Multi-Attractor Gene Reordering for Graph Bisection

Inwook Hwang
School of Computer Science & Engineering
Seoul National University
Sillim-dong, Gwanak-gu,
Seoul, 151-744 Korea
hwang@soar.snu.ac.kr

Yong-Hyuk Kim
Department of Mathematical Sciences & Research Institute of Mathematics
Seoul National University
Sillim-dong, Gwanak-gu,
Seoul, 151-747 Korea
yhdfly@soar.snu.ac.kr

Byung-Ro Moon
School of Computer Science & Engineering
Seoul National University
Sillim-dong, Gwanak-gu,
Seoul, 151-744 Korea
moon@soar.snu.ac.kr

ABSTRACT

We propose a new gene reordering scheme for the graph bisection problem. Our gene reordering starts with two or more vertices to capture the clustering structure of graphs effectively. We devised a chromosome repairing method for hybrid genetic search, which helps exploit clusters when combined with gene reordering. Experimental tests showed that the suggested reordering scheme significantly improves the performance of genetic algorithms over previous reordering methods.

Categories and Subject Descriptors

G.1.6 [**Numerical Analysis**]: Optimization—*Global optimization*

General Terms

Algorithms

Keywords

Gene reordering, genetic algorithm, graph partitioning, repairing scheme, schema preprocessing, vertex ordering

1. INTRODUCTION

Let $G = (V, E)$ be an unweighted undirected graph, where V is the set of vertices and E is the set of edges. A balanced k-way partition $\{C_1, C_2, ..., C_k\}$ of the graph G is a partitioning of the vertex set V into k disjoint subsets where the difference of cardinalities between the largest subset and the smallest one is at most one, i.e., for all $1 \leq i, j \leq k$, $||C_i| - |C_j|| \leq 1$. The *cut size* of a partition is defined to be the number of edges whose endpoints belong to different subsets of partition. The *k-way graph partitioning* problem is the problem of finding a balanced k-way partition with minimum cut size. In particular, it is called the *graph bisection* problem when k is equal to 2.

It is known that the bisection problem for general graphs is NP-hard [8]. Bui and Jones [4] showed that even finding good approximation solutions for general graphs is also NP-hard. So, heuristic methods were used for the problem such as Kernighan-Lin algorithm (KL) [13], Fiduccia-Mattheyses algorithm (FM) [7], simulated annealing (SA) [12], genetic algorithm (GA) [5], and tabu search [2]. Recently, there have been a number of studies using GAs, reporting notable results [14] [16] [17] [19].

Bui and Moon [5] introduced a gene reordering[1] for the graph partitioning problem. They reported superior results to GAs without reordering. As a schema preprocessing technique, gene reordering helps genetic algorithms effectively create and preserve high-quality schemata. It consequently improves the performance of GAs. In this paper, we propose a new gene reordering scheme that captures the clustering structure of graphs more effectively. One of the distinguishing features of our reordering method is that it starts with two or more vertices (*multi-attractor*). We also devised an improved chromosome repairing scheme which exploits the clustering information obtained by reordering. In experiments, the suggested reordering scheme and repairing method excelled compared to other traditional approaches.

The remainder of this paper is organized as follows. We give the concepts of vertex reordering and schema disruptivity in Section 2. In Section 3, we explain our multi-attractor gene reordering. In Section 4, we describe our genetic algorithm for the graph partitioning problem and give the concepts of contiguous repairing for more effective search. Our experimental results are presented in Section 5. Finally, the conclusion is given in Section 6.

2. PRELIMINARIES

2.1 Vertex Reordering

Given the set of vertices $V = \{v_1, v_2, \ldots, v_n\}$, a vertex reordering $\{v_{\sigma(1)}, v_{\sigma(2)}, \ldots, v_{\sigma(n)}\}$ is defined by the bijective map $\sigma : \{1, 2, \ldots, n\} \rightarrow \{1, 2, \ldots, n\}$. Vertex v_i is the j^{th} vertex in the vertex reordering if $\sigma(j) = i$.

[1]In this paper, we use "gene reordering" and "vertex reordering" interchangeably.

GECCO'06, July 8–12, 2006, Seattle, Washington, USA.

A number of studies on vertex reordering have been conducted [1] [18]. BFS reordering performs a breadth first search (BFS) on the input graph starting at a random vertex. The visiting order of vertices in BFS is used to reorder the vertices. In Max-Adjacency reordering [18], starting at a random vertex, the vertex with the most edges incident to the already ordered vertices is iteratively added to the reordering.

Choose a random number $a \in \{1, 2, \ldots, n\}$;
$\sigma(1) = a$;
for $i = 2$ **to** n {
  Find an unordered vertex v_r
    such that $Attract(v_r)$ is maximal;
  $\sigma(i) = r$;
}
Reorder the vertices using the bijective map σ;

Figure 1: Framework of traditional reordering

Figure 1 shows the framework of traditional reordering [1]. $Attract(v_i)$ means the *attraction* from v_i to the set of ordered vertices [1]. It can be represented as follows.

- BFS Reordering: $Attract(v_i) = min\{j \mid (v_i, v_{\sigma(j)}) \in E,\ v_{\sigma(j)}$ is ordered$\}$
- Max-Adjacency Reordering: $Attract(v_i) = |\{(v_i, v_j) \in E \mid v_j$ is ordered$\}|$

As shown in Figure 1, traditional reordering starts with a random vertex. We call this initial vertex an *attractor*. In Section 3, we will introduce a generalized reordering that starts with two or more attractors.

2.2 Schema and Its Disruptivity

In genetic algorithm for the graph bisection problem, each solution is usually represented by a binary string with the length of $|V|$. Each solution is called a *chromosome*; that is, a chromosome corresponds to a partition of the graph. Each gene corresponds to a vertex in the graph. It has the value $i-1$ if the corresponding vertex belongs to set C_i. A genetic algorithm for this problem evolves a group of chromosomes under a genetic process. A schema is a pattern of genes that can be described by a template consisting of $\{0, 1\}$ and asterisks; here, zero and one represent the pattern and the asterisks represent "don't care."

Schema is the important terminology which can explain a working principle of GA. According to the *building block hypothesis* [9], a GA recombines and forms strings of higher-order schemata from short, low-order schemata in the early stages.

In the case of the graph bisection problems, a genetic algorithm starts with a group of random initial binary strings (solutions). Of course, the quality of the solutions is low in the early stages of the genetic algorithm. However, most low-quality solutions contain some schemata common to high-quality solutions. The crossover operators of genetic algorithms generate larger schemata by juxtaposition of smaller schemata. So, it is important to preserve the valuable schemata.

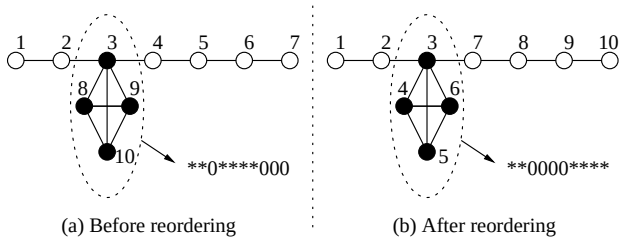


Figure 2: Decrease of defining length of schema

However, they are prone to be destroyed by the crossover operators if the positions forming the schema are scattered[2].

In Figure 2a, schema **0****000 (or **1****111) represents the set of solutions in which four vertices of the clique belong to the same partition. This schema is a good example of a schema with high quality. In Figure 2, reordering decreased the defining length (distance between the first and the last specific string position) of the schema from 7 to 3. So the survival probability of the schema increases under the traditional crossover, e.g., n-point crossover.

The objective of a vertex reordering (or gene reordering) in genetic partitioning algorithm is to preserve the clustering structure of the graphs and to decrease the defining lengths of good schemata. Experimental results in Section 3.3 will show that vertex reordering decreases the defining lengths of good schemata.

3. MULTI-ATTRACTOR REORDERING ALGORITHM

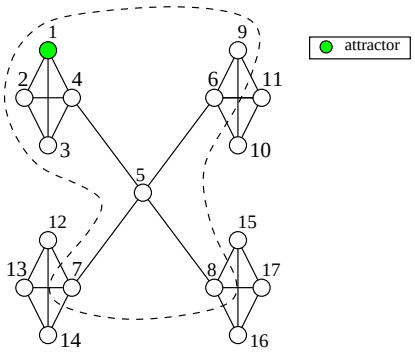
All the traditional reorderings are one-attractor reorderings, which means that the reorderings start with one attractor. Figure 3a is an example that demonstrates how one-attractor reorderings do not work well. Vertices of most cliques were not closely indexed. In particular, on large graphs, one-attractor reorderings are prone to scatter the specific symbols of the valuable schemata. As shown in Figure 3b, reordering with two or more attractors can preserve the clustering structure well in this case. Our motivation in undertaking this study starts from this observation.

3.1 Algorithm Overview

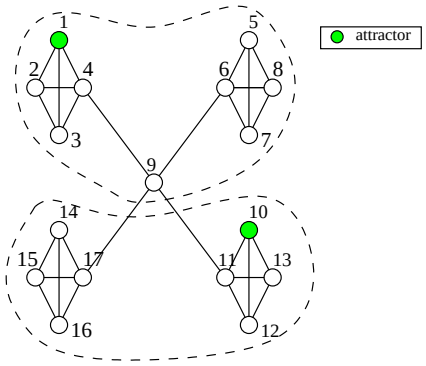
Figure 4 shows our reordering heuristic with two attractors. Like in previous methods [1] [18], starting with the random attractors, it iteratively adds a new vertex to the ordering based on the already ordered vertices. The main difference lies in how the already ordered vertices are classified. We use two classes (A and B). One (A) is the former-half ordered vertices and the other (B) is the latter-half ordered vertices. Maintaining the balance of two classes, the heuristic allows 10 percent of skewness between the sets A and B.

Let $E(v, S)$ be the number of vertices adjacent to the vertex v in a vertex set S. We define $Attract_A(v)$ and $Attract_B(v)$ to be $E(v, A) - E(v, B)$ and $E(v, B) - E(v, A)$, respectively. The vertex with the maximal value of $Attract_A$

[2]Holland [11] showed that schemata with shorter defining lengths have higher survival probabilities in a single-point crossover. Also in multi-point crossover with odd numbers of crossover points, it was shown that shorter schemata have higher chances to survive [6].



(a) A result of one−attractor reordering



(b) A result of two−attractor reordering

Figure 3: Need of new reordering

is added to the former-half ordering and the set A. Similarly, the vertex with the maximal value of $Attract_B$ is added to the latter-half ordering and the set B. We name this reordering heuristic *2-attractor reordering*. Multi-attractor reordering with 2^n attractors is based on 2-attractor reordering. It first reorders vertices with 2-attractor reordering and recursively applies 2-attractor reordering to each partition. In our experiments, 2-attractor reordering and 4-attractor reordering were used.

FM algorithm for graph partitioning uses a bucket data structure to maintain the gains of vertices in linear time [7]. Using a bucket data structure like in FM algorithm to compute attraction, multi-attractor reordering heuristic takes $\Theta(|V|+|E|)$ time. When a tie occurs, we use the stack-based management[3] (i.e., the most recently inserted vertex has the highest priority in each bucket) in our reordering and Max-Adjacency reordering.

3.2 Finding Good Attractors

Traditional vertex reorderings start with an attractor. This attractor affects the quality of reordering. Figure 5 shows examples of reordering with a good attractor and a bad attractor. We devised some metrics for evaluating the quality of reordering (one for BFS reordering and another for multi-attractor reordering). As the computational cost of reordering is low[4], we can run BFS and multi-attractor heuristic several times. Among the results of reordering, our GA chooses an attractor with the longest diameter in BFS reordering and a pair of attractors with the minimum cut size between the two classes in 2-attractor reordering, respectively.

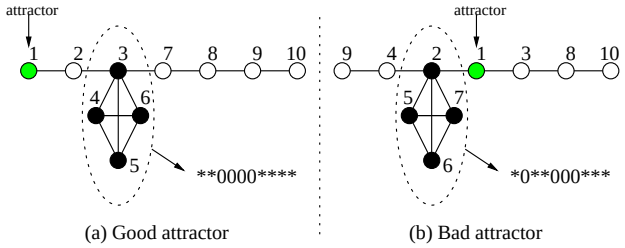
Table 1 shows the correlation coefficient[5] between these metrics and the results of GAs. In each class of graphs, only four largest graphs in Section 5.2 are included. More specifically, in the case of BFS reordering, the correlation

[3]This management is traditional and attractive [10].

Choose two different random numbers $a, b \in \{1, 2, \ldots, n\}$;
$\sigma_A(1) = a, \sigma_B(1) = b$;
$A = \{v_a\}, B = \{v_b\}$;
for $i = 3$ **to** n {
  Find an unordered vertex $v_r^A \in V \backslash (A \cup B)$
    such that $Attract_A(v_r^A)$ is maximal;
  Find an unordered vertex $v_r^B \in V \backslash (A \cup B)$
    such that $Attract_B(v_r^B)$ is maximal;
  if $|B|/|A| > \alpha^*$ **then**
    $\sigma_A(|A|+1) = v_r^A,\ A = A \cup \{v_r^A\}$;
  else if $|A|/|B| > \alpha^*$ **then**
    $\sigma_B(|B|+1) = v_r^B,\ B = B \cup \{v_r^B\}$;
  else if $Attract_A(v_r^A) \geq Attract_B(v_r^B)$ **then**
    $\sigma_A(|A|+1) = v_r^A,\ A = A \cup \{v_r^A\}$;
  else
    $\sigma_B(|B|+1) = v_r^B,\ B = B \cup \{v_r^B\}$;
}
for $i = 1$ **to** n {
  if $i \leq |A|$ **then**
    $\sigma(i) = \sigma_A(i)$
  else
    $\sigma(i) = \sigma_B(i - |A|)$
}
Reorder the vertices using the bijective map σ;

$*$ In our experiments, we set the skewness parameter α to be 1.1.

Figure 4: Two-attractor reordering



(a) Good attractor (b) Bad attractor

Figure 5: Reordering examples

[4]In the experiments, 50 repetitions of BFS reordering took 0.6 percent of total running time and 50 repetitions of 2-attractor reordering took 2.4 percent of total running time, on average. However, the total running time decreased, instead. See experimental results in Section 5.2.

[5]The correlation coefficient of X and Y is

$$\rho(X,Y) = \frac{E[(X - E(X))(Y - E(Y))]}{\sqrt{Var(X)Var(Y)}}.$$

Table 1: Correlation coefficient between each metric and the bisection result of GA

Graph	BFS	2-Att†
G1000.2.5	0.00	−0.01
G1000.05	−0.02	−0.03
G1000.10	0.05	0.04
G1000.20	−0.04	0.01
U2000.05	−0.47	0.57
U2000.10	0.01	0.37
U5000.05	−0.25	0.87
U5000.10	−0.42	0.65
pcart.1052	−0.35	0.28
pcart.5252	−0.37	0.38
prcart.994	−0.50	0.22
prcart.5114	−0.40	0.20

∗ Average over 1,000 runs.

† Two-attractor reordering.

coefficient is between the size of the diameter and the bisection cut size of the GA, in the case of 2-attractor reordering, it is between the cut size of the attraction sets and the bisection cut size of the GA. We can observe a strong correlation between these metrics and the performance of GAs. The results show that, except on random graphs (G*.*), BFS reordering with long diameters and 2-attractor reordering with low cut sizes improve the performance of GAs. We could not find an appropriate metric for Max-Adjacency reordering. In our experiments, we set the number of trials to be 50, both in BFS reordering and multi-attractor reordering.

3.3 Effect of Schema Preprocessing

We examined the effect of vertex reordering by measuring the defining lengths of the schemata associated with clusters before and after reordering. For measuring the defining lengths of the clusters, we located 10 maximal cliques of size 3 or more for each run. We compared the defining lengths of the schemata corresponding to these cliques before and after gene reordering.

Table 2: The defining lengths of schemata representing cliques before and after reordering

Graph	None	BFS	Max-Adj	2-Att	4-Att	Clique size
G500.2.5	369.57	81.86	75.11	48.38	34.30	3.00
G500.05	327.83	161.47	90.34	131.92	120.74	3.00
G500.10	326.17	209.88	169.11	166.55	161.90	3.00
G500.20	315.47	237.65	204.11	195.38	183.34	3.05
G1000.2.5	554.89	190.86	113.35	135.92	112.12	3.00
G1000.05	534.41	334.29	176.34	228.06	215.90	3.00
G1000.10	682.01	420.12	295.10	346.76	330.67	3.00
G1000.20	637.61	470.08	384.09	384.26	383.27	3.01
U500.05	324.24	15.30	15.51	10.23	10.29	4.72
U500.10	388.94	33.59	25.97	29.29	28.32	6.40
U1000.05	646.20	14.03	10.73	11.62	11.46	4.41
U1000.10	758.83	50.68	30.88	36.00	31.28	6.36
U2000.05	1310.37	23.44	17.49	14.43	11.68	4.57
U2000.10	1516.67	72.33	60.57	50.87	46.48	6.31
U5000.05	3217.93	49.81	37.04	23.40	17.01	4.36
U5000.10	3817.27	111.88	107.36	80.29	50.49	6.50

∗ Average over 100 runs (a total of 1,000 cliques).

† There is no clique of size 3 or more in caterpillar graphs.

Table 2 shows the effect of each reordering scheme on random graphs (G*.*) and random geometric graphs (U*.*). The column "None" shows the result without reordering, and "Max-Adj" shows with Max-Adjacency reordering. The columns labeled "2-Att" and "4-Att" show the results of 2-attractor reordering and 4-attractor reordering, respectively. Each reordering significantly decreased the defining lengths of schemata associated with the cliques, which are expected to be of high quality. The fact is also remarkable that there are few cliques of size 4 or more in random graphs.

4. GENETIC FRAMEWORK

4.1 Overview

A genetic algorithm generates a set of initial solutions and evolves over a number of iterations. When GA meets some condition, the best solution is returned and the algorithm is terminated. Our genetic algorithm generates only one offspring per generation (steady-state GA) and use a local improvement heuristic after crossover and repairing step (hybrid GA). In the following, our genetic algorithm for the graph bisection problem is described.

- Encoding: A binary string for each chromosome to represent a bisection is used. Each gene in a chromosome has the value 0 or 1 depending on the partition which the matching gene belongs to.
- Initialization: When GA starts, thirty chromosomes are created at random.
- Selection: The roulette-wheel-based proportional selection is used. The probability that the best chromosome is chosen was set to four times higher than the probability that the worst chromosome is chosen. The fitness value f_i of the i^{th} chromosome can be represented as follows, where c_b, c_w, and c_i are the cut sizes of the best chromosome, the worst chromosome, and the i^{th} chromosome, respectively.

  $f_i = (c_w - c_i) + (c_w - c_b)/3.$
- Crossover and mutation: We used the crossover operator proposed in [5]. It first produces two offspring. One is from traditional 5-point crossover and the other is from 5-point crossover which copies the part of the first parent and the complement values of the part of the second parent. After crossover, chromosomes are usually unbalanced. So the repairing process is necessary, which is described in Section 4.2 (as it produces some mutation effect, we did not add any specific mutation). Among two repaired offspring, better one is chosen as the final offspring.
- Replacement: We used the replacement scheme of [5]. If the offspring is better than its closer parent in Hamming distance, GA replaces its closer parent. If not, the other parent is replaced if the offspring is better. If not again (i.e., the offspring is worse than both parents), we replace the worst member of the population.
- Stopping criterion: For stopping, we use the number of consecutive fails to replace one of the parents. We set the number to be 30.
- Local optimization: KL algorithm [13] is used for local optimization. KL proceeds in a series of *passes*. During each pass, the algorithm improves on an initial

solution by swapping pairs of vertices to create a new solution. This process is repeated on the new bisection until no improvement can be obtained.

Let (A, B) be an initial bipartition of $G = (V, E)$. Define the *gain* g_v of a vertex v to be the cut-size reduction by moving v to the opposite set. The gain $g(a, b)$ as a result of swapping vertices $a \in A$ and $b \in B$ is as follows:

$$g(a, b) = g_a + g_b - 2\delta(a, b)$$

where

$$\delta(a, b) = \begin{cases} 1, & \text{if } (a, b) \in E \\ 0, & \text{otherwise.} \end{cases}$$

The pair (a, b) which maximizes $g(a, b)$ is selected. Once a and b are selected, they are assumed to be exchanged and not considered any more for further exchange. A sequence of pairs $(a_1, b_1), \ldots, (a_{n/2-1}, b_{n/2-1})$ are selected in this way, where n is the number of vertices. The algorithm chooses a pair (X, Y), $X = \{a_1, \ldots, a_k\}$ and $Y = \{b_1, \ldots, b_k\}$, such that $\sum_{i=1}^{k} g(a_i, b_i)$ is maximized. The algorithm exchanges X and Y. This is a pass of KL. KL repeats the above pass until no improvement is possible.

In our genetic algorithms, we used the fast variation of KL proposed in [5]. Its time complexity is $O(|E|)$.

4.2 Repairing Scheme

```
Contiguous-Repairing (addset)
{
    gab = |V|/2 − |C_addset|;
    Choose a random number r ∈ {0, 1, ..., n − 1};
    p_−1 = r, p_1 = r;
    direction = 1;
    while (gab > 0) {
        if (chromosome[r + 1] = 1 − addset) {
            chromosome[r + 1] = addset;
            gab = gab − 1;
        }
        else
            direction = −direction;
        p_direction = p_direction + direction;
        r = p_direction mod n;
    }
}
```

Figure 6: Contiguous repairing

After the crossover in genetic partitioning algorithms, an offspring may not have the same number of 0s and 1s in most cases. So, repairing is necessary and the repairing strategy can affect the quality of solutions.

In general, the random repairing scheme is used, which selects the required number of random 1s (or 0s) and changes them to 0s (or 1s). But with reordering, GA can do repairing more effectively. In Bui and Moon's study [5], GA selects a random point on the chromosome and repairs the required number of 1s to 0s (or 0s to 1s) starting at that point on to the right (and wrapping around if necessary). But the study only mentioned the mutation effect of repairing. Experimental results in Section 5.1 will show the importance of repairing scheme in genetic algorithms for the graph bisection problem.

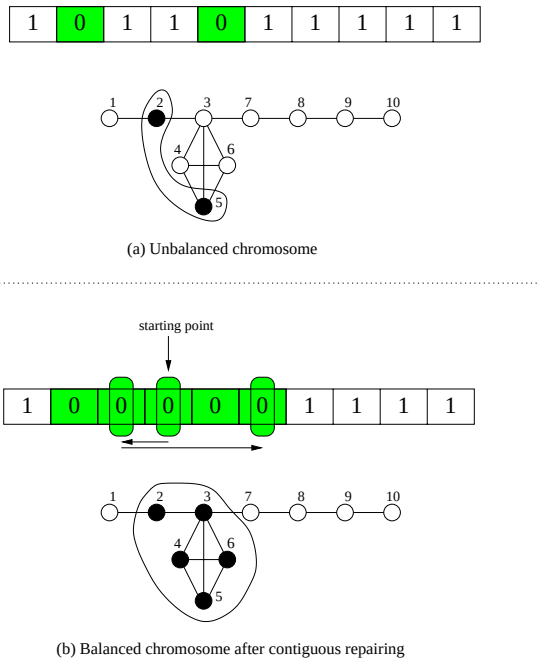


Figure 7: An example of contiguous repairing

Figure 6 shows our contiguous repairing algorithm, an improved version of [5] and Figure 7 shows an example of the repairing. First, GA selects a random starting point on the unbalanced chromosome. It changes 1s to 0s (or 0s to 1s) to the right until the consecutive string of 1s (or 0s) ends. GA changes 1s to 0s from the starting point to the left until consecutive string of 1s (or 0s) on the left side ends. This process is repeated until the required number of 1s (or 0s) are changed to 0s (or 1s).

The main difference from [5] lies in that our repairing minimizes the length of the repaired part of the chromosome. Contiguous repairing increases the probability of a good schema being generated in the repairing process.

5. EXPERIMENTAL RESULTS

In this section, we give our experimental results on the graphs that were used in [5] and [14]. Test graphs are composed of 8 random graphs (G*.*) [12], 8 random geometric graphs (U*.*) [12], and 8 caterpillar graphs (pcart.* and prcart.*) [5]. We omit results on quite easy benchmark graphs such as dense geometric graphs (U*.20, U*.40), random regular graphs [3] and grid graphs [5], on which our GAs found the best known solutions in most runs[6]. All benchmark graphs are available on the web[7].

The code was programmed in C language on a Pentium IV 2.8GHz computer with Linux. It was compiled using GNU's *gcc* compiler.

Since we performed 1,000 runs for all the experiments, the confidence intervals of the results are quite narrow.

[6]More than 90 percent of trials.

[7]`http://soar.snu.ac.kr/benchmark/`

5.1 Comparison of Repairing Schemes

Table 3 shows the bisection cut sizes using the random repairing and the contiguous repairing. In each class of graphs, only four largest graphs are included. In GA without reordering, repairing scheme did not affect results. When combined with reordering, the contiguous repairing scheme improved the performance of genetic bisection algorithms dramatically on geometric graphs and caterpillar graphs.

5.2 Results of Bisection

Table 4 shows the performance of genetic bisection algorithms without reordering and using BFS, Max-Adjacency, 2-attractor, 4-attractor reordering, respectively. As reported by Bui and Moon [5], the improvement resulting from the reordering heuristic is dramatic, yet simple and low-cost. The contiguous repairing heuristic augmented the improvement in our experiments. Except on G1000.2.5 and U500.10, one of the multi-attractor reorderings performed better than the other reorderings within comparable time. GAs preprocessed by multi-attractor reorderings separately outperformed GAs with the other two reorderings, on almost all the benchmark graphs.

In particular, on large geometric graphs such as U2000.* and U5000.*, multi-attractor reorderings considerably outperformed the others. On U2000.05, U5000.05 and U5000.10, only the GAs with multi-attractor reorderings found the best known value in [14]. On U5000.05 and U5000.10, GAs preprocessed by multi-attractor reorderings obtained the average cut sizes even better than the best of the 1,000 runs of GAs preprocessed by the other traditional reorderings.

Compared to GA without reordering, GAs with multi-attractor reordering slightly decreased cut sizes of random graphs. A study of the problem space [15] hints why improving the performance of genetic partitioning algorithm by reordering on random graphs is hard.

Mostly, GAs preprocessed by reorderings had similar running times to GAs without reordering. Furthermore, on large geometric and caterpillar graphs, reordering considerably decreases total running time.

6. CONCLUSIONS

In this paper, we proposed a multi-attractor gene reordering method for graph bisection. We also suggested the contiguous chromosomal repairing as a good combination with the reordering method. Experimental results on bisection problem showed that the new reordering scheme and repairing scheme significantly improves the performance of GAs.

Further applications of multi-attractor reordering such as the multi-way partitioning problem or the circuit partitioning problem are left for future study.

7. ACKNOWLEDGMENTS

This work was supported by the Brain Korea 21 Project in 2006. This was also partly supported by grant No. (R01-2003-000-10879-0) from the Basic Research Program of the Korea Science and Engineering Foundation. The ICT at Seoul National University provided research facilities for this study.

8. REFERENCES

[1] C. Alpert and A. B. Kahng. A general framework for vertex orderings, with applications to netlist clustering. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 63–67, 1994.

[2] R. Battiti and A. Bertossi. Greedy, prohibition, and reactive heuristics for graph partitioning. *IEEE Transactions on Computers*, 48(4):361–385, 1999.

[3] T. N. Bui, S. Chaudhuri, F. T. Leighton, and M. Sipser. Graph bisection algorithms with good average case behavior. *Combinatorica*, 7(2):171–191, 1987.

[4] T. N. Bui and C. Jones. Finding good approximate vertex and edge partitions is NP-hard. *Information Processing Letters*, 42:153–159, 1992.

[5] T. N. Bui and B. R. Moon. Genetic algorithm and graph partitioning. *IEEE Transactions on Computers*, 45(7):841–855, 1996.

[6] K. DeJong and W. Spears. A formal analysis of the role of multi-point crossover in genetic algorithms. *Annals of Mathematics and Artificial Intelligence*, 5:1–26, 1992.

[7] C. Fiduccia and R. Mattheyses. A linear time heuristics for improving network partitions. In *Proceedings of the 19th ACM/IEEE Design Automation Conference*, pages 175–181, 1982.

[8] M. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

[9] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[10] L. Hagen, J. H. Huang, and A. B. Kahng. On implementation choices for iterative improvement partitioning algorithms. *IEEE Transactions on Computer-Aided Design*, 16(10):1199–1205, 1997.

[11] J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[12] D. S. Johnson, C. Aragon, L. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation, Part 1, graph partitioning. *Operations Research*, 37:865–892, 1989.

[13] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 49:291–307, Feb. 1970.

[14] Y. H. Kim and B. R. Moon. Lock gain based graph partitioning. *Journal of Heuristics*, 10(1):37–57, 2004.

[15] Y. H. Kim and B. R. Moon. New topologies for genetic search space. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1393–1399, 2005.

[16] P. Merz and B. Freisleben. Fitness landscapes, memetic algorithms, and greedy operators for graph bipartitioning. *Evolutionary Computation*, 8(1):61–91, 2000.

[17] H. Mühlenbein and T. Mahnig. Evolutionary optimization and the estimation of search distributions with applications to graph bipartitioning. *International Journal of Approximate Reasoning*, 31(3):157–192, 2002.

Table 3: The effect of contiguous repairing scheme

Graph	None		BFS		Max-Adj		2-Att		4-Att	
	random	contiguous	random	contiguous	random	contiguous	random	contiguous	random	contiguous
G1000.2.5	97.53	**97.43**	97.57	**97.44**	97.46	**97.41**	97.44	**97.42**	**97.42**	97.45
G1000.05	**451.97**	452.04	451.89	**451.85**	451.74	**451.54**	451.56	**451.40**	451.56	**451.47**
G1000.10	1368.16	**1368.15**	**1367.74**	1367.89	1367.68	**1367.43**	1367.44	**1367.36**	**1367.23**	1367.25
G1000.20	3387.74	**3387.53**	3387.32	**3387.15**	3386.82	**3386.61**	3387.06	**3386.22**	3386.59	**3386.21**
U2000.05	35.09	**34.91**	29.38	**7.89**	28.65	**8.09**	29.25	**4.13**	28.52	**4.17**
U2000.10	75.89	**75.23**	**59.26**	62.68	58.89	**54.60**	56.60	**47.83**	57.33	**48.32**
U5000.05	**128.48**	129.11	124.00	**21.24**	118.33	**22.50**	114.83	**4.99**	114.08	**5.14**
U5000.10	**253.16**	255.18	182.54	**125.97**	171.95	**147.07**	165.64	**82.47**	163.34	**81.97**
pcart.1052	17.52	**17.29**	3.81	**2.64**	5.00	**2.68**	4.15	**2.12**	4.28	**2.52**
pcart.5252	**91.39**	91.77	6.94	**2.75**	8.54	**2.84**	7.72	**2.35**	7.56	**2.70**
prcart.994	**5.94**	5.98	2.63	**1.96**	2.89	**2.37**	2.34	**1.50**	2.74	**1.87**
prcart.5114	**20.58**	20.65	2.99	**2.48**	3.70	**2.60**	3.08	**1.96**	3.20	**2.31**

∗ Average over 1,000 runs.

† Bisection cut sizes.

Table 4: The results of bisection

Graph	None		BFS		Max-Adj		2-Att		4-Att	
	Average(Best)	CPU	Average(Best)	CPU	Average(Best)	CPU	Average(Best)	CPU	Average(Best)	CPU
G500.2.5	51.36(49)	0.32	51.43(49)	0.31	51.47(49)	0.32	**51.30**(49)	0.33	51.31(49)	0.33
G500.05	218.94(218)	0.55	218.98(218)	0.58	218.99(218)	0.60	**218.69**(218)	0.59	218.72(218)	0.61
G500.10	627.89(626)	0.93	627.84(626)	0.96	627.78(626)	0.99	627.71(626)	1.01	**627.67**(626)	1.04
G500.20	1746.64(1744)	1.73	1746.48(1744)	1.77	1746.50(1744)	1.81	1746.24(1744)	1.87	**1746.17**(1744)	1.91
G1000.2.5	97.43(93)	0.94	97.44(95)	0.97	**97.41**(94)	0.98	97.42(93)	1.01	97.45(93)	1.05
G1000.05	452.04(445)	1.90	451.85(445)	1.98	451.54(445)	2.04	**451.40**(445)	2.07	451.47(445)	2.12
G1000.10	1368.15(1362)	3.53	1367.89(1362)	3.61	1367.43(1362)	3.72	1367.36(1362)	3.79	**1367.25**(1362)	3.90
G1000.20	3387.53(3382)	6.23	3387.15(3382)	6.38	3386.61(3382)	6.53	3386.22(3382)	6.51	**3386.21**(3382)	6.64
U500.05	5.37(2)	0.54	2.65(2)	0.51	4.07(2)	0.49	2.07(2)	0.47	**2.06**(2)	0.48
U500.10	26.70(26)	0.44	26.08(26)	0.58	**26.02**(26)	0.51	26.05(26)	0.53	26.04(26)	0.54
U1000.05	12.05(1)	1.63	1.12(1)	1.13	1.63(1)	1.13	**1.01**(1)	1.12	1.04(1)	1.15
U1000.10	46.77(39)	1.41	44.11(39)	1.75	43.51(39)	1.60	**39.75**(39)	1.55	39.76(39)	1.56
U2000.05	34.91(13)	5.35	7.89(4)	3.34	8.09(4)	3.21	**4.13**(3)	2.79	4.17(3)	2.88
U2000.10	75.23(47)	4.95	62.68(47)	5.50	54.60(47)	5.25	**47.83**(47)	4.64	48.32(47)	4.49
U5000.05	129.11(74)	26.02	21.24(12)	13.02	22.50(10)	11.76	**4.99**(4)	9.55	5.14(4)	9.60
U5000.10	255.18(109)	28.53	125.97(84)	22.07	147.07(88)	22.55	82.47(73)	17.60	**81.97**(73)	17.35
pcart.352	4.75(1)	0.09	2.22(1)	0.06	2.47(1)	0.07	**1.82**(1)	0.07	2.08(1)	0.08
pcart.702	11.05(5)	0.24	2.55(1)	0.15	2.62(1)	0.17	**2.00**(1)	0.19	2.35(1)	0.19
pcart.1052	17.29(5)	0.48	2.64(1)	0.26	2.68(1)	0.29	**2.12**(1)	0.30	2.52(1)	0.31
pcart.5252	91.77(59)	8.21	2.75(1)	2.39	2.84(1)	2.62	**2.35**(1)	2.62	2.70(1)	2.72
prcart.134	1.38(1)	0.02	**1.01**(1)	0.02	1.09(1)	0.02	**1.01**(1)	0.02	1.02(1)	0.02
prcart.554	3.79(1)	0.11	1.66(1)	0.08	2.05(1)	0.08	**1.34**(1)	0.10	1.56(1)	0.10
prcart.994	5.98(1)	0.23	1.96(1)	0.16	2.37(1)	0.18	**1.50**(1)	0.19	1.87(1)	0.20
prcart.5114	20.65(11)	3.24	2.48(1)	1.83	2.60(1)	2.22	**1.96**(1)	2.28	2.31(1)	2.29

∗ Data from 1,000 runs.

[18] H. Nagamochi and T. Ibaraki. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM Journal of Discrete Mathematics*, 5(1):54–66, Feb 1992.

[19] A. G. Steenbeek, E. Marchiori, and A. E. Eiben. Finding balanced graph bi-partitions using a hybrid genetic algorithm. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 90–95, 1998.