An Edge-Set Representation Based on a Spanning Tree for Searching Cut Space

Kisung Seo, Soohwan Hyun, and Yong-Hyuk Kim, *Member, IEEE*

***Abstract*—The encoding or representation scheme in evolutionary algorithms is very important because it can greatly affect their performance. Most previous evolutionary algorithms for solving graph problems have traditionally used a vertex-based encoding in which each gene corresponds to a vertex. In this paper, addressing the well-known maximum cut problem, we introduce an edge-set encoding based on the spanning tree—a kind of edge-based encoding. In our encoding scheme, each gene corresponds to an edge subset derived from a spanning tree. In contrast to a traditional edge-based encoding in which each gene corresponds to only one edge, our encoding scheme has the advantage of representing only feasible solutions, so there is no need to apply a repair step. We present a genetic algorithm based on this new encoding. We have conducted various experiments on a large set of test graphs including commonly used benchmark graphs and have obtained performance improvement on sparse graphs, which frequently appear in real-world applications such as social networks and systems biology, in comparison with a scheme using a vertex-based encoding.**

***Index Terms*—Basis change, encoding, genetic algorithm (GA), graph, maximum cut, representation, spanning tree.**

I. Introduction

In genetic algorithms (GAs), different encodings lead to completely different searches of the solution space, so the encoding can affect performance dramatically. There have been many studies that emphasize the importance of representation in GAs (see, for example, Rothlauf [1] or Kim *et al.* [2]). Kim *et al.* improved the performance of GAs on various problems by rearranging related gene positions to be close to one another (improving linkage). This gene rearrangement can be seen as a simple type of transformed encoding. There have also been more generalized studies of encoding transformations to make the relationships between genes be as independent as possible by applying invertible linear transformations [3], [4]. These studies showed the importance of encoding transformations, but they failed to show concrete transformation methods. As an extension of these studies, there has been a trial to find better encodings using a meta-GA [5]. However, it also failed to provide good guidance about how to transform the encoding of a given problem.

Edge-set encoding is a systematic edge-based encoding scheme for cut space—a sort of transformation of vertex-based encoding. It is expected to overcome some limitations of previous encodings. In particular, the method can be applied efficiently to the search of cut space for the MAX CUT problem.

Most studies about graph problems such as graph partitioning and MAX CUT have been vertex-centric when dealing with partitions and representing them [6]–[21]. Intuitive techniques based on vertices, which are easy to manage, have been the most common representations used to solve graph problems. However, when dealing with partitions, there have been studies [22]–[26] using methods based on edge representation, which is a dual of a vertex representation. In particular, Armbruster *et al.* [27] and Yoon *et al.* [28] used an edge representation for solving graph partitioning, which maps a solution to an edge set, not a vertex set. In their representation, each location of an encoding is assigned to 1 if its corresponding edge is on the cut and 0 otherwise. This representation is well adapted to their integer programming formulation, but it is very crucial and difficult to check whether or not a given encoding forms a valid graph partition.

In this paper, we considerably extend our preliminary work [29]. We present a new GA based not on a vertex encoding but on an edge-set encoding built upon a spanning tree [30], [31]—a different kind of edge-based encoding. Contrary to a general edge-based encoding, edge-set encoding based on a spanning tree represents only feasible partitions. As a target problem, we adopted the MAX CUT problem, which is well known as a representative NP-hard problem, and examined the performance of this representation experimentally. This method was expected to perform well on sparse graphs. In particular, if we consider that graphs appearing in real-world applications such as social networks and systems biology are typically sparse, the new method holds promise for addressing real-world graph problems.

The remainder of this paper is organized as follows. Section II introduces the MAX CUT problem and previous work on it. Section III describes the proposed edge-set encoding scheme based on a spanning tree. Section IV presents experimental results on various graph sets, and Section V concludes the paper.

Manuscript received October 10, 2013; revised February 25, 2014 and May 16, 2014; accepted July 7, 2014. Date of publication July 10, 2014; date of current version July 28, 2015. This work was supported by the National Research Foundation of Korea funded by the Korea government under Grant NRF-2011-0009958. The present research has been conducted by the Research Grant of Kwangwoon University in 2015. *(Corresponding author: Y.-H. Kim.)*

K. Seo was with Michigan State University, East Lansing, MI 48823 USA. He is now with the Department of Electronics Engineering, Seokyeong University, Seoul 136-704, Korea (e-mail: ksseo@skuniv.ac.kr).

S. Hyun is with Hyundai Heavy Industries Research Institute, Yongin 136-749, Korea (e-mail: xjavalov@shhyun.com).

Y.-H. Kim is with the Department of Computer Science and Engineering, Kwangwoon University, Seoul 139-701, Korea (e-mail: yhdfly@kw.ac.kr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TEVC.2014.2338076

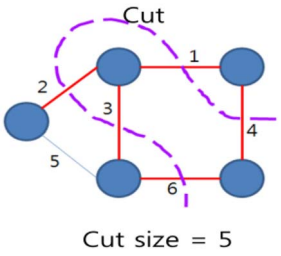


Fig. 1. Example of a cut.

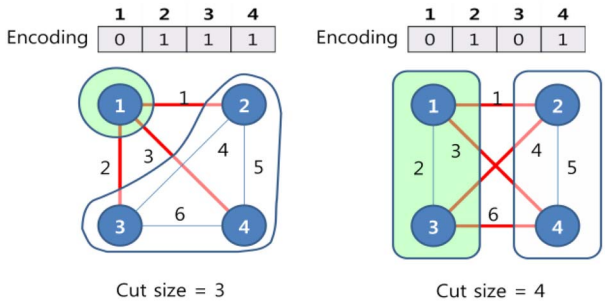


Fig. 2. Two examples of vertex-based encoding.

II. Max Cut

It is important in combinatorial optimization to partition the vertices into two disjoint subsets of nearly equal size such that the sum of weights of edges with their vertices in different subsets (i.e., the cut size) is maximized or minimized. Fig. 1 shows an example of a cut. Given an undirected graph $G = (V, E)$ with edge weights, the MAX CUT problem is that of finding a subset $S \subset V$ which maximizes the sum of edge weights in the cut $(S, V–S)$.

Every graph has a finite number of cuts, so one can find the minimum or maximum weight cut in a graph by an exhaustive search that enumerates the sizes of all the cuts. This is not a practical approach for the large graphs that arise in real-world applications, since the number of cuts in a graph grows exponentially with the number of vertices. Although we can solve the min-cut problem without balance requirement in polynomial time using the maxflow-mincut algorithm [32], we have no such fortune when it comes to the MAX CUT problem. There is no known way to solve the problem optimally other than by exhaustive enumeration. The MAX CUT problem is one of Karp's original NP-complete problems [33] and has been known to be NP-complete even if the problem is unweighted [34].

Since there is no algorithm that guarantees an optimal solution, a typical approach to solve such a problem is to find a ρ-approximation algorithm that delivers a solution at least ρ times the optimal value in polynomial time. Sahni and Gonzales [35] presented a 1/2-approximation algorithm for the MAX CUT problem. Their greedy approach iterates through the vertices and decides which placement (S or $V - S$) maximizes the cut of vertex v_i with respect to vertices v_1 to v_{i-1}. Since [35], many researchers have presented approximation algorithms for the MAX CUT problem [36]–[39], but little progress has been made. For more than twenty years a factor of 0.5 was the best-known polynomial-time performance guarantee for the MAX CUT problem. An algorithm by Goemans and Williamson (GW) [40] guarantees a factor of 0.878 of the optimum. The significant improvement was due to the technique of positive semidefinite programming and randomized rounding. However, solving semidefinite programming is computationally expensive. Homer and Peinado [41] gave a parallelized version of GW. In [41], GW was improved by combining with simulated annealing (SA) [42]. Afterward, Kim *et al*. [43] successfully applied GAs to the MAX CUT problem. In practice, when the GA was combined with lock-gain-based local search [14], the hybrid GA could outperform GW (the best previously known approximation algorithm) combined with SA. It is known that GAs can perform well on the MAX CUT problem [43], [44]; we have therefore chosen the MAX CUT problem to test our new encoding scheme in this paper.

The MAX CUT problem has many applications in various fields. It has been observed that one of the phases (the layer assignment problem) in the design process for VLSI chips and printed circuit boards (PCB) can be reduced to the MAX CUT problem [45], [46]. One of the most famous applications of the problem comes from a classical application to statistical physics [45]. It is concerned with the exact determination of a minimal energy configuration of a spin glass under no exterior field and under a continuously varying exterior magnetic field. Poljak and Tuza [47] provided a comprehensive survey of the MAX CUT problem.

III. Encoding and Evaluation

Each solution is represented by a chromosome, which is a binary string. In this section, we consider two different binary encodings to represent solutions for the MAX CUT problem.

A. Vertex-Based Encoding

When we use vertex-based encoding in GAs, the number of genes in the chromosome equals n, which is the number of vertices in the graph. Each gene corresponds to a vertex in the graph. A gene has value 0 if the corresponding vertex is in S, and has value 1 otherwise.

To evaluate the cut size of a solution, we should compute the number of cut edges, which are edge whose end-vertices are in different subsets. For each edge, to determine if it is a cut edge, we just check that the values of genes corresponding to its end-vertices are different, i.e., (0, 1) or (1, 0). Fig. 2 shows two examples of vertex-based encoding. In the left example, v_1 is in S and cut edges are e_1, e_2, and e_3. In the right example, v_1 and v_3 are in S and cut edges are e_1, e_3, e_4, and e_6.

B. Edge-Set Encoding Based on Spanning Tree

If $\{V_1, V_2\}$ is a partition of V, the set $E(V_1, V_2)$ of all the edges of G crossing between V_1 and V_2 is called a *cut*. *Cut space* consisting of all the cuts together with $\emptyset$ is proven to be a vector space (see the following proposition). It means that an arbitrary cut can be represented by a linear combination of basis elements of cut space. The summation operator in each coordinate is the same as the XOR operator in Boolean algebra, i.e., $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, and $1 \oplus 1 = 0$.

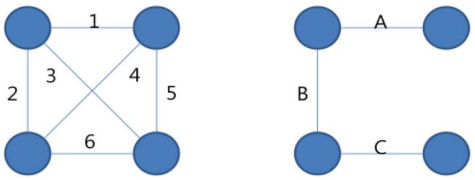


Fig. 3. Example of a graph (left) and its spanning tree (right).

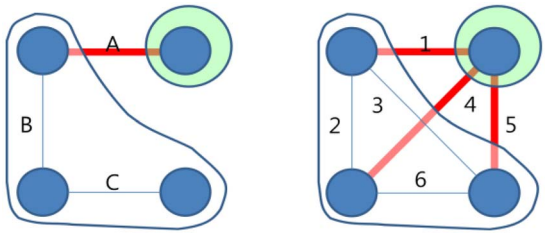


Fig. 4. Edge e_A in a spanning tree, and its corresponding cut.

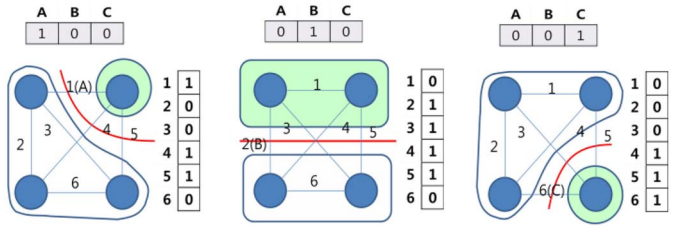


Fig. 5. Three basis elements e_A, e_B, and e_C in an edge-set encoding based on spanning tree.

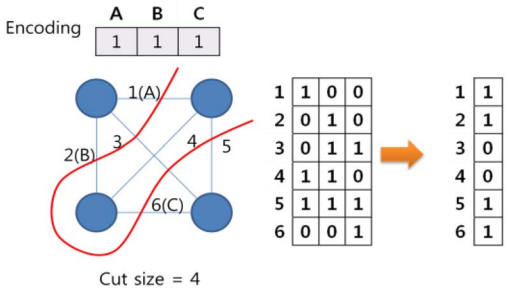


Fig. 6. Example of an edge-set encoding based on spanning tree, and its cut edges.

Proposition 1: Let $C(G)$ be the set of all cuts in $G = (V, E)$, together with $\emptyset$. Then $C(G)$ is a vector space [30].

Proof: It is enough to show that, for all C, C' in $C(G)$, $C+C'$ (= C–C') lies in $C(G)$. Since $C+C = \emptyset$ is in $C(G)$ and $C+\emptyset = C$ is in $C(G)$, we may assume that C and C' are distinct and nonempty. Let $\{V_1, V_2\}$ and $\{V_1', V_2'\}$ be the corresponding partitions of V. Then $C + C'$ consists of all the edges that cross one of these partitions but not the other. These are precisely the edges between $(V_1 \bigcap V_1')\bigcup(V_2 \bigcap V_2')$ and $(V_1 \bigcap V_2')\bigcup(V_2 \bigcap V_1')$, and by $C \neq C'$ these two sets form another partition of V. Hence $C + C'$ is in $C(G)$, and $C(G)$ is indeed a vector space. ■

In the case that graph G is connected,[1] we can derive a basis of the cut space from a spanning tree of G. Finding a basis of the cut space based on a spanning tree is nontrivial. General graph traversal algorithms such as depth-first search (DFS) and breadth-first search (BFS) can produce spanning trees of G (see Fig. 3). Let T be a spanning tree of G. For each edge e of the n – 1 edges in T, the graph $T - e$ has exactly two components, and the edge set C_e of edges in G between the two components forms a cut. Since none of the edges e in T lies in $C_{e'}$ for $e' \neq e$, these n – 1 cuts are linearly independent. From the fact that the dimension of the cut space is n – 1 [30], the cuts exactly form a basis of cut space.

When we use an edge-set encoding based on a spanning tree in GAs, the number of genes in the chromosome equals n – 1, the number of edges in T. Each gene corresponds to an edge in T. To evaluate the cut size of a solution, we should compute the number of cut edges, the set of cut edges is easily computed by summing C_e on $\mathbb{Z}_2^{|E|}$ for each edge e in T whose gene value is 1. Then the cut size becomes the cardinality of the set of cut edges (see Fig. 4).

Fig. 5 shows the basis elements in an edge-set encoding based on a spanning tree for the example given in Fig. 3. In the example, the spanning tree has three edges, so there are three basis elements derived from edges e_A, e_B, and e_C.

[1]In this paper, we assume that G is connected for convenience.

Each basis element has its corresponding edge-set C_e (see the length-$|E|$ binary code in the right part of each element).

Fig. 6 shows an example of an edge-set encoding based on a spanning tree and its interpretation in terms of cut edges. In the example, the final cut edge set is obtained from the vector summation of $C_{e_A} + C_{e_B} + C_{e_C}$ on the $|E|$-dimensional vector space $\mathbb{Z}_2^{|E|}$.

The following proposition hints at some advantages of using the proposed edge-set encoding based on spanning tree on sparse graphs.

Proposition 2: Assume that a given graph $G = (V, E)$ is connected. Let the ratio of vertices to edges, i.e., the number of vertices/the number of edges, be $n/(n-1)$, where n is the number of vertices. That is, G is an extremely sparse graph. Then, the problem of finding a maximum cut using an edge-set encoding based on spanning tree is the one-max problem of size $n-1$.

Proof: If the ratio of vertices to edges is $n/(n-1)$, the given graph becomes a tree with $n-1$ edges. So its spanning tree is the given graph itself. Each element in an edge-set encoding based on spanning tree has its edge-set including only one edge. Since elements in the encoding are linearly independent, we can get the maximum cut when all the elements of the encoding are valued at 1, which means that the cut includes all the edges of the given graph. Hence the problem becomes exactly the one-max problem of size $n-1$. ■

Considering that the one-max problem is known to be quite easy to tackle using GAs [48], we can expect that the edge-set encoding based on spanning tree presented here will be a good choice for sparse graphs.

IV. Simulation and Analysis

A. Experiment Environments

This section describes how we developed the GA approach used with the edge-set encoding based on spanning tree for

TABLE I
GA PARAMETERS IN OPENBEAGLE SOFTWARE

Stopping criterion 100 generations
Population size 100
Crossover rate 0.9
Mutation rate 0.1
Tournament size 7
Generational model in which offspring replace their parents
Elitism with the best individual

the MAX CUT problem. The GA parameters are shown in Table I. One-point crossover and random bit-wise mutation were used, and tournament selection was used to select candidates for crossover and mutation. The proposed algorithm was implemented using open beagle [49] with the boost graph library [50]. Every GA run was repeated 30 times for each case. The proposed GA was tested on a total of 65 graphs, among which 31 graphs are from well-known benchmark data set and the others are randomly generated with various densities. Every GA was performed 30 independent runs for each graph (each test case). From the 30 runs for each case, we obtained the results of the average and the standard deviation.

B. Sparse and Dense Graphs

We can classify graphs into two categories: sparse graphs and dense ones, according to the ratio of vertices to edges. In sparse graphs, the average degree is small, but in dense graphs, the average degree is large. The ratio of vertices to edges in sparse graphs is greater than in dense graphs. For connected graphs, we can get the following range for the ratio of vertices to edges. It suggests a criterion for classifying graphs into the two groups.

Fact 1: Assume that a given graph $G = (V, E)$ is connected. Let n be the number of vertices. Then the ratio of vertices to edges is in the range $[2/(n-1), n/(n-1)]$.

Proof: For a connected graph, when the graph is a tree, the number of edges is the smallest, and when the graph is complete, the number of edges is the largest. Then the number of edges is between $n-1$ and $n(n-1)/2$. Hence the ratio of vertices to edges is between $2/(n-1)$ and $n/(n-1)$. ■

In general, there is no agreed-upon ratio threshold for classifying graphs into sparse or dense graphs. In this paper, we regard as sparse graphs the cases that the ratio of vertices to edges is more than 50%.

C. Experiment Results I

In the first experiment, various sets on a total of 17 graphs are tested. A Kruskal-like algorithm is used to find spanning trees. The classes of graphs that we tested our algorithms on are described below.

1) G*n.p:* A random graph on n vertices with edge probability p. E.g., G1000.01 is a 1,000-vertex graph with $p = 0.01$.
2) U*n.d:* a geometric random graph on n vertices and expected degree d. E.g., U500.10 is a 500-vertex geometric graph with "expected degree" 10.
3) cat.*n:* A caterpillar graph on n vertices, with each vertex having six legs. rcat.*n* is a caterpillar graph with n vertices, where each vertex on the spine has $\sqrt{n}$ legs.
4) grid*n.b:* A grid graph on n vertices and whose optimal minimum cut size is known to be b. w-grid*n.b* denotes the same grid but the boundaries are wrapped around.

Please refer to [14] and [51] for details on how these classes of graphs are generated.

TABLE II
EXPERIMENTAL RESULTS ON SPARSE GRAPHS

Instances	Vertex / Edge ratio (%)	Vertex encoding		Edge encoding		Improvement(%)
		Cut size	Std dev	Cut size	Std dev	
cat.352	100%	224.1	2.9	**242.1**	2.1	8.05%
cat.702	100%	419.0	3.3	**444.6**	3.5	6.10%
cat.1052	100%	606.8	5.0	**644.9**	5.6	6.28%
rcat.134	100%	99.4	1.5	**106.1**	1.5	6.74%
rcat.554	100%	339.0	3.1	**360.2**	2.5	6.26%
rcat.994	100%	577.9	5.2	**611.3**	4.7	5.78%
grid100.0	55.25%	130.1	2.4	**133.2**	2.0	2.36%
grid500.21	52.30%	556.9	4.3	**569.4**	4.3	2.24%
grid1000.20	51.79%	1075.5	5.6	**1094.6**	7.2	1.77%
w-grid100.20	49.50%	145.1	2.6	**146.8**	1.9	1.22%
w-grid500.42	49.90%	582.1	4.6	**594.0**	5.1	2.04%
w-grid1000.40	49.95%	1113.8	9.0	**1133.2**	7.0	1.74%

The tabular results for sparse graphs are provided in Table II. Sparse graphs are graphs in which the ratio of vertices to edges is larger than or equal to 0.5, which means the average vertex is connected by two or fewer edges. The results show that an average 6% improvement in cut size was obtained on the caterpillar graph set and about 2% improvement was obtained on the grid graph set, using the edge encoding based on a spanning tree, compared to the same number of evaluations using the vertex encoding.

The tabular results for dense graphs, in which the ratio of vertices to edges is less than 0.5, are shown in Table III. We can see that the proposed edge-set encoding performed worse than the vertex-based encoding. Fig. 7 summarizes the results provided in Tables II and III. Tested graphs are categorized into sparse and dense graphs.

D. Experiment Results II

Table IV shows the results obtained on a set of various graphs that were randomly generated according to the ratio of vertices to edges. For example, r100.1.0 in the first row means a random graph with 100 vertices in which the ratio of vertices to edges is 1.0, and r500.0.1 in the last row means a random graph with 500 vertices in which the ratio of vertices to edges is 0.1. For ratios of vertices to edges between 0.4 and 1.0, we obtained large improvements in performance. For

TABLE III
EXPERIMENTAL RESULTS ON DENSE GRAPHS

Instances	Vertex / Edge ratio (%)	Vertex encoding		Edge encoding		Improvement(%)
		Cut size	Std dev	Cut size	Std dev	
G500.02	21.20%	**1303.1**	7.2	1293.3	7.4	-0.75%
G500.04	9.74%	**2747.9**	10.0	2718.3	10.5	-1.08%
G1000.01	19.74%	**2711.0**	8.5	2699.2	7.7	-0.44%
U500.20	10.97%	**2410.2**	5.7	2407.4	5.4	-0.11%
U500.40	5.68%	**4575.6**	7.1	4563.2	8.2	-0.27%

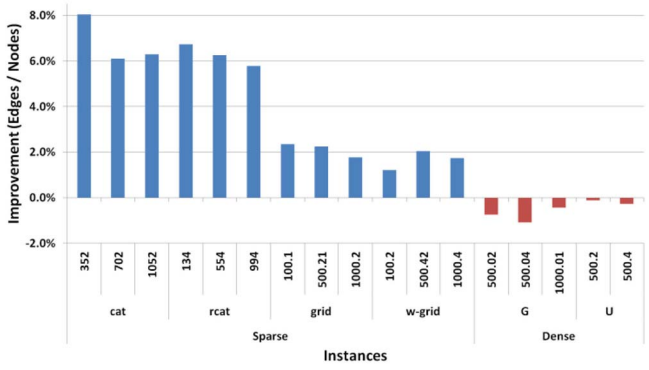


Fig. 7. Summary of performance for sparse and dense graphs (I).

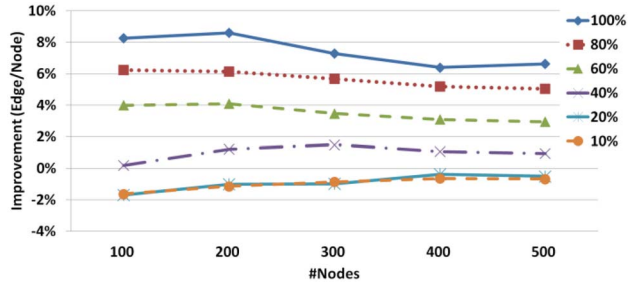


Fig. 8. Summary of performance for sparse and dense graphs (II).

the ratios 0.1 and 0.2, we obtained performance degradation by at most 1.7%. Fig. 8 gives the comparison of the results given in Table IV. The algorithm appears to show decreasing performance with the number of vertices. For example, the improvements were much smaller for sets with more vertices, such as r1000.10, r1000.0.8, and smaller yet for r2000.1.0, r2000.0.8 in Table IV. 100 random graphs are generated in each case of experiments.

E. Experiment Results III

The tabular results for the extended sets on a total of 31 graphs are provided in Table V. Three methods—Kruskal-like, DFS, and BFS—were used to find spanning trees. The new class of graphs that we tested our algorithms on is described below.

TABLE IV
EXPERIMENTAL RESULTS ON RANDOM GRAPH

Instances	Vertex / Edge ratio (%)	Vertex encoding		Edge encoding		Improvement(%)
		Cut size	Std dev	Cut size	Std dev	
r100.1.0	100%	75.9	1.0	**82.2**	1.3	8.26%
r200.1.0	100%	135.7	1.7	**147.4**	1.6	8.60%
r300.1.0	100%	193.4	2.1	**207.5**	2.3	7.29%
r400.1.0	100%	250.2	3.1	**266.2**	2.7	6.39%
r500.1.0	100%	305.8	3.7	**326.1**	2.9	6.63%
r1000.1.0	100%	565.3	3.8	**581.6**	5.1	2..81%
r2000.1.0	100%	1093.9	6.4	**1114.9**	6.0	1.74%
r100.0.8	80%	91.5	1.5	**97.2**	1.4	6.23%
r200.0.8	80%	165.1	2.1	**175.3**	2.6	6.14%
r300.0.8	80%	236.2	3.1	**249.6**	2.3	5.67%
r400.0.8	80%	306.1	3.5	**322.0**	3.3	5.18%
r500.0.8	80%	374.6	3.9	**393.4**	4.0	5.04%
r1000.0.8	80%	700.9	7.3	**714.1**	5.5	1..85%
r2000.0.8	80%	1355.5	8.8	**1375.7**	7.3	1..47%
r100.0.6	60%	117.0	1.9	**121.7**	1.9	3.99%
r200.0.6	60%	212.5	2.5	**221.2**	3.1	4.09%
r300.0.6	60%	306.9	3.5	**317.5**	3.1	3.47%
r400.0.6	60%	397.8	4.2	**410.1**	4.4	3.09%
r500.0.6	60%	488.7	3.5	**503.0**	4.8	2.93%
r100.0.4	40%	167.5	3.2	**167.8**	2.4	0.18%
r200.0.4	40%	307.0	3.2	**310.8**	3.3	1.22%
r300.0.4	40%	442.6	4.3	**449.2**	3.6	1.49%
r400.0.4	40%	578.6	4.2	**584.7**	4.9	1.06%
r500.0.4	40%	713.5	4.8	**720.2**	4.6	0.93%
r100.0.2	20%	**308.1**	3.6	302.8	3.4	-1.70%
r200.0.2	20%	**581.6**	5.8	575.7	5.1	-1.01%
r300.0.2	20%	**848.8**	7.3	840.4	4.6	-0.99%
r400.0.2	20%	**1111.1**	6.2	1107.0	6.9	-0.37%
r500.0.2	20%	**1374.8**	6.7	1367.8	8.0	-0.51%
r100.0.1	10%	**579.8**	4.7	570.4	4.6	-1.63%
r200.0.1	10%	**1110.3**	6.2	1097.6	4.9	-1.14%
r300.0.1	10%	**1635.6**	7.8	1621.7	10.6	-0.85%
r400.0.1	10%	**2155.8**	7.4	2142.0	13.5	-0.64%
r500.0.1	10%	**2676.1**	8.8	2657.9	14.7	-0.68%

breg*n.b:* a regular random graph on *n* vertices in which each vertex has degree 3 and the optimal bisection size is *b* with high probability, i.e., probability approaches 1 as *n* approaches infinity.

Please refer to [14] and [51] for details on how this class of graphs is generated.

For the case of sparse graphs in which the ratio of vertices to edges is more than 38.9%, the performance of edge-set encoding based on spanning tree is superior to the results of vertex-based encoding. The improvement increases for greater

TABLE V
COMPARISON OF ENCODING SCHEMES FOR EXTENDED SET OF GRAPHS

Instances	Vertex / Edge ratio (%)	Vertex encoding		Edge encodings								
				Kruskal-like			DFS			BFS		
		Cut size	Std dev	Cut size	Std dev	Improvement(%)	Cut size	Std dev	Improvement(%)	Cut size	Std dev	Improvement(%)
G500.005	79.9%	375.0	3.7	391.3	3.0	4.35	**392.1**	**3.2**	**4.55**	388.9	3.8	3.71
G500.01	40.8%	700.1	4.5	706.1	4.6	0.87	**708.1**	**5.7**	**1.15**	707.0	4.4	1.00
G500.02	21.2%	**1303.1**	**7.2**	1293.3	7.4	-0.75	1294.7	7.9	-0.64	1295.7	7.5	-0.57
G500.04	9.7%	**2747.9**	**10.0**	2718.3	10.5	-1.08	2711.1	7.9	-1.34	2722.2	12.1	-0.94
G1000.005	40.0%	1373.4	8.4	**1383.6**	**5.4**	**0.75**	1383.5	6.9	0.74	1383.2	4.3	0.71
G1000.01	19.7%	**2711.0**	**8.5**	2699.2	7.7	-0.44	2695.9	9.7	-0.56	2702.0	7.8	-0.33
G1000.02	9.9%	**5307.4**	**10.9**	5271.3	13.6	-0.68	5265.6	15.0	-0.79	5280.5	15.2	-0.51
U500.05	38.9%	597.2	3.6	**606.3**	**3.8**	**1.52**	605.0	3.7	1.31	606.0	3.4	1.47
U500.10	20.3%	1276.1	4.8	1282.3	4.5	0.49	1279.3	4.0	0.25	**1283.8**	**6.5**	**0.61**
U500.20	11.0%	**2410.2**	**5.7**	2407.4	5.4	-0.11	2396.0	7.6	-0.59	2402.1	7.5	-0.34
U500.40	5.7%	**4575.6**	**7.1**	4563.2	8.2	-0.27	4547.1	6.6	-0.62	4557.5	7.2	-0.40
U1000.05	41.7%	1309.0	7.6	**1324.9**	**4.6**	**1.21**	1324.7	5.3	1.20	1323.1	5.0	1.08
U1000.20	10.7%	**4877.9**	**10.4**	4871.1	10.6	-0.14	4851.2	12.1	-0.55	4861.2	12.5	-0.34
U1000.40	5.5%	**9292.5**	**12.2**	9261.0	8.8	-0.34	9234.2	12.2	-0.63	9252.2	15.6	-0.43
breg500.12	66.5%	445.4	4.8	460.8	3.6	3.47	**461.8**	**3.3**	**3.67**	459.9	3.7	3.25
breg500.16	66.5%	444.5	3.4	**462.5**	**4.7**	**4.04**	462.3	3.8	4.01	460.0	3.5	3.49
breg500.20	66.5%	444.9	4.6	460.8	3.1	3.57	460.7	3.2	3.55	**461.2**	**4.3**	**3.66**
cat.352	100.0%	224.1	2.9	242.1	2.1	8.05	241.5	2.5	7.80	**242.7**	**2.8**	**8.33**
cat.702	100.0%	419.0	3.3	444.6	3.5	6.10	**446.2**	**3.9**	**6.49**	445.8	3.0	6.40
cat.1052	100.0%	606.8	5.0	**644.9**	**5.6**	**6.28**	**644.9**	**5.6**	**6.28**	**644.9**	**5.7**	**6.28**
cat.5252	100.0%	2804.5	10.0	2890.8	10.0	3.07	2885.3	10.7	2.88	**2891.2**	**10.2**	**3.09**
rcat.134	100.0%	99.4	1.5	**106.1**	**1.5**	**6.74**	**106.1**	**1.5**	**6.74**	**106.1**	**1.6**	**6.74**
rcat.554	100.0%	339.0	3.1	**360.2**	**2.5**	**6.26**	**360.2**	**2.5**	**6.26**	**360.2**	**2.6**	**6.26**
rcat.994	100.0%	577.9	5.2	**611.3**	**4.7**	**5.78**	**611.3**	**4.7**	**5.78**	**611.3**	**4.8**	**5.78**
rcat.5114	100.0%	2736.6	10.3	2817.4	10.0	2.95	**2817.6**	**9.7**	**2.96**	2816.1	9.5	2.91
grid100.10	55.2%	130.1	2.4	133.2	2.0	2.36	132.9	2.4	2.10	**133.7**	**1.8**	**2.77**
grid500.21	52.3%	556.9	4.3	569.4	4.3	2.24	**571.7**	**6.0**	**2.66**	570.2	5.5	2.38
grid1000.20	51.8%	1075.5	5.6	1094.6	7.2	1.77	**1096.3**	**5.1**	**1.93**	1095.9	7.7	1.89
w-grid100.20	49.5%	145.1	2.6	**146.8**	**1.9**	**1.22**	145.2	2.7	0.07	144.3	2.4	-0.55
w-grid500.42	49.9%	582.1	4.6	594.0	5.1	2.04	**594.5**	**5.4**	**2.12**	591.3	3.9	1.58
w-grid1000.40	50.0%	1113.8	9.0	**1133.2**	**7.0**	**1.74**	1130.2	5.1	1.47	1129.5	5.8	1.40

ratios of vertices to edges, in general. Examples include breg*n.b*, grid*n.b*, cat.*n*, rcat.*n*, and w-grid*n.b*. The cat.*n* and rcat.*n* graph sets, with nearly 100% ratios of vertices to edges, show an average of approximately 6% improvement using edge-set encoding based on spanning tree.

On the other hand, the performance of the vertex-based encoding is better than that of edge-set encoding based on spanning tree for dense graphs in which ratio of vertices to edges is less than 20%. Some G*n.p* and U*n.d* graph sets are examples of this category. The performances for graphs with ratios near 20% are irregular. For example, the performance of edge-set encoding based on spanning tree is superior in U500.10, which has a 20.3% ratio, but the performance of vertex-based encoding is better in G500.02, which has a 21.2% ratio. Geometric random graphs are closer to real-world problems than random graphs, considering that the randomness of geometric random graphs is less than that of random graphs and the vertices connected with edges in geometric random graphs are locally clustered.

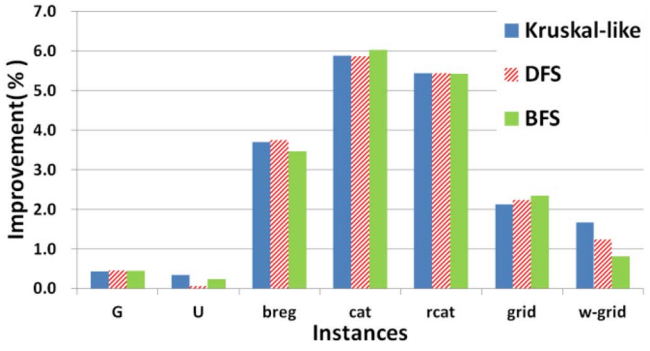


Fig. 9. Summary of performance for Table IV.

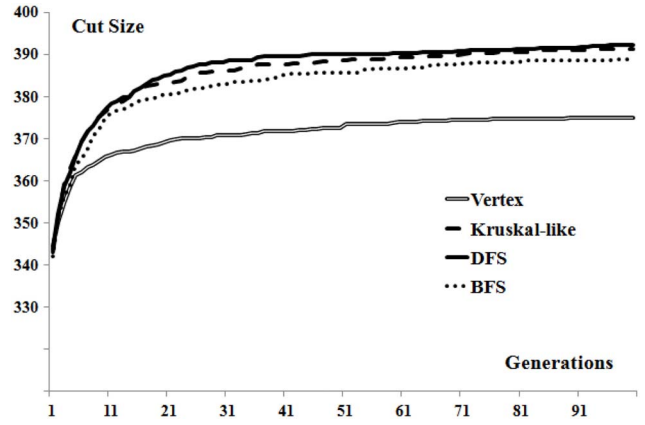


Fig. 10. Convergence diagram for the mean best of 30 runs on G500.005.

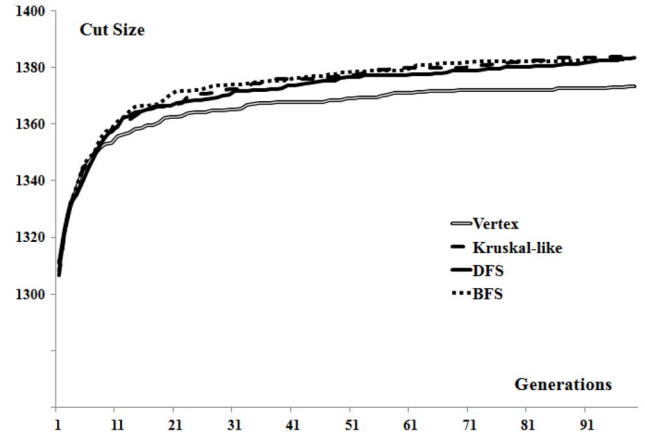


Fig. 11. Convergence diagram for the mean best of 30 runs on G1000.005.

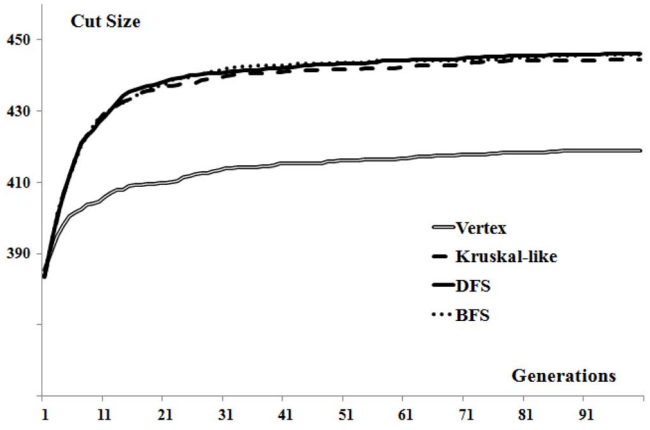


Fig. 12. Convergence diagram for the mean best of 30 runs on cat.702.

Therefore, the superiority of edge-set encoding based on spanning tree will be expected for real-world problems which consist of sparse graphs having more than a 20% ratio of vertices to edges. In the paper, three methods—Kruskal-like, DFS, and BFS—are used to obtain spanning trees and the results differ slightly for each method, but they are not very different. Determining which kind of algorithms, for finding a spanning tree, are efficient and how these algorithms influence the performance appears to be a topic of interest. Fig. 9 summarizes the results of Table IV.

TABLE VI
COMPARISON OF AVERAGE CUT SIZES FOR GEN100/GEN300

Instance	Vertex Encoding	Edge Encodings		
		Kruskal-like	DFS	BFS
r1000.1.0	565.27/ 569.73	581.63/ 585.33	582.40/ 586.50	560.10/ 563.73
r2000.1.0	1093.93/ 1098.70	1114.90/ 1120.87	1116.10/ 1122.60	1079.50/ 1085.17
r1000.0.8	700.93/ 705.20	714.13/ 719.63	713.73/ 718.63	708.00/ 714.37
r2000.0.8	1355.47/ 1362.93	1375.67/ 1382.80	1378.00/ 1383.60	1361.60/ 1368.87

Convergence diagrams are shown in Figs. 10–12 for the average of 30 runs with G500.005, G1000.005, and cat.702, which appeared to be typical GA runs. We have tested several cases with more generations and confirmed that only small improvements occur, relative to the large improvements made earlier in the run, Table VI shows results comparing some longer runs (300 generations) with the "standard" 100-generation runs.

V. CONCLUSION

We have proposed a new encoding method and investigated its performance comparing it to a widely-used method for the MAX CUT problem, which is one of the well-known NP-hard problems. This work is the first trial of applying edge-set encoding based on spanning tree as an optimization method for graph problems. To demonstrate the effectiveness of the proposed approach, experiments on three edge-set encodings derived from different spanning trees were conducted for benchmark graphs. Improvement in cut size was obtained for sparse graphs.

We conducted three experiments: tests on benchmark graphs, those on random graphs we generated, and those using three different methods of getting spanning trees. In all of the experiments, we got consistent results. The proposed edge-set encoding based on spanning tree has strong merits for addressing sparse graphs typical of real-world problems.

We also found that a change of encoding method could be beneficial for improving performance. That is, edge-set encoding based on spanning tree is advantageous for sparse graphs and vertex-based encoding is preferable for dense graphs.

The proposed approach can be applied to other graph partitioning problems, e.g., ratio-cut graph partitioning [11], [52], which are similar to the MAX CUT problem. In particular, the proposed edge-set encoding scheme based on spanning tree has merit for partitioning of sparse graphs, which appear widely in real-world applications (see [53]). Further study will aim to refine the encoding schemes, extend the experiments, and apply the encodings to various graph sets.

ACKNOWLEDGMENT

The authors would like to thank Prof. Yourim Yoon and Prof. Erik D. Goodman for their valuable suggestions for improving this paper. They would also like to thank Mr. Byeongyong Hyeon and Mr. Dongyeon Kim for their efforts to edit the paper.

REFERENCES

[1] F. Rothlauf, *Representations for Genetic and Evolutionary Algorithms*. Berlin, Germany: Springer, 2006.

[2] Y.-H. Kim, Y.-K. Kwon, and B.-R. Moon, "Problem-independent schema synthesis for genetic algorithms," in *Proc. Genet. Evol. Comput. Conf.*, Jul. 2003, pp. 1112–1122.

[3] Y.-H. Kim, "Linear transformation in pseudo-Boolean functions," in *Proc. Genet. Evol. Comput. Conf.*, Jul. 2008, pp. 1117–1118.

[4] Y.-H. Kim and Y. Yoon, "Effect of changing the basis in genetic algorithms using binary encoding," *KSII Trans. Internet Inf. Syst.*, vol. 2, no. 4, pp. 184–193, Aug. 2008.

[5] Y. Yoon and Y.-H. Kim, "A mathematical design of genetic operators on $GL_n(Z_2)$," *Math. Probl. Eng.*, vol. 2014, Apr. 2014, Art. ID 540936.

[6] C. J. Alpert and A. B. Kahng, "Recent directions in netlist partitioning: A survey," *Integr. VLSI J.*, vol. 19, nos. 1–2, pp. 1–81, Aug. 1995.

[7] T. N. Bui and C. Jones, "Finding good approximate vertex and edge partitions is NP-hard," *Inf. Process. Lett.*, vol. 42, no. 3, pp. 153–159, May 1992.

[8] L. H. Clark, F. Shahrokhi, and L. A. Székely, "A linear time algorithm for graph partition problems," *Inf. Process. Lett.*, vol. 42, no. 1, pp. 19–24, Apr. 1992.

[9] U. Feige, M. Karpinski, and M. Langberg, "A note on approximating Max-Bisection on regular graphs," *Inf. Process. Lett.*, vol. 79, no. 4, pp. 181–188, Aug. 2001.

[10] P. O. Fjällström, "Algorithms for graph partitioning: A survey," in *Linköping Electronic Articles in Computer and Information Science*, vol. 3. Linköping, Sweden: Linköping Univ. Electron. Press, 1998.

[11] L. Hagen and A. B. Kahng, "New spectral methods for ratio cut partitioning and clustering," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 11, no. 9, pp. 1074–1085, Sep. 1992.

[12] I. Hwang, Y.-H. Kim, and B.-R. Moon, "Multi-attractor gene reordering for graph bisection," in *Proc. Genetic Evol. Comput. Conf.*, Seattle, WA, USA, Jul. 2006, pp. 1209–1215.

[13] Y.-H. Kim, "An enzyme-inspired approach to surmount barriers in graph bisection," in *Proc. Int. Conf. Comput. Sci. Its Appl.*, LNCS 5072. Perugia, Italy, Jun. 2008, pp. 841–851.

[14] Y.-H. Kim and B.-R. Moon, "Lock-gain based graph partitioning," *J. Heuristics*, vol. 10, no. 1, pp. 37–57, Jan. 2004.

[15] Y.-H. Kim and B.-R. Moon, "Investigation of the fitness landscapes in graph bipartitioning: An empirical study," *J. Heuristics*, vol. 10, no. 2, pp. 111–133, Mar. 2004.

[16] L. Kuèera, "Expected complexity of graph partitioning problems," *Discrete Appl. Math.*, vol. 57, nos. 2–3, pp. 193–212, Feb. 1995.

[17] A. Moraglio, Y.-H. Kim, Y. Yoon, and B.-R. Moon, "Geometric crossovers for multiway graph partitioning," *Evol. Comput.*, vol. 15, no. 4, pp. 445–474, 2007.

[18] D. L. Powers, "Graph partitioning by eigenvectors," *Linear Algebra Appl.*, vol. 101, pp. 121–133, Apr. 1988.

[19] J.-T. Yan and P.-Y. Hsiao, "A fuzzy clustering algorithm for graph bisection," *Inf. Process. Lett.*, vol. 52, no. 5, pp. 259–263, Dec. 1994.

[20] Y. Yoon and Y.-H. Kim, "New bucket managements in iterative improvement partitioning algorithms," *Appl. Math. Inf. Sci.*, vol. 7, no. 2, pp. 529–532, Mar. 2013.

[21] Y. Yoon and Y.-H. Kim, "Vertex ordering, clustering, and their application to graph partitioning," *Appl. Math. Inf. Sci.*, vol. 8, no. 1, pp. 135–138, Jan. 2014.

[22] S. M. Antonio, D. Abraham, J. P. Juan, and C. Raúl, "High-performance VNS for the MAX-CUT problem using commodity graphics hardware," in *Proc. 18th Mini Euro Conf. VNS*, 2005.

[23] J. Cong, W. J. Labio, and N. Shivakumar, "Multiway VLSI circuit partitioning based on dual net representation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 15, no. 4, pp. 396–409, Apr. 1996.

[24] S. Guattery and G. L. Miller, "On the quality of spectral separators," *SIAM J. Matrix Anal. Appl.*, vol. 19, no. 3, pp. 701–719, Jul. 1998.

[25] J. Michel, F. Pellegrini, and J. Roman, "Unstructured graph partitioning for sparse linear system solving," in *Proc. 4th Int. Symp. Solv. Irregularly Struct. Probl. Parallel,* LNCS 1253. Paderborn, Germany, 1997, pp. 273–286.

[26] V. Venkatakrishnan, "Parallel computation of Ax and A^Tx," *Int. J. High Speed Comput.*, vol. 6, no. 2, pp. 325–342, Jun. 1994.

[27] M. Armbruster, M. Fügenschuh, C. Helmberg, N. Jetchev, and A. Martin, "Hybrid genetic algorithm within branch-and-cut for the minimum graph bisection problem," in *Evolutionary Computation in Combinatorial Optimization* LNCS 3906. Berlin, Germany: Springer, 2006, pp. 1–12.

[28] Y. Yoon, Y.-H. Kim, and B.-R. Moon, "A note on edge-based graph partitioning and its linear algebraic structure," *J. Math. Model. Algorith.*, vol. 10, no. 3, pp. 269–276, Sep. 2011.

[29] K. Seo, S. Hyun, and Y.-H. Kim, "A spanning tree-based encoding of the MAX CUT problem for evolutionary search," in *Proc. 12th Int. Conf. Parallel Probl. Solv. Nature—PPSN XII,* LNCS 7492. Sep. 2012, pp. 510–518.

[30] N. Biggs, *Algebraic Graph Theory*, 2nd ed. Cambridge, U.K.: Cambridge Univ. Press, 1994.

[31] R. Diestel, *Graph Theory* (Graduate Texts in Mathematics), vol. 173, 3rd ed. Heidelberg, Germany: Springer-Verlag, 2005.

[32] L. R. Ford, Jr., and D. R. Fulkerson, *Flows in Network*. Princeton, NJ, USA: Princeton Univ. Press, 1962.

[33] R. M. Karp, *Reducibility Among Combinatorial Problems*. New York, NY, USA: Plenum Press, 1972, pp. 85–103.

[34] M. R. Garey, D. S. Johnson, and L. J. Stockmeyer, "Some simplified NP-complete graph problems," *Theor. Comput. Sci.*, vol. 1, no. 3, pp. 237–267, 1976.

[35] S. Sahni and T. Gonzalez, "P-complete approximation problems," *J. ACM*, vol. 23, no. 3, pp. 555–565, Jul. 1976.

[36] P. M. B. Vitányi, "How well can a graph be *n*-colored?" *Discrete Math.*, vol. 34, no. 1, pp. 69–80, Jan. 1981.

[37] S. Poljak and Z. Tuza, "A polynomial algorithm for constructing a large bipartite subgraph, with an application to a satisfiability problem," *Can. J. Math.*, vol. 34, no. 3, pp. 519–524, Jun. 1982.

[38] D. J. Haglin and S. M. Venkatesan, "Approximation and intractability results for the maximum cut problem and its variants," *IEEE Trans. Comput.*, vol. 40, no. 1, pp. 110–113, Jan. 1991.

[39] T. Hofmeister and H. Lefmann, "A combinatorial design approach to MAXCUT," in *Proc. 13th Symp. Theor. Aspects Comput. Sci.,* Grenoble, France, 1995.

[40] M. X. Goemans and D. P. Williamson, "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming," *J. ACM*, vol. 42, no. 6, pp. 1115–1145, Nov. 1995.

[41] S. Homer and M. Peinado, "Design and performance of parallel and distributed approximation algorithms for MAXCUT," *J. Parallel Distrib. Comput.*, vol. 46, no. 1, pp. 48–61, Oct. 1997.

[42] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[43] S.-H. Kim, Y.-H. Kim, and B.-R. Moon, "A hybrid genetic algorithm for the MAX CUT problem," in *Proc. Genetic Evol. Comput. Conf.*, Jul. 2001, pp. 416–423.

[44] Q. Wu and J.-K. Hao, "A memetic approach for the MAX-CUT problem," in *Proc. 12th Int. Conf. Parallel Probl. Solv. Nature (PPSN XII),* LNCS 7492. Taormina, Italy, Sep. 2012, pp. 297–306.

[45] F. Barahona, M. Grotschel, M. Junger, and G. Reinelt, "An application of combinatorial optimization to statistical physics and circuit layout design," *Oper. Res.*, vol. 36, no. 3, pp. 493–513, 1984.

[46] R. Y. Pinter, "Optimal layer assignment for interconnect," *J. VLSI Comput. Syst.*, vol. 1, pp. 123–137, 1984.

[47] S. Poljak and Z. Tuza, "Maximum cuts and largest bipartite subgraphs," *Discrete Math. Theor. Computer Sci. Amer. Math. Soc*, (Combinatorial Optimization DIMACS), vol. 20, pp. 181–244, 1995

[48] T. Jones and S. Forrest, "Fitness distance correlation as a measure of problem difficulty for genetic algorithms," in *Proc. 6th Int. Conf. Genet. Algorith.*, 1995, pp. 184–192.

[49] (2013, Jan.). *Boost Library* [Online]. Available: http://boost.org

[50] C. Gagné and M. Parizeau, "Genericity in evolutionary computation software tools: Principles and case study," *Int. J. Artif. Intell. Tools*, vol. 15, no. 2, pp. 173–194, 2006.

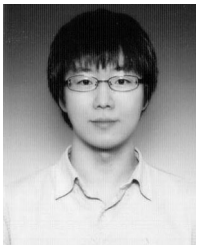
[51] T. N. Bui and B.-R. Moon, "Genetic algorithm and graph partitioning," *IEEE Trans. Comput.*, vol. 45, no. 7, pp. 841–855, Jul. 1996.

[52] T. N. Bui and B.-R. Moon, "GRCA: A hybrid genetic algorithm for circuit ratio-cut partitioning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 17, no. 3, pp. 193–204, Mar. 1998.

[53] Y.-H. Kim, S. Seo, Y.-H. Ha, S. Lim, and Y. Yoon, "Two applications of clustering techniques to Twitter: Community detection and issue extraction," *Discrete Dyn. Nature Soc.*, vol. 2013, Oct. 2013, Art. ID 903765.

Kisung Seo received the B.S., M.S., and Ph.D. degrees in electrical engineering from Yonsei University, Seoul, Korea, in 1986, 1988, and 1993, respectively.

He was a full-time Lecturer and Assistant Professor of Industrial Engineering with Seokyeong University, Seoul, in 1993 and 1995, respectively. He was with the Genetic Algorithms Research and Applications Group and Case Center for Computer-Aided Engineering and Manufacturing, Michigan State University, East Lansing, MI, USA, from 1999 to 2003, first as a Research Associate, then as a Visiting Assistant Professor in Electrical and Computer Engineering, from 2002 to 2003. He visited Michigan State University again, from 2011 to 2012, where he did collaborative research in the BEACON Center for the Study of Evolution in Action, an NSF Science and Technology Center headquartered there. He has been an Associate Professor of Electronics Engineering, Seokyeong University, since 2004. His research interests involve genetic programming, evolutionary algorithm, evolutionary design of gait generation for quadruped and humanoid robots, evoultionary detection of image features, and evolutionary prediction method for weather systems.



Soohwan Hyun received the B.S. and M.S. degrees in electronic engineering from Seokyeong University, Seoul, Korea, in 2010 and 2012, respectively.

He is a Research Engineer with Hyundai Heavy Industries Research Institute, Yongin, Korea. His research interests include genetic programming, evolutionary algorithm, and evolutionary robotics.

Yong-Hyuk Kim (M'13) received the B.S. degree in computer science, and the M.S. and Ph.D. degrees in computer science and engineering from Seoul National University, Seoul, Korea, in 1999, 2001, and 2005, respectively.

Since 2007 he has been a Professor with the Department of Computer Science and Engineering, Kwangwoon University, Seoul. His research interests include algorithm design/analysis, discrete mathematics, optimization theory, combinatorial optimization, evolutionary computation, operations research, data/web mining, machine learning, smart grids, and sensor networks.

Dr. Kim has served as an Editor of the KSII Transactions on Internet and Information Systems from 2010 to 2013. He is a member of the IEEE SMC Society and the IEEE Communications Society.