International Journal of Innovative Computing, Information and Control
Volume **9**, Number **2**, February **2013**
ICIC International ©2013 ISSN 1349-4198
pp. **641–650**

K-CUT CROSSOVER USING GRAPH THEORY IN GENETIC NETWORK PROGRAMMING

Hiroaki Murata[1], Makoto Koshino[2] and Haruhiko Kimura[3]

[1]Institute of Nature and Environmental Technology
[3]Graduate School of Natural Science
Kanazawa University
Kanazawa 920-1192, Japan
hiro@blitz.ec.t.kanazawa-u.ac.jp

[2]Department of Electronics and Information Engineering
Ishikawa National College of Technology
Kahoku-gun 929-0392, Japan

Received December 2011; revised April 2012

Abstract. *In this study, we focus on Genetic Network Programming (GNP) which is the graph-based evolutionary algorithm. Similar to Genetic Algorithm (GA) and Genetic Programming (GP), GNP applies genetic operators to an individual, which is represented by a directed graph, in order to solve a given problem. GNP is usually applied to automatic generation of programs which control a mobile robot. Since the crossover exchanges a sub-graph of parent individuals, a selection of a sub-graph is an important factor. Some selection methods are proposed in previous work. However, the selection method based on the graph theory is not proposed even though the individual is represented by a graph. In this study, we propose a k-cut crossover based on the graph theory. The proposed k-cut crossover selects a sub-graph by using a minimum k-cut algorithm which finds a minimum graph partition on weighted graph. We applied the GNP with the k-cut crossover to the automatic generation of programs in the tileworld, and the experimental result shows the advantage of the k-cut crossover.*

Keywords: K-cut crossover operator, Genetic network programming, GNP, Minimum cut problem, Frequency of edge usage, Graph theory

1. Introduction.

Since it is very difficult to obtain an optimum solution in many real world problems, we try to obtain a near-optimum solution. The accuracy of a required solution varies in the use of the solution. Evolutionary Algorithm (EA) is able to find a near-optimum solution by improving a solution iteratively. The evolutionary algorithm is widely applied and studied. In recent years, Katagiri et al. proposed a new graph-based evolutionary algorithm named Genetic Network Programming (GNP) [1]. Similar to Genetic Algorithm (GA) and Genetic Programming (GP), GNP applies genetic operators to an individual, which is represented a directed graph, in order to solve a given problem. The individual in GNP is composed of judgment nodes, processing nodes and start node, which are connected to each other. Judgment nodes have if-then type branch decision function, which return judgment results for assigned inputs and determine the next node. Processing nodes have an action or a processing function, which are defined according to the given problem. For example, when a mobile robot control is given problem, those functions are defined as “turn left”, “turn right”, and so on. There is only one start node in one individual, and this node has no function. The only role of this is to determine the first node to be executed.

Initially, the GNP was applied to automatic generation of programs which control a mobile robot [1, 2], and then came to be applied to data mining such as association rule mining [3, 4, 5, 6]. Up to now, GNP has been successfully applied to many fields such as elevator supervisory control systems [7], stock market prediction [8, 9, 10, 11, 12, 13] and traffic prediction [14, 15, 16].

Additionally, several improved GNP algorithms are also proposed [17, 18, 19, 20]. In [17], the multi-start node is introduced in order to create plural programs simultaneously in one individual. In [18, 19], the local search using reinforcement learning is proposed. In [20], the GNP using subroutine is proposed. In contrast, there are only a handful of studies on basic genetic operator such as mutation and crossover operator [21]. In addition, the crossover operator based on graph theory is not proposed even though an individual is represented by a graph.

In this study, we propose a k-cut crossover based on the graph theory. In the k-cut crossover, exchanged sub-graph is determined by using a minimum k-cut algorithm which finds a minimum graph partition on weighted graph. We consider the weight of arc as the frequency of usage of arc during implementation, and that the k-cut crossover keeps a frequently used arc. We applied the GNP with the k-cut crossover to the automatic generation of programs in the tileworld which is one of the garbage collection problems, and the experimental results show the merits of k-cut crossover.

2. Genetic Network Programming.

2.1. The architecture of the GNP.

GNP is defined by the tuple $< \mathbf{I}, \mathbf{S}, \mathbf{R}, \mathbf{G}, \mathbf{F}, \mathbf{D} >$. The components are as follows.

$\mathbf{I}$: The population initialization rule.
$\mathbf{S}$: The selection rule of individual.
$\mathbf{R}$: The reproduction rule.
$\mathbf{G}$: The set of genetic operators.
$\mathbf{F}$: The fitness evaluation function.
$\mathbf{D}$: The set of an elemental node which is component of a graph. The elemental node is categorized as follows.
the *start node* S_n
the *judgment nodes* $J_n = \{j_1, j_2, \cdots, j_{n_j}\}$
the *processing nodes* $P_n = \{p_1, p_2, \cdots, p_{n_p}\}$

An elemental node has a function and is categorized according to the kind of node's function. The *start node* has an empty function and is created to implementation of an individual as a program (see in Section 2.3). The *judgment node* has a judgment function indicated a branch on condition, and the *judgment node* has a number of external arc corresponding to the possible judgment result. The *processing node* has an action or a processing function, and the *processing node* has one external arc. Those functions are defined according to the given problem.

A program of GNP is an arbitrary directed graph, composed of nodes connected to each other by directed arcs. The program of GNP is defined by the tuple $< N, A >$, where N is composed of one start node and kind of $|N| - 1$ nodes. A is a set of arc connected to each node in N. The example of a program of GNP is shown in Figure 1.

2.2. Individual structure.

In GNP, an individual represents a directed graph. Figure 2 shows the example of an individual, and this individual represents the directed graph in Figure 2. The individual contains the following parameter for each node. NID_i is the *node identification number* of node i, which is assigned uniquely for each node in the program. NT_i is the node type (0: start node, 1: judgment node, 2: processing node) of

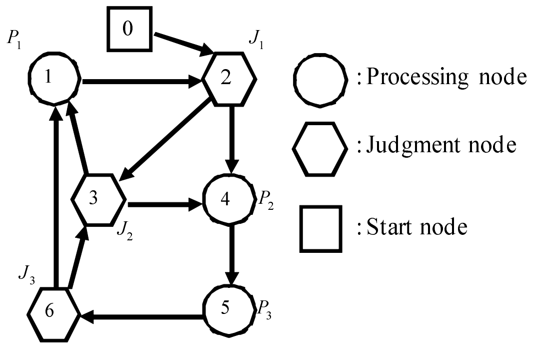
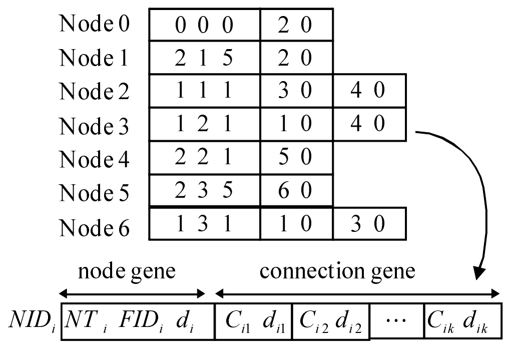


FIGURE 1. The example of program of GNP



Node 0	0 0 0	2 0	
Node 1	2 1 5	2 0	
Node 2	1 1 1	3 0	4 0
Node 3	1 2 1	1 0	4 0
Node 4	2 2 1	5 0	
Node 5	2 3 5	6 0	
Node 6	1 3 1	1 0	3 0

FIGURE 2. The example of individual of GNP

node i. FID_i is the *function identification number* of node i. The combination of NT_i and FID_i determines the elemental node. For example, the pair of $NT_i = 1$ and $FID_i = 2$ indicate the elemental node j_2. C_{ij} indicates a *node identification number* connected by the jth arc of node i. d_i indicates the delay time required to judge or process at node i. d_{ij} is the delay time required to transfer *control* from node i to the node connected by jth arc of node i.

2.3. **Implementation of a program.** We consider the *control* which transitions between nodes in an individual, and the *control* executes the function which current node has. Such behavior of the *control* is considered an implementation of a program. Firstly, the *control* initializes at the start node. Since the start node has an empty function, the *control* transitions to the next node according to the start node's arc. Then the *control* executes the function which the current node has and transitions to the next node according to the current node's arc. If the current node is a judgment node, the arc is selected by the judgment result.

2.4. **Evolution of an individual.** Similar to GA and GP, GNP applies genetic operators in order to evolve the individuals. A population of GNP is composed of an elite individual, N_m individuals produced by using the mutation operator and N_c individuals produced by using the crossover operator. An individual, which is applied to the genetic operator, is selected by using tournament selection.

2.4.1. *Mutation operator.* The mutation operator reproduces a new individual by changing an arc in the individual at random. The procedure of the mutation operator is as follows.

1. Select one individual by using tournament selection and reproduce it as a parent.

2. Each arc is selected with the probability p_m. The *node identification number* connected by the selected arc is changed to other node number randomly.
3. The reproduced new individual becomes the new one of the next generation.

2.4.2. *Crossover operator.* The crossover operator is performed between two parents and reproduces two new individual. The procedure of the basic crossover operator is as follows.

1. Select two individuals by using tournament selection twice and reproduce them as parents.
2. Each node i is selected with probability of p_c.
3. Two parents exchange the selected node, i.e., the nodes with the same node number.
4. The reproduced new individuals become the new ones of the next generation.

Since a node is selected at random, this crossover is corresponding to it uniform crossover in GA. Therefore, we call this crossover "uniform crossover".

3. Previous Crossover Operator.

Since an individual in GNP represents a graph, the crossover operator is also able to exchange a sub-graph. Some crossover operators, which exchange a sub-graph, are proposed by Katagiri et al. [21]. Those crossover operators randomly select a node, named crossover point, and determine a sub-graph by application of tree traversal that begins crossover point. Three crossover operators proposed by Katagiri et al. [21] use random width first search, width first search and random depth first search, respectively. According to Katagiri et al. [21], however, the uniform crossover operator is better than three crossover operators. It is seem that the reason of this is those crossover operators determine a sub-graph without regard for the behavior of the program. Therefore, we propose new crossover operator with consideration for the behavior of the program.

4. The Proposal Method: K-Cut Crossover.

4.1. Main idea.

According to the building block hypothesis in GA, the combination of a good sub-solution could be a good solution. We assume this hypothesis into GNP. In GNP, a sub-solution means a sub-graph in an individual, and we consider that a *good sub-graph* is the frequently used sub-graph in a program. The genetic operator is not only combine a *good sub-graph* but also has probability of breaking a *good sub-graph*. Since the mutation operator and the uniform crossover do not consider a sub-graph, the probability of breaking a *good sub-graph* increases. In particular, since the crossover operator is more influential than the mutation operator, the crossover operator's probability of breaking a *good sub-graph* is higher than the mutation operator's probability. In addition, since the previous crossover method is selected a node at random without consideration of sub-graph, there is a high probability of breaking a good sub-graph.

Therefore, in this study, we propose a k-cut crossover considering the *good sub-graph*. The k-cut crossover selects a sub-graph based on graph theory to keep a *good sub-graph*. Similar to the uniform crossover, the k-cut crossover reproduces two individuals from two parent individuals. Each parent individual partitions into k sub-graph, and two individuals are reproduced by exchanging the sub-graph. In this paper, we consider the most simplest k-cut crossover as k = 2.

4.2. The minimum k-cut problem.

In order to select a sub-graph, we use the minimum k-cut problem in graph theory. So, we describe to the minimum k-cut problem in this subsection. Given a graph with a vertex set V and an edge set E, we wish to partition the vertices into non-empty sets so as to minimize the number (or total weight) of edges crossing between them. More formally, a $cut(A, B)$ of a graph G is a partition of the vertices of G into two nonempty set A and B. An edge (v, w) *crosses* cut (A, B) if one of v and w is in A and the other in B. The *value* of a cut is the number of edges that cross the cut or, in a weighted graph, the sum of the weights of the edges that cross the cut. The minimum cut problem is to find a cut of minimum value. The usual approach to solve this problem is to use its close relationship to the maximum flow problem. The famous Max-Flow-Min-Cut-Theorem by Ford and Fulkerson [22] showed the duality of the maximum flow and the so-called minimum s-t-cut. There, s and t are two vertices that the source and the sink in the flow problem and have to be separated by the cut, that is, they have to lie in different parts of the partition. Until recently all cut algorithms were essentially flow algorithms using this duality. Finding a minimum cut without specified vertices to be separated can be done by finding minimum s-t-cut for a fixed vertex s and all $|V|-1$ possible choices of $t \in V\backslash\{s\}$ and then selecting the lightest one. Recently Hao and Orlin [23] showed how to use the maximum flow algorithm by Goldberg and Tarjan [24] in order to solve the minimum cut problem in time $O(|V||E|\log(|V|^2/|E|))$, which is nearly as fast as the fastest maximum flow algorithms so far [25, 26, 27]. Nagamochi and Ibaraki [28] published the first deterministic minimum cut algorithm that is not based on a flow algorithm, has the slightly better running time of $O(|V||E| + |V|^2 \log |V|)$, but is still rather complicated. In the unweighted case, they use a fast-search technique to decompose a graph's edge set E into subsets $E_1, \dots, E_\lambda$ such that the union of the first k E_i's is a k-edge-connected spanning subgraph of the given graph and has a most $k|V|$ edges. They simulate this approach in the weighted case, Their work is one of a small number of papers treating questions of graph connectivity by non-flow-based methods [29, 30].

The minimum cut algorithm we used is proposed by Nagamochi and Ibaraki [28] and uses the maximum ordering (MA) ordering [29] of vertices of a graph. An ordering $v_1, v_2, \dots, v_n$ of vertices is called an *MA ording* if an arbitrary vertex is chosen as v_1, and after choosing the first i vertices $v_1, \dots, v_i$, the $(i+1)$th vertex v_{i+1} is chosen from the vertices u that have the largest number of edges between $\{v_1, \dots, v_i\}$ and u. An important property of an MA ordering is that it identifies a minimum cut between some two vertices, which are specified by the ordering (see [31] for details).

4.3. **The crossover using minimum k-cut algorithm.**

The k-cut crossover selects sub-graphs by partitioning a graph and then keeps a frequently used sub-graph. Keeping a frequently used sub-graph is equal to cut a less frequently used arc. In order to accomplish such task, we solve a minimum k-cut problem in graph theory. On the weighted graph, the minimum k-cut problem is to find a partition the nodes into k non-empty sets so as to minimize the total weight of arcs crossing between them. For k is arbitrary variable, the minimum k-cut problem is known to be NP-hard. For k is constant, however, the minimum k-cut problem is able to solve in polynomial time. In particular, for k = 2 the problem is able to be solved by using the maximum flow problem.

All arcs may not use in an implementation of an individual. The used arcs are selected by using the minimum k-cut algorithm. However, the unused arcs are not selected. Thus, the k-cut crossover selects unused arc at random. The procedure of k-cut crossover is as follows.

1. Select two individuals using tournament selection twice and reproduce them as parents.
2. Select a sub-graph in each parent if used arc, by using minimum k-cut algorithm, otherwise, at random.
3. Label all arc as *internal* if they connect to a node in the same sub-graph, otherwise as *external.*

4. All *external* arcs are connected to randomly selected node in other sub-graph.
5. The reproduced new individual becomes the new one of the next generation.

4.4. **Comparison with previous crossover method.** The previous crossover method exchanges node selected at random. On the other hand, the proposed crossover method selects edge based on k-cut problem, and exchanges node which is belonged selected edges. Therefore, in the proposed crossover method, the probability of cutting important edge is lower than the previous crossover method.

5. Experiment.

5.1. Experimental setting.

5.1.1. *The tileworld.* In order to examine the effectiveness of the proposed method, we applied the k-cut crossover method to the tileworld which is a virtual environment for agents in 2-dimensional latticed world according to Katagiri et al. [21]. The tileworld consists of some agents, tiles, obstacles, holes and floors. Figure 3 shows ten tailworlds in our experiment. In this world, agents can move to an adjacent front cell in one step; they can also turn right or left in one step. When an agent moves to an adjacent tile cell, the agent can push the tile. When a tile is pushed into a hole, both the hole and the tail vanish. The task of an agent is to drop all the tiles into the holes as quickly as they can.

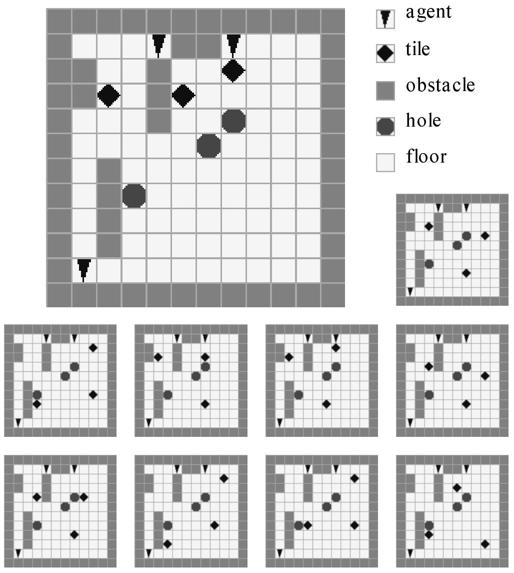


FIGURE 3. The tileworlds in our experiment

TABLE 1. The judgment node sets

symbol	description
JF	judge forward
JB	judge backward
JL	judge left side
JR	judge right side
TD	direction on the nearest tile from the agent
HD	direction on the nearest hole from the agent
THD	direction on the nearest hole from the nearest tile
STD	direction on the second nearest tile from the agent

Table 2. The processing node sets

symbol	description
MF	move forward
TR	turn right
TL	turn left
ST	stay

5.1.2. *The node sets.* According to Katagiri et al. [21], we define the judgment node and the processing node as follows. The judgment node $J_n = \{JF, JB, JL, JR, TD, HD, TH D, STD\}$. The processing node $P_n = \{MF, TR, TL, ST\}$. Each description of nodes shows Table 1 and Table 2. The result of JF, JB, JL, JR is the kind of an object in the tileworld: agent, tile, obstacle, hole and floor. The result of TD, HD, THD, STD is the rough direction which is the following four directions: right, front, left and back.

5.1.3. *Fitness and parameters.* In this study, we used a homogeneous strategy. In other words, all agents move based on an identical program of GNP. The agents can move untill all tiles have been dropped or within maximum number of steps. In our experiment, the maximum number of steps is 60 for each agent. The fitness value is defined by Equation (1) so as to consider how many tiles have been dropped, how far the tiles have approached the holes and how fast the task has been completed for all tileworlds.

$$fitness = \Sigma_{t \in TW}(100 \times DT_t + 20 \times AD_t + RS_t) \tag{1}$$

where TW is the set of tileworlds, the number of tileworlds is 10, DT_t is the total number of tiles which have been dropped into holes within the maximum number of steps on tileworld t. AD_t means how far the tiles have approached the holes and it is difference between the final state's *total distance tile-hole* and the initial state's. Here, the *total distance tile-hole* is calculated by sum of the distance from the tile to the hole. If the distance to the hole from the tile is longer than initial state, AD_t becomes a minus value. RS_t is the remaining step if agents have dropped all tiles within the maximum number of steps, otherwise 0.

According to Katagiri et al. [21], we use the following parameters: the maximum generation is 500; the population size is 301 ($N_m = 180$ and $N_c = 120$); $p_m = 0.01$; $p_c = 0.1$; tournament size is 100; $d_i = 1$ for all judgment nodes, $d_i = 5$ for all processing nodes, and all $d_{ij} = 0$. The judgment or processing of agents at one step is restricted by d_i, d_{ij}. The internal allowed time is five. We executed ten times with different random seeds in AMD Althon 64 X2 Dual Core Processor 3800+ 2.01 GHz, 1.00GB.

5.2. **Experimental results.** Table 3 shows experimental result. The best fitness value is the fitness value of the best of solution obtained by each trial. The average fitness value, worst fitness value and standard deviation fitness value are calculated in like fashion, respectively. The CPU time means the average time in trials of ten times. Figure 4 shows the progress of the value which is the best fitness value in the current generation. Figure

Table 3. The experimental result

Method	Fitness value				CPU time (m)
	Best	Average	Worst	STD	
uniform crossover	3379.00	2954.50	2410.00	293.37	234.97
k-cut crossover	3720.00	3057.63	2600.00	291.28	374.51

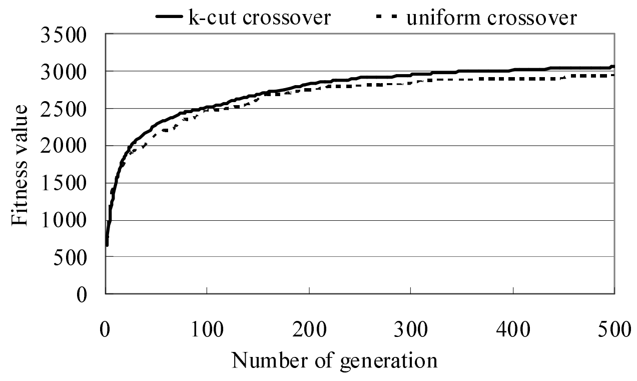


FIGURE 4. Progress of the best fitness value

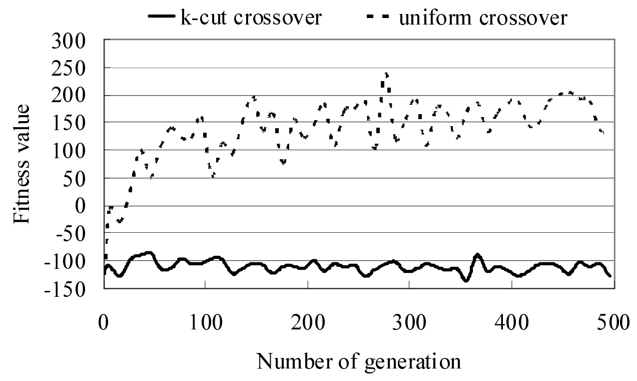


FIGURE 5. Progress of the value which is the worst fitness value in current generation

5 shows the progress of the value which is the fitness value of worst solution in the current generation. Note that Figure 4 and Figure 5 show the average value of trials of ten times.

Table 3 shows that the best fitness value, average fitness value and worst fitness value of the k-cut crossover are better than uniform crossover. On the other hand, the standard deviation of fitness value of k-cut crossover is smaller than uniform crossover. Therefore, it is seemed that the k-cut crossover averagely is able to obtain better solution than uniform crossover. Figure 4 shows that the k-cut crossover obtains a better solution than the solution obtained by the uniform crossover in almost generation. It is seemed that the k-cut crossover is able to reproduce a superior individual. However, Figure 5 shows that the worst fitness value of the k-cut crossover is lower than one of the uniform crossover. This result shows that the k-cut crossover has a probability of reproducing an inferior individual. For above two results, we consider that the k-cut crossover reproduces an extreme individual which means a superior individual or an inferior individual. The reason of this characteristic is the k-cut crossover partitions into sub-graphs based on the frequency of usage sub-graph. All movement of an agent may not have influence task of the agent. When the k-cut crossover finds the sub-graph indicated influential movement, the k-cut crossover reproduces a superior individual based on the building block hypothesis. In contrast, when the k-cut crossover finds the sub-graph indicated uninfluential movement, the k-cut crossover keeps the uninfluential movement and reproduces by connection of uninfluential movement. In this case, the reproduced individual is considered as an inferior individual. However, since the k-cut crossover improves the best individual, the k-cut crossover has higher probability of producing a superior individual than probability of producing an inferior individual.

Table 3 shows that the computational time of the k-cut crossover is slower than the uniform crossover. This result indicates the disadvantage of the use of minimum k-cut algorithm in the k-cut crossover.

6. Conclusion.

In this study, we focus on Genetic Network Programming (GNP) which is graph-based evolutionary algorithm. Similar to GA and GP, GNP applies to genetic operators to an individual, which is represented a directed graph, in order to solve a given problem. The crossover of GNP exchanges a sub-graph of parent individuals. A selection of sub-graph is important in the crossover, and some selection methods are proposed by Katagiri et al. [21]. However, the selection method based on graph theory is not proposed even though an individual is represented a graph. In this study, we propose the k-cut crossover based on graph theory. The k-cut crossover selects a sub-graph by using a minimum k-cut algorithm which finds a minimum graph partition on weighted graph.

In this study, we show the effectiveness of the k-cut crossover by applying to the tile-world which is a virtual environment for agent in 2-dimentioanl latticed world. In the experimental result, we show that k-cut crossover is able to reproduce a superior individual. However, it is necessary to compute more than the previous crossover. k-cut crossover has higher probability of producing a good solution than the GNP with other crossover method. In the future, we consider to improve the speed of computation in the k-cut crossover.

REFERENCES

[1] H. Katagiri, K. Hirasawa and J. Hu, Genetic network programming – Application to intelligent agent –, *Proc. of IEEE International Conf. on Systems, Man and Cybernetics*, vol.5, pp.3826-3834, 2000.

[2] 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.

[3] G. Yang, K. Shimada, S. Mabu and K. Hirasawa, A nonlinear model to rank association rules based on semantic similarity and genetic network programming, *IEEJ Transactions on Electrical and Electronic Engineering*, vol.4, no.2, pp.248-256, 2009.

[4] G. Yang, S. Mabu, K. Shimada and K. Hirasawa, A novel evolutionary method to search interesting association rules by keywords, *Expert Systems with Applications*, vol.38, no.10, pp.13378-13385, 2011.

[5] T. Karla, G. Eloy, K. Shimada, S. Mabu, K. Hirasawa and H. Jinglu, Association rule mining for continuous attributes using genetic network programming, *IEEJ Transactions on Electrical and Electronic Engineering*, vol.3, no.2, pp.1931-4981, 2008.

[6] Y. Yuchen, S. Mabu, K. Shimada and K. Hirasawa, Fuzzy intertransaction class association rule mining using genetic network programming for stock market prediction, *IEEJ Transactions on Electrical and Electronic Engineering*, vol.6, no.4, pp.353-360, 2011.

[7] L. Yu, S. Mabu, K. Shimada and K. Hirasawa, Multicar elevator group supervisory control system using genetic network programming, *IEEJ Transactions on Electrical and Electronic Engineering*, vol.6, no.S1, pp.S65-S73, 2011.

[8] Y. Chen, S. Mabu, K. Shimada and K. Hirasawa, A genetic network programming with learning approach for enhanced stock trading model, *Expert Systems with Applications*, vol.36, no.10, pp.12537-12546, 2009.

[9] Y. Chen, S. Mabu, K. Shimada and K. Hirasawa, A model of portfolio optimization using time adapting genetic network programming, *Comput. Oper. Res.*, vol.37, no.10, pp.1697-1707, 2010.

[10] Y. Chen, O. Etsushi, S. Mabu, K. Shimada and K. Hirasawa, A portfolio optimization model using genetic network programming with control nodes, *Expert Syst. Appl.*, vol.36, no.7, pp.10735-10745, 2009.

[11] S. Mabu, Y. Chen and K. Hirasawa, Generating stock trading rules using genetic network programming with flag nodes and adjustment of importance indexes, *Electronics and Communications in Japan*, vol.94, no.1, pp.25-33, 2011.

[12] Y. Chen, S. Mabu and K. Hirasawa, Genetic relation algorithm with guided mutation for the large-scale portfolio optimization, *Expert Systems with Applications*, vol.38, no.4, pp.3353-3363, 2011.

[13] E. Ohkawa, Y. Chen, Z. Bao, S. Mabu, K. Shimada and K. Hirasawa, Buying and selling stocks of multi brands using genetic network programming with control nodes, *IEEJ Transactions on Electronics, Information, and Systems*, vol.128, no.12, pp.1811-1819, 2008.

[14] J. Zhou, L. Yu, S. Mabu, K. Shimada, K. Hirasawa and S. Markon, A traffic-flow-adaptive controller of double-deck elevator systems using genetic network programming, *IEEJ Transactions on Electrical and Electronic Engineering*, vol.3, no.6, pp.703-714, 2008.

[15] L. Yu, S. Mabu, K. Shimada, K. Hirasawa and T. Ueno, Analysis of energy consumption of elevator group supervisory control system based on genetic network programming, *IEEJ Transactions on Electrical and Electronic Engineering*, vol.6, no.5, pp.414-423, 2011.

[16] 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*, vol.6, no.5, pp.457-467, 2011.

[17] S. Mabu and K. Hirasawa, Efficient program generation by evolving graph structures with multi-start nodes, *Applied Soft Computing*, vol.11, no.4, pp.3618-3624, 2011.

[18] F. Ye, S. Mabu and K. Hirasawa, Adaptive mutation in SARSA learning of genetic network programming with individual reconstruction, *Transaction of the Japanese Society for Evolutionary Computation*, vol.2, no.1, pp.12-28, 2011.

[19] S. Mabu, K. Hirasawa and J. Hu, A graph-based evolutionary algorithm: Genetic network programming (GNP) and its extension using reinforcement learning, *Evol. Comput.*, vol.15, no.3, pp.369-398, 2007.

[20] B. Li, S. Mabu and K. Hirasawa, Genetic network programming with subroutines for automatic program generation, *IEEJ Transactions on Electrical and Electronic Engineering*, vol.7, no.2, pp.197-207, 2012.

[21] H. Katagiri, K. Hirasawa, J. Hu and J. Murata, Comparing some graph crossover in genetic network programming, *Proc. of the SICE Annual Conf. 2002*, vol.2, pp.1263-1268, 2002.

[22] L. R. Ford and D. R. Fulkerson, Maximal flow through a network, *Can. J. Math.*, vol.8, pp.399-404, 1956.

[23] J. Hao and J. B. Orlin, A faster algorithm for finding the minimum cut in a graph, *Proc. of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, no.10, pp.165-174, 1992.

[24] A. V. Goldberg and R. E. Tarjan, A new approach to the maximum-flow problem, *J. ACM*, vol.35, no.4, pp.921-940, 1988.

[25] N. Alon, Generating pseudo-random permutations and maximum flow algorithms, *Information Processing Letters*, vol.35, no.4, pp.201-204, 1900.

[26] R. K. Ahuja, J. B. Orlin and R. E. Tarjan, Improved time bounds for the maximum flow, *SIAM J. Comput.*, vol.18, pp.939-954, 1989.

[27] J. Cheriyan, T. Hagerup and K. Mehlhorn, Can a maximum flow be computed in o(nm) time, *Proc. of ICALP*, pp.235-248, 1990.

[28] H. Nagamochi and T. Ibaraki, Computing the edge-connectivity of multigraphs and capacitated graphs, *SIAM Journal on Discrete Mathematics*, vol.5, pp.54-66, 1990.

[29] H. Nagamochi and T. Ibaraki, A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph, *Algorithmica*, vol.7, pp.583-596, 1992.

[30] D. W. Matula, A linear time $2 + \epsilon$ approximation algorithm for edge connectivity, *Proc. of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp.500-504, 1993.

[31] H. Nagamochi and T. Ibaraki, Graph connectivity and its augmentation: Applications of MA orderings, *Discrete Appl. Math.*, vol.123, no.1-3, pp.447-472, 2002.