9

GA-Based Q-Learning to Develop Compact Control Table for Multiple Agents

Tadahiko Murata and Yusuke Aoki
Kansai University
Japan

1. Introduction

Recently reinforcement learning has received much attention as a learning method (Sutton, 1988; Watkins & Dayan, 1992). It does not need a priori knowledge and has higher capability of reactive and adaptive behaviors. However there are some significant problems in applying it to real problems. Some of them are deep cost of learning and large size of action-state space. The Q-learning (Watkins & Dayan, 1992), known as one of effective reinforcement learning, has difficulty in accomplishing learning tasks when the size of action-state space is large. Therefore the application of the usual Q-learning is restricted to simple tasks with the small action-state space. Due to the large action-state space, it is difficult to apply the Q-learning directly to real problems such as control problem for robots with many redundant degrees of freedom or multiple agents moving cooperatively one another.

In order to cope with such difficulty of large action-state space, various structural and dividing algorithms of the action-state space were proposed (Holland, 1986; Svinin et al., 2001; Yamada et al., 2001). In the dividing algorithm, the state space is divided dynamically, however, the action space is fixed so that it is impossible to apply the algorithm to the task with a large action space. In the classifier system, "don't care" attribute is introduced in order to create general rules. But, that causes partially observable problems. Furthermore, an ensemble system of general and special rules should be prepared in advance.

Considering these points, Ito & Matsuno (2002) proposed a GA-based Q-learning method called "Q-learning with Dynamic Structuring of Exploration Space Based on Genetic Algorithm (QDSEGA)." In their algorithm, a genetic algorithm is employed to reconstruct an action-state space which is learned by Q-learning. That is, the size of the action-state space is reduced by the genetic algorithm in order to apply Q-learning to the learning process of that space. They applied their algorithm to a control problem of multi-legged robot which has many redundant degrees of freedom and a large action-state space. By applying their restriction method for the action-state space, they successfully obtained the control rules for a multi-legged robot by their QDSEGA. However, the way to apply a genetic algorithm in their approach seems so straightforward. Therefore we have proposed a crossover for QDSEGA (Murata & Yamaguchi, 2005; Murata & Yamaguchi, 2008). Through their computer simulations on a control problem of a multi-legged robot, they could make about 50% reduction of the number of generations to obtain a target state of the problem.

Source: New Achievements in Evolutionary Computation, Book edited by: Peter Korosec,
ISBN 978-953-307-053-7, pp. 318, February 2010, INTECH, Croatia, downloaded from SCIYO.COM

In this chapter, we apply the QDSEGA with the neighboring crossover to control multiple agents. An application of QDSEGA to multiple agent system has been considered (Ito and Gofuku, 2003; Ito et al., 2004) though, they still applied genetic operators straightforward. We apply the neighboring crossover to Multi Agent Simulations (MAS) problem and show its effectiveness to reduce the number of actions in a Q-table. We also propose a deletion algorithm to make more compact Q-table in MAS problem. We employ the application in Ito et al. (2004) where a Q-table is developed for homogeneous multiple agents. Computer simulation results show that the size of Q-table can be reduced by introducing the proposed neighboring crossover and the deletion algorithm.

2. QDSEGA

In this section, we briefly explain the outline of QDSEGA (Ito & Matsuno, 2002; Ito & Gofuku, 2003; Ito et al., 2004; Murata & Yamaguchi, 2005). QDSEGA has two dynamics. One is a learning dynamics based on Q-learning and the other is a structural dynamics based on Genetic Algorithm. Figure 1 shows the outline of QDSEGA. In QDSEGA, each action is represented by an individual of a genetic algorithm. According to actions defined by a set of individuals, an action-state space called Q-table is created. Q-learning is applied to the created Q-table. Then the learned Q-table is evaluated through simulations. A fitness value for each action is assigned according to Q-table. After that, each individual (i.e., each action) is modified through genetic operations such as crossover and mutation. We show some details in these steps in the following subsections, and show our proposed method for crossover and a deletion algorithm in the next section.

2.1 Action encoding

Each individual expresses a selectable action on the learning dynamics. It means that a set of individuals is selected by genetic operations, and a learning dynamics is applied to the subset. After the evaluation of the subset of actions, a new subset is restructured by genetic operations.

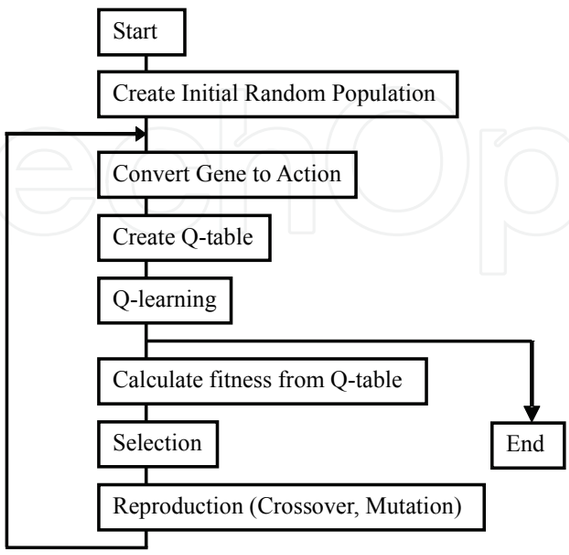


Fig. 1. Outline of QDSEGA

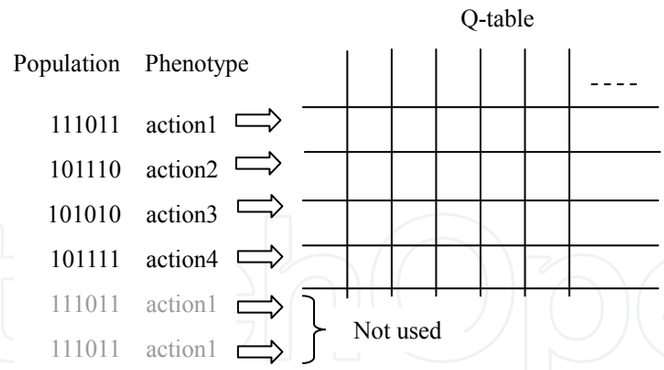


Fig. 2. Q-table created from a set of individuals

2.2 Q-table

An action-state space called Q-table is created from the set of individuals. When several individuals are the same code, only one action is used in the action-state space to avoid the redundancy of actions. Figure 2 shows this avoidance process.

2.3 Learning dynamics

In QDSEGA, the conventional Q-learning (Watkins & Dayan, 1992) is employed as a learning dynamics. The dynamics of Q-learning are written as follows:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\{r(s,a) + \gamma \max_{a'} Q(s',a')\}\,, \tag{1}$$

where $Q(s,a)$ is a Q-value of the state s and the action a, r is the reward, α is the learning rate, and γ is the discount rate.

2.4 Fitness

The fitness $fit(a_i)$ for each action is calculated by the following equation:

$$fit(a_i) = fit_Q(a_i) + k_f \cdot fit_u(a_i)\,, \tag{2}$$

where $fit_Q(a_i)$ is a fitness value for action a_i calculated from Q-table, $fit_u(a_i)$ is a fitness value for action a_i calculated from the frequency of use, and k_f is a non-negative constant value to determine the ratio of $fit_Q(a_i)$ and $fit_u(a_i)$. We show the detail explanation of these factors in this subsection.

(a) Fitness of Q-table

The fitness of Q-table $fit_Q(a_i)$ is calculated from Q-values in the current Q-table. In order to calculate $fit_Q(a_i)$ for each action a_i the following normalization is taken place in advance as for the Q-values in the current Q-table.

First, calculate the maximum and minimum value of each state as follows:

$$V_{\max}(s) = \max_{a'}(Q(s,a'))\,, \tag{3}$$

$$V_{\min}(s) = \min_{a'}(Q(s,a')) . \quad (4)$$

Then Q' of the normalized Q-table is given as follows:

$$\text{If } Q(s,a) \geq 0 \text{ Then } Q'(s,a) = \frac{1-p}{V_{\max}(s)} Q(s,a) + p , \quad (5)$$

$$\text{Else } (Q(s,a) < 0) \text{ Then } Q'(s,a) = -\frac{p}{V_{\min}(s)} Q(s,a) + p , \quad (6)$$

where p is a constant value which means the ratio of reward to penalty. After this normalization process, we fix the action a_i and sort $Q'(s,a_i)$ according to their value from high to low for all states. We define the sorted $Q'(s,a_i)$ as $Q'_s(s,a_i)$, and $Q'_s(1,a_i)$ means the maximum value of $Q'(s,a_i)$, and $Q'_s(N_s,a_i)$ means the minimum value of $Q'(s,a_i)$, where N_s is the size of states. Using the normalized and sorted Q-value $Q'_s(s,a_i)$, the fitness of action a_i is calculated as follows:

$$fit_Q(a_i) = \sum_{j=1}^{Ns} (w_j \frac{\sum_{k=1}^{j} Q'_s(k,a_i)}{j}) , \quad (7)$$

where w_j is a weight which decides the ratio of special actions to general actions.

(b) Fitness of Frequency of Use

The fitness of frequency of use $fit_u(a_i)$ is introduced to save important actions. That fitness is defined as follows:

$$fit_u(a_i) = N_u(a_i) / \sum_{j=1}^{N_a} N_u(a_j) , \quad (8)$$

where N_a is the number of all actions of one generation and $N_u(a_i)$ is the number of times which a_i was used for in the Q-learning of this generation. Important actions are used frequently. Therefore the actions with high fitness value of $fit_u(a_i)$ are preserved by this fitness value.

2.5 Genetic algorithm and neighboring crossover

Ito & Matsuno (2002) says "the method of the selection and reproduction is not main subject so the conventional method is used." They employed a crossover that exchanges randomly selected bits between the parent individuals according to the crossover probability P_c. They mutated each bit according to the mutation probability P_m. They did not replace parent individuals with offspring. Therefore the number of individuals is increased by the genetic operations. As for the elite preserving strategy, they preserve 30% individuals with the highest fitness value.

Since they did not modify genetic operators for QDSEGA, we have proposed a crossover operation for the multi-legged robot control problem (MRC problem) in (Murata & Yamaguchi, 2005; Murata & Yamaguchi, 2008). We developed a neighboring crossover for QDSEGA for MRC problems.

The crossover employed in QDSEGA (Ito & Matsuno, 2002) causes drastic change in the phenotype of a solution since randomly selected bits are changed between two solutions. If

the change of solution in phenotype is so drastic, the good part of the solution may be broken. In order to avoid causing such drastic change among solutions, we proposed a crossover between similar parent solutions. We define the similarity by the number of the same genes in the same locus of a chromosome. We introduced a parameter k_{sim} to denote the similarity. Thus, the crossover is applied among individuals that have the same genes more than k_{sim} .

This kind of the restriction for the crossover has been proposed in the research area of distributed genetic algorithms (DGAs). Researches on DGAs can be categorized into two areas: coarse-grained genetic algorithms (Tanese, 1989; Belding, 1995) and fine-grained genetic algorithms (Mandelick & Spiessens, 1989; Muhlenbein et al., 1991; Murata et al., 2000). In the coarse-grained GAs, a population, that is ordinarily a single, is divided into several subpopulations. Each of these subpopulations is individually governed by genetic operations such as crossover and mutation, and subpopulations communicate each other periodically. Algorithms in this type are called the island model because each subpopulation can be regarded as an island. On the other hand, several individuals are locally governed by genetic operations in fine-grained GAs. In a fine-grained GA, each individual exists in a cell, and genetic operations are applied to an individual with individuals in neighboring cells. The DGAs are known to have an advantage to keep the variety of individuals during the execution of an algorithm, and avoid converging prematurely.

While we don't define any solution space such as cells or islands in our proposed crossover, our restriction in crossover operation may have the same effect of keeping variety in a population and attain the effective search.

3. Transportation task using Q-learning

3.1 Transportation task

We consider a transportation task shown in Ito & Gofuku (2003) and Ito et al. (2004). Figure 3 shows a transportation task used in this chapter. There is a world with 25 cells and a goal cell shown in "G" where five agents exist in Cell 0 and Cell 4. The aim of the transportation task is to convey a load shown in "L1" to the goal cell. In order to carry "L1" to "G", the

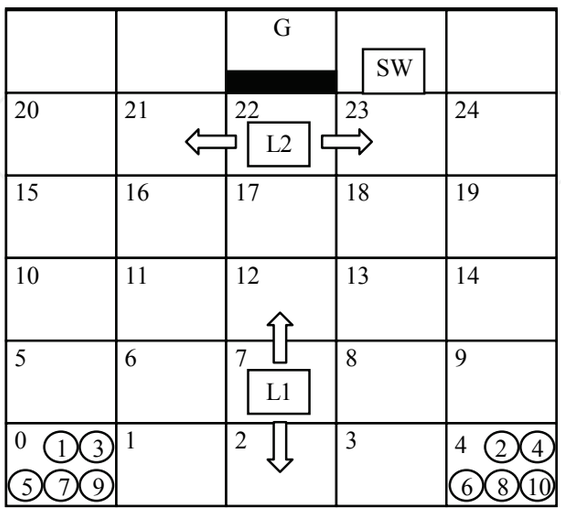


Fig. 3. Transportation task

other load shown in "L2" should be removed from Cell 22. Simultaneously, the door of "G" should be opened before carrying "L1". To open the door, the switch shown in "SW" should be pushed by an agent in Cell 23.

Each load has a mobile direction. As shown in Figure 3, "L1" can be moved only in the vertical direction, and "L2" only in the horizontal direction. To move a load, more than one agent should push it toward the same movable direction. Therefore, to convey "L1" to "G", agents should remove "L2" from Cell 22, open the door, and move "L1" to the goal.

In order to control actions of each agent, Ito & Gofuku, (2003); Ito et al., (2004) employed their QDSEGA where the state of the agent is handled as a chromosome of an individual to which genetic operators are applied. Figure 4 shows the chromosome representation of the agent location in Figure 3. Each chromosome consists of genes with the same number of agents. The figure in each gene shows the identification number of cell where the agent locates. Since the agents with odd number locate in Cell 0, all genes for those agents have 0 as its value.

	1	2	3	4	5	6	7	8	9	10
Chromosome	0	4	0	4	0	4	0	4	0	4

Fig. 4. Chromosome representation for the agents in Figure 3

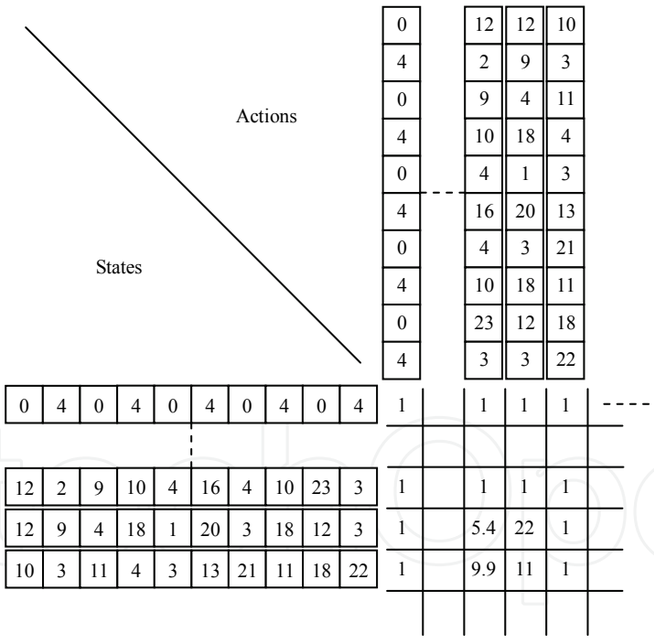


Fig. 5. An example of Q-table with a set of chromosomes

3.2 Q-learning in our simulation

Q-table of the Q-learning is generated using a set of chromosomes. Figure 5 shows an example of Q-table that shows the relations of agent locations.

In Figure 5, each column in the Q-table shows a chromosome generated by genetic operations. Rows of the table consist of the same chromosomes of the columns. That is, the chromosomes in the rows act as states in the Q-table, and the chromosomes in the column

act as actions the agent can take in the fired state. When the ten agents locates in the start position (Cell 0 or Cell 4 as in Fig. 3), the current position is shown as (0, 4, 0, 4, 0, 4, 0, 4, 0, 4) in the table. If the target position (10, 3, 11, 4, 3, 13, 21, 11, 18, 22) is selected as an action from the current position, Agent "1" moves from Cell 0 to Cell 10, Agent "2" moves from Cell 4 to Cell 3, and so on.
With this Q-table, Q-learning is applied. In order to move each agent to the target position, Ito & Gofuku, (2003) and Ito et al., (2004) proposed the following rules.

<**R1**: The rule to decide a path to a target position>

If $x(i) \neq x_t$ **Then** $x(i+1) = x(i) - \mathrm{sgn}(x(i) - x_t)\Delta x$, $y(i+1) = y(i)$,
Else if $y(i) \neq y_t$ **Then** $y(i+1) = y(i) - \mathrm{sgn}(y(i) - y_t)\Delta y$, $x(i+1) = x(i)$,
Else $x(i+1) = x(i)$, $y(i+1) = y(i)$.

where x_t, y_t are the coordinates of the target position, and $x(i), y(i)$ are the coordinates of the current position of the agent in time *i*. Using this rule, the agent moves in horizontal direction first, then it moves vertically to the target position.

<**R2**: The rule to avoid a collision>

If *obstacle is on the course that is given by R1* **Then**
  If *the obstacle is load* **Then** *Employ R3*
  Else *Don't move*
Else *Move using R1*

Since the collision between agents is assumed to avoid using traffic rules, Ito & Gofuku, (2003); Ito et al., (2004) considered only the collision between an agent and a load. If the load can not be carried by the agent alone, it should stop until other agents come.

<**R3**: The rule to move the load>

If *Load is on the course that is given by R1* **Then** *Push the Load to the way that the agent has to go*
Else *Move using R1*

If the way that the agent has to go is not the direction to which the load can be moved, the agent should stop beside the load.

<**R4**: The rule to open the door>

If *Switch is in a cell where the agent stops* **Then** *Turn on the switch to open the door*
Else *Nothing is done*

3.3 A deletion algorithm to create more compact control table

When we observe a Q-table developed by QDSEGA, some actions or chromosomes are not used in moving multiple agents. That is, unnecessary actions are generated through genetic operations. In order to make a compact Q-table, we mark the chromosomes that are not used for a prespecified term in Q-Learning process.

4. Computer simulation

4.1 Parameter specifications

In this section, we show the simulation results to compare the conventional QDSEGA and the QDSEGA with the neighboring crossover shown in Subsection 2.5 and the deletion

algorithm in Subsection 3.3. The neighboring crossover can be applied to the parent solutions that have the same genes more than k_{sim}. In this paper, we employed $k_{sim} = 0, 2, 4, 6, 8$. Since $k_{sim} = 10$ means the crossover between the same chromosomes, we did not use. When $k_{sim} = 0$, the crossover is applied between any parent solutions. The deletion algorithm is applied when the reward for the developed Q-table becomes larger than 100. This means that the deletion algorithm is applied after attaining the goal by multiple agents.

We employed the same parameter specifications as shown in Ito & Gofuku (2003) and Ito et al. (2004) except the learning rate and the discount rate in Equation (1). We found better specifications for those parameters by preliminary simulations:

[Genetic Algorithm]

The number of individuals: 300,
Selection: Roulette selection,
Type of crossover: uniform crossover,
The probability of crossover: 0.2,
Type of mutation: change the value among valid cell number,
The probability of mutation: 0.001,
The number of generations: 100,
Weights in Equation (2): $k_f = 200$,
Weights in Equation (7): $w_1 = 0.5, w_{N_s} = 0.5$, $w_i = 0 (i = 2,..., N_{s-1})$.

[Q-learning]

Reward: When "L1" reaches the goal, $r = 100$,
When "L1" moves up or "L2" is removed, $r = 20$,
When "L1" moves down or "L2" blocks the course of "L1", $r = -20$,
When any agent can not move to the target position, $r = -10$,
Learning rate in Equation (1): $\alpha = 0.9$,
Discount rate in Equation (1): $\gamma = 0.1$,
ε-greedy action selection: 10% random action,
The number of trials of each learning dynamics: 10,000.

4.2 Simulation results

Figures 6 and 7 show that the average reward for the obtained Q-table and an average number of actions (or situations) in Q-table. The average reward for Q-table is calculated over the last 100 trials among 10,000 trials. The maximum average reward is 130. These figures show that the proposed QDSEGA with $k_{sim} = 2, 4$ could obtain the better or similar average reward with comparing to the algorithm without the neighboring crossover. As for the number of actions, the larger k_{sim} enables the less number of actions as shown in Figure 7. This shows that a compact Q-table can be obtained using the proposed neighboring crossover. Obtaining a compact Q-table enables users to find important actions to control the multiple agents.

In order to obtain a compact Q-table with high average reward, we apply our proposed neighboring crossover after the average reward becomes larger than 100. Since the

neighboring crossover is applied to the similar parent solutions, that crossover often produces the offspring that is the same chromosome. This causes the reduction of the size of Q-table. As shown in Figure 7, the number of actions in the Q-table reduced rather than the previous QDSEGA. However, this reduction may prevent improving the performance in the average reward.

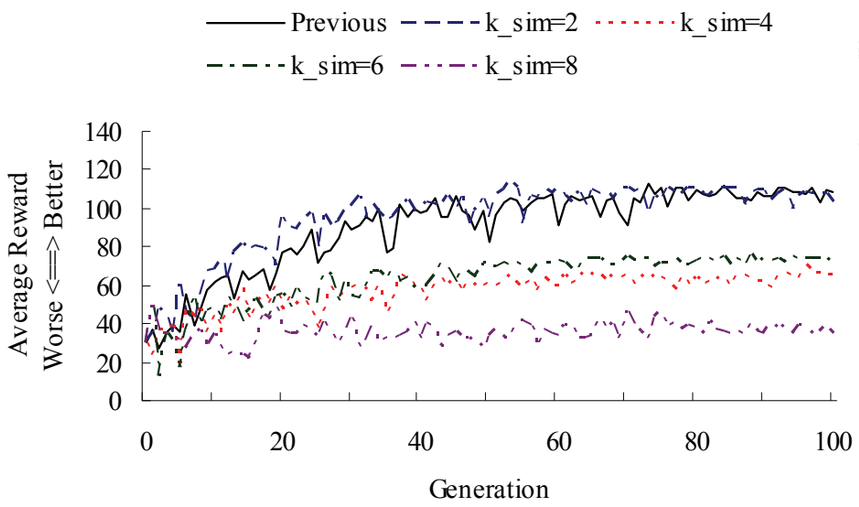


Fig. 6. Gain attained by Q-learning generated by QDSEGA

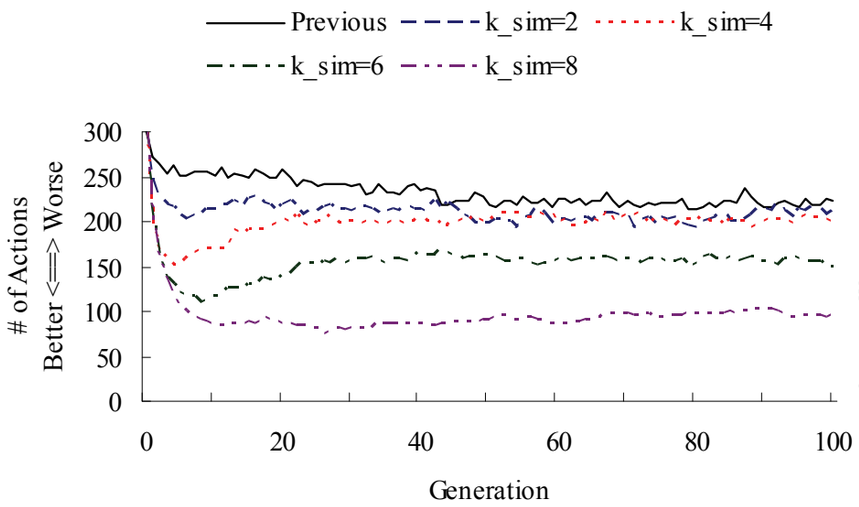


Fig. 7. The number of actions in Q-table generated by QDSEGA

Figures 8 and 9 show that the average reward and the average number of actions in Q-table. From these figures, we can see that the proposed QDSEGA can keep the high average reward with any value of k_{sim}. Figure 9 shows that the large value of k_{sim} enables to reduce the number of actions in Q-table.

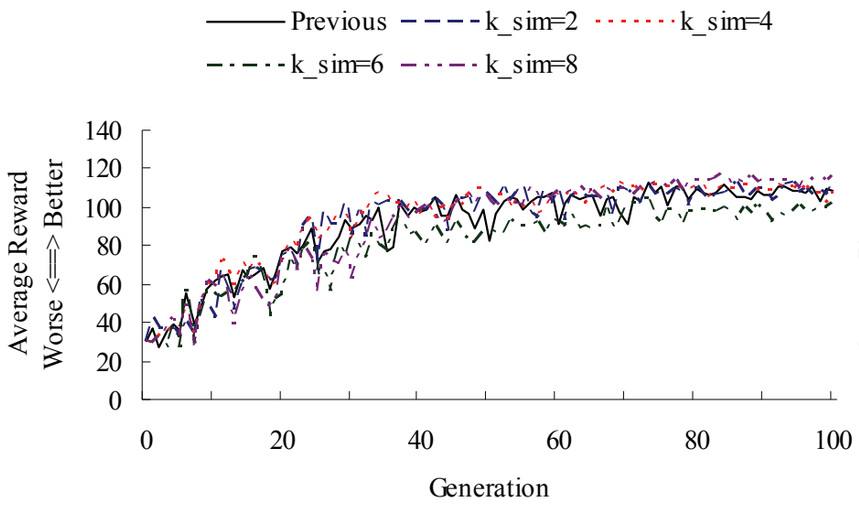


Fig. 8. Gain attained by Q-learning generated by QDSEGA applied the neighboring crossover when obtaining 100 reward

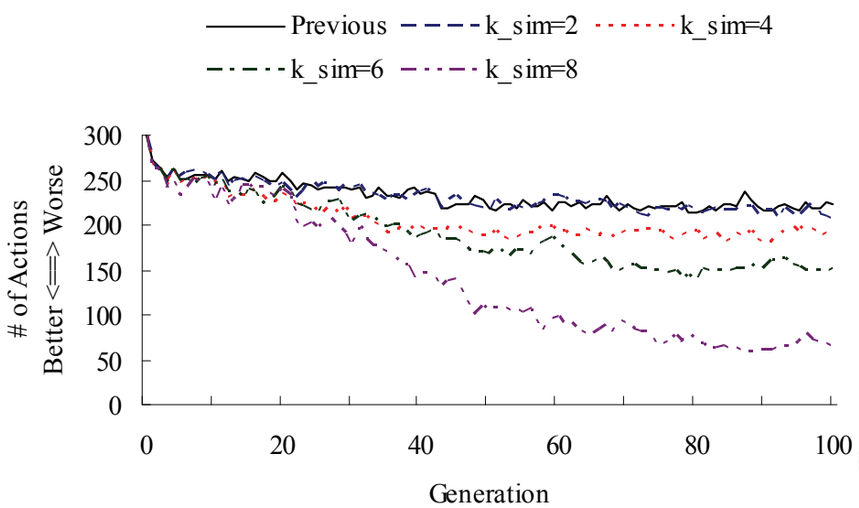


Fig. 9. The number of actions in Q-table generated by QDSEGA applied the neighboring crossover when obtaining 100 reward

Although the neighboring crossover has an effect to reduce the number of actions, there are some actions that are not used in moving agents. Therefore, we apply the deletion algorithm in Subsection 3.3. Figures 10 and 11 show the results of QDSEGA with neighboring crossover and the deletion algorithm. From these figures, we can see that the deletion algorithm does not degrade the performance in the average reward but have a fine effect to reduce the number of actions. By combining the neighboring crossover and the deletion algorithm, we could obtain more compact control table with high performance than using the previous algorithms.

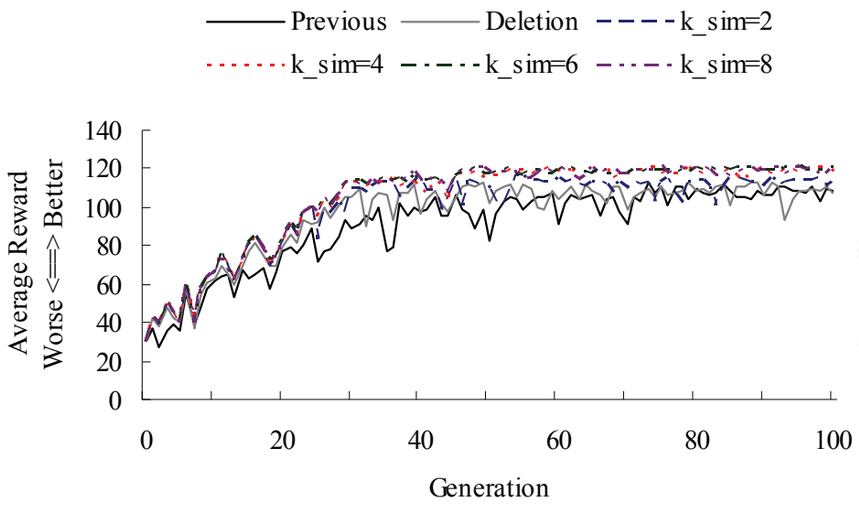


Fig. 10. Gain attained by Q-learning generated by QDSEGA applied the neighboring crossover and the deletion algorithm when obtaining 100 reward

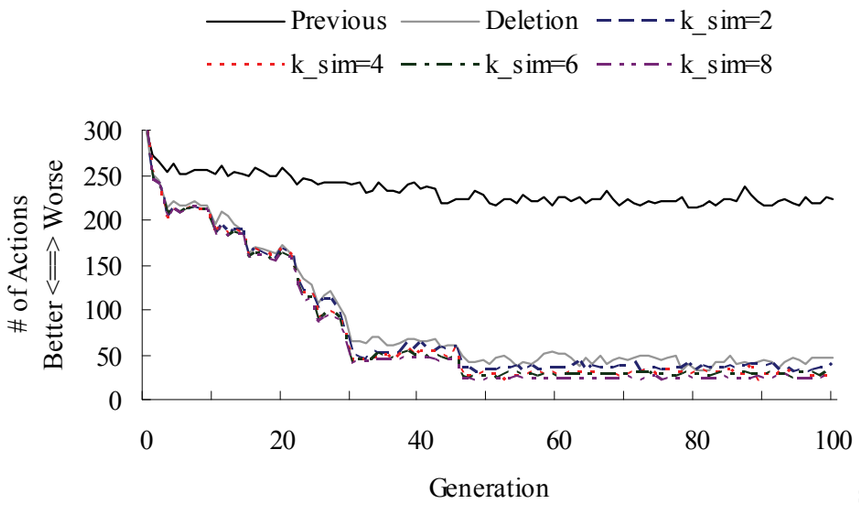


Fig. 11. The number of actions in Q-table generated by QDSEGA applied the neighboring crossover and the deletion algorithm when obtaining 100 reward

Table 1 shows the average number of actions obtained at the final generation. From this table, we can see that the number of actions is reduced by the neighboring crossover and the deletion algorithm. Especially the deletion algorithm could reduce it without degrading the performance of the developed control table using neighboring crossover.

After obtaining a compact control table, we can examine the states and actions that are used to reach the goal. We can see that in order to achieve the task to bring "L1" to the goal, only two actions are required from the initial states shown in Figure 3. For example, the two actions in Figure 12 are enough to convey "L1" to the goal with ten agents. Figure 13 shows the states or positions of the agents according to the obtained states shown in Figure 12.

Without Deletion Algorithm	Previous	2	4	6	8
# of actions	222.2	210.8	196.9	149.2	96.5

With Deletion Algorithm	Previous	2	4	6	8
# of actions	46.6	39.0	28.0	30.2	25.1

Table 1. Size of the Q-table at the final generation

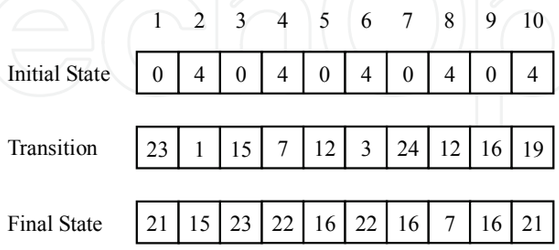


Fig. 12. Succession of the states to achieve the goal

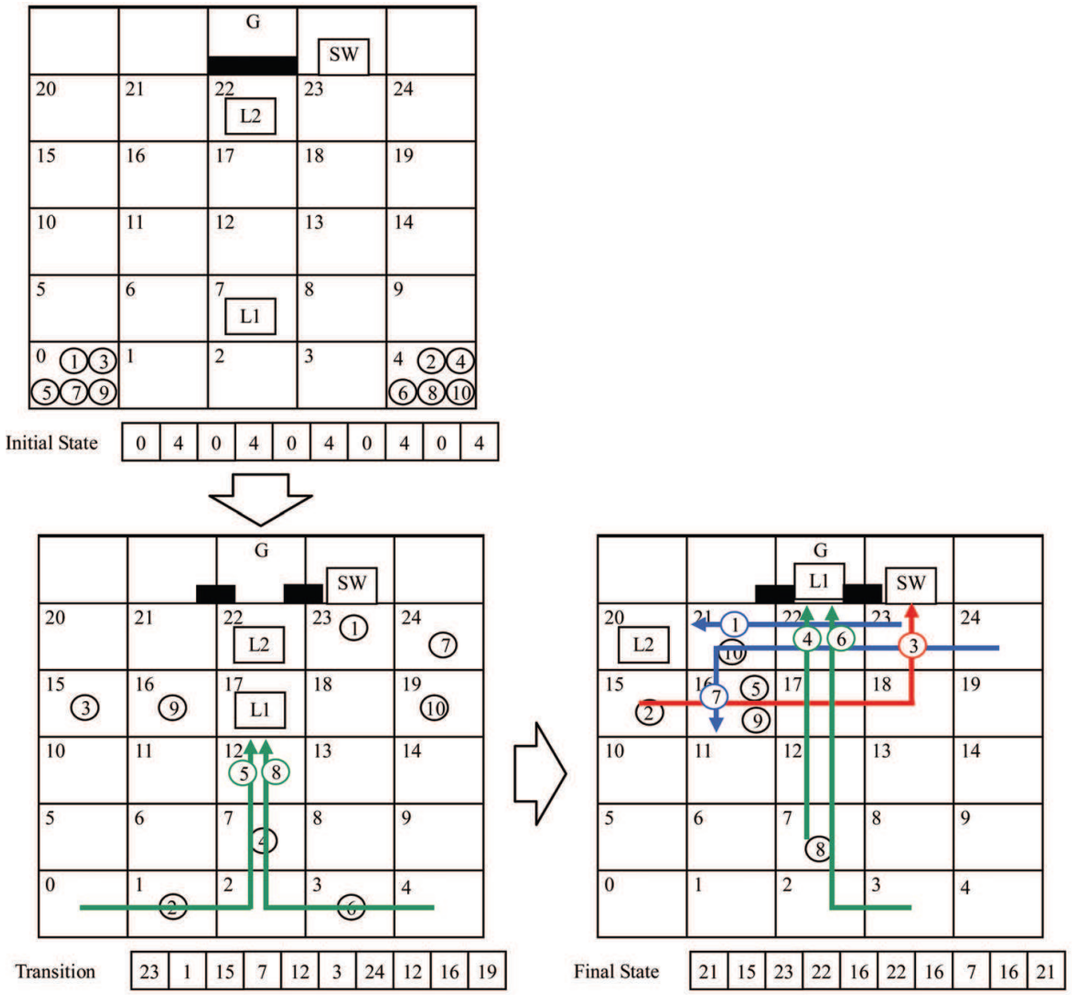


Fig. 13. Achievement of carrying the load to the goal

5. Conclusion

In this chapter, we show the effectiveness of the neighboring crossover and the deletion algorithm especially in reducing the size of the Q-table. By reducing the Q-table, it becomes easy to read the Q-table that is required for attaining the objective to reach the goal and minimizes the memory to store the developed control table.
As for other further study, we can bring other objective functions to achieve the goal. In Figures 6, 8, and 10, we compared the average reward as shown in the previous study (Ito and Gofuku, 2003; Ito et al., 2004). From these figures, we could minimize the total moving cost of all the agents to achieve the goal.
Furthermore, Ito and Gofuku (2003) examined the effectiveness of QDSEGA for multi-agent system with heterogeneous ability. We can show the effectiveness of the neighboring crossover in that problem too.

6. Acknowledgments

This work was partially supported by the MEXT, Japan under Collaboration with Local Communities Project for Private Universities starting 2005.

7. References

Belding, T. C. (1995). The distributed genetic algorithm revisited. *Proceedings of 6th International Conference on Genetic Algorithms*, pp. 114-121, ISBN 1558603700, University of Pittsburgh, July 1995, Morgan Kaufmann Publishers, Inc., San Francisco, USA

Holland, J. H. (1986). Escaping brittleness: the possibilities of general purpose learning algorithms applied to parallel rulebased system, *Machine Learning: An artificial intelligence approach*, Vol. 2, pp. 593-623, Morgan Kaufmann Publishers Inc., ISBN 0934613001, San Francisco, USA

Ito, K. & Matsuno, F. (2002). A study of reinforcement learning for the robot with many degrees of freedom –Acquisition of locomotion patterns for multi legged robot-, *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 392-397, ISBN 0780372727, Washington, D.C., USA, May 2002, Institute of Electrical and Electronics Engineers, Inc., Piscataway, USA

Ito, K. & Gofuku, A. (2003). Hybrid autonomous control for heterogeneous multi-agent system, *Proceedings of 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2500-2505, ISBN 0780378601, Las Vegas, October 2003, Institute of Electrical and Electronics Engineers, Inc., Piscataway, USA

Ito, K., Gofuku, A. & Takeshita, M. (2004). Hybrid autonomous control for multi mobile robots, *Advanced Robotics*, Vol. 18, No. 1, pp. 83-99, ISSN 01691864

Mandelick, B. & Spiessens, P. (1989). Fine-grained parallel genetic algorithms, *Proceedings of 3rd International Conference on Genetic Algorithms*, pp. 428-433, ISBN 1558600063, George Mason University, June 1989, Morgan Kaufmann Publishers, Inc., San Francisco, USA

Muhlenbein, H., Schomisch, M. & Born, J. (1991). The parallel genetic algorithm as function optimizer, *Proceedings of 4th International Conference on Genetic Algorithms*, pp. 271-

278, ISBN 1558602089, San Diego, July 1991, Morgan Kaufmann Publishers, Inc., San Francisco, USA

Murata, T. & Yamaguchi, M. (2005). Neighboring Crossover to Improve GA-Based Q-Learning Method for Multi-Legged Robot Control, *Proceedings of Genetic and Evolutionary Computation Conference 2005*, pp. 145-146, ISBN 1595930108, Washington, D.C., June 2005, The Association for Computing Machinery, Inc., New York , USA

Murata, T. & Yamaguchi, M. (2008). Multi-Legged Robot Control Using GA-Based Q-Learning Method With Neighboring Crossover, In: *Frontiers in Evolutionary Robotics*, Iba (Ed.), pp. 341-352, I-Tech Education and Publishing, ISBN 9783902613196, Vienna, Austria

Murata, T., Ishibuchi, H. & Gen, M. (2000). Cellular genetic local search for multi-objective optimization, *Proceedings of Genetic and Evolutionary Computation Conference 2000*, pp. 307-314, ISBN 1558607080, Las Vegas, July 2000, The Association for Computing Machinery, Inc., New York, USA

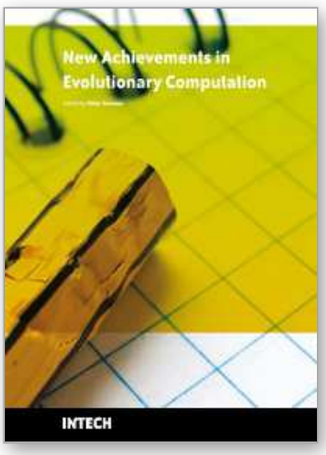
Sutton, R. S. (1988). *Reinforcement Learning: An Introduction*, The MIT Press, ISBN 0262193981, Cambridge, USA

Svinin, M., Ushio, S., Yamada, K. & Ueda, K. (2001). An evolutionary approach to decentralized reinforcement learning for walking robots, *Proceedings of the 6th International Symposium on Artificial life and Robotics*, pp. 176-179, Tokyo, January 2001

Tanese, R. (1989). Distributed genetic algorithms, *Proceedings of 3rd International Conference on Genetic Algorithms*, pp. 434-439, ISBN 1558600063, George Mason University, June 1989, Morgan Kaufmann Publishers, Inc., San Francisco

Watkins, C.J.C.H. & Dayan, P. (1992). Technical note q-learning, *Machine Learning*, Vol. 8, pp. 279-292

Yamada, K., Ohkura, K., Svinin, M. & Ueda, K. (2001). Adaptive segmentation of the state space based on bayesian discrimination in reinforcement learning, *Proceedings of the 6th Int. Symposium on Artificial life and Robotics*, pp. 168–171, Tokyo, January 2001



New Achievements in Evolutionary Computation

Edited by Peter Korosec

ISBN 978-953-307-053-7
Hard cover, 318 pages
Publisher InTech
Published online 01, February, 2010
Published in print edition February, 2010

Evolutionary computation has been widely used in computer science for decades. Even though it started as far back as the 1960s with simulated evolution, the subject is still evolving. During this time, new metaheuristic optimization approaches, like evolutionary algorithms, genetic algorithms, swarm intelligence, etc., were being developed and new fields of usage in artificial intelligence, machine learning, combinatorial and numerical optimization, etc., were being explored. However, even with so much work done, novel research into new techniques and new areas of usage is far from over. This book presents some new theoretical as well as practical aspects of evolutionary computation. This book will be of great value to undergraduates, graduate students, researchers in computer science, and anyone else with an interest in learning about the latest developments in evolutionary computation.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen