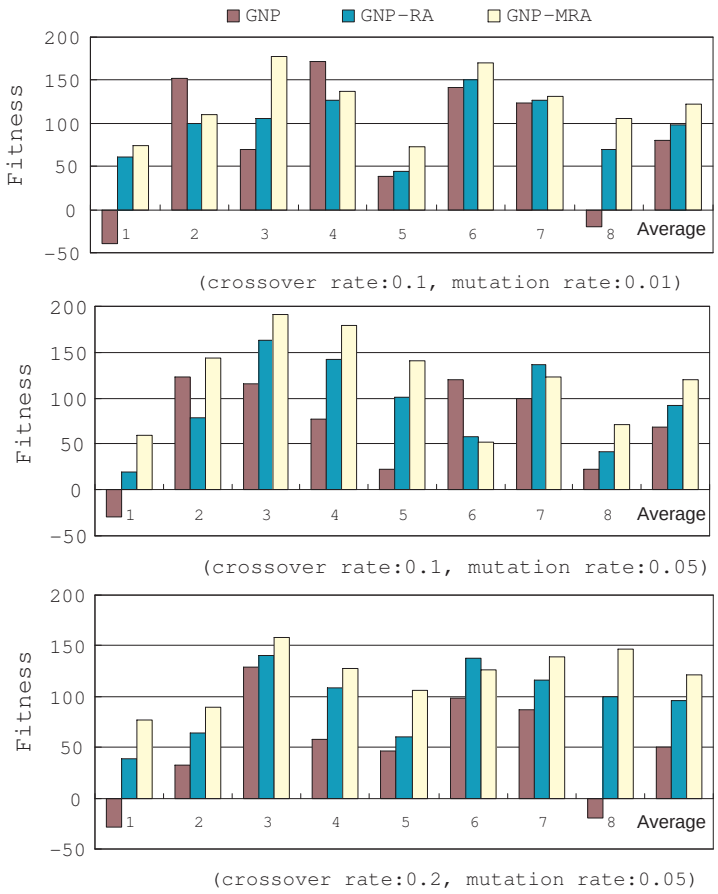

Figure 3.10: Testing results of different methods

rates. It is discovered that GNP-MRA outperforms GNP-RA in most tile-worlds. This is because the first order rule of GNP-RA contains only the current environment information, which serves as the "if-then" decision rule. Tile-world contains many aliasing situations, and agents' current information is not enough to take a proper action. The second order rule contains both the current and past information on the environments and agents' actions, which is like the sequence of the "if-then" decision rules. Agents could "recall" their previous memory when taking the current action. This additional information helps agents to distinguish many different aliasing situations. Therefore, the performance of GNP-MRA is better than GNP-RA.

It is also noticed that when crossover and mutation rates increase, the average fitness of GNP decreases from 79.6 to 50.4 (36.7%), while that of GNP-RA decreases from

Table 3.1: Average fitness of different methods

crossover &mutation rate	GNP	GNP-RA	GNP-MRA
(0.1, 0.01)	79.6	97.9	122.4
(0.1, 0.05)	69.0	92.0	121.9
(0.2, 0.05)	50.4	95.5	124.0

Table 3.2: Average fitness and standard deviation(in brackets) of GNP-MRA

matching methods	1^{st} order	2^{nd} order	3^{rd} order
completely matching	89.1 (23.7)	108.5 (20.2)	76.2 (32.5)
partially matching	94.4 (21.3)	134.9 (19.3)	152.3 (18.9)

97.9 to 95.5 (2.5%) and GNP-MRA increases from 122.4 to 124.0 (0.3%). GNP-RA and GNP-MRA exhibit smaller changes compared with GNP. This is because GNP uses the best individual to guide agents' actions, whose gene structure is sensitive, which means that a small change in the gene structure could affect the fitness largely. However, GNP-RA and GNP-MRA guide agents' actions by using a large number of rules generated and accumulated generation by generation. Therefore, many experienced situations which are similar to the current situation could be retrieved and used to make proper decisions. Since GNP-RA and GNP-MRA don't use the best individual directly, genetic operators have small influence on the performance of GNP-RA and GNP-MRA. This means that the rule-based model could achieve more stable and robust results than the individual-based model in non-Markov environments in terms of genetic operations.

This simulation studies how different matching methods affect the proposed method. Fig. 3.11 shows the performance of GNP-MRA using different matching methods, where 100 randomly generated tile-worlds are used as the testing instances. Table. 3.2 lists the average fitness values of the 100 tile-worlds.

In Fig. 3.11, as for the completely matching method, the second order rules outperform the first order rules and its reason has been discussed in the previous subsection. However, when the rule order turns to three, the performance of GNP-MRA decreases(low average and high deviation). This is because when the rule order increases, the number of judgment nodes and processing nodes in the multi-order rules increases. The completely matching method uses only the completely matched rules (all

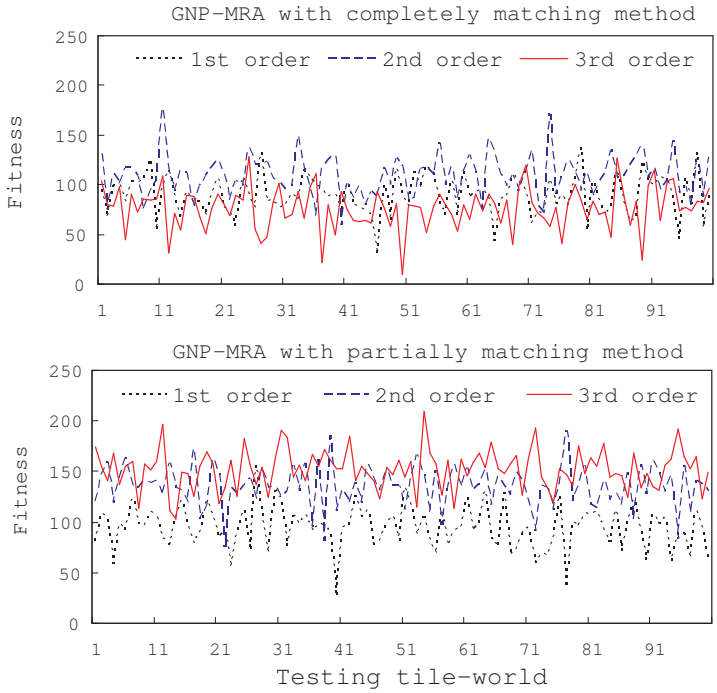

Figure 3.11: Testing results of different matching methods

the nodes and the corresponding results are exactly the same as the current situation). As the rule order increases, completely matching becomes more difficult to match. As a result, the number of matched rules decreases dramatically. Fig. 3.12 shows the proportions of completely matched, partially matched and mismatched rules. It is clearly seen that only 17.3% of the second order rules and 4.1% of the third order rules could be used to guide agents' actions. These few rules could not provide enough experiences to guide agents' actions. Thus, the performance is not satisfactory as the rule order increases.

On the other hand, it is noticed from Fig. 3.12 that the proportion of partially matched rules is relatively stable(around 30%). The partially matching method uses both the completely matched and partially matched rules to determine agents' actions. Fig. 3.13 shows the number of accumulated rules with different orders in the training phase. Apparently, more rules could be generated if the order of rules is larger since there are more combinations of judgments and processings in the rules with higher order. Therefore, the partially matching method could obtain more experiences from the previous actions than the completely matching method. Furthermore, the larger the rule order is, the more information each multi-order rule could memorize, and the more

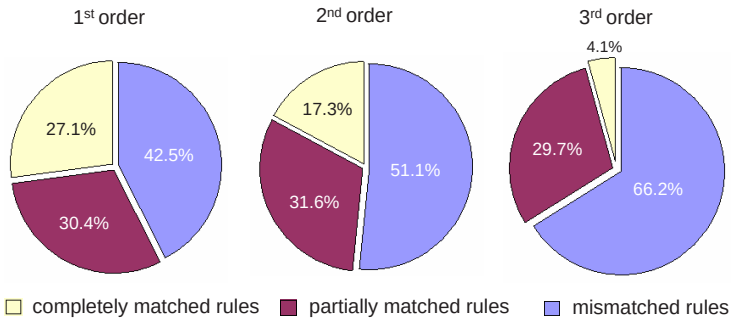

Figure 3.12: Proportion of differently matched rules

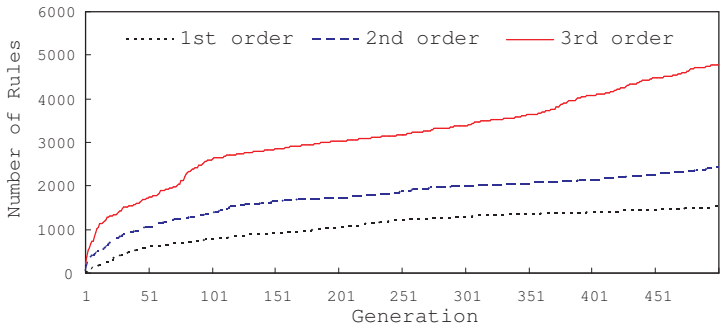

Figure 3.13: Number of rules with different orders

easily agents could distinguish the aliasing situations in non-Markov environments. This helps agents to take proper actions and achieve better performance. Therefore, the performance of GNP-MRA gradually increases as the order of rules increases when using the partially matching method. This also demonstrates the effectiveness of the proposed method.

Table. 3.3 shows the calculation time of GNP-MRA using the partially matching method as an example. The training time is relatively long because evolution takes time. Besides, when the order of rules is larger, it takes more time to store multi-order rules. However, in the testing phase, matching rules with data is fast, which costs only several seconds. Moreover, as the rule order increases from 1 to 3, the calculation time increases from 5.82 seconds to 8.51 seconds, which is still much faster than that of the training phase. This means that the proposed method is very efficient for multi-agent control once the rules are generated.

Table 3.3: Time cost of the GNP-MRA

time(seconds)	1^{st} order	2^{nd} order	3^{rd} order
training	974	1255	1732
testing	5.82	7.36	8.51

3.4 Summary

In this paper, a novel rule-base model named *Multi-order Rule Accumulation* is proposed for multi-agent control, which is very different from the conventional methods and the previous research. The purpose is to solve the perceptual aliasing problem and improve the performance in non-Markov environments. Each multi-order rule memorizes the past information on the environment and agent's actions, which is the additional information that helps agents to distinguish different aliasing situations. In addition, multi-order rules contain only necessary judgments for an action and don't require complete information. Besides, multi-order rules are easy to obtain in the node transitions of GNP. These general and flexible rules could handle POMDP in non-Markov environments efficiently. Furthermore, multi-order rules are accumulated in the rule pool which could cover many experienced situations throughout the generations. These large number of rules could be matched with data and used to guide agents' actions, which also improves the performance in non-Markov environments.

Simulation results on tile-world show that the higher the rule order is, the easier it can help agents to distinguish different aliasing situations, and the better performance it can achieve. They also reveal that, as the rule order increases, the partially matching method is more appropriate to retrieve relevant rules and determine agents' actions, since completely matching becomes very difficult. The calculation time of the rule matching (testing) is much less than the training time, which means that the proposed method is very efficient once the rules are generated. The efficiency and effectiveness of GNP-MRA has been proved.

4

Genetic Network Programming with Rule Accumulation and Pruning

4.1 Introduction

In the previous GNP-RA method, rules are created by genetic operations such as crossover and mutation. Genetic operations could generate rough rules because crossover and mutation could change the gene structure largely. In the rule of GNP-RA, which node to use, how many nodes are necessary and what action to take in the current environment are determined only by evolution. Fitness is used as a standard to evaluate the performance of the individuals. However, fitness is an off-line evaluation since it represents the performance of the entire individual. In other words, fitness is obtained by a serial of rules, and how much each rule contributes to the final fitness is unclear. In creating the action rules for agents, it is expected to select the really useful judgments and combine them to take a correct action. Therefore, the judgment nodes for a particular processing should be determined during the running of the program. The obtained knowledge during the running of the program is beneficial in creating efficient rules. On the other hand, in GNP-RA, all rules generated by the best individuals are regarded as good rule and stored in the rule pool. Actually, these rules contribute differently to the final fitness. Even the best individual can generate some unimportant rules which contribute to nothing, or even negatively to the final fitness. These rules represent some random or misleading actions and should be pruned from the rule pool.

4.1.1 Problem to be solved

The main problem of GNP-RA is that the quality of its rules are not good enough. The reasons are as follows. Firstly, GNP-RA creates rules only by evolution, which is an off-line method. The obtained knowledge obtained during the program running(on-line learning) is lost and could not be used to create efficient rules. Secondly, rules are generated only by the best individuals, where good rules and bad rules are hard to distinguish. The bad rules should be pruned in order to improve the quality of the rule pool. Besides, there exist some specific rules which can cause the over-fitting problem easily. These rules should also be pruned and how to prune them becomes a problem to be solved in this paper.

4.1.2 Motivation

Various research revealed that combining EC and RL is an effective way to bring together their advantages and overcome their weaknesses(73, 74). This is because combining off-line evolution and on-line learning helps to create better chromosomes and more reasonable rules. On the other hand, many studies proved that utilizing bad individuals could benefit the decision making. For example, (75) theoretically showed that the use of infeasible solutions could transform an EA-hard problem to an EA-easy problem. (76) directly used a part of infeasible individuals to construct probabilistic models, and obtained better performance in some problems. In (77), infeasible individuals are used to filter sample errors in EDA to improve the convergence speed.

4.1.3 Novelties of this chapter

Inspired by the above research, a new rule-based model for agent control is proposed in this paper, and it is named as "GNP with Rule Accumulation and Pruning"(GNP-RAP). Compared with GNP-RA, the features and uniqueness of GNP-RAP are as follows.

- On-line and Off-line learning. GNP-RAP combines evolution and reinforcement learning, i.e., off-line evolution and on-line learning in its program. The really important judgment nodes could be selected for a processing considering different situations during the program running. The obtained knowledge could be used to make rules more reasonable. This contributes to generating more efficient rules.

- Pruning techniques. GNP-RAP extracts rules not only from the best individuals, but also from the worst individuals. Rules generated from the worst individuals are used to prune the generated rules in the rule pool and four different pruning methods are designed and compared.

The rest of this chapter is organized as follows. Section 4.2 shows the framework and detailed algorithm of the proposed method. Section 4.3 demonstrates the effectiveness of the proposed method using the tile-world problem, and analyzes the simulation results. Section 4.4 is devoted to summary.

4.2 GNP with Rule Accumulation and Pruning(GNP-RAP)

4.2.1 General framework of GNP-RAP

In this section, the algorithm of the proposed method is described in detail, which consists of three stages: 1), evolve GNP individuals using Sarsa-learning and generate rules. 2), prune rules to improve the quality of the rule pool. 3), use the rules to guide agent's actions by using a matching calculation. The following three subsections will show the detail of them. Fig. 4.1 shows the flowchart of the proposed method, in which A, B and C correspond to the subsections 4.2.2, 4.2.3 and 4.2.4 of Section 4.2, respectively.

4.2.2 Evolution and learning of GNP-RAP

GNP-RAP combines both evolution and reinforcement learning to create efficient rules, while evolution is to make the rough gene structure statically and reinforcement learning is to use the obtained knowledge during the program running dynamically to make the rough gene structure more reasonable, which contributes to creating more efficient rules.

Evolution is to change the gene structure of GNP-RAP largely through genetic operators in order to generate rules quickly. Genetic operators are similar to those of GA and GP. For crossover, two parents are selected and exchange their gene segments to generate two new offspring for the next generation. All connections of the selected nodes are exchanged. For mutation, the connection or function of a particular node is changed randomly to another one, so that a new individual could be generated. At the end of each generation, the elite individuals with higher fitness values are selected and preserved, while the rest individuals are replaced by the new ones generated by crossover and mutation. Fig. 4.2 shows the genetic operators of GNP-RAP.

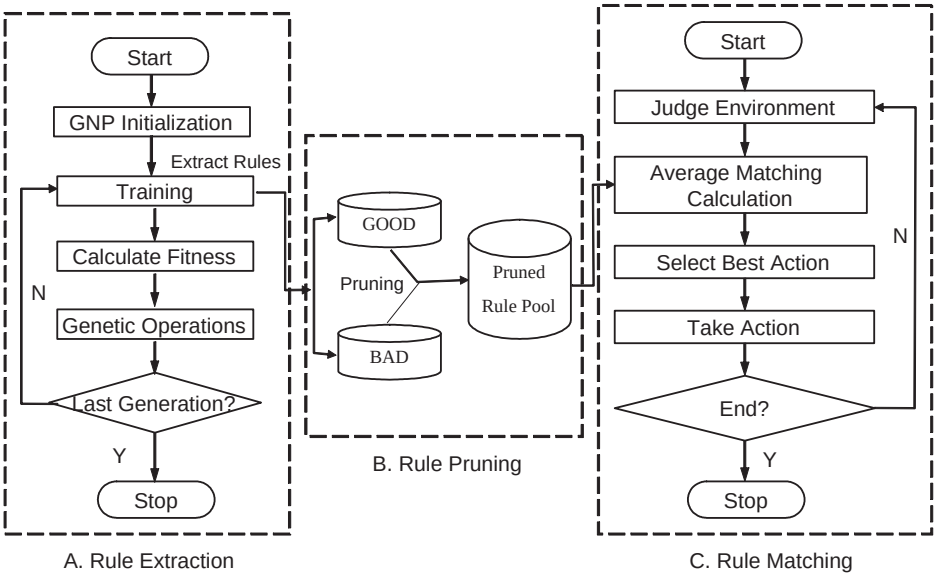

Figure 4.1: Flowchart of the proposed method.

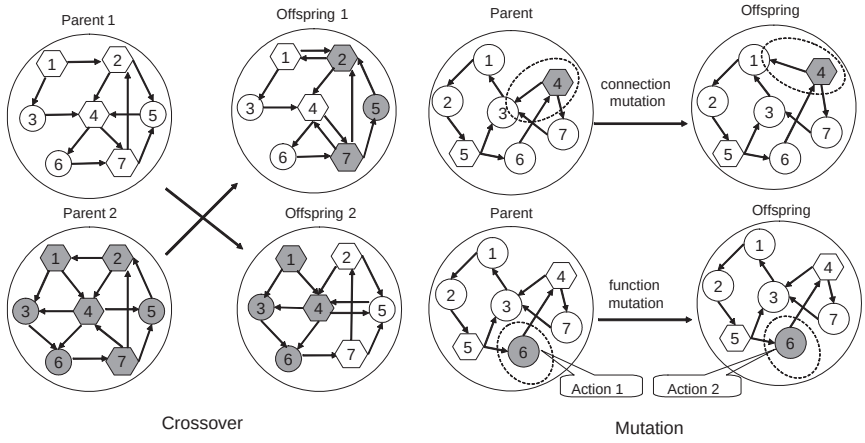

Figure 4.2: Genetic operators of GNP-RAP

Reinforcement learning is a machine learning technique which enables agents to interact with its environment through "trial-and-error"s. It can build a bridge between the dynamic execution of GNP-RAP and its static structure. Furthermore, the obtained knowledge during the program running could be obtained to create more effective rules, thus, in this paper, the diversified search of evolution (off-line) and intensified search of RL (on-line) are combined in order to improve the quality of the generated rules.

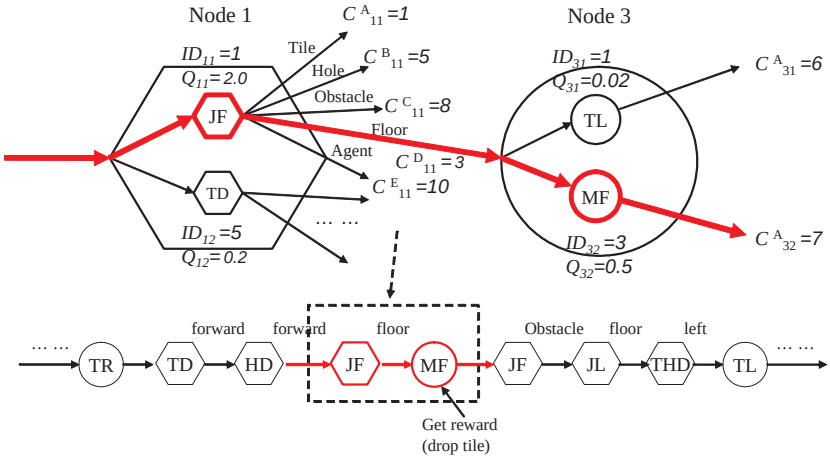

Figure 4.3: Node structure and transition of GNP-RAP

Among RL, Sarsa is an on-policy approach(42) which updates its state-action pairs based on the really taken actions, i.e., the real information during task execution. Therefore, Sarsa-learning is more appropriate for this paper and is selected as the learning strategy. In order to realize Sarsa-learning, sub-nodes are introduced into the judgment and processing nodes. Each node is a "state" and selection of a sub-node is regarded as an "action". The Q-value of each subnode estimates the sum of discounted rewards to be obtained in the future. Fig. 4.3 shows the node structure and node transition of GNP-RAP. The notation ID_{ij} is the identification number of subnode j in node i, and Q_{ij} is the Q-value of selecting this subnode; C^A_{ij}, C^B_{ij}, ... denote the next nodes to which subnode j in node i connects according to different judgment results A, B, ..., and so on. The selection of the sub-node is based on the ε-greedy policy, i. e., the sub-node with the highest Q-value is selected with the probability of 1-ε, or a sub-node is selected randomly with the probability of ε.

The Q-value of each state-action pair is updated by the following equation.

$$Q(i,a) \leftarrow Q(i,a) + \alpha[r(t) + \gamma Q(j,a') - Q(i,a)], \tag{4.1}$$

where, $Q(i,a)$ is the Q-value of state i when taking action a. a' is an action taken at state j. State j is the next node of node i. At the beginning of learning, all the Q-values in the Q-table are initialized at 0.

However, because of the ε-greedy policy of Sarsa, some unreasonable connections could be visited, bringing some bad rules. Therefore, pruning of these rules becomes necessary. In the previous research, rules are only extracted from the best individuals

and store them as good experiences. In this chapter, the rules are extracted from the best R individuals (with the highest fitness) throughout the generations and are stored into the GOOD rule pool, which represents the good experience set of agent's historical behaviors. Meanwhile,the rules are also extracted from the worst R individuals and stored into the BAD rule pool, which represents the bad experiences, i.e., the passive precepts that tell the agent to avoid some particular actions. For simplicity, the order of rules extracted by GNP-RAP is set at one.

4.2.3 Rule pruning

Generally speaking, the rules in the GOOD rule pool have good experiences which are helpful for guiding agent's actions. However, it also contains some bad rules which contribute to nothing, or even mislead the agent. On the other hand, among all the rules in the rule pool, it is hard to tell which one is good and which one is bad. It is noticed that the bad rules appear more frequently in the worst individuals. Thus, we could use the BAD rule pool to find out the bad rules in the GOOD rule pool. A simple method is to check the rules in the GOOD and BAD rule pool one by one, and find the overlapped rules. Four methods are proposed to deal with the overlapped rules.

Method 1: Deleting Overlapped Rules. The overlapped rules contain some bad rules generated by the good individuals. If they are pruned from the GOOD rule pool, the really good ones could stand out and contribute more to the final reward. A simple method is to delete all the overlapped rules, and use the rest rules in the GOOD rule pool for decision making.

However, because rules are extracted from R individuals of both the best and the worst individuals. When R increases, there exists a possibility that the worst R individuals could generate some good rules, which may be pruned mistakenly by simply deleting them. Meanwhile, in the overlapped rules, there exist some neutral rules, i.e., those who don't directly contribute to the final reward, but are essential for taking actions. For example, in the tile-world problem, some "stay" actions of agents don't directly contribute to dropping the tiles into holes, but they are essential for agents' cooperation to achieve the global optimal solutions. Some reasonable stays of agents could bring more rewards in the future. In order to avoid deleting these neutral rules, the partial use of the overlapped rules becomes sensible. A simple approach is to decrease the strengths of the overlapped rules by the following methods.

Method 2: Pruning using Subtracted Strength. In human society, a person is evaluated by both his good behaviors and bad behaviors. His good behaviors are discounted if he commits some bad behaviors. Similarly, each overlapped rule has

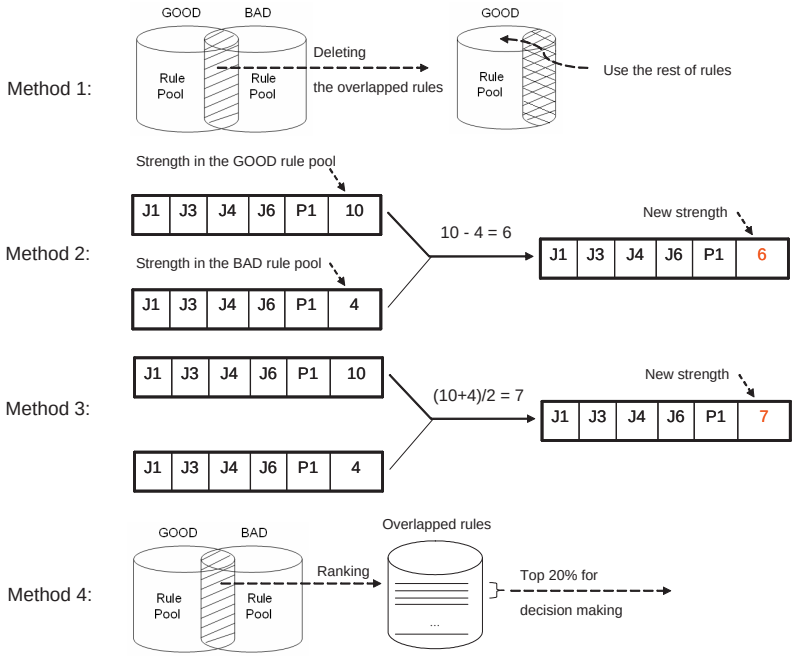

Figure 4.4: Examples of different pruning methods.

its strength value corresponding to the positive and negative contributions to agent's actions. The strength of the rule in the GOOD rule pool is subtracted by its strength in the BAD rule pool, so that its contribution to agent's behaviors could be fairly evaluated. For example, if the strength of an overlapped rule in the GOOD rule pool is a and its strength in the BAD rule pool is b, then the strength of the rule is set at a-b.

Method 3: Pruning using Averaged Strength. Method 3 regards the two strength values of the same rule as the contributions to the higher and lower fitness, respectively. Actually, the worst individuals could evolve and the extracted rules by them also contribute to some rewards. In order to fairly evaluate the rule, the average of the two strength values is calculated to represent its real contribution to the final fitness. For example, if the strength of an overlapped rule in the GOOD rule pool is a and its strength in the BAD rule pool is b, the strength of this rule becomes (a+b)/2.

Method 4: Pruning using Ranking. Since the overlapped rules have different strength values, a reasonable method is to rank all the overlapped rules according to their strength values in the GOOD rule pool and use only highly ranked ones for decision making. In this paper, the top 20% of them are selected. Fig. 4.4 gives some

examples on the different pruning methods.

4.2.4 Rule matching

This step focuses on how to make use of the good rules in the pruned rule pool. The matching method is the same as chapter 3, rules are matched through an "average matching degree calculation". The matched rules are picked up to calculate the average matching degree, which represents the general similarity between the current environment and all different situations recorded in the rules. Finally, the class with the most similar situations to the current situation is selected, and its processing is executed.

4.3 Simulation

4.3.1 Simulation environment

Dynamic tile-world is chosen as the simulation environment to test the performance of GNP-RAP. In dynamic tile-worlds, the locations of the tiles and holes are randomly distributed. Agents should distinguish different objects and take appropriative actions in different situations. The task of the agents is to move tiles into holes as many as and as quickly as possible without hitting obstacles or dropping themselves into the holes. After a tile is dropped into a hole, they disappear to form a floor, and a new tile and a new hole appear at random positions. This dynamic tile-world is a great challenge to agent control problems because the world information is unpredictable beforehand and agents have limited sight. Therefore, the cooperation between agents becomes more difficult and challengeable than the static tile-world. Furthermore, a broader map with the size of 20×20 is used as the simulation environment. Fig. 4.5 shows an example of the dynamic tile-world.

The number of rule generators R is set at 5. The rules from the best five individuals are extracted and stored into the GOOD rule pool. Likewise, the rules extracted from the worst five individuals are stored into the BAD rule pool. The evolution of GNP lasts for 1500 generations, and 300 time steps are assigned for agents to take actions. The number of agents is initialized at 3. 10 tile-worlds are used as training instances in order to extract enough rules, where 30 tiles and 30 holes are randomly distributed at the beginning of evolution. In the testing phase, 10 new tile-worlds with different locations of tiles and holes are used to test the performance of the proposed method. The program is executed for 30 times with different random seeds in order to get reliable results. The parameter configuration for the simulation is described by Table 4.1.

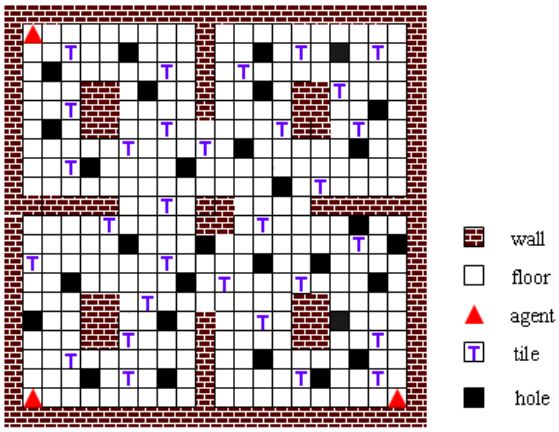

Figure 4.5: An example of the tile-world

In the simulation, the proposed method is compared with the classical methods such as GP and GNP. For GP, the function nodes are selected from the judgment nodes J1∼J8, and the terminal nodes are selected from the processing nodes P1∼P4. The function nodes have multiple arguments corresponding to different judgment results and the terminal nodes have no argument. The maximum depth of GP is set at 4 in order to avoid bloating. For fair comparison, parameters of GP are also set as Table 4.1. Since each non-terminal node has five outputs, the total number of nodes in GP is 781. The best individual of GP is selected to guide agents' actions. For GNP, the total number of nodes is set at 60, including 40 judgment nodes and 20 processing nodes.

4.3.2 Fitness function

Fitness function of the dynamic tile-world is defined as the sum of total rewards obtained within the assigned time steps, as shown by Eq. (4.2).

$$fit = \sum_{j=1}^{N} reward[j], \tag{4.2}$$

where, $reward[j]$ is the j^{th} reward and N is the total number of rewards. Reward is 1 if a tile is successfully dropped into a hole, otherwise it is 0.

4.3.3 Results and analysis

In the following simulation, GNP-RAP is compared with some conventional methods such as GP and GNP. Fig. 4.6 shows the training results of different methods, and Fig.

Table 4.1: Simulation conditions

Evolution		Sarsa-learning	
Population	300		
Mutation	170		
Crossover	120		
Elite Number	5		
Worst Number	5		
Mutation Rate P_m	0.01	Learning Rate α	0.9
Crossover Rate P_c	0.1	Discount Rate γ	0.9
Generations	1500	ε	0.01, 0.1, 0.2

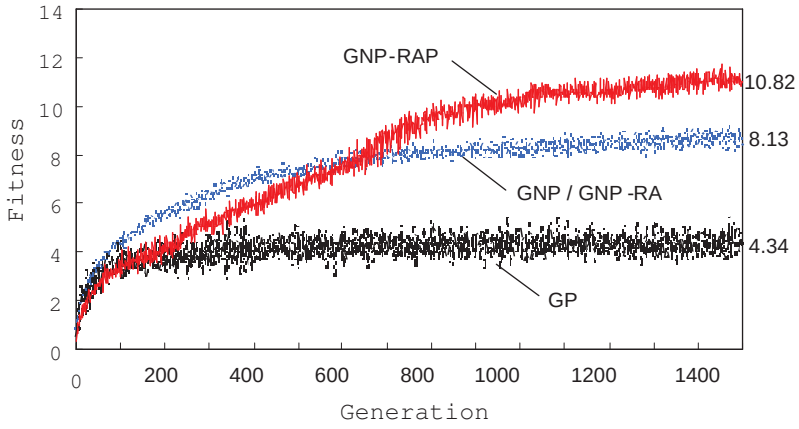

Figure 4.6: Training results of the proposed method

4.7 shows the testing results which are averaged over 30 independent simulations.

GP increases its fitness relatively slowly and also its testing results are not satisfactory. This is because GP has too many nodes. For example, the total number of nodes in GP is 781 if the maximum depth is 4. The decision tree becomes very large and hard to evolve, which over-fits the training data easily. That is why GP can not increase its fitness too much in later generations. In the testing phase, such a large structure can not adapt to the environment changes easily, and its generalization ability is not good.

GNP shows better performance than GP(in the training phase, GNP and GNP-RA are the same). This is because GNP can reuse the nodes to create compact programs.

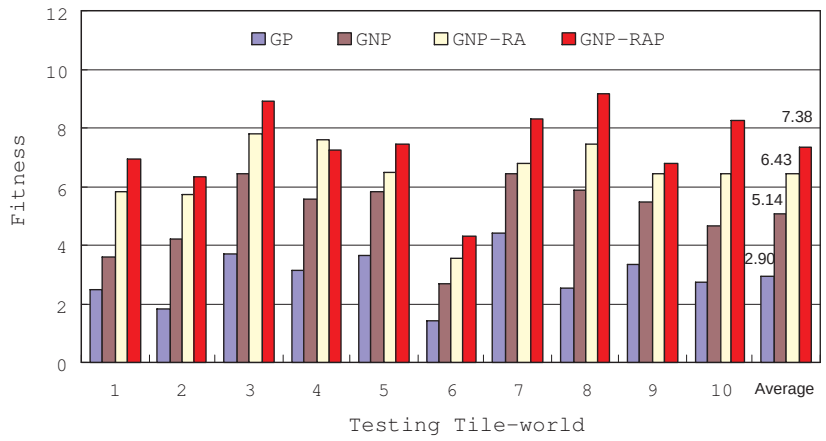

Figure 4.7: Testing results of the proposed method

60 nodes are enough for GNP to deal with complicated problems and GNP never causes bloating. However, GNP has many connections which make the gene structure very complex. Therefore, the generalization ability of GNP is not good and its testing results are relatively low.

GNP-RAP achieves lower fitness than GNP in earlier generations because it has to update its state-action pairs gradually. In later generations, it outperforms GNP. This is because GNP-RAP combines evolution and reinforcement learning in its program. The obtained knowledge during the running of the program could be used to create more reasonable combinations of judgments for processings (rules). In the testing phase, GNP-RAP and GNP-RA could get higher fitnesses than GP and GNP, which demonstrates the effectiveness of the rule-based model. This is because, GNP-RAP and GNP-RA use a large number of rules to guide agents' actions. These rules are very general which can overcome the over-fitting problem. Furthermore, many past experiences could be retrieved to make the current decision, which can guide agents' actions more accurately.

In this simulation , GNP-RAP is compared with the previous GNP-RA method. How ε affects the proposed method in both training and testing is also studied. The pruning method of GNP-RAP is to simply delete the overlapped rules.

Fig. 4.8 shows the training results of GNP-RAP when ε is 0.01, 0.1 and 0.2, respectively. From the results, it is noticed that the case of $\varepsilon = 0.2$ increases the fitness value fastest and $\varepsilon = 0.01$ increases it slowest in earlier generations. This is because relatively larger ε enables GNP-RAP to search for the optimal solutions in a broader

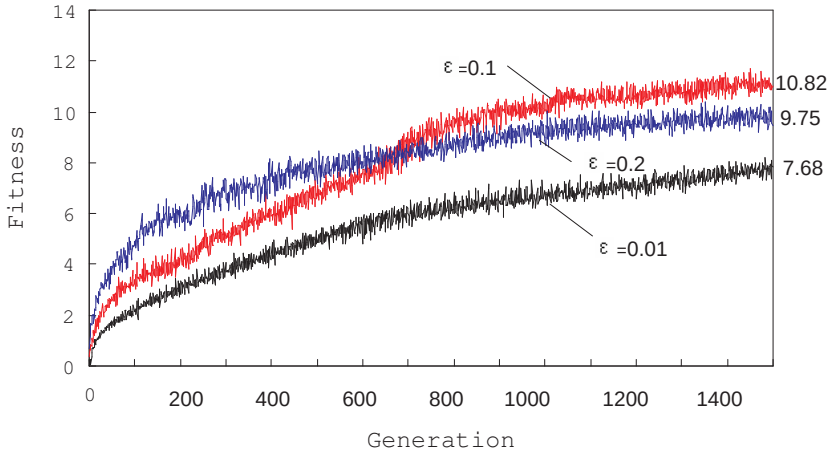

Figure 4.8: Training results of GNP-RAP

space, which helps to increase the possibility of generating good solutions. In later generations, the results of $\varepsilon = 0.01$ and $\varepsilon = 0.2$ are worse than that of $\varepsilon = 0.1$. $\varepsilon = 0.01$ enables GNP-RAP to over-exploit the obtained information and its exploration ability is not satisfactory, thus it can be easily trapped into the local minima. $\varepsilon = 0.2$ explores the solution space sufficiently, but its exploitation ability is not good. The case of ε=0.1 could get comparatively better results in later generations probably because its exploration and exploitation ability is well balanced.

Fig. 4.9 shows the testing results of GNP-RA and GNP-RAP with different εs. The x axis is the testing tile-world 1-10 and the y axis is the fitness. GNP-RAP shows better performance than GNP-RA in all three cases. This is because GNP-RAP could prune the unimportant rules, which helps to improve the quality of the rule pool. Furthermore, some specific rules which represent the specific situations could be deleted. This contributes to improving the generalization ability of the rule-based model.

In this simulation, the effectiveness of 4 different pruning methods is studied and analyzed. The results of the previous GNP-RA method (without rule pruning) are also provided as a benchmark.

Fig. 4.10 shows the testing results of GNP-RA and GNP-RAP(Method 1 to Method 4). Generally speaking, the four pruning methods outperform GNP-RA without rule pruning in most testing instances. Table 4.2 shows the average fitness values of the five methods under different εs. From the table we could see that the average fitness

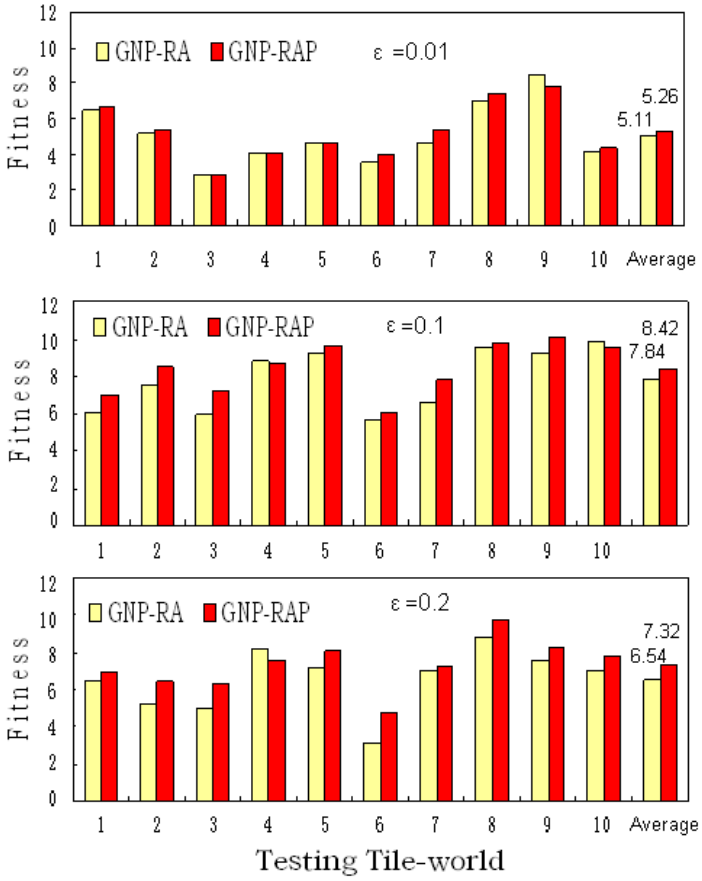

Figure 4.9: Testing results of GNP-RA and GNP-RAP

values of the four pruning methods are higher than that of the conventional GNP-RA method, which demonstrates the effectiveness of the proposed method. After pruning the overlapped rules, the really good rules in the rule pool could stand out and contribute more to the final reward. Method 1 is a little better than GNP-RA, but the improvement is not satisfactory. This is because deleting all the overlapped rules could reduce the impact of the bad rules. Actually, some good rules generated by bad individuals as well as the useful neutral rules could also be deleted mistakenly. Method 4 doesn't improve the performance too much because in earlier generations, the strength values of the good rules are generally low. Simply ranking the overlapped rules could not distinguish the good rules among them, and some good rules in earlier generations are lost. Method 2 and Method 3 could get relatively higher results because reducing the strength values of the overlapped rules could reduce the influence of the bad rules,

Table 4.2: Average fitness value of the testing results

ε	GNP-RA	Method 1	Method 2	Method 3	Method 4
0.01	5.11	5.26	5.72	5.90	5.01
0.1	7.84	8.42	9.31	9.64	8.55
0.2	6.54	7.32	8.87	9.29	7.70

Table 4.3: Number of rules under different εs

ε	GOOD Pool	BAD Pool	Overlapped Rules	Proportion
0.01	3662	4269	138	3.76%
0.1	4983	5576	396	7.94%
0.2	7453	8626	865	11.61%

so that they don't interfere the decision making so much. On the other hand, it could avoid deleting the neutral rules in the overlapped rules by mistake.

It is noticed that when ε is larger, the improvements of the proposed method become more obvious. Table 4.3 shows the number of rules in the GOOD and BAD rule pool as well as the proportions of the overlapped rules under different εs. It is noticed that when ε increases from 0.01 to 0.2, the number of rules in both GOOD rule pool and BAD rule pool increases. Meanwhile, the proportion of the overlapped rules is also increasing. This is because larger εs enable agents to search for the optimal solutions in a broader space, so that state-action pairs with lower Q-values could be visited, bringing more rules into the rule pool. However, when ε is too large, the quality of the rule pool may decrease. When ε is 0.01, the proportion of the overlapped rules is merely 3.76%. Pruning such few rules doesn't make too much difference, so that the improvements of the four methods are not obvious. However, when ε turns to 0.2, the proportion of the overlapped rules comes to 11.61%, thus the improvements of the proposed method become more apparent.

Since rules are extracted from both the best and worst R individuals, and how R affects the proposed method is studied.

Table 4.4 shows the average fitness values of GNP-RA and GNP-RAP under different Rs. For simplicity, we choose Method 1 as the pruning method. It is noticed that when R increases, both the performance of GNP-RA and GNP-RA with rule pruning

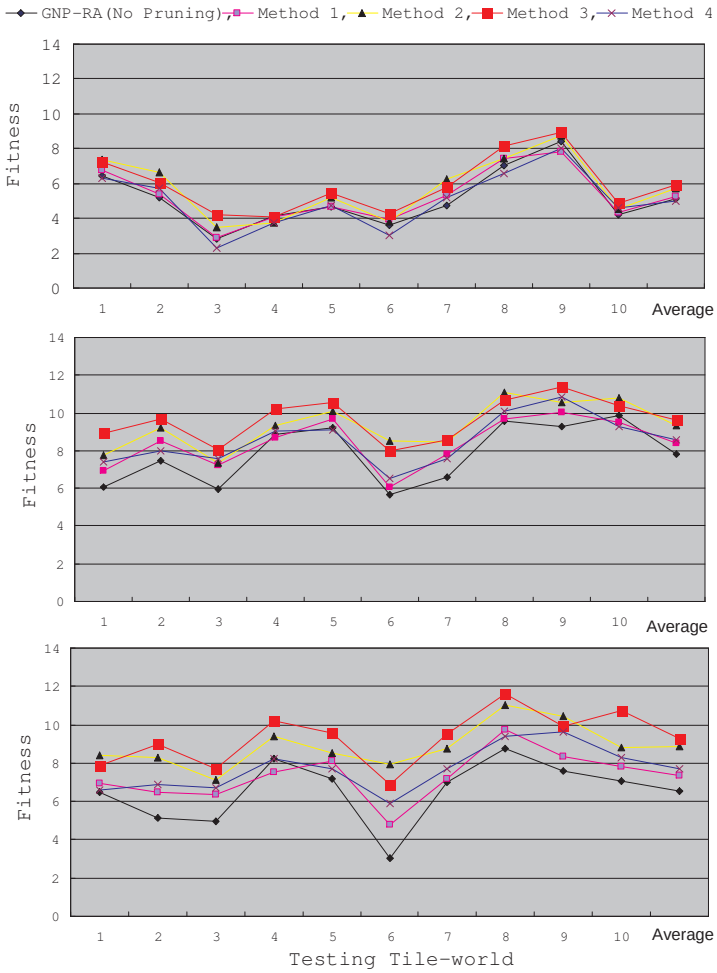

Figure 4.10: Testing results of different pruning methods with different εs

increases. This is because more individuals could generate more rules which means more experiences for determining actions. However, when R is 50, the performance decreases because rules generated from not so good individuals decrease the quality of the rule pool. Meanwhile, the proportion of the overlapped rules in the GOOD rule pool increases together with R, because larger Rs increase the possibility of generating more common rules between the best and worst individuals. When R is 1, the improvement rate of the proposed method over conventional GNP-RA method is 4.63%, which is not obvious enough because the proportion of overlapped rules is merely 2.7%. Pruning of such few rules doesn't work too much. When R turns to 5 and 10, the improvements are more satisfactory since a good number of bad rules are pruned to increase the quality

Table 4.4: Average fitness value under different Rs

R	GNP-RA	GNP-RAP	Overlapped Rules	Improvement
1	5.62	5.88	2.7%	4.63%
5	7.84	8.42	7.94%	7.40%
10	8.23	8.72	8.68%	5.95%
50	8.07	8.39	11.23%	3.97%

of the rule pool. Nevertheless, when R turns to 50, the improvement rate decreases. This is because the worst individuals could also generate some good rules, and too large R could increase the chance to prune some good rules mistakenly. Thus, selecting a proper number of rules in the rule pools is important for the proposed method. Rs of 5 and 10 seem to be the optimal ones in this simulation.

4.4 Summary

In this paper, a new rule-based model named "GNP with Rule Accumulation and Pruning" is proposed. The purpose is to improve the performance of GNP-RA by improving the quality of its rules. In order to realize this, GNP-RAP combines off-line evolution and on-line learning to create more efficient rules. During the rule generation, the diversified search of evolution and intensified search of reinforcement learning work together to create more reasonable combinations of the judgement nodes for a processing. The obtained knowledge (Q table) during the program running could be used to create more efficient rules, which helps to improve the quality of the rule pool. After the rules are generated, a unique rule pruning approach using bad individuals is designed in this chapter. The general idea is to use bad individuals as rule monitors to filter the bad rules in the rule pool. Four methods for pruning the overlapped rules are designed, and their performances are systematically analyzed. Simulations on the tileworld demonstrate the effectiveness of the proposed method over the classical methods and the previous work.

5

Genetic Network Programming with Updating Rule Accumulation to Improve Adaptability in Dynamic environments

5.1 Introduction

In this chapter, it is studied how to improve the adaptability of GNP-RA in accordance with the changing environments. As an important feature of agent-based systems, adaptability (78, 79, 80, 81) is to be understood as the ability to adapt themselves quickly and efficiently to changed environments. An adaptive system is therefore an open system(82, 83) that is able to fit its behaviors or update parts of itself according to the changes in its environment. Adaptability is also an important feature in real-world applications, for example, when a best trained product with the optimal policy is used, it is expected to adapt to the new environment.

5.1.1 Problem to be solved

One problem of GNP-RA is that *its adaptability is very poor*. In GNP-RA, the best individuals in each generation are used to generate action rules, which are stored into the rule pool. These rules remain the same without any change throughout the testing

phase. However, the environment is dynamic and changing all the time, where new situations appear frequently. The old rules in the rule pool are becoming incompetent for guiding agent's actions, which cannot deal with changing environments efficiently. Therefore, updating them becomes necessary. *How to update the rules in the rule pool in accordance with the environment changes* is the main problem to be solved in this chapter.

5.1.2 Motivation

Adaptability in changing environments has becomes a hot topic for decades. EC proves to be an effective method to solve dynamic optimization problems because of its global search ability in uncertain environments(84, 85). Recently, machine learning (ML) is witnessed to play an important role to enhance EC methods(86, 87). For example, EC with memory schemes(88) could store historical information into memory to generate better solutions. A variable relocation approach(89) was proposed to re-initialize the population after a change occurred. Reinforcement learning (RL)(90) could also improve the robustness through trial-and-error interaction with dynamic environments. However, its state-actions pairs will exponentially increase when the problem is complicated, which decrease its performance(87). In real world applications, RL is often combined with EC to create more robust individuals(91), since it combines the diversified search of EC and intensified search of RL. Up to now, most research focuses on the individual level, i.e., to generate the most adaptive individuals for dynamic environments. It remains a new topic on how to improve the adaptability through generating and updating rules.

5.1.3 Novelty of this chapter

Inspired by the above mentioned methods, this chapter proposes a novel rule-base model named GNP with Updating Rule Accumulation (GNP-URA) for the multi-agent control in changing environments. Different from the conventional methods, this chapter focuses on how to build a more adaptive rule pool, i.e., how to adapt the rule pool to the changing environments, so that *"rule pool adapting to changing environments"* is the novel point of this chapter.

In order to realize this, Sarsa-learning is used as a tool to generate *better*, *more* and *newer* rules for GNP-URA, which is helpful to increase its adaptability. Firstly, Sarsa can select the really important judgments and actions in the current situation, generating better rules. Secondly, the ε-greedy policy of Sarsa enables GNP-URA to explore the solution space sufficiently, so that some unvisited transitions could be

visited. This contributes to generating more rules, which means more experiences for agent control. Most importantly, when evolution ends, Sarsa could also updates its policy by on-line learning in the new environments. Some new situations could be recorded as new rules to update the old ones in the rule pool. Thus, the rule pool could adapt to new environments.

The rest of this chapter is organized as follows. Section 5.2 describes the general framework and the detailed algorithm of the proposed method. Section 5.3 compares the performance of different methods, and analyzes the reasons for improvements. Section 5.4 is devoted to a brief summary.

5.2 Genetic Network Programming with Updating Rule Accumulation

In this section, a rule-based model with higher adaptability named GNP-URA is proposed to guide agent's actions in changing environments. How to generate rules through evolution and learning in the training phase, how to update the rules through Sarsa-learning in the testing phase and how to guide agent's actions using the rule pool are explained in details.

5.2.1 General framework of GNP-URA

Fig. 5.1 shows the framework of the proposed method, which contains three phases. In the training phase, the population is trained by dynamic tile-worlds, and rules generated by the best individual are stored in the rule pool generation by generation. When evolution ends, it comes to the testing phase, where the environment keeps changing. The best individual continues to learn in such environments and the rule generation is done successively. The generated new rules in the new environments are used to update the old rules in the rule pool. In the testing phase, all rules in the updated rule pool are used for determining agent's actions through a unique matching calculation.

5.2.2 Node structure of GNP-URA

Fig. 5.2 shows the node structure of the proposed method. In order to realize Sarsa-learning, each node in GNP-URA has a macro node. Sub-nodes are introduced into each macro node as alternative functions. In this structure, each macro node is regarded as a "state" and selection of a sub-node is regarded as an "action". ID_{ij} is the identical number of sub-node j in node i, and Q_{ij} is the Q-value of selecting this

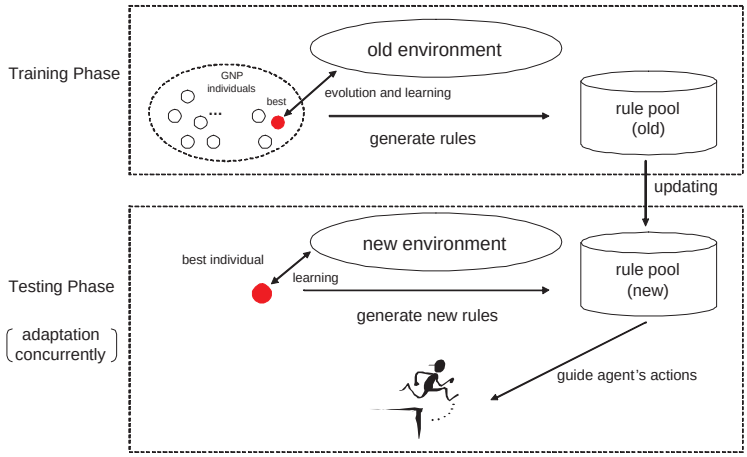

Figure 5.1: General framework of GNP-URA

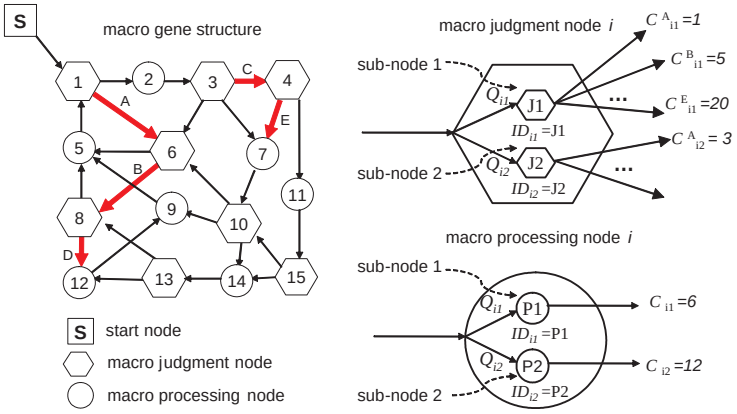

Figure 5.2: Node structure of the proposed method

sub-node (the number of sub-nodes is set at 2 in this chapter); C_{ij}^{A}, C_{ij}^{B}, ... denote the next nodes to which sub-node j in node i connects according to different judgment results A, B, ..., and so on. Q-values estimate the sum of discounted rewards to be obtained in the future, which represents the sub-node's importance. The selection of the sub-node is based on the ε-greedy policy, that is, the sub-node with the highest Q-value is selected with the probability of 1-ε, or a sub-node is selected randomly with the probability of ε.

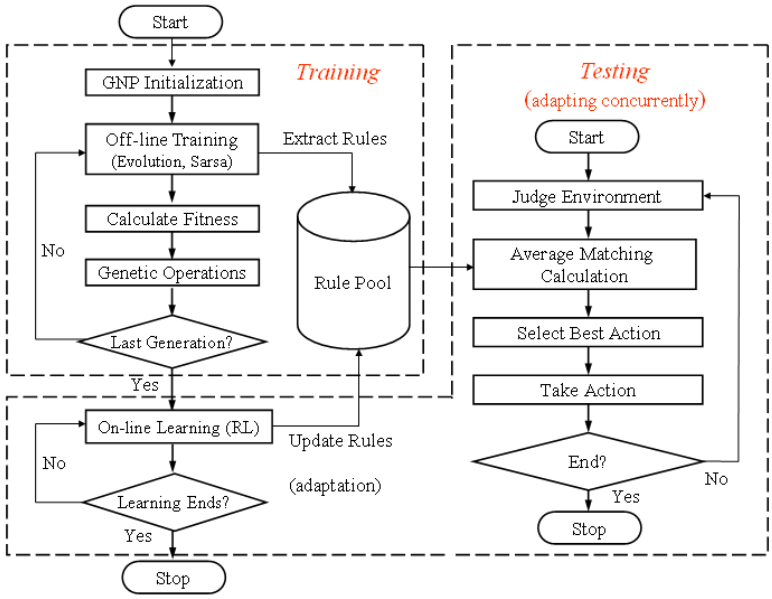

Figure 5.3: Flowchart of GNP-URA

5.2.3 Rule description

In GNP-RA, the rule is defined as a sequence of successive (macro)judgment nodes with their judgment results and the succeeding (macro)processing node. For example, in Fig. 5.2, $J_1^A \rightarrow J_6^B \rightarrow J_8^D \rightarrow P_{12}$ is a rule, and $J_3^C \rightarrow J_4^E \rightarrow P_7$ could be regarded as another rule, where *A, B, C, D and E* in the superscripts denote the judgment results. However, in GNP-URA, the actually visited sub-nodes are recorded and regarded as the ingredients of the rule.

5.2.4 Flowchart of GNP-URA

Fig. 5.3 describes the flowchart of the proposed method, whose detailed algorithms are explained in the following three sub-sections.

5.2.5 Rule generation in the training phase

In the training phase, evolution (diversified search) is combined with Sarsa-learning (intensified search) in order to generate better rules. Genetic operators such as crossover and mutation could change the gene structure largely, so that the best combinations of judgments with a processing could be obtained. Evolution enables GNP-URA to generate a great number of useful rules quickly.

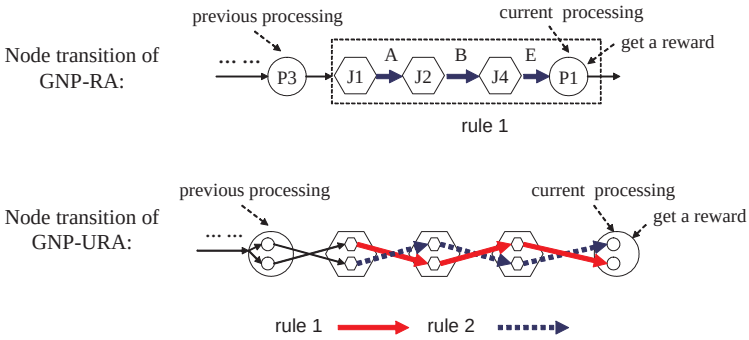

Figure 5.4: Comparison of different node transitions

Sarsa-learning could update its policies, i.e., Q-values of the state-action pairs based on the real information obtained during task execution, since the really important judgments for a processing in the current situation could be selected. This helps GNP-URA to generate more reasonable rules for different situations, namely, better rules. Furthermore, the ε-greedy policy of Sarsa-learning could bring more rules for decision making. Fig. 5.4 shows the node transitions of GNP-RA and GNP-URA, from which it can be seen that only one rule could be generated in the node transition of GNP-RA, while two(or even more) rules could be generated by GNP-URA.

The strength of the rules is defined the same as that in the previous chapter.

5.2.6 Rule updating in the testing phase

When the training phase terminates, it comes to the adapting phase, in which the rule pool is updated depending on the changing environments by updating the rules in it. The best individual obtained in the training phase continues learning in the changing environments. There is no genetic operation such as crossover and mutation, so that the gene structure of the best individual doesn't change. However, its node transition could be changed by updating the Q-values of the sub-nodes in each judgment and processing node. The best individual interacts with the new environments and get rewards. Once a reward is obtained, Q-values of the visited sub-nodes which reflect this reward are updated as follows.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)], \tag{5.1}$$

where, $Q(s_t, a_t)$ is the Q-value of the pair of state s_t and action a_t, r_t is a reward from the new environment, α is a learning rate and γ is a discount rate.

Changing the Q-values of the sub-nodes could change the node transition of GNP-URA. In the new node transition, the visited nodes are recorded and stored as new rules in the rule pool. These new rules represent some unexperienced situations, which are different from those in the training phase. These rules are used to update the old rules in the rule pool, so that the rule pool could "learn" and adapt to the changing environments.

Once a new rule is generated, the rules in the rule pool are updated under the following policy: If the new rule and the old rule in the rule pool are the same(this rule already exists in the rule pool), and if the strength of the new rule is higher than that of the old rule, update the strength of the old rule to the higher one, otherwise the strength remains the same. If a new rule doesn't exist in the rule pool, then directly add it into the rule pool.

5.2.7 Rule matching and action taking in the testing phase

The average matching degree method is also used in this chapter to match rules with the environment data in different classes. The class with the highest average matching degree is picked up, and its action is taken by the agent. After that, the environment is updated and a new environment data could be obtained for the next step.

5.3 Simulations

5.3.1 Dynamic Tile-world

Dynamic tile-world is used in this chapter to testify the adaptability of the proposed method in comparison with the conventional methods. In the dynamic tile-world, tiles and holes are randomly distributed in different positions. After a tile is dropped into a hole, they disappear to form a floor, and a new tile and a new hole appear at random locations. Dynamic tile-world is an excellent test-bed of adaptability since the environment keeps changing all the time, and the changes are unpredictable before hand. The size of the dynamic tile-world is set at 20×20, where 30 tiles and 30 holes are randomly distributed. The number of agents is set at 3. In the training phase, evolution lasts for 1000 generations with each generation having 300 action steps. After that, the environment keeps changing continuously for 1000 episodes and each episode has 300 action steps. The performances of different methods are tested in such environments.

Table 5.1: Parameter configuration for the simulation

Evolution		Sarsa-learning	
Population	300		
Mutation	175		
Crossover	120		
Best Individual	5		
Mutation Rate P_m	0.01	Learning Rate α	0.9
Crossover Rate P_c	0.1	Discount Rate γ	0.9
Generations	1000	ε	0.01, 0.1, 0.15, 0.2

5.3.2 Simulation configuration

In this simulation, GNP-URA is compared with GP, GNP and the conventional reinforcement learning method. For GNP-URA, the rule pool is used to guide the actions of the three agents. In the testing phase, after one episode ends, the rule pool is updated for the testing. Then, it comes to the next episode. Therefore, GNP-URA could react immediately once the change happens. For GP, the function nodes are selected from the judgment nodes J1∼J8, and the terminal nodes are selected from the processing nodes P1∼P4. The function nodes have multiple arguments corresponding to different judgment results and the terminal nodes have no argument. In order to avoid bloating, the maximum depth of GP is set as 4. The best individual of GP is selected to guide agents' actions. For conventional reinforcement learning method, Sarsa-learning is chosen for comparison in the testing phase. Here, "state" is defined as the full information of judgment nodes J1∼J8, and "action" is defined as one processing from P1∼P4. Since J1 to J4 return {1,2,3,4,5}, and J5 to J8 return {1,2,3,4}, the possible number of state-action pairs is $5^4 \times 4^4 \times 4 = 640,000$. Q-values of the state-action pairs are updated in the same way as Eq. (5.1). Table 5.1 shows the parameter configuration for the simulation.

5.3.3 Results and analysis

5.3.4 Training results of different methods

Fig. 5.5 shows the training results of different methods which are averaged from 30 independent simulations. The x-axis is generation(for RL, the x-axis is episode) and the y-axis is fitness. It can be seen that in earlier generations, GNP-RA increases its

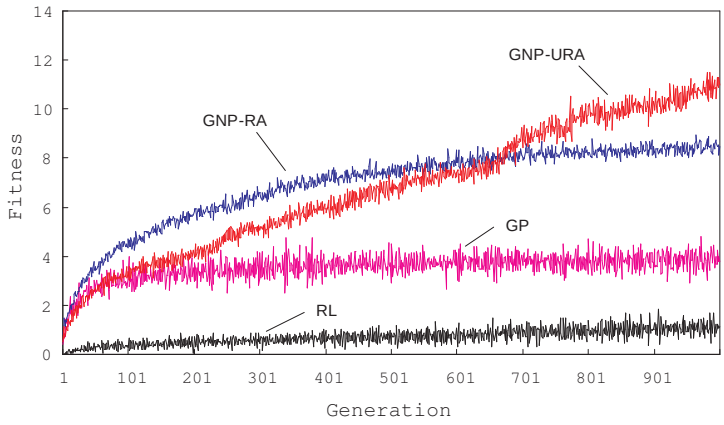

Figure 5.5: Training results of different methods

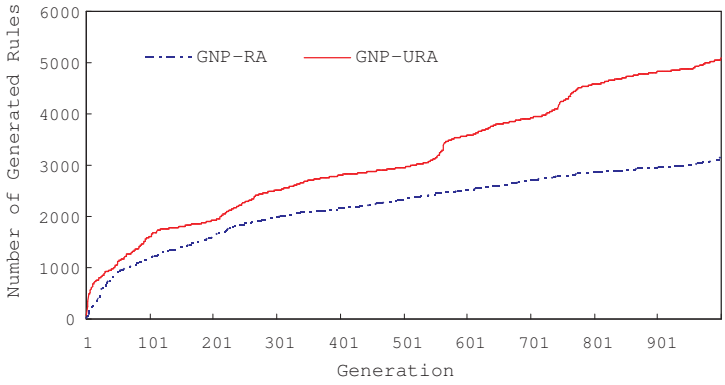

Figure 5.6: Number of generated rules in training

fitness faster than GNP-URA. This is because GNP-URA has to update its state-action pairs gradually through Sarsa-learning. However, in later generations, GNP-URA outperforms GNP-RA since it can make use of the information during task execution to generate more reasonable combinations of its nodes. This contributes to generating better rules. Furthermore, the ε-greedy policy of Sarsa could bring more useful rules into the rule pool. Fig. 5.6 shows the number of rules in the rule pool in each generation, from which it is noticed that GNP-URA could generate more rules than GNP-RA.

GP increases quickly in earlier generations, but converges to low fitness in later generations. This is because the expression ability of its tree structure is limited,

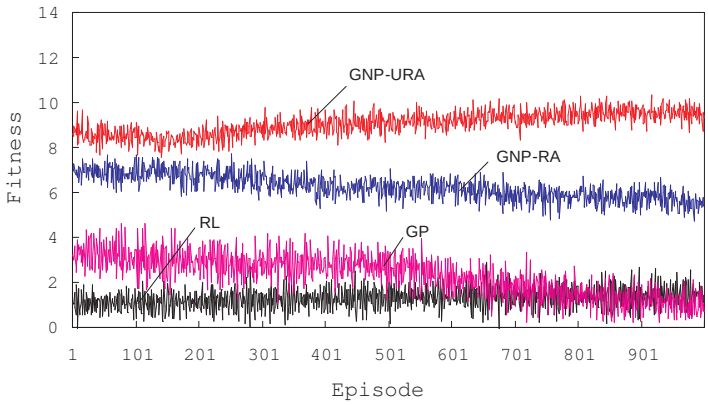

Figure 5.7: Testing results of different methods

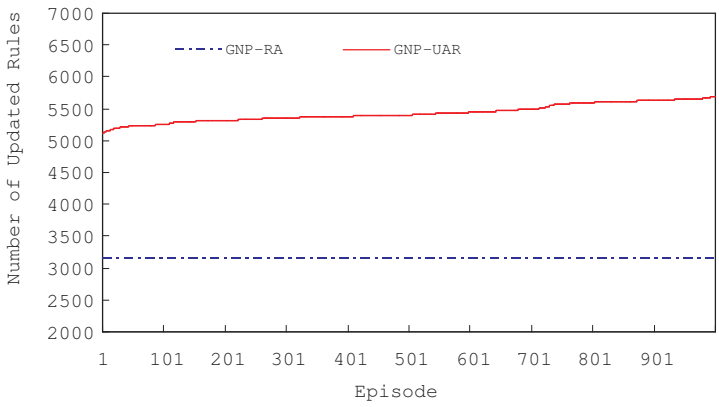

Figure 5.8: Number of rules in the testing phase

and tile-world is very difficult to solve using such a tree structure. However, if the depth of the tree is increased, the cost of memory and calculation time would increase dramatically. GNP-RA and GNP-URA could get relatively better training results since they can reuse the nodes to avoid bloating, and create general programs through partial observable processes. RL increases its fitness very slowly because it has too many state-action pairs, and the obtained rewards are not enough to update them.

5.3.5 Testing results of different methods

Fig. 5.7 describes the testing results of different methods. The x-axis is episode and y-axis is fitness. Generally speaking, GNP-URA could obtain higher testing results

than GNP-RA. This is because GNP-URA could generate better and more rules than GNP-RA in the training, which also helps to improve the adaptability in the testing. Moreover, it is noticed that the fitness curve of GNP-RA gradually decreases along with the environment changes. This is because the rules generated in the training phase fail to reflect the new situations in the changing environments. Many unexperienced situations appear frequently, therefore, the old rules are incompetent for guiding agent's actions. On the contrary, the fitness curve of GNP-URA gradually increases, which could be explained by the following reasons. Firstly, the same rule with higher strength as the one in the rule pool is used to update the old rule in the rule pool, which contributes to improve the performance of the rule. Secondly, some new rules which represent the unexperienced situations of the changing environment are added into the rule pool so that they could "learn" and adapt to the environment changes. Thirdly, the total number of rules in the rule pool is increasing, which means more experiences are accumulated for making decisions. Fig. 5.8 shows the number of rules in the testing phase, from which it is seen that the number of rules for GNP-URA is increasing, while that of GNP-RA remains the same.

GP achieves relatively low fitness which is decreasing in the testing phase. The limited size of its gene fails to generate adaptive individuals for the tile-world problem. Since there is no genetic operations such as crossover or mutation in the testing phase, it is impossible to change the gene structure of the best individual to adapt to the environment changes. Therefore, the adaptability of GP is not good. For RL, although it performance is gradually increasing in the testing episodes, its fitness is very low. This is because there are too many state-action pairs of RL and updating them takes very long time. The 1000 episodes are not enough for updating its Q-table.

5.3.6 Parameter discussion

As a sensitive parameter, ε balances the exploration and exploitation of reinforcement learning. If ε is too small, the program would be trapped into local minima with high probability. If ε is too large, the program would take too much random actions and the results are unstable. Fig. 5.9 shows the training results when ε is (0.01, 0.1, 0.15, 0.2). It can be seen that in earlier generations, when ε is large, the fitness curve increases quickly and fluctuates a lot. This is because large ε helps to explore the solution space sufficiently and generate best solutions quickly. However, its performance is not good in later generations since its exploitation ability is not satisfactory. On the other hand, lower ε takes too much time for exploring the search space, so its fitness

Table 5.2: Number of rules and average fitness with different εs

	ε=0.01	ε=0.1	ε=0.15	ε=0.2
Generated Rules	3427	5086	5743	7956
Updated Rules	254	610	726	1159
Updated Proportion	7.41%	11.99%	12.64%	14.57%
Average Fitness	7.25	9.36	9.04	6.48
Stand Deviation	0.615	0.448	0.519	0.641

curve increases slowly. 0.1 seems to be the optimal value of ε in training because it could well balance exploration and exploitation.

Fig. 5.10 shows the testing results with different εs. It is noticed that when ε is 0.1 and 0.15, the performance of GNP-URA increases during testing, while when it is 0.01 and 0.2, the performance decreases. This is because when ε is 0.01, not enough rules are generated to update the rule pool, so that it cannot adapt to the changes of environments. On the other hand, when ε is 0.2, GNP-URA takes too many random actions. The state-action pairs with very low Q-values could be selected with high probability. Therefore, many unimportant rules or even bad rule which mislead the agents are generated. These rules decrease the quality of the rule pool, so that its adaptability and robustness decrease. The cases of ε being 0.1 and 0.15 could get higher fitness values because they can balance the number of updated rules and the quality of these rules.

Fig. 5.11 and Fig. 5.12 show the number of rules in the training and testing phase, respectively. Table 5.2 shows the total number of generated rules and updated rules, and the average fitness values in the testing phase. It is noticed that when ε is large, GNP-URA could generate rules quickly and generate more rules in both training and testing. However, this does not mean the more the better. A proper number of rules with high quality are helpful for guiding agents' actions, so that balancing the number of rules and the quality of these rules is necessary. ε being 0.1 and 0.15 could obtain better results in the testing phase.

5.3.7 Agents' traces

Fig. 5.13 shows the traces of different agents in the changing environments. For simplicity, we use a small map(10×10) which has 3 tiles and 3 holes. 60 action steps are assigned to 3 agents for taking actions. When tiles are dropped into holes, new

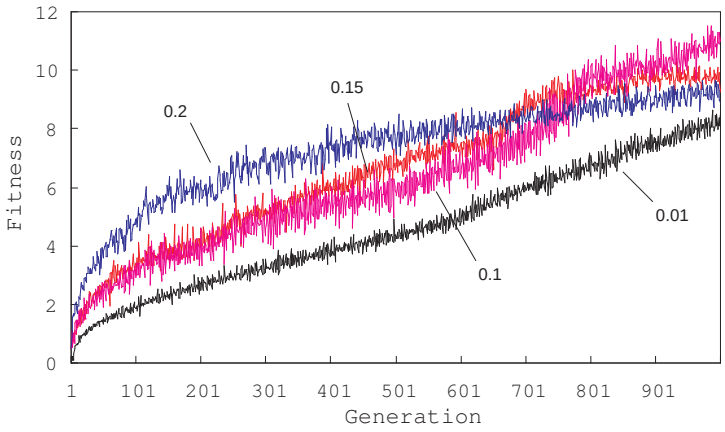

Figure 5.9: Training results with different εs

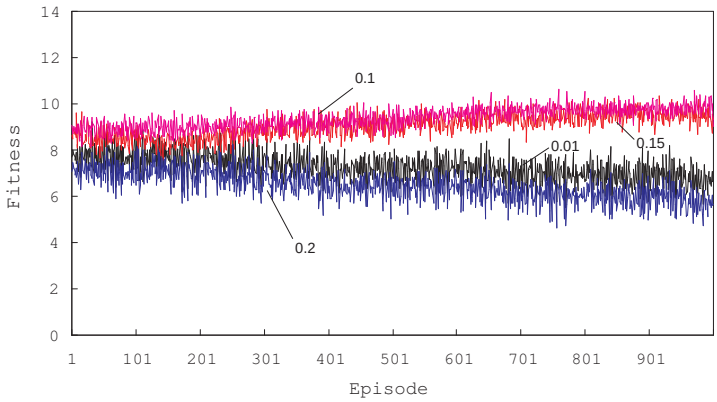

Figure 5.10: Testing results with different εs

tiles and new holes appear at different positions, then the old environment turns to a new one. GNP-RA could drop 3 tiles within the 60 steps in the old environment, while in the new environment, it only drops 1. Besides, the traces of the three agents don't change too much. This is because GNP-RA always uses the old rules to guide agents' actions, which cannot cover the new situations in the new environment. Therefore, the adaptability of GNP-RA to the changing environments is not good. On the other hand, GNP-URA successfully drops all the tiles into holes, although the environment is different from the old one. Furthermore, GNP-URA could change the traces of agents largely to adjust to the environment changes. This is because GNP-URA could update its rule pool through Sarsa-learning and generate many new rules which implies what

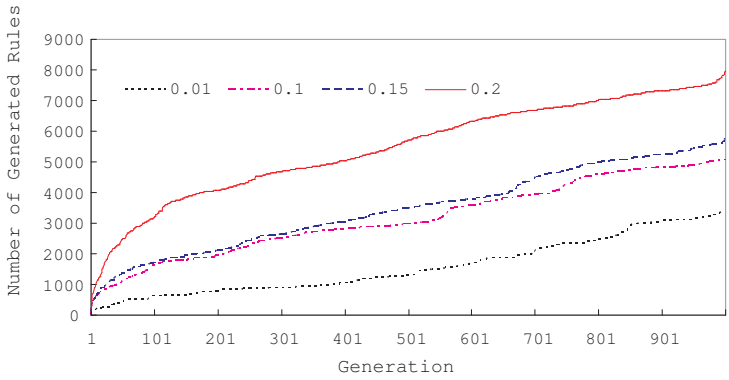

Figure 5.11: Number of generated rules in training

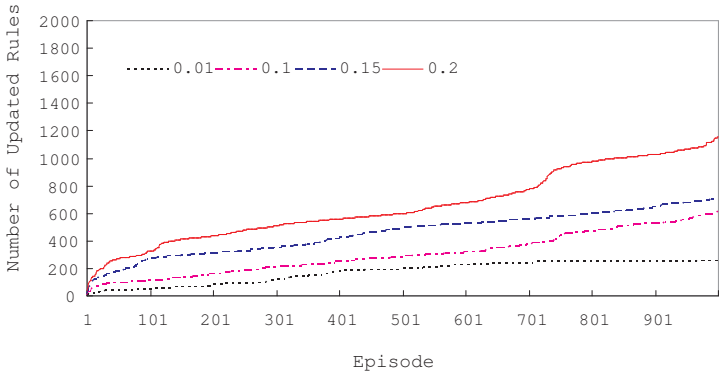

Figure 5.12: Number of updated rules in testing

agents should do in new environments. Thus, the adaptability of the proposed method to the changing environment is improved.

5.4 Summary

In this chapter, a more adaptive rule-based model named "GNP with Updating Rule Accumulation" is proposed for multi-agent control in changing environments. The purpose is to increase the adaptability and robustness of the rule pool to cope with the environment changes. In order to realize this, Sarsa-learning is introduced into the rule-based model to generate new rules to update the old ones in the testing phase. In addition, combining evolution and Sarsa-learning could generate better and more rules in the training phase, which also helps to improve the adaptability in the testing

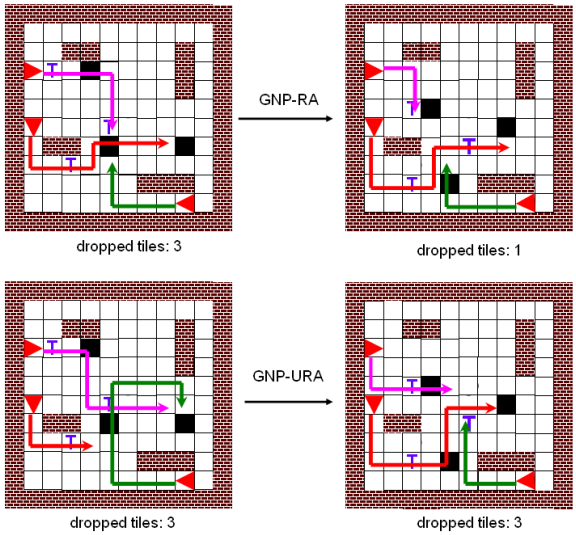

Figure 5.13: Agent' traces using different methods

phase. In new environments, some unexperienced situations are recorded into the new rules, which are used to replace the old rules in the rule pool. Therefore, the rule could adapt itself gradually to the environment changes. Simulations on the tile-world problem demonstrate the effectiveness of the proposed method over the previous work, GP and reinforcement learning. The proposed rule-based model could guide agents' actions properly even if the environments keep changing. The results also show that proper ε should be tuned in order to balance the number of updated rules and the quality of these rules.

6

Credit Genetic Network Programming with Rule Accumulation to Improve the Generalization Ability

6.1 Introduction

In this chapter, we focus on how to improve the generalization ability of GNP-RA in both training and testing. Generalization ability(92, 93, 94, 95) refers to the ability of an agent based system to cope with unexpected situations and continue to work efficiently without retraining. It is an important standard to evaluate the agent based systems in new and unexperienced environments.

6.1.1 Problem to be solved

The generalization ability of both GNP and GNP-RA is unsatisfactory because of *the existence of some harmful nodes in the program.*

In GNP, the function of the nodes and the number of nodes are assigned by the designer. It is difficult to know the optimal number of nodes for a particular problem. These nodes are not equally used during the program running(96), and some harmful nodes exist in the program of GNP. As a result, the program over-fits the training data easily and can not generalize well in the testing instances. Furthermore, it always uses the fixed nodes and can not distinguish the important nodes from the unimportant

ones. How to select the really useful nodes and a proper number of nodes in different situations is essential to improve the generalization ability of GNP.

In GNP-RA, the generated rules contain some harmful attributes (nodes) which are not useful for the processing. These rules are relatively long and represent some specific situations. They can not guide agents' actions efficiently in dynamic environments, where many inexperienced situations exist. The generalization ability of the rule pool is not good. Therefore, the harmful nodes should be pruned in order to create more general rules.

The difficulty is, whether a node is harmful or not depends on different environments. The seldom used nodes in the old environment may be used frequently in a new environment. Therefore, they can not be deleted simply. The really important thing is to select the really useful nodes flexibly considering different environments.

6.1.2 Motivation

In has been proved in both Neural Networks(NNs)(97, 98, 99) and EC(100, 101) that pruning the useless information in the program could improve its generalization ability. In NNs, pruning the useless nodes in the hidden layers helps to increase the generalization ability of the network and avoid the over-fitting(97). For example, a Fourier amplitude sensitivity test method(98) was used to rank the nodes in a hidden layer, and the nodes with lower rankings are pruned. In (99), PSO was incorporated into NN to optimize its parameters in the pruning process. However, choosing suitable parameters for PSO is difficult. A statistical pruning heuristic is proposed in (100), and nodes with low variances in sensitivity are regarded as irrelevant and removed. In GP, the nodes with unexecuted partial trees are regarded as useless and pruned in (101), which improves the search ability of GP. Although these methods are difficult to be used directly in GNP, they reveal a fact that pruning the useless information in the program could increase the generalization ability.

6.1.3 Novelty of this chapter

In this chapter, how to prune the harmful nodes to create more general programs and more general rules are studied. A new rule-based model named "Credit GNP with Rule Accumulation"(CGNP-RA) is proposed, where Credit GNP is used as the rule generator. The unique points of Credit GNP are as follows.

- Credit GNP has a unique gene structure, where each node has an additional branch. This gene structure has more exploration ability than that of the conventional GNP.

- Credit GNP combines evolution and reinforcement learning, i.e., off-line evolution and on-line learning in its program to prune the harmful nodes, which could speed up evolution and increase fitness. Credit GNP is a more efficient rule generator compared with GNP.
- Credit GNP could select the really useful nodes and prune the useless ones dynamically and flexibly considering different situations. Even if the environment changes suddenly, Credit GNP could select the really useful nodes to determine an action in the new environment.
- Credit GNP doesn't simply delete the useless nodes, which is different from the conventional pruning methods. It can create more general programs and more general rules which helps to improve the performance of GNP-RA. Therefore, CGNP-RA could be regarded as a unique tool for rule pruning.

In this chapter, Section 6.2 describes the algorithm of Credit GNP in terms of its node structure, evolution and reinforcement learning mechanism. Section 6.3 shows how to use Credit GNP to generate action rules for multi-agent control. A new rule-based model named "Credit GNP with Rule Accumulation"(CGNP-RA) is proposed. Section 6.4 demonstrates the effectiveness of CGNP-RA in both training and testing phases. The dynamic tile-world is chosen as the simulation environment. Section 6.5 is devoted to the summary.

6.2 Credit GNP

In this section, the basic concepts of Credit GNP are introduced, including it gene structure, genetic operators and reinforcement learning mechanism.

6.2.1 Node structure of Credit GNP

Fig. 6.1 compares the differences between the node structure of GNP and Credit GNP. The features of Credit GNP are as follows.

- Besides the normal branches, each node of Credit GNP has a credit branch.
- The credit branch serves as an additional connection of each node, therefore, Credit GNP has more exploration ability than the conventional GNP. This helps Credit GNP to explore solutions in a broader space and improve the search ability.

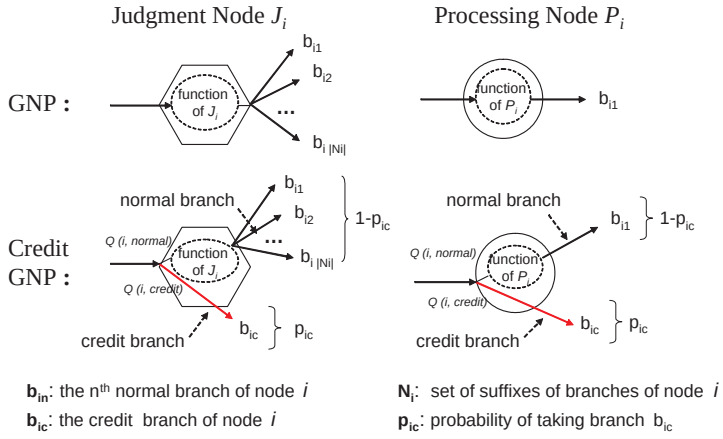

Figure 6.1: Node structure of GNP and Credit GNP

- If the credit branch is visited, the function of the node is not executed, i.e., its function is neglected and regarded as non-existent, and the next node is picked up for execution. In addition, the reward of taking credit branch is equal to zero when Q-values of credit branch are updated.
- The probability of selecting the credit branch, and which node the credit branch connects to are determined by evolution (crossover and mutation) and reinforcement learning.

6.2.2 Node execution of Credit GNP

The node execution of Credit GNP is different from that of GNP. When the node transition comes to a node in Credit GNP (suppose node i), it should firstly determine whether to take credit branch of node i or not. If credit branch is not taken, the node transition goes to the normal branches of node i, which is the same as in GNP; If credit branch is taken, the node transition skip node i and goes directly to the next node, which means node i is neglected. Whether to neglect node i or not depends on *Q(i, normal)* and *Q(i, credit)*, i.e., the Q-values of node i in the Q-table, which will be discussed in the following sections.

6.2.3 Evolution of Credit GNP

The aim of evolution is to find the best-fitted individual with the optimal gene structure. Genetic operators could change the gene structure of Credit GNP largely, so

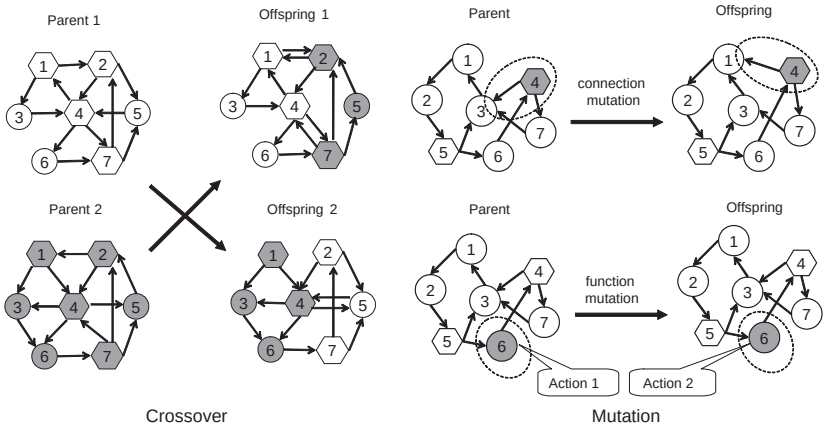

Figure 6.2: Genetic operators of Credit GNP

that the best individual could be obtained quickly. For crossover, two parents exchange the corresponding nodes and their connections, and the generated new offspring become parents of the next generation. For mutation, a parent randomly changes the function or connection of its nodes, and a new offspring is generated. Fig. 6.2 shows the genetic operators. Please note that the credit branch is changed in the same way as the normal branches by crossover and mutation. If the credit branch is taken, to which node it is connected is also determined by crossover and mutation. Tournament selection and elite selection are used as the selection policies.

6.2.4 Reinforcement learning of Credit GNP

In each node, the probability of choosing the normal branch or credit branch is determined by reinforcement learning, i.e., learning from the environments during the execution of the program. Reinforcement learning builds a bridge between the static structure of Credit GNP and the dynamic execution of its program. Q-learning and Sarsa-learning are the most famous reinforcement learning techniques. However, Q-learning is an off-line policy since the selection of the maximum Q-value at each state is necessary. Sarsa-learning is an on-line approach because it updates its Q values based on the really obtained actions during the task execution. Therefore, Sarsa-learning is more appropriate for the learning of Credit GNP.

In Sarsa-learning, the "state"and "action"should be defined firstly. The node of Credit GNP is defined as the "state"and the selection of the normal branches or credit branch is regarded as an "action". The Q-value of each state-action pair is updated by

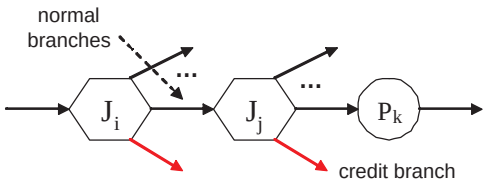

Figure 6.3: Node transition of Credit GNP

the following equation.

$$Q(i,a) \leftarrow Q(i,a) + \alpha[r(t) + \gamma Q(j,a') - Q(i,a)], \tag{6.1}$$

where, $Q(i,a)$ is the Q-value of state i when taking action $a \in$ {normal branch, credit branch}. a' is an action taken at state j. State j is the next node of node i as shown in Fig. 6.3. As mentioned before, the reward of taking credit branch is zero.

The useless and harmful nodes could be distinguished and neglected gradually through updating the Q-values. For example, suppose the Q-values of selecting the normal branches or the credit branch at node i are *Q(i, normal)* and *Q(i, credit)*, respectively.

- If the normal branch is selected and node is useful, then *Q(i, normal)* is increased, which increases the chance to visit node i the next time.
- If the normal branch is selected and the node is useless, then *Q(i, normal)* would decrease, which increases the probability to select the credit branch and to skip node i next time.
- If the credit branch is selected, then the node is skipped and *Q(i, credit)* is increased, which avoids the possible negative effects of carrying out node i and starts the new transition. This is also beneficial to the future rewards.

After calculating the Q-values, we adopt the following two policies for selecting the normal branches or credit branch. How they affect the learning of Credit GNP is studied in the simulation.

- $\varepsilon - greedy$: the branch with the largest Q value is selected with the probability of $1 - \varepsilon$, or a branch is randomly selected with the probability of ε.
- *Boltzmann Distribution*(102): the probability of selecting the credit branch is P_{ic} and the normal branches are selected with the probability of $1 - P_{ic}$, and P_{ic} is calculated as follows.

$$P_{ic} = \frac{e^{\frac{Q(i,credit)}{T}}}{e^{\frac{Q(i,normal)}{T}} + e^{\frac{Q(i,credit)}{T}}}, \tag{6.2}$$

where, T is the temperature parameter. At the beginning of the learning, the initial Q-values of the normal branches and credit branch are set at 1 and 0, respectively in order to avoid the situation that many useful nodes are neglected.

Credit GNP is more efficient than GNP because of the following reasons. Firstly, Credit GNP could generate more flexible node transitions because of the probabilistic selection of the normal branches or the credit branch. This enables Credit GNP to have more exploration ability which increases its search ability. While in GNP, the node transition is fixed which is changed only by evolution. Secondly, the node transition could be changed by updating the Q-values (Q(normal) and Q(credit)) and the harmful node could be pruned automatically and dynamically. This means that Credit GNP could select the really useful nodes and neglect the useless nodes during the programming running in different situations. For example, after the normal branch is selected, if a negative reward is given, then Q(normal) decreases. The credit branch could be used when the node is visited the next time, and this node is skipped because it is useless. This also increases the generalization ability in dynamic environments.

6.3 Rule accumulation based on Credit GNP

In this section, how to generate action rules through Credit GNP for agent control in dynamic environments is explained in details. The new method is named as "Credit GNP based Rule Accumulation"(CGNP-RA). Fig. 6.4 shows the flowchart of this model, which contains two phases. In the training phase, the node transitions of the best individuals in the Credit GNP population are recorded as action rules and accumulated into the rule pool generation by generation. In the testing phase, all the rules in the rule pool are used to guide agents' actions through the average matching degree calculation.

6.3.1 Rule generation in the training phase

In the training phase, the node transitions of the best individuals are recorded as rules and stored in the rule pool every generation. The rule of CGNP-RA is defined in the same way as that of GNP-RA. However, the different point is, in recording the node transitions, if the credit branch of a node is visited, this node is not recorded into the rule of CGNP-RA. This is because this node is neglected by visiting its credit branch, and its function is not executed. A large number of rules could be obtained

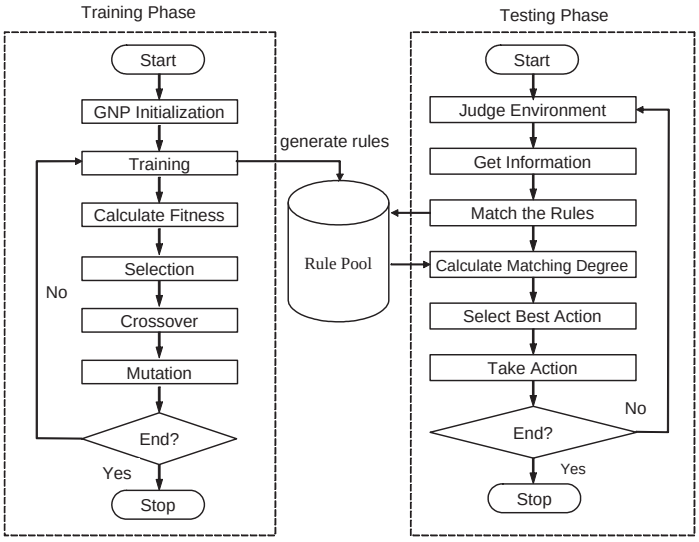

Figure 6.4: Flowchart of CGNP-RA

by training Credit GNP. The rule pool covers many good experiences of all the best individuals. Besides, it records many experienced situations in each generation.

Another advantage of the rule of CGNP-RA is, since the useless nodes are pruned by Credit GNP, these rules become more general. This helps to avoid to match with some specific rules and increase the generalization ability. Furthermore, even if some sudden changes of the environment occur, in the new environment, the rules after pruning are easier to match (since they are more general). This also contributes to improving the performance of the rule-based model.

6.3.2 Rule usage in the testing phase

This phase concerns how to use rules to control agents. The average matching degree calculation is used in this phase to match rules and determine agents' actions.

Firstly, all the rules in the rule pool are categorized into $|K|$ classes according to their processing nodes, i.e., each class represents a specific action. Then, the matching degree of new environment data d with rule r in class k is calculated as follows.

$$Match_k(d,r) = \frac{N_k(d,r)}{N_k(r)} str_k(r), \tag{6.3}$$

where, $N_k(d,r)$ is the number of matched judgment nodes with data d in the antecedent part of rule r in class k; $N_k(r)$ is the number of judgment nodes in the antecedent part of rule r in class k; $str_k(r)$ is the strength of rule r in class k.

For each class k, a threshold T_k is set to filter the lower matched rules. Only the rules whose matching degree are higher than the threshold could be regarded as selected.

$$T_k = mean_k + 0.1 * std_k, \tag{6.4}$$

$$mean_k = \frac{1}{|R_k|} \sum_{r \in R_k} Match_k(d, r), \tag{6.5}$$

$$std_k = \sqrt{\frac{1}{|R_k|} \sum_{r \in R_k} (Match_k(d, r) - mean_k)^2}, \tag{6.6}$$

where, T_k is the threshold of class k; $mean_k$ and std_k are the mean value and standard deviation of the matching degrees of class k; R_k is the set of suffixes of rules in class k.

Secondly, the average matching degree of environment data d with all the rules in class k is calculated.

$$m_k(d) = \frac{1}{|M_k|} \sum_{r \in M_k} Match_k(d, r), \tag{6.7}$$

where, M_k represents the set of suffixes of selected rules in class k.

Finally, class k which has the highest average matching degree is picked up and its corresponding action is taken.

$$k = arg \max_{k \in K} m_k(d), \tag{6.8}$$

where, K is the set of suffixes of classes (actions). After an action is taken, the environment is updated and a new environment data d is obtained, as a result, agents could take the next action.

6.4 Simulation

6.4.1 Simulation conditions

The dynamic tile-world is selected as the simulation environment to testify the generalization ability of the proposed method. In dynamic tile-world, the environment information could change suddenly, and this change is unpredictable before hand. In the testing tile-worlds, the locations of the tiles and holes are randomly set, which are quite different from the training instances. The testing tile-worlds imitate the sudden change of the environment, and the performance of Credit GNP and CGNP-RA is

Table 6.1: Parameter configurations

Evolution		Sarsa-learning	
Population	300		
Mutation Individuals	175		
Crossover Individuals	120	Action Steps	300
Best Individuals	5	Learning Rate α	0.9
Mutation Rate P_m	0.01	Discount Rate γ	0.9
Crossover Rate P_c	0.1	T	$1 \sim 5$
Generations	1000	ε	$0.01 \sim 0.2$

studied. The world size is set at 20×20. Table 6.1 shows the parameter configurations for the simulations.

In this simulation, Credit GNP is compared with some classical methods such as GNP, GP and standard Reinforcement Learning(RL). The total number of nodes for Credit GNP is set at 60, including 40 judgment nodes and 20 processing nodes. For GP, the function nodes are selected from the judgment nodes J1∼J8, and the terminal nodes are selected from the processing nodes P1∼P4. The function nodes have multiple arguments corresponding to different judgment results and the terminal nodes have no argument. In order to avoid bloating, the maximum depth of GP is set at 4. Since each non-terminal node has five outputs, the total number of nodes in GP is 781. The best individual of GP is selected to guide agents' actions. For RL, Sarsa-learning is chosen as the learning policy, "state"is defined as the full information of judgment nodes J1∼J8, and "action"is defined as the processing nodes from P1∼P4. Since J1 to J4 return {1,2,3,4,5}, and J5 to J8 return {1,2,3,4}, the possible number of state-action pairs is $5^4 \times 4^4 \times 4 = 640,000$. Q-values of the state-action pairs are updated in the same way as Eq. (6.1).

6.4.2 Results and analysis

6.4.2.1 Comparison with the classical methods

Fig. 6.5 and Fig. 6.6 show the training and testing results of RL, GP, GNP and Credit GNP, respectively, which are averaged from 30 independent simulations.

RL increases its fitness very slowly in the training and exhibits the low fitness in the testing. The main problem is that it has too many state-action pairs. The tile-world is

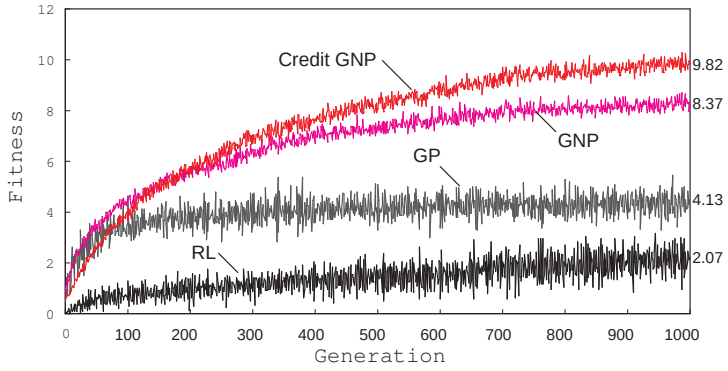

Figure 6.5: Training results of different methods

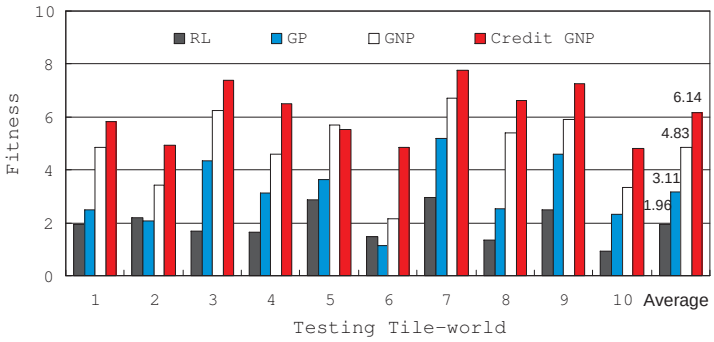

Figure 6.6: Testing results of different methods

very complex, where many different objects exist. Therefore, the search space(Q-table) becomes extremely large and 1000 generations are not enough to update them.

GP performs better than RL, but its fitness is still low. This is because GP has too many nodes(the total number of nodes is 781 if the maximum depth is 4). The decision tree becomes very large and hard to evolve, which over-fits the training data easily. That is why GP can not increase its fitness too much in later generations. In the testing phase, such a large structure can not cope with environment changes easily, and its generalization ability is not good.

GNP shows higher fitness in the training, because it can reuse the nodes to create compact programs. 60 nodes are sufficient for GNP to fulfil the given tasks. Moreover, the number of nodes is fixed and GNP never causes bloating. The graph structure has stronger expression ability, which enables GNP to solve the tile-world problem efficiently. However, due to the existence of the harmful nodes, the gene structure of

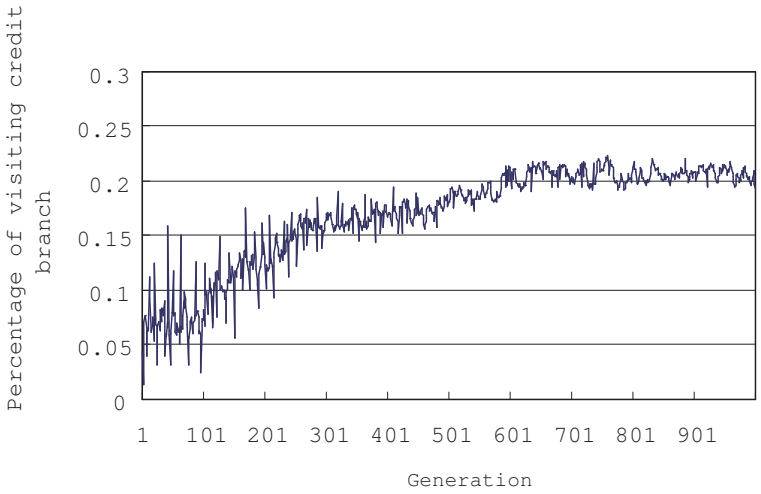

Figure 6.7: Percentage of selecting the credit branch

GNP becomes complicated, which degrade its generalization ability. It is noticed that the testing results of GNP are relatively low.

Credit GNP achieves lower fitness than GNP in earlier generations because it has to update the state-action pairs gradually. In later generations, it outperforms the conventional GNP. This is because Credit GNP could distinguish the important nodes and unimportant nodes through evolution and reinforcement learning. The really important nodes are selected for taking an action and the unimportant nodes are neglected, which enables Credit GNP to use its nodes flexibly and sufficiently. Furthermore, after pruning the harmful nodes, the program becomes more general, which can cope with the new environments easily. Therefore, Credit GNP could get higher fitness in both the training and testing.

Fig. 6.7 shows the percentage of selecting the credit branch during the node transition in different generations. The curve fluctuates a lot in earlier generations since it has to update the Q-table gradually through reinforcement learning to distinguish which node is useful and which node is harmful. This percentage converges to a certain number (approximately 20%) in later generations through enough learning, which means that about 20% of the nodes could be pruned as harmful nodes. Therefore, the program becomes more general after pruning which could handle different environments efficiently. While in GNP, the number of nodes is fixed because it can not prune the harmful nodes through genetic operations such as crossover and mutation.

Fig. 6.8 depicts the agents' traces of different methods. For simplicity, a small map with the size of 10×10 is used and the maximum time step is set at 60. It is

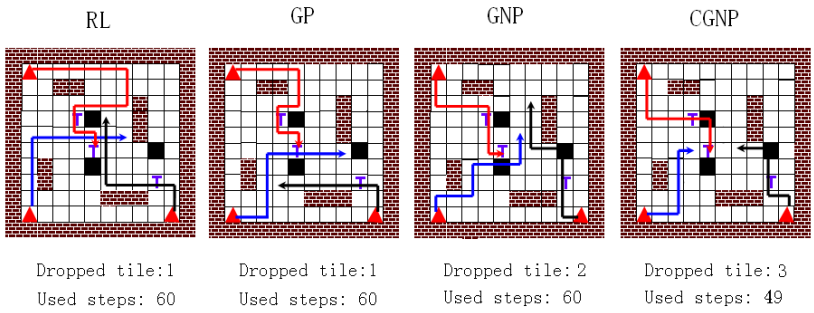

Figure 6.8: Agents' traces of different methods

noticed that Credit GNP could drop more tiles using less steps, which demonstrates its effectiveness.

6.4.2.2 Node usage study

The generalization ability of GNP and Credit GNP is compared by studying the usage of their nodes. In the training phase, GNP and Credit GNP are trained by the training tile-world. In the testing phase, a new tile-world is used to test the performance of the two methods. Agents' traces in both the training and testing are recorded and analyzed. For simplicity, the world size is set at 10×10 and the time steps are set at 60. Fig. 6.9 shows the ratio of used nodes by GNP and Credit GNP in the training and testing tile-worlds. The x-axis is the node index, and there are 5 nodes for each judgment and processing. The y-axis is the ratio of the used nodes.

Firstly, it is noticed that in GNP, the ratio of used nodes vary largely. Some nodes are used very frequently, while many others are seldom used. It is impossible to prune the harmful nodes through crossover and mutation. On the other hand, Credit GNP could distinguish useful nodes and useless nodes and prune the useless nodes by reinforcement learning. For example, the second node in J1, the fifth node in J2, etc., could be pruned. This contributes to generating more general programs and more general rules.

Secondly, let's compare the ratio changes in the training and testing tile-worlds. It is noticed that in GNP, there is a very small change of the ratio of used nodes in the training and testing. This is because the gene structure of GNP could not be changed. Therefore, the nodes of GNP in the testing are used in almost the same way as the training, although the environment has changed a lot. This decreases the generalization of GNP. On the contrary, the ratio of used nodes of Credit GNP changed a lot in the testing compared with the training. For example, in the training, the second node of

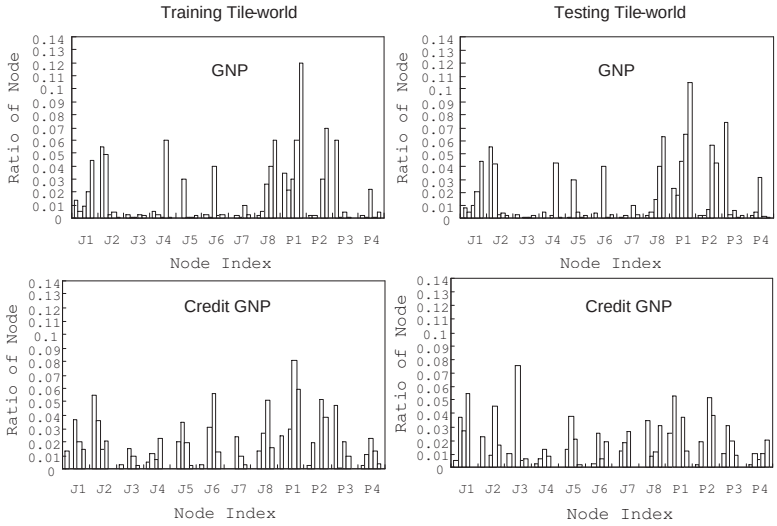

Figure 6.9: Ratio of nodes in training and testing

J1 is pruned, while in the testing, the fifth node in J1 is pruned. The fourth node of J6 is used frequently in the training; however, it is seldom used in the testing. This means that Credit GNP could select the really useful nodes considering different situations. Credit GNP could change the node transitions flexibly by updating its Q table through learning in the new environments, although its gene structure could not be changed. This also helps to improve its generalization ability.

Thirdly, it is noticed that Credit GNP could use its nodes more evenly, while in GNP, the ratio is very different between the nodes. This means that Credit GNP could distribute the task to different nodes. However, GNP always uses some particular nodes for a given task. This is because Credit GNP has an additional branch which can jump to any node if necessary. Therefore, Credit GNP has more exploration ability than GNP, and it can make use of its gene structure more sufficiently.

Fig. 6.10 shows the agents' traces of GNP and Credit GNP in the training and testing tile-worlds, respectively. GNP could drop 3 tiles in the training tile-world, but dropped only 1 tile in the testing tile-world. Besides, the agents' traces don't change too much between the training and testing. GNP over-fits the training data, since it always uses the same nodes for decision making and can not prune the harmful nodes. On the other hand, Credit GNP dropped all the tiles in the testing tile-world although the map has changed. Furthermore, it changed the agents' traces largely to handly the new situations. Credit GNP exhibits more generalization ability since it can prune the harmful nodes to create more flexible solutions. Also, Credit GNP can make full

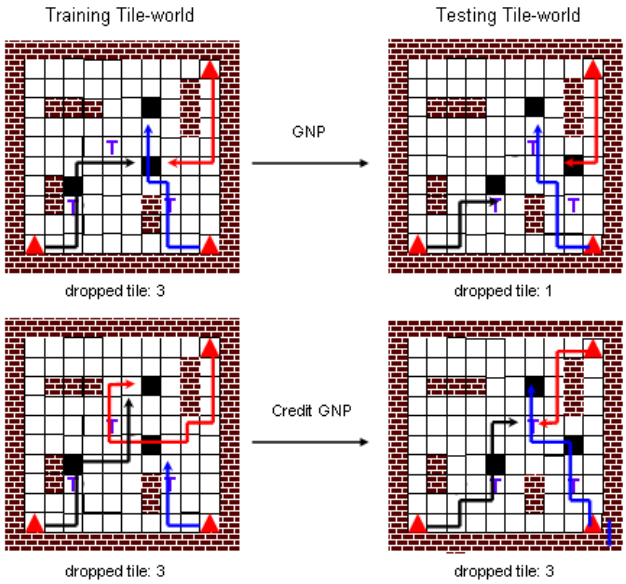

Figure 6.10: Agents' traces in training and testing

use of its nodes and change the node transitions flexibly using the obtained knowledge during learning. Therefore, it can change the usage of its nodes largely to cope with the sudden changes of the environment.

6.4.2.3 Parameter tuning

In the following simulation, how $\varepsilon - greedy$ and Boltzmann Distribution affects Credit GNP, i.e., the training results of CGNP-RA is studied. The optimal parameter for each selection method is tuned for the next simulation.

Fig. 6.11 shows the training results of Boltzmann Distribution with different temperatures. The temperature T is sensitive since, if T is very low, the selection becomes too greedy; and if it is very high, the selection becomes too random. Credit GNP is trained when T is from 1 to 5, where P_{ic} (the probability of selecting the credit branch in each node i) of 26.9%, 37.8%, 41.7%, 43.8% and 45% is obtained, respectively. When T is 1, the fitness curve is stable in earlier generations but fluctuates a lot in later generations. This is because during learning, larger Q-values become dominant and the selection becomes too greedy. When T is 5, it fluctuates a lot in earlier generations and increases relatively slowly. This is because the credit branch is visited with high probability and too many random actions are taken. As a result, many useful nodes

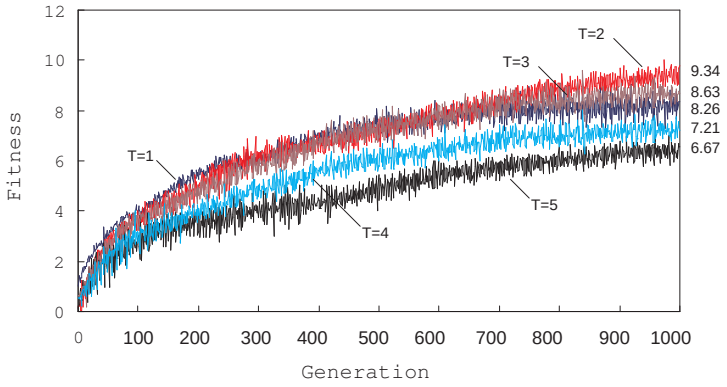

Figure 6.11: Training results of Boltzmann Distribution

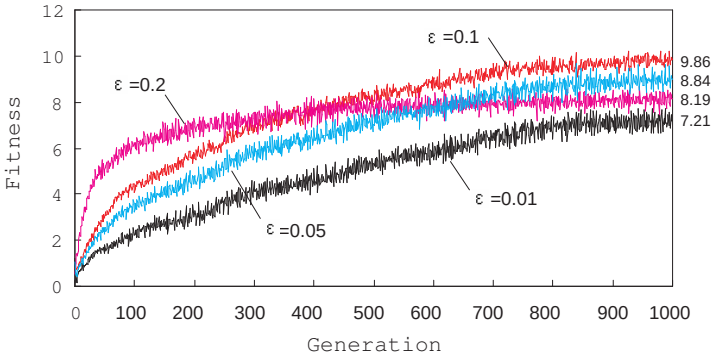

Figure 6.12: Training results of $\varepsilon - greedy$ policy

are neglected mistakenly, which results in the low fitness. T=2 seems to be the optimal parameter for Boltzmann Distribution.

Fig. 6.12 shows the training results of $\varepsilon - greedy$ policy. ε balances the exploration and exploitation of reinforcement learning. If ε is too small, the program would be trapped into local minima with high probability. If it is too large, the program would take too much random actions and the results are unstable. ε being 0.2 increases the fitness fast because large ε explores the solution space sufficiently and generate better solutions quickly. However, its fitness is low in later generations since its exploitation ability is not good. On the other hand, when ε is 0.01, the fitness curve increases slowly because it over-exploits the obtained knowledge, while its exploration ability is not good. 0.1 seems to be the optimal ε since it could well balance the exploration and exploitation. Compared with Boltzmann Distribution, $\varepsilon - greedy$ policy could get relatively better results because it selects the best action with a stable probability.

Table 6.2: Average fitness and standard deviation of the random test

	GNP	GNP-RA	CGNP-RA
AVG	4.81	6.70	8.13
STD	1.226	0.937	0.845
p-values (t-test)	1.27×10^{-5}	3.46×10^{-6}	

For Boltzmann Distribution, the temperature T is a bit hard to control. Therefore, $\varepsilon - greedy$ policy is chosen as the selection policy and ε is set at 0.1.

6.4.2.4 Comparison with the previous research

In the following simulation, CGNP-RA is compared with the previous GNP-RA method. How Credit GNP affects the quality of the rules is studied. In GNP-RA, important nodes and unimportant nodes are all recorded in the rules, while in CGNP-RA, unimportant nodes are pruned from the rules. The results of GNP are also provided as the benchmark. In order to fully test the performances of the two methods, 100 randomly generated tile-worlds are used as the testing cases.

Fig. 6.13 shows the testing results of the 100 tile-worlds and Table. 6.2 lists the average fitness (AVG) and standard deviation (STD) of the three methods. The p-values of the t-test between CGNP-RA and the other two methods are also listed in Table. 6.2. The small p-values suggest that the improvement is significant. Besides, CGNP-RA outperforms GNP-RA 82 times out of the 100 test cases. The average fitness of CGNP-RA is 8.13, while that of GNP-RA is 6.70. This demonstrates the effectiveness of the proposed method. This is because CGNP-RA could prune the unimportant nodes in the antecedent part of the rules. Some unnecessary judgments are eliminated, which may mislead the agents. As a result, the really important judgments could be picked up for taking an action. This could guide agents' actions more accurately. Furthermore, after pruning the unimportant nodes, these rules become more general, which work more efficiently in dynamic environments. Fig. 6.14 shows the average rule length of CGNP-RA and GNP-RA in different generations. It is noticed that the rules of CGNP-RA are becoming shorter(more general) since the harmful nodes are pruned gradually, while the length of the rules in GNP-RA almost remain the same.

Fig. 6.15 shows the number of rules generated by CGNP-RA and GNP-RA. Although the number of rules of CGNP-RA are less than those of GNP-RA, they could work better and get higher fitness. It means that Credit GNP could generate more

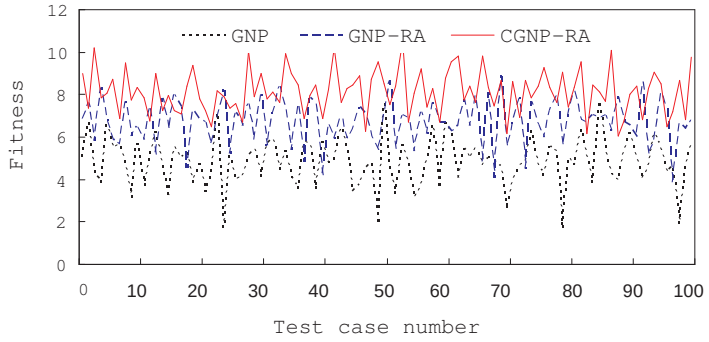

Figure 6.13: Random test of different methods

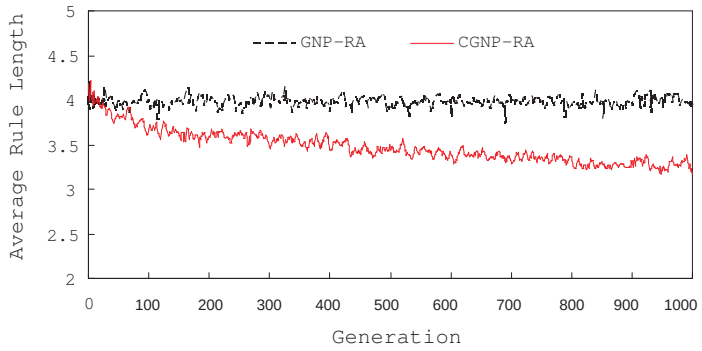

Figure 6.14: Average length of the generated rules

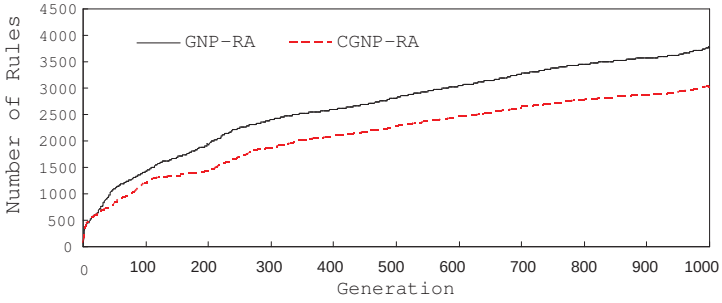

Figure 6.15: Number of generated rules

general rules for agent control. This also demonstrates the effectiveness of this pruning method.

6.5 Summary

In this chapter, a unique rule-based model named "Credit GNP based Rule Accumulation" (CGNP-RA) is proposed for multi-agent control, where Credit GNP is used as the rule generator. The purpose is to prune the harmful nodes and improve the generalization ability of GNP-RA. In the training phase, CGNP-RA could prune the harmful nodes in the gene structure and create more general programs. The obtained knowledge during the running of the program is used to distinguish the useful nodes from the useless ones. Which node is useless and how many nodes are useless are determined by both evolution and reinforcement learning concerning different situations. Furthermore, CGNP-RA could generate more general rules which are more efficient to cope with the unexperienced new environments. Simulation results on the dynamic tile-world problem demonstrate the effectiveness of CGNP-RA over the conventional GNP, GP and reinforcement learning methods. They also prove that CGNP-RA could build more general rule pool and achieve better performances than GNP-RA.

7

Conclusion

In this thesis, a GNP-based explicit memory scheme for multi-agent control is proposed, which is named as "Genetic Network Programming with Rule Accumulation". Different from the population-based methods which evolve the population and generate the best individual for decision making, this rule-based model uses a large number of rules to guide agents' actions. Two major advantages of GNP-RA are, firstly, rules are very simple and general, which could guide agents' actions accurately and avoid over-fitting. Secondly, good experiences from all the best individuals throughout the generations are accumulated into the rule pool. Many experienced situations could be recorded into rules, which helps agents to take correct actions in the current situation.

In chapter 2, the rule of GNP-RA is defined, and the general framework including rule generation in the training phase and rule application in the testing phase is proposed. The nodes transitions of the best GNP individuals are recorded as rules and stored in the rule pool. After that, all the rules in the rule pool are used to determine agents' actions through an average strength calculation. The most important point is, compared with only one rule (node transition route) of GNP, GNP-RA could use a large number of rules to make decisions. Simulation results on the static tile-world prove that GNP-RA could achieve higher fitness and better generalization ability than GNP.

In chapter 3, GNP with Multi-order Rule Accumulation (GNP-MRA) is proposed to solve the perceptual aliasing problem and improve performance in non-Markov environments. Historical information of the environment data and agent's actions is recorded into the multi-order rules, which serves as the additional information to distinguish the aliasing situations. Two methods are designed to match the multi-order rules, the completely matching and partially matching. Simulation results reveal that GNP-MRA outperforms GNP-RA since the historical information in the previous steps helps

agents to distinguish different aliasing situations and take correct actions. They also prove that as the rule order increases, partially matching becomes more efficient since they can retrieve many multi-order rules whose situations are similar to the current situation, while completely matching becomes too difficult.

In chapter 4, GNP with Rule Accumulation and Pruning (GNP-RAP) is proposed to improve the quality of the generated rules. GNP-RAP improves the quality of the rules from two aspects. During the rule generation, evolution is combined with reinforcement learning in order to create more efficient rules, since the obtained knowledge during the program running can create more reasonable combinations of judgments for a processing. After the rules are generated, a post pruning approach is proposed to filter the generated bad rules. The unique point is that bad individuals are used to generate the bad rules, which are used to locate the bad rules in the rule pool. The overlapped rules are reevaluated, which improves the quality of the rule pool.

In chapter 5, GNP with Updating Rule Accumulation (GNP-URA) is proposed to improve the adaptability of GNP-RA in dynamic environments. Reinforcement learning is combined with evolution in order to realize this, and Sarsa-learning is adopted as the learning policy. In the off-line evolution, Sarsa enables GNP to generate better and more rules. More importantly, when evolution terminates, Sarsa enables the best individuals of GNP to learn in the changing environments, during which some new rules could be obtained. These rules represent some unexperienced situations and are used to update the old rules in the rule pool. Therefore, the rule pool could cope with the environment changes. Simulation results on dynamic tile-worlds demonstrate the effectiveness of GNP-URA in adapting to the changing environments.

In chapter 6, Credit GNP with Rule Accumulation(CGNP-RA) is proposed in order to improve the generalization ability of GNP-RA. Different from GNP-RA, each node of CGNP-RA has an additional branch named "credit branch" whose function is to skip this node if it is considered harmful or useless. Reinforcement learning is combined with evolution in CGNP-RA in order to prune the harmful nodes. Which node is harmful and how many nodes are harmful are determined automatically considering different situations. Therefore, CGNP-RA could use its nodes flexibly and sufficiently, which helps to improve its generalization ability. Furthermore, the harmful nodes in the antecedent part of the rule could be pruned, and the rules become shorter and more general. These rules work better in dynamic environments. Some randomly generated tile-worlds are used to test the generalization ability of CGNP-RA, and simulation results prove that CGNP-RA could generalize well and achieve better performance than GNP-RA.

References

[1] K. D. Jong. **Evolutionary Computation: A Unified Approach**. *Cambridge, MA: MIT Press*, 2005. 1

[2] T. Back. **Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms**. *London, U.K.: Oxford Univ. Press*, 1996. 1

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

[4] D. E. Goldberg. **Genetic Algorithm in Search Optimization and Machine Learning**. *Reading, MA: Addison-Wesley*, 1989. 1

[5] J. R. Koza. **Genetic Programming, on the Programming of Computers by Means of Natural Selection**. *MIT Press, Cambridge, MA*, 1992. 1

[6] J. R. Koza. **Genetic Programming II, Automatic Discovery of Reusable Programs**. *MIT Press, Cambridge, MA*, 1994. 1, 5

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

[8] J. R. Koza. **Routine Human-Competitive Machine Intelligence**. *Kluwer Academic Pub.*, 2003. 1

[9] D. B. Fogel. **An introduction to simulated evolutionary optimization**. *IEEE Transactions on Neural Networks*, **5(1)**:3–14, 1994. 1

[10] L. J. Fogel, J. M. Zurada, R. J. Marks II, and C. Goldberg. **Evolutionary programming in perspective: the top-down view, Computational Intelligence: Imitating Life**. *Eds. Piscataway, NJ: IEEE Press*, 1994. 1

[11] I. Rechenberg, J. M. Zurada, R. J. Marks II, and C. Goldberg. **Evolution strategy, Computational Intelligence: Imitating Life**. *Eds. Piscataway, NJ: IEEE Press*, 1994. 1

[12] T. Back, F. Hoffmeister, and H. P. Schwefel. **Evolution Strategies-a comprehensive introduction**. *Natural Computing*, **1(1)**:3–52, 2002. 1

[13] W. B. Langdon and R. Poli. **Fitness causes bloat: mutation**. *Genetic Programming*, **1391**:37–48, 1998. 2

[14] W. B. Langdon. **The evolution of size in variable length representations**. *In Proc. of the IEEE International Conference on Evolutionary Computation*, pages 663–682, 1998. 2

[15] J. Kennedy and R. Eberhart. **Particle swarm optimization**. *In Proc. of the IEEE International Conference on Evolutionary Computation*, pages 1942–1948, 1995. 2

[16] R. Poli, J. Kennedy, and T. Blackwell. **Particle swarm optimization: an overview**. *Swarm Intelligence*, **1(1)**:33–57, 2007. 2

[17] M. Dorigo, M. Birattari, and T. Stutzle. **Ant colony optimization**. *IEEE Computational Intelligence Magazine*, **1(4)**:38–49, 2006. 2

[18] M. Dorigo, , and L. D. Bruxelles. **Ant colony optimization: a new meta-heuristic**. *In Proc. of the IEEE Congress on Evolutionary Computation*, pages 1470–1477, 1999. 2

[19] F. Bergh and A. P. Engelbrecht. **A cooperative approach to particle swarm optimization**. *IEEE Transactions on Evolutionary Computation*, **8(3)**:225–239, 2004. 2

[20] C. Juang. **A Hybrid of Genetic Algorithm and Particle Swarm Optimization for Recurrent Network Design**. *IEEE Transaction on Systems, Man, and Cybernetics, Part B*, **34(2)**:997–1006, 2004. 2

[21] M. Dorigo, V. Maniezzo, and A. Colorni. **Ant system: optimization by a colony of cooperating agents**. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, **26(1)**:29–41, 1996. 2

[22] S. Mabu, K. Hirasawa, and J. Hu. **A Graph-Based Evolutionary Algorithm: Genetic Network Programming (GNP) and Its extension Using Reinforcement Learning**. *Evolutionary Computation*, **15(3)**:369–398, 2007. 2, 3, 4

[23] T. Eguchi, K. Hirasawa, J. Hu, and N.Ota. **A study of Evolutionary Multiagent Models Based on Symbiosis**. *IEEE Trans. on Systems, Man and Cybernetics, Part B*, **35(1)**:179–193, 2006. 2

[24] K. Hirasawa, M. Okubo, J. Hu, and J. Murata. **Comparison between Genetice Network Programming (GNP) and Genetic Programming (GP)**. *In Proc. of the IEEE Congress on Evolutionary Computation*, pages 1276–1282, 2001. 2

[25] H. Katagiri, K. Hirasawa, J. Hu, and J. Murata. **Genetice Network Programming and Its Application to the Multiagent System**. *Trans. IEE of Japan*, **122-C**:2149–2156, 2002. 2

[26] D. M. Hawkins. **The Problem of Overfitting**. *Journal of Machine Learning Research*, **44(1)**:1–12, 2004. 2

[27] G. C. Cawley and N. L. C. Talbot. **Preventing Over-Fitting during Model Selection via Bayesian Regularisation of the Hyper-Parameters**. *Journal of Machine Learning Research*, **8**:841–861, 2007. 2

[28] 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)*, **3(2)**:121–129, 2010. 3

[29] Y. Yang, J. Li, S. Mabu, and K. Hirasawa. **GNP-Sarsa with subroutines for trading rules on stock markets**. *In Proc. of IEEE International Conference on System, Man and Cybernetics*, pages 1161–1165, 2010. 3

[30] L. Yu, J. Xhou, S. Mabu, K. Hirasawa, J. Hu, and S. Markon. **Double-deck elevator supervisory control system using genetic network programming with ant colony optimization with evaporation**. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, **11(9)**:1149–1158, 2007. 3

[31] K. Hirasawa, T. Eguchi, J. Zhou, L. Yu, and S. Markon. **A double-deck elevator group supervisory control system using Genetic Network Programming**. *IEEE Transactions on Systems, Man and Cybernetics, Part C*, **38(4)**:535–550, 2008. 3

[32] S. Mabu and K. Hirasawa. **Enhanced rule extraction and classification mechanism of genetic network programming for stock trading signal generation**. *In Proc. of the Genetic and Evolutionary Computation Conference*, pages 1659–1666, 2011. 3

[33] H. Zhou, S. Mabu, K. Shimada, and K. Hirasawa. **MBFP Generalized Association Rule Mining and Classification in Traffic Volume Prediction**. *IEEJ Transactions on Electrical and Electronic Engineering*, **6(5)**:457–467, 2011. 3

[34] S. Mabu, C. Chen, N. Lu, K. Shimada, and K. Hirasawa. **An intrusion-detection model based on fuzzy class-association-rule mining using Genetic Network Programming**. *IEEE Transaction on Systems, Man and Cybernetics, Part C*, **41(1)**:130–139, 2011. 3

[35] G. Yang, S. Mabu, K. Shimada, and K. Hirasawa. **An evolutionary approach to rank class association rules with feedback mechanism**. *Expert Systems with Applications*, **38(12)**:15040–15048, 2011. 3

[36] A. L. C. Bazzan. **Opportunities for multiagent systems and multiagent reinforcement learning in traffic control**. *Autonomous Agents and Multi-Agent Systems*, **18(3)**:342–375, 2009. 3

[37] P. Stone and M. Veloso. **Multi-agent systems: a survey from a machine learning perspective**. *Autonomous Robots*, **8(3)**. 3

[38] L. Busoniu, R. Babuska, and B. D. Schutter. **A comprehensive survey of multi-agent reinforcement learning**. *IEEE Trans. on System, Man and Cybernetics, Part C*, **38(2)**:156–172, 2008. 3

[39] C. Watkins. **Learning from Delayed Rewards**. *Ph.D thesis, Cambridge University, King's College*, 1989. 3

[40] G. Rummery and M. Niranjan. **On-line Q-learning using connectionist systems**. *Tech. Rep. CUED/F-INFENG/TR 166, Cambridge University, Engineering Department*, 1994. 3

[41] J. Peng and R. J. Williams. **Incremental multi-step Q-learning**. *Machine Learning*, **22(1)**:283–290, 1996. 3

[42] G. A. Rummery and M. Niranjan. **On-line Q-learning using connectionist systems**. *Cambridge University Press*, 1994. 3, 47

[43] J. J. Grefenstette, D. E. Moriarty, and A. C. Schultz. **Evolutionary Algorithms for Reinforcement Learning**. *Journal Of Artificial Intelligence Research*, **11**:241–276, 1999. 3

[44] F. Liu and G. Zeng. **Study of genetic algorithm with reinforcement learning to solve the TSP**. *Expert Systems with Applications*, **36(3)**:6995C7001, 2009. 3

[45] H. Y. Chen, C. Y. Hsu, and T. K. Chao. **Reinforcement learning of robotic motion with genetic programming, simulated annealing and self-organizing map**. *In Proc. of the Conference on Technologies and Applications of Artificial Intelligence*, pages 292–298, 2011. 3

[46] J. P. Rosca and D. H. Ballard. **Discovery of Subroutines in Genetic Programming**. *Advances in Genetic Programming*, **MIT Press Cambridge, MA, USA**:177–201, 1996. 3

[47] A. Aamodt and E. Plaza. **Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches**. *Computer Science and Artificial Intelligence*, **7(1)**:39–59, 1994. 4

[48] R. L. D. Mantaras. **Case-Based Reasoning**. *Machine Learning and Its Applications*, **2049**:127–145, 2001. 4

[49] I. Watson. **Applying case-based reasoning: techniques for enterprise systems**. *Morgan Kaufmann Publishers Inc. San Francisco, CA, USA*, 1998. 4

[50] C. Stanfill and D. Waltz. **Toward memory-based reasoning**. *Communications of the ACM*, **19(12)**:1213–1228, 1986. 4

[51] 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*, pages 299–306, 1997. 4

[52] J. Branke. **Memory enhanced evolutionary algorithms for changing optimization problems**. *in Proc. of the 1999 Congress on Evolutionary Computation*, pages 1875–1882, 1999. 4

[53] 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*, pages 168–174, 2003. 4

[54] S. Yang. **Memory-based immigrants for genetic algorithms in dynamic environments**. *In Proc. of the 2005 Genetic Evolutionary Computation Conference*, pages 1115–1122, 2005. 4

[55] D. E. Goldberg and R. E. Smith. **Non-stationary function optimization using genetic algorithms with dominance and diploidy**. *In Proc. of the 2nd International Conference on Genetic Algorithms*, pages 59–68, 1987. 4

[56] K. P. Ng and K. C. Wong. **A new diploid scheme and dominance change mechanism for non-stationary function optimization**. *In Proc. of the 6th International Conference on Genetic Algorithms*, pages 159–166, 1995. 4

[57] 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*, pages 139–48, 1998. 4

[58] 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*, pages 99–102, 1996. 5

[59] C. L. Ramsey and J. J. Grefenstette. **Case-based initialization of genetic algorithms**. *In Proc. of the 5th International Conference on Genetic Algorithms*, pages 84–91, 1993. 5

[60] C. L. Ramsey and J. J. Grefenstette. **Case-based initialization of genetic algorithms**. *In Proc. of the 5th International Conference on Genetic Algorithms*, pages 84–91, 1993. 5

[61] J. Han and M. Kamber. **Data Mining: Concepts and Techniques**. *San Francisco, CA*, 2001. 9, 14

[62] 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, AAAI Press*, pages 183–189, 1990. 17

[63] S. D. Whitehead and L. J. Lin. **Reinforcement learning of non-Markov decision processes**. *Artificial Intelligence*, **73(1)**:271–306, 1995. 26

[64] P. L. Lanzi and S. W. Wilson. **Toward Optimal Classifier System Performance in Non-Markov Environments**. *Evolutionary Computation*, **8(4)**:393–418, 2000. 26

[65] T. Jaakkola, S. P. Singh, and M. I. Jordan. **Reinforcement learning algorithm for partially observable Markov decision problems**. *MIT Press, Cambridge, MA*, 1995. 26

[66] S. P. Sinph, T. Jookkola, and M. I. Jordan. **Learning without state-estimation in partiall observable markov decision processes**. *MIT Press, Cambridge, MA*, 1994. 26

[67] P. L. Lanzi and S. W. Wilson. **Toward optimal classifier system performance in non-Markov environments**. *Evolutionary Computation*, **8(4)**:393–418, 2000. 26

[68] P. Smyth. **Hidden Markov models for fault detection in dynamic systems**. *Pattern Recognition*, **27(1)**:149C164, 1994. 27

[69] W. S. Lovejoy. **A survey of algorithmic methods for partially observed Markov decision processes**. *Annals of Operations Research*, **28(1)**:47–56, 1991. 27

[70] F. Gomez, J. Schmidhuber, and R. Miikkulainen. **Accelerated neural evolution through cooperatively coevolved synapses**. *Journal of Machine Learning Research*, **9**:937–965, 2008. 27

[71] S. W. Wilson. **ZCS: a zeroth level classifier system**. *Evolutionary Computation*, **1**:1–18, 1994. 28

[72] S. D. Whitehead. **Reinforcement learning of non-Markov decision processes**. *Artificial Intelligence*, **73**:271–306, 1995. 28

[73] J. J. Grefenstette, D. E. Moriarty, and A. C. Schultz. **Evolutionary Algorithms for Reinforcement Learning**. *Journal Of Artificial Intelligence Research*, **11**:241–276, 1999. 44

[74] D. E. Moriarty and R. Mikkulainen. **Efficient reinforcement learning through symbiotic evolution**. *Machine Learning*, **22**:11–32, 1996. 44

[75] Y. Yu and Z. H. Zhou. **On the usefulness of infeasible solutions in evolutionary search: A theoretical study**. *In Proc. of the IEEE Congress on Evolutionary Computation*, pages 835–840, 2008. 44

[76] S. Brownlee, J. McCall, Q. Zhang, and D. Brown. **Approaches to selection and their effect on fitness modelling in an estimation of distribution algorithm**. *In Proc. of the IEEE Congress on Evolutionary Computation*, pages 2621–2628, 2008. 44

[77] Y. Hong, G. Zhu, S. Kwong, and Q. Ren. **Estimation of distribution algorithms making use of both high quality and low quality individuals**. *In Proc. of the IEEE International Conference on Fuzzy Systems*, pages 1806–1818, 2009. 44

[78] X. Li, J. Branke, and T. Blackwell. **Particle swarm with speciation and adaptation in a dynamic environment**. *In Proc. of the 8th annual conference on genetic and evolutionary computation*, pages 51–58, 2006. 59

[79] J. A. Lepine, J. A. Colquitt, and A. Erez. **Adaptbility to changing task contexts: effects of general cognitive ability, conscientiousness and openness to experience**. *Personnel Psychology*, **53(3)**:563–593, 2000. 59

[80] E. B. Jacob. **Bacterial self-organization: co-enhancement of complexification and adaptability in a dynamic environment**. *Philosophical transactions of the Royal Society of London*, **361**:1283–1313, 2003. 59

[81] Y. Liu and X. Yao. **Evolutionary design of artificial neural networks with different nodes**. *In Proc. of IEEE International Conference on Evolutionary Computation*, pages 670–675, 1996. 59

[82] B. H. Liu, L. Li, Y. F. Huang, C. F. Li, G. C. Guo, E. M. Laine, H. P. Breuer, and J. Piilo. **Experimental control of the transition from Markovian to non-Markovian dynamics of open quantum systems**. *Nature Physics*, **7**:931–934, 2011. 59

[83] L. Viola, E. Knill, and S. Lloyd. **Dynamical decoupling of open quantum systems**. *Physical Review Letters*, **82(12)**:2417–2421, 1999. 59

[84] Y. Jin and J. Branke. **Evolutionary optimization in uncertain environments-a survey**. *IEEE Transactions on Evolutionary Computation*, **9(3)**:303–317, 2005. 60

[85] D. B. Fogel. **Evolutionary Computation: Toward a New Philosophy of Machine Intelligence**. *Piscataway, NJ: IEEE Press*, 2006. 60

[86] J. Zurada, M. Mazurowski, R. Ragade, A. Abdullin, J. Wojtudiak, and J. Gentle. **Building virtual community in computational intelligence and machine learning**. *IEEE Computational Intelligence Magazine*, **4(1)**:43–54, 2009. 60

[87] J. Zhang, Z. Zhang, Y. Lin, N. Chen, Y. Gong, J. Zhong, H.S.H. Chung, Y. Li, and Y. Shi. **Evolutionary computation meets machine learning: a survey**. *IEEE Computational Intelligence Magazine*, **6(4)**:68–75, 2011. 60

[88] S. Yang and X. Yao. **Population-based incremental learning with associative memory for dynamic environments**. *IEEE Transactions on Evolutionary Computation*, **12(5)**:542–561, 2008. 60

[89] Y. G. Woldesenbet and G. G. Yen. **Dynamic evolutionary algorithm with variable relocation**. *IEEE Transactions on Evolutionary Computation*, **13(3)**:500–513, 2009. 60

[90] R. S. Sutton and A. G. Barto. **Reinforcement Learning: An Introduction**. *MIT Press, Cambridge, MA*, 1998. 60

[91] D. E. Goldberg. **Genetic Algorithms in Search, Optimization and Machine Learning**. *Reading, MA: Addison-Wesley*, 1989. 60

[92] T. Back, U. Hammel, and H. P. Schwefel. **Evolutionary computation: comments on the history and current state**. *IEEE Transactions on Evolutionary Computation*, **1(1)**:3–17, 1997. 74

[93] Y. Jin. **A comprehensive survey of fitness approximation in evolutionary computation**. *Soft Computing*, **9**:3–12, 2005. 74

[94] W. M. Spears, K. A. D. Jong, T. Back, D. B. Fogel, and H. D. Garis. **An overview of evolutionary computation**. *Machine Learning*, **667**:442–459, 1993. 74

[95] D. E. Goldberg and J. H. Holland. **Genetic algorithm and machine learning**. *Machine Learning*, **3(2)**:95–99, 1988. 74

[96] S. Mabu, Y. Chen, S. Eto, K. Shimada, and K. Hirasawa. **Genetic network programming with intron-like nodes**. *IEEJ Trans. on Electronics, Information and Systems*, **128(8)**:1312–1319, 2008. 74

[97] R. Reed. **Pruning algorithms-a survey**. *IEEE Transactions on Neural Networks*, **4(5)**:740–747, 1993. 75

[98] P. Lauret, E. Fock, and T. A. Mara. **A node pruning algorithm based on a Fourier amplitude sensitivity test method**. *IEEE Transactions on Neural Network*, **17(2)**:273–293, 2006. 75

[99] J. Tu, Y. Zhan, and F. Han. **A neural network pruning method optimized with PSO algorithm**. *In Proc. of the International Conference on Computer Modeling and Simulation*, pages 257–259, 2010. 75

[100] A. P. Engelbrecht. **A new pruning heuristic based on variance analysis of sensitivity information**. *IEEE Transactions on Neural Networks*, **12(6)**:1386–1399, 2001. 75

[101] Y. Maeda and S. Kawaguchi. **Redundant node pruning and adaptive search method for Genetic Programming**. *In Proc. of the Genetic and Evolutionary Computation Conference (GECCO 2000)*, pages 572–573, 2000. 75

[102] E. A. Guggenheim. **Boltzmann's distribution law**. *North-Holland Publishing Comp., Amsterdam*, 1955. 79

Research Achievements

< Journal Papers >

1. L. Wang, S. Mabu, W. Xu and K. Hirasawa, Pruning of Redundant Information to Improve Performance for Agent Control in A Changing Environment, IEEJ Transaction on Electronics, Information and Systems, Vol. 132, No. 11, Part C, 2012.(accepted)

2. L. Wang, S. Mabu and K. Hirasawa, Multi-order Rule Accumulation for an Agent Control Problem in Non-markov Environments, IEEJ Transaction on Electronics, Information and Systems, Vol. 132, No. 11, pp. 1819-1828, 2012.

3. 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, 2010/3.

4. L. Wang, S. Mabu, F. Ye, S. Eto, X. Fan, K. Hirasawa, Genetic Network Programming with Rule Accumulation and Its Application to Tile-World Problem, Journal of Advanced Computational Intelligence and Intelligent Informatics, Vol. 13, No. 5, pp. 551-572, 2009/9.

< International Conference Papers >

1. W. Xu, L. Wang, S. Mabu and K. Hirasawa, Genetic Network Programming with Credit, In Proc. of the SICE International Annual Conference (SICE 2012), pp. 1769-1777, Akita, Japan, 2012/8.

2. L. Wang, W. Xu, S. Mabu and K. Hirasawa, Rule Accumulation Method Based on Credit Genetic Network Programming, In Proc. of the IEEE Congress on Evolutionary Computation (CEC2012), pp. 3651-3658, Brisbane, Australia, 2012/6.

3. L. Wang, S. Mabu and K. Hirasawa, A Pruning Method for Accumulated Rules by Genetic Network Programming, In Proc. of the SICE International Annual Conference (SICE 2011), pp. 1744-1749, Tokyo, Japan, 2011/9.

4. L. Wang, S. Mabu and K. Hirasawa, Genetic Network Programming with Updating Rule Accumulation, In Proc. of the IEEE World Congress on Evolutionary Computation (CEC2011), pp. 2259-2266, New Orleans, U.S., 2011/6.

5. F. Ye, S. Mabu, L. Wang and K. Hirasawa, Genetic Network Programming with Route Nodes, In Proc. of the IEEE TENCON International Conference (TENCOM 2010), pp. 1404-1409, Fukuoka, Japan, 2010/11.

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

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

8. L. Wang, S. Mabu, Q. Meng and K. Hirasawa, Genetic Network Programming with Generalized Rule Accumulation, In Proc. of the IEEE World Congress on Evolutionary Computation (CEC2010), pp. 2681-2687, Barcelona, Spain, 2010/7.

9. L. Wang, S. Mabu, F. Ye, K. Hirasawa, Rule Accumulation Method with Modified Fitness Function based on Genetic Network Programming, In Proc. of the ICROS-SICE International Joint Conference (2009), pp. 1000-1005, Fukuoka, Japan, 2009/8.

10. F. Ye, S. Mabu, L. Wang and K. Hirasawa, Genetic Network Programming with General Individual Reconstruction, In Proc. of the ICROS-SICE International Joint Conference (SICE 2009), pp. 3474-3479, Fukuoka, Japan, 2009/8.

11. L. Wang, S. Mabu, F. Ye and K. Hirasawa, Genetic Network Programming with Rule Accumulation Considering Judgment Order, In Proc. of the IEEE Congress on Evolutionary Computation (CEC 2009), pp. 3176-3182, Trondheim, Norway, 2009/5.

12. F. Ye, S. Mabu, L. Wang, S. Eto and K. Hirasawa, Genetic Network Programming with Reconstructed Individuals, In Proc. of the IEEE Congress on Evolutionary Computation (CEC 2009), pp. 854-859, Trondheim, Norway, 2009/5.