SICE Annual Conference 2011
September 13-18, 2011, Waseda University, Tokyo, Japan

A Pruning Method for Accumulated Rules by Genetic Network Programming

Lutao Wang*, Shingo Mabu*, Kotaro Hirasawa*
*Graduate School of Information, Production and Systems, Waseda University
Hibikino 2-7, Wakamatsu-ku, Kitakyushu, Fukuoka 808-0135, Japan
Email: wanglutao@fuji.waseda.jp, mabu@aoni.waseda.jp, hirasawa@waseda.jp

***Abstract*—Genetic Network Programming(GNP) is a newly developed evolutionary computation method. A GNP based rule accumulation method(GNP-RA) is also proposed to generate decision rules and accumulate them into the rule pool, which serves as an experience set for agent control problems. Elite individuals are regarded as evolving rule generators and the extracted rules are viewed as solutions, which is different from the conventional evolutionary computation methods. However, even the best individual could generate some bad rules, thus the interesting rules and uninteresting rules are hard to distinguish. This paper proposed a method to prune the uninteresting rules in the rule pool so that the interesting ones could stand out, which helps to increase the accuracy of decision making. The efficiency and effectiveness of the proposed method is verified by the tile-world problem, which is an excellent benchmark in multi-agent systems.**

***Keywords*—Genetic Network Programming, Rule Accumulation, Rule Pruning, Multi-agent System, Tile-world**

I. Introduction

Genetic Network Programming[1] is an extension of GA[2] and GP[3]. Unlike the string structure of GA and the tree structure of GP, GNP adopts a directed graph structure as its chromosome, which is its unique feature. The advantages of GNP over other EC methods are: 1), the nodes of GNP are categorized into judgment nodes and processing nodes, thus the gene structure is very compact. 2), GNP's nodes could be reused many times during the node transition, which helps to reduce the memory occupation. 3), GNP uses only the necessary environment information and is competent for dealing with partially observable problems. Recently, GNP is adopted to many real world applications, such as: elevator supervisory control systems, stock market prediction, traffic volume forecasting, online auction system, and class association rule mining in data mining field, ect.

Conventional GNP method has no memory scheme, so that the past experiences could not be recorded and used later. The elite individuals could not experience all the situations, thus the generalization ability of GNP is not satisfactory. In order to solve this, GNP-RA[4] has been proposed in the previous work to accumulated good experiences of elite individuals (rules) generation by generation into the rule pool, and reuse them for guiding agent's actions. The intrinsic feature of GNP-RA is to record various rules in the evolutionary period and use them for decision making in the future. Related research could be Case-Based Reasoning (CBR)[5] and Memory-Based Reasoning (MBR)[6], which aims to store historical information on good examples and used them later. However, a very different point of GNP-RA is, in stead of directly storing the elite individuals as cases, it extracts and accumulates generalized rules into the rule pool, which represent good solutions. Currently, GNP-RA has been applied to the stock prediction problem for generating stock trading rules[7].

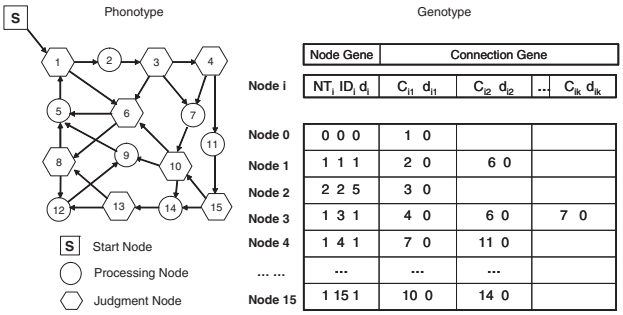

Fig. 1. Phenotype and genotype of GNP

The disadvantage of GNP-RA is that good rules and bad rules are hard to distinguish because of the following reasons. Firstly, since the reward is obtained by a series of actions which correspond to different rules, but how much they contribute to the reward is a bit hard to determine. Secondly, because of the ε-greedy policy of Sarsa-learning, some unimportant branches with low Q-values could be visited, generating some unimportant rules, which decreases the quality of the rule pool. Thirdly, because the fitness value is obtained at the end of the generation, it fails to represent the importance of each rule generated during evolution. This brings difficulties to the rule evaluation. Currently, we merely extract rules from the elite individuals and use them as good experiences, however, the experiences from the worst individuals are ignored. Even the best individual could generate bad rules. Therefore, pruning of the accumulated rules in the rule pool becomes necessary.

The rest of the paper is organized as follows. Section II presents some work that is related to the current research. Section III designed a framework for the proposed method and shows the detailed algorithm. Section IV introduces the simulation environment and analyzes the simulation results. Finally, Section V is devoted to the conclusion and future work.

PR0001/11/0000-1744 ¥400

II. Related Work

A. Genetic Network Programming

The gene structure of GNP is composed of three types of nodes, i.e., a start node, judgment nodes and processing nodes. The start node with no particular function is used to choose the entry of the program. Judgment nodes are designed to judge the partial information from the environments. Processing nodes are used for agents to take actions which enable agents to respond to the changing environments. These nodes are connected by directed links to form a compact graph structure, so that they could be reused many times, which helps to reduce the memory cost. The number of nodes and their function set are predetermined by designers before running the program, and their links are changed by evolution to generate optimal solutions. Fig. 1 shows the phenotype and genotype of GNP. A two-dimensional array is adopted to store the gene of nodes which contains both node information and connection information. In the node gene segment, 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. ID_i is the identification number of node *i*. 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}, ... represent the time delays spent on the transition from node *i* to node C_{i1}, C_{i2}, ..., respectively.

Genetic operators of GNP 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. 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. Tournament Selection and Elite Selection are adopted as the selection policies in this paper.

The node transitions of GNP correspond to the behaviors of agents. However, not all the nodes and connections are used equally in the execution of GNP. It is noticed that some transitions of judgments and processing appear frequently in elite individuals, which serve as "if-then" decision rules. These rules represent the good experiences of agents' past behaviors, and could be reused for guiding their future actions. We proposed a GNP based Rule Accumulation method (GNP-RA) to make use of them in the previous research, however, this method needs to be improved since good rules and bad rules are hard to distinguish. This paper focuses on how to distinguish them to improve the quality of the rule pool.

B. Memory Scheme

The research on discovering historical experiences on good examples and reuse them later has been conducted for many years, among which the most famous ones are Case-based Reasoning (CBR) and Memory-based Reasoning (MBR). CBR is an Artificial Intelligence approach to learning and problem solving based on past experiences. It combines the aspects from the knowledge-based systems with the ones from the machine learning field. The intrinsic feature is to solve new problems by retrieving relevant prior cases and adapting them to new situations. Similarly, MBR intensively uses memory to recall specific episodes from the past instead of rules. Two important factors of MBR are Distance Function and Combination Function. Distance Function is designed to find the most similar episodes in memory, while Combination Function is used to combine the attributes of the similar episodes for future prediction. The advantage of this memory scheme is that the past experiences could be retrieved and reused. The disadvantage is that it needs a large number of historical data, otherwise it is incompetent for accurate prediction. Storing and retrieving these data cost time. Another disadvantage is that it is more suitable for static environments or simple problems. For highly dynamic environments and very complicated problems, it is impossible to store all the cases into the memory because the solution space becomes almost infinite. In this case, memory scheme should be combined with rule based approaches, that is, inductive reasoning and deductive reasoning should be integrated.

C. Reinforcement Learning

Sarsa-learning is brought into GNP-RA to improve the quality of rule generators and generate reasonable decision rules. One advantage is that it can produce rules using the current information during task execution. The other advantage is a combination of the diversified search of evolution and the intensified search of RL, which contributes to generating better solutions. In order to realize Sarsa-learning, sub-nodes are introduced into the judgment and processing nodes, as described in Fig.2 using the tile-world model. The notation ID_{ij} is the identical number of subnode *j* in node *i*, and Q_{ij} is the Q-value of this subnode; C_{ij}^A, C_{ij}^B, ... denote the next nodes to which subnode *j* in node *i* connects according to different judgment results A, B, ... In this structure, each node is a "state" and selection of a sub-node is regarded as an "action". Q-values estimate the sum of discounted rewards to be obtained in the future. 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.

Nevertheless, because of the ε-greedy policy of Sarsa, some unreasonable connections could be visited, bringing some unimportant or even misleading rules. Therefore, pruning of these rules becomes necessary. In the previous research, we only extract the rules from elite individuals and store them as good experiences. In this paper, we also extract the rules from the worst individuals, which could be viewed as bad experiences. Rules in the good rule pool and bad rule pool are checked one by one and their common rules are picked up. These rules include some random actions or even misleading behaviors of agents. Thus, they should be deleted or their strength should be decreased. In this paper, we simply delete these rules and use the rest of the high quality rules in the rule pool for making decisions.

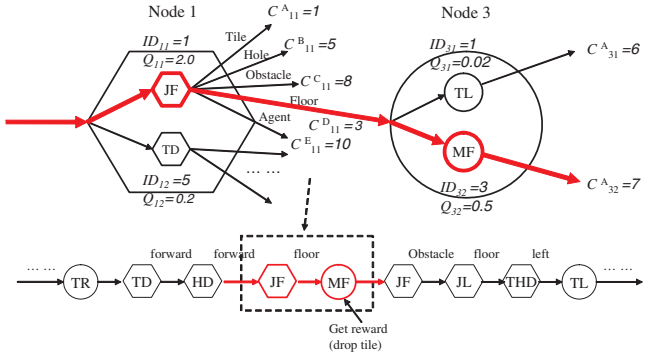

Fig. 2. Node structure and transition of GNP-Sarsa

III. The Proposed Method

The proposed method 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 unique matching calculation. The following three subsections will show the detail of them.

A. Rule Generation

The rule of GNP-RA is defined as a sequence of successive judgment nodes together with their judgment results and the subsequent processing node. The anticipant part of the rules represents the information on the judgment results of the environments, which is necessary for taking a particular action. The consequent part denotes what the agent should do under such environments. In Fig. 1, the node transition of 3→4→7 could be regarded as a rule, and 1→6→8→12 could be viewed as another rule.

We extract the rules from the elite individuals throughout the generations and store them into the GOOD rule pool, which represents the good experience set of agent's historical behaviors. Meanwhile, we also extract rules from the worst individuals and store them into the BAD rule pool, which represents the bad experiences, i.e., the passive precepts that tell the agent to avoid some particular actions. Rules are stored into the memory in the form of one-dimensional strings.

Then, the strength of the rule is defined to represent its importance by Eq. (1).

$$str(r) = fit(r) + C * N(r), \tag{1}$$

where, $str(r)$ is the strength of rule *r*; $fit(r)$ is the fitness of the elite individual from which rule *r* is extracted and *N(r)* is the number of extracted times of rule *r*.

B. Rule Pruning

Generally speaking, the rules in the GOOD rule pool are 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. We notice 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 common rules. These common rules are pruned from the GOOD rule pool, so that the really good ones could stand out and contribute more to the final reward. Fig. 3 shows an example of rule pruning.

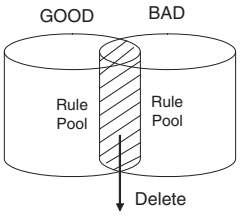

Fig. 3. An example of rule pruning.

C. Rule Matching

A unique average matching degree calculation is designed to make use of the good rules in the pruned rule pool. Firstly, we separate all the rules in the rule pool into $|C|$ classes according to their processing nodes, i.e., each class represents a specific action, where *C* is the set of suffixes of classes. Then, the matching degree of the new environment data *d* with rule *r* in class *k* is calculated as follows.

$$Match_k(d, r) = \frac{N_k(d, r)}{N_k(r)}, \tag{2}$$

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*.

Then, the mean and standard deviation of the matching degrees of data *d* with the rules in class *k* is calculated in order to set the threshold T_k to filter the lower matched rules in class *k* as follows. Only the rules whose matching degree is higher than the threshold could be picked up for the average matching degree calculation.

$$mean_k = \frac{1}{|R_k|} \sum_{r \in R_k} Match_k(d, r), \tag{3}$$

$$std_k = \sqrt{\frac{1}{|R_k|} \sum_{r \in R_k} (Match_k(d, r) - mean_k)^2}, \tag{4}$$

$$T_k = mean_k + 0.1 * std_k, \tag{5}$$

where, R_k is the set of suffixes of rules in class *k*.

After that, the average matching degree of environment data *d* with the picked up rules in class *k* is calculated as follows.

$$m_k(d) = \frac{1}{|R_k'|} \sum_{r \in R_k'} Match_k(d, r) * str_k(r), \tag{6}$$

where, R_k' is the set of suffixes of picked up rules in class *k*.

Finally, class k that has the highest average matching degree is selected and its corresponding action is taken by the agent.

$$k = arg \max_{k \in C} m_k(d), \tag{7}$$

After the agent takes actions, the environment is updated and a new environment data d is obtained. As a result, agents could take the next action. Fig. 4 shows the flowchart of the proposed method, in which A, B and C correspond to the subsection A, B and C of Section III, respectively.

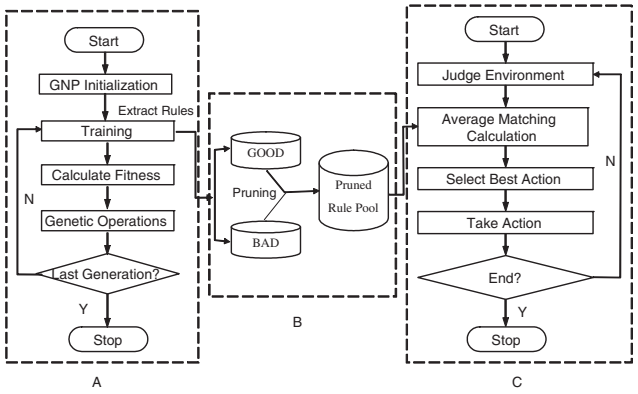

Fig. 4. Flowchart of the proposed method.

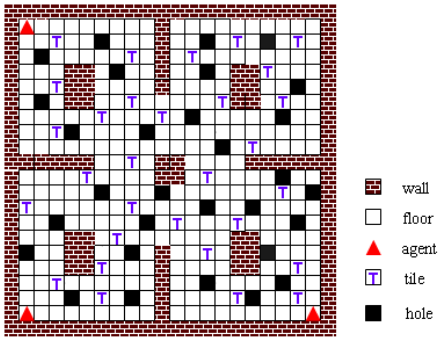

Fig. 5. An example of the tile-world

TABLE I
FUNCTION SET OF GNP NODES

Node	Symbol	Function
J1	JF	Judge Forward
J2	JB	Judge Backward
J3	JL	Judge Left
J4	JR	Judge Right
J5	TD	Judge direction of the nearest Tile
J6	HD	Judge direction of the nearest Hole
J7	THD	Judge direction of nearest Hole from nearest Tile
J8	STD	Judge direction of the second nearest Tile
P1	MF	Move Forward
P2	TL	Turn Left
P3	TR	Turn Right
P4	ST	Stay

IV. Simulation

A. Simulation Environment

Tile-world problem[8] is a good test-bed for the multi-agent problems, and is chosen as the simulation environment to confirm the effectiveness of the proposed method. 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 to its neighboring cell. The task of agents is to move tiles into holes as many and quickly as possible without hitting obstacles or dropping itself into holes. When 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 unknown beforehand and agents have limited sight. Fig. 5 shows an example of the tile-world whose size is 22×22. We set 8 kinds of judgment nodes (J1∼J8) and 4 kinds of processing nodes (P1∼P4) for the tile-world problem. J1 to J4 return the object information {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, 5: nothing}. Table I shows the function set of GNP nodes.

We extract rules from the best five individuals and store them 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 to 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 rule application stage, 10 new tile-worlds with different locations of tiles and wholes are used to test the performance of the proposed method. The parameter configuration for the simulation is described by Table II.

B. Results and Analysis

Fig. 6 shows the training results of the proposed method with ε being 0.1 and 0.2, respectively. From the results, it is noticed that the case of ε being 0.2 increases faster in earlier generations. This is because relatively larger ε enables GNP to search for optimal solutions in a broader space, which helps to increase the possibility of generating good individuals. In other words, the exploration ability of GNP is strengthened. However, the results are not satisfactory in later generations due to the lack of exploitation ability. ε=0.1 could get comparatively better results in later generations probably because it can well balance the exploration and exploitation.

TABLE II
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	0.1, 0.2

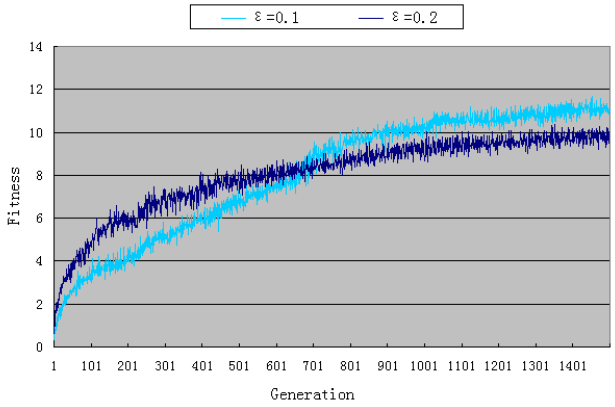

Fig. 6. Training results of the proposed method

TABLE III
AVERAGE FITNESS VALUE OF THE TESTING RESULTS

ε	GNP	GNP-RA	GNP-RA(Pruning)	Improvement Rate
0.1	6.05	7.84	8.46	7.91%
0.2	5.42	6.54	7.92	21.10%

Fig. 7 and Fig. 8 show the testing results of the proposed method under different εs. Three approaches, i.e., the conventional GNP method, GNP-RA without rule pruning and GNP-RA with rule pruning are compared and their performances are evaluated. The x axis is the testing tile-world 1-10 and the y axis is the fitness. Generally speaking, the results of GNP-RA is better than that of GNP. This is because the rule accumulation mechanism enables GNP to obtain the historical experiences from evolution, and many rules could be generated for making decisions. Furthermore, GNP-RA with rule pruning is better than GNP-RA without rule pruning in 7 cases out of 10. This could be explained by the fact that GNP-RA with rule pruning deleted the bad rules existing in the good rule pool, so that the good rules could stand out and contribute more to the final reward. Moreover, the average fitness value of GNP-RA with rule pruning is higher than that of the conventional GNP and GNP-RA without rule pruning, which demonstrates the effectiveness of the rule pruning approach. In the case of ε being 0.2, the improvement is more apparent. This is because ε=0.2 generates more random actions, bringing more common rules between the GOOD rule pool and the BAD rule pool. These rules decrease the quality of the GOOD rule pool and affect the process of decision making, thus pruning them becomes necessary. Table III shows the average fitness values of the three methods, and Table IV shows the number of rules in the GOOD and BAD rule pool as well as the proportions of the common rules under different εs.

Another simple pruning method could rank all the accumulated rules in the GOOD rule pool and delete the lowest ranking ones. For fair comparison, we select 7.94% of the rules in the GOOD rule pool as the worst ones and delete them, which is the same amount as the common rules between the GOOD rule pool and the BAD rule pool when ε=0.1. The performance of this ranking and deleting approach and GNP-RA without rule pruning is compared in Fig. 9. The ranking

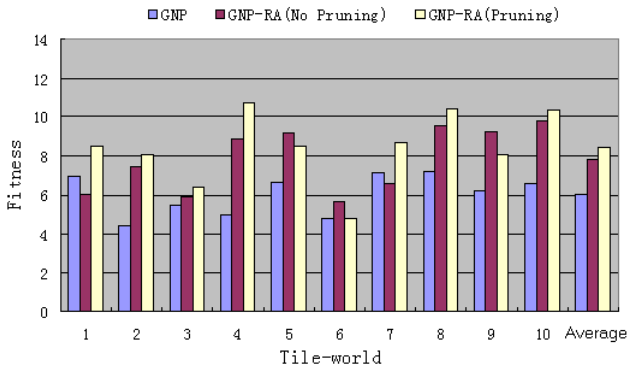

Fig. 7. Testing results of the proposed method when ε is 0.1

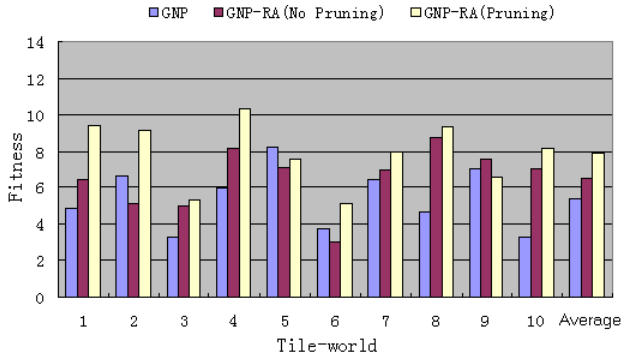

Fig. 8. Testing results of the proposed method when ε is 0.2

and deleting approach outperforms the conventional GNP-RA method in 4 cases out of 10, shows almost no difference in 3 cases and worse than the conventional method in 3 cases. The average fitness value of the ranking and deleting method is 8.02 and that of GNP-RA without rule pruning is 7.84. The improvement is not satisfactory in comparison with the proposed method. This is because the ranking and deleting approach may delete some good rules generated in earlier generations, during which the strength of the rules are low. This comparison demonstrates the efficiency and effectiveness of the proposed method from another point of view.

V. Conclusion and Future Work

A unique pruning method is designed in this paper to distinguish the good rules and bad rules accumulated by GNP-RA. The novel point is that not only good rules extracted from the best individuals are considered, but also the bad rules from the worst individuals are used to prune the bad rules in the GOOD rule pool. The rules representing random and misleading actions are removed from the GOOD rule pool, so that the quality of the rule pool is improved. The efficiency and effectiveness of the proposed method are proved in comparison with the conventional GNP-RA method without rule pruning and ranking and deleting method.

In the common rules between the GOOD rule pool and the BAD rule pool, there exit some neutral rules which don't directly contribute to the reward, but are essential to make

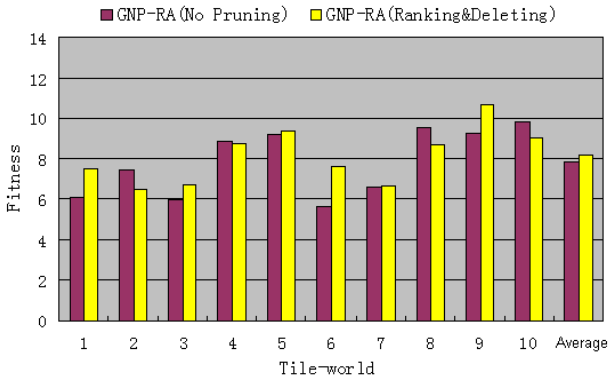

Fig. 9. The performance of rule pruning by ranking and deleting

TABLE IV
NUMBER OF RULES UNDER DIFFERENT εS

ε	GNP	GOOD Pool	BAD Pool	Common Rules	Proportion
0.1	0	4983	5576	396	7.94%
0.2	0	7453	8626	865	11.61%

decisions. In the future work, other ways for dealing with the common rules between the GOOD rule pool and the BAD rule pool are worth trying. For example, rather than simply deleting them, we could reduce their strength because they contribute both to the good behaviors and bad behaviors. In addition, we could extract rules from different number of individuals to see how it affects the proposed method.

REFERENCES

[1] S. Mabu, K. Hirasawa and J. Hu, "A Graph-Based Evolutionary Algorithm: Genetic Network Programming (GNP) and Its extension Using Reinforcement Learning", *Evolutionary Computation,* Vol. 15, No. 3, pp. 369-398, 2007.

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

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

[4] L. Wang, S. Mabu, F. Ye, S. Eto, X. Fan and 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. 551-560, 2009.

[5] A. Aamodt and E. Plaza, "Case-based reasoning: Foundational issues, methodological variations, and system approaches", *Artificial Intelligence Communications,* IOS Press, Vol. 7, No. 1, pp. 39-59, 1994.

[6] N. Mori, H. Kita and Y. Nishikawa, "Adaptation to Changing Environments by Means of the Memory Based Thermodynamical Genetic Algorithm," *Transactions of the Institute of Systems, Control and Information Engineers,* Vol. 14, No. 1, pp. 33-41, 2001.

[7] S. Mabu, Y. Lian, Y. Chen and K. Hirasawa, "Generating Stock Trading Signals Based on Matching Degree with Extracted Rules by Genetic Network Programming", *In Proc. of the SICE Annual Conference 2010, The Grand Hotel, Taipei, Taiwan,* pp. 1164-1169, 2010.

[8] M. E. Pollack and M. Ringuette, "Introducing the tile-world: Experimentally evaluating agent architectures", In T. Dietterich and W. Swartout, editors, *In Proc. of the conference of the American Association for Artificial Intelligence,* pp. 183-189, AAAI Press, 1990.