Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 238529, 10 pages
http://dx.doi.org/10.1155/2015/238529

Research Article

Moving Clusters within a Memetic Algorithm for Graph Partitioning

Inwook Hwang,[1] Yong-Hyuk Kim,[2] and Yourim Yoon[3]

[1]*Technical Laboratory, Atto Research, 225-18 Pangyoyeok-ro, Bundang-gu, Seongnam-si, Gyeonggi-do 463-400, Republic of Korea*
[2]*Department of Computer Science & Engineering, Kwangwoon University, 20 Kwangwoon-ro, Nowon-gu, Seoul 139-701, Republic of Korea*
[3]*Department of Computer Engineering, College of Information Technology, Gachon University, 1342 Seongnamdaero, Sujeong-gu, Seongnam-si, Gyeonggi-do 461-701, Republic of Korea*

Correspondence should be addressed to Yourim Yoon; yryoon@gachon.ac.kr

Received 23 September 2014; Accepted 7 January 2015

Academic Editor: John Gunnar Carlsson

Most memetic algorithms (MAs) for graph partitioning reduce the cut size of partitions using iterative improvement. But this local process considers one vertex at a time and fails to move clusters between subsets when the movement of any single vertex increases cut size, even though moving the whole cluster would reduce it. A new heuristic identifies clusters from the population of locally optimized random partitions that must anyway be created to seed the MA, and as the MA runs it makes beneficial cluster moves. Results on standard benchmark graphs show significant reductions in cut size, in some cases improving on the best result in the literature.

1. Introduction

Consider an unweighted undirected graph $G = (V, E)$, where V is a set of n vertices, and E is the set of edges that connect them. A *k-way partition* $\{P_1, P_2, \ldots, P_k\}$ of the graph G is a partitioning of the vertex set V into k disjoint subsets. A partition is said to be *balanced* if the difference in size between the largest and the smallest subset is at most 1, that is, for all $1 \leq i, j \leq k$, $||P_i| - |P_j|| \leq 1$. The *cut size* of a partition is defined to be the number of edges connecting vertices in different subsets of the partition. The *k-way graph partitioning* problem is the problem of finding a balanced k-way partition with the minimum cut size. If $k = 2$, it can be called bipartitioning and if $k > 2$, multiway partitioning. These problems arise in applications such as sparse matrix factorization, network partitioning, layout and floor planning, circuit placement, social network analysis, and software-defined networking [1, 2].

For general graphs, partitioning is known to be NP-hard [3]. Bui and Jones [4] have shown that even finding good approximate solutions is also NP-hard.

Therefore, many heuristic methods have been proposed: some of them work well, but they cannot of course guarantee optimality. The simplest heuristic is iterative improvement partitioning (IIP) [5, 6], exemplified by the Kernighan-Lin (KL) [7] and the Fiduccia-Mattheyses (FM) algorithms [8], but these algorithms only produce solutions which are approximations to *local* optima; however, this limitation can be overcome by hybridizing them with metaheuristics, such as simulated annealing [9], genetic algorithms (GAs) [10], tabu search [11, 12], or ant colony optimization [13]. Recently, a number of techniques based on GAs have achieved notable results for $k = 2$ [14–19] and $k > 2$ [20–26]. Kim et al. [27] have surveyed this work.

The use of IIP for local optimization of partitioning produced by a GA becomes less effective as the graph becomes larger. We will show that this is because IIP often fails to move densely interconnected subgraphs, called *clusters*, between partitions, and hence fails to find partitions with small cut sizes.

The goal of the work reported in this paper is to overcome the barriers to effective search which are presented

by clusters, by modifying the GA so that it contributes to move clusters appropriately. We present a memetic algorithm (MA), which is a GA combined with local optimization, in which a heuristic finds clusters in some of the positions in each generation, by examining population of individuals, each of which represents a position, rather than trying to identify them directly from a single graph. It moves some of these clusters. This heuristic supplements the well-known ability of MAs to provide attractive initial points for local optimization. Experimental results show that this approach can substantially improve the performance of an MA. The contributions of this work are summarized as follows.

(i) We provide a detailed explanation of the difficulty of moving clusters in graph partitioning and provide experimental results quantifying the impact of clusters on the search for partitions with a small cut size.

(ii) We present a heuristic for detecting and moving clusters, which is based on a new, population-based, measure of the distance between vertices called *genic distance*.

(iii) We show that this heuristic substantially improves the ability of an MA to find good partitions.

The remainder of this paper is organized as follows. In Section 2 we briefly introduce IIP algorithms and the test graphs used in our experiments. In Section 3 we investigate the difficulty of moving clusters in graph partitioning. In Section 4 we describe our new cluster-handling heuristic, and an MA that uses this heuristic is described in Section 5. In Section 6 we present experimental results, and draw conclusions in Section 7.

2. Preliminaries

2.1. Iterative Improvement Algorithms in Bipartitioning. Iterative improvement partitioning starts with a random partition. This is refined in a series of passes. At the start of each pass, all the vertices are free to move between subsets. IIP selects vertices and moves them, but each vertex is only moved once during a pass. At the end of the pass, the best partition found during the pass is identified and used as the input to the next pass. Passes continue until there is no further improvement.

There are a number of IIP algorithms, of which KL [7] is often considered to be the first reasonable heuristic for bipartitioning. In KL, the movement of vertices during a pass is restricted to the swapping of a pair of vertices between subsets.

Let $\{A, B\}$ be a partition of V into two subsets A and B. We define the gain g_v associated with a vertex v to be the reduction in cut size obtained by moving v to the other subset. By extension, the gain $g(a, b)$ obtained by swapping vertices $a \in A$ and $b \in B$ can be expressed as follows:

$$g(a, b) = g_a + g_b - 2\delta(a, b), \tag{1}$$

where

$$\delta(a, b) = \begin{cases} 1, & \text{if } (a, b) \in E, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

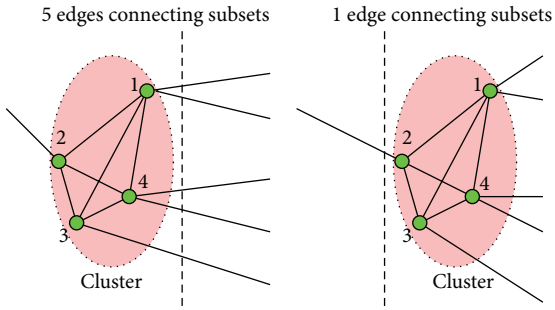
KL selects the pair (a, b) with the highest value of $g(a, b)$ and effects the exchange. The vertices a and b are not considered again during the current pass. A sequence of pairs $(a_1, b_1), (a_2, b_2), \ldots, (a_{n/2-1}, b_{n/2-1})$ are selected in this way. The algorithm chooses l that maximizes $\sum_{i=1}^{l} g(a_i, b_i)$ and exchanges $\{a_1, a_2, \ldots, a_l\}$ and $\{b_1, b_2, \ldots, b_l\}$. KL performs further passes until no improvement is possible.

FM is another widely used IIP algorithm, which is similar to KL, except that it only moves one vertex at a time. This makes FM faster than KL, with little loss in partition quality. Several variants of KL and FM exist [15, 28, 29].

2.2. Local Optimization Algorithms for Multiway Partitioning. There are three main schemes for multiway partitioning, which are developments of the recursive, pair-wise, and direct approaches [21] to bipartitioning. The recursive KL algorithm bisects the graph recursively until there are k subsets. The pair-wise KL [7] starts with an arbitrary k-way partition. It picks two subsets at a time from the k subsets and performs bipartitioning to reduce the cut size between those pairs. Sanchis [30] extended the FM algorithm to multiway partitioning and showed that the direct method performed better than recursion. The extended algorithm considers moving each vertex from its current subset to every other subset. To perform local optimization in the proposed MA for multiway partitioning, we use a variant of this algorithm, called EFM (extended FM) [21]. The time complexity of EFM is $O(k|E|)$.

2.3. Local Search in Memetic Algorithms. It is already clear that combining a GA with local optimization algorithms is an effective approach to the graph partitioning problem [15]. Some authors have explored fast but weak local optimization algorithms. For example [31, 32], 2-opt was used to relocate border vertices, which are those with edges that connect to vertices in other subsets. Bui and Moon [10] obtained better results with KL by allowing only a single pass, while restricting the number of vertices to be swapped. Conversely, other authors have reported notable improvements by enhancing local optimization algorithms. For bipartitioning, Kim and Moon [15] suggested a new KL-based local optimization algorithm, formulated using a new type of gain, called *lock gain*, which only takes into account the edges that connect a vertex to the vertices that have already been moved. Combined with a GA, this algorithm obtained impressive results on most benchmark graphs. For multiway partitioning, the combination of MAs with specialized local optimization algorithms showed good results [20, 21]. Steenbeek et al. [18] proposed what they called a cluster enhancement heuristic, which they combined with an MA, and reported successful results. Their MA uses a vertex swap heuristic to identify clusters. The MA only handles the moving of clusters between subsets.

2.4. Test Graphs. We tested our MA on Johnson's benchmark graphs [9], which have been widely used in other studies [10, 11, 14–17, 20, 21, 23, 33–36]. They are composed of 8 random



(a) Before moving a cluster (b) After moving a cluster

Figure 1: An example of cluster moving, in which the cut size of a partition is reduced by 4.

graphs G$n.d$ and 8 random geometric graphs U$n.d$. The two different classes of graphs are briefly described below.

(i) G$n.d$: a random graph on n vertices, with an independent probability p that any two vertices are connected by an edge. The probability p is biased so that the expected vertex degree, $p(n-1)$, is d.

(ii) U$n.d$: a random geometric graph on n vertices that lie in the unit square and whose coordinates are chosen uniformly from the unit interval. Every pair of vertices separated by a distance of t or less is connected by an edge. The expected degree of a vertex is $n\pi t^2$.

3. Difficulty of Moving Clusters

Suppose that the cluster shown in Figure 1(a) is involved in a bipartitioning problem. The four vertices in this cluster are fully interconnected, and they all belong to the same subset. Moving this cluster to the other subset, across the dotted line in Figure 1(b) will reduce the cut size of the partition by 4. However, there is no motivation to move any single vertex, because they all have negative gain: the gains of v_1, v_2, v_3, and v_4 are −1, −4, −2, and −1, respectively. This example illustrates how IIP algorithms may miss a significant reduction in cut size that could be achieved by moving several vertices together.

The baleful effect of clusters on local search algorithms trying to solve the graph partitioning problem motivated this study. Kim [37, 38] indicated that graph partitioning is hard primarily due to the difficulty of moving clusters. Dutt and Deng [39, 40] have also observed that an IIP method applied to circuit partitioning can fail because of the difficulty of dealing with clusters that straddle subsets.

3.1. Experimental Support. We designed experiments to quantify the effect of clusters on IIP algorithms, represented by the KL algorithm. Using the cluster detection method to be described in Section 4.1, we find clusters in the graph and select one randomly. We then take a locally optimum bipartition s obtained by KL and move the selected cluster to the other subset, creating a perturbed partition t_{cluster}. Applying KL to t_{cluster}, we obtain a new local optimum u_{cluster}.

Table 1: Probability that KL fails to return vertices moved from one subset of a partition to the other, when the vertices are in cluster (P_{cluster}) or chosen at random (P_{random}), over 1,000 runs.

Graph	P_{cluster} (%)	P_{random} (%)	Average number of vertices moved (q)
G500.2.5	16.90	2.50	6.41
G500.05	5.70	1.30	5.50
G500.10	1.90	0.30	5.05
G500.20	0.50	0.10	4.78
G1000.2.5	12.80	1.90	5.89
G1000.05	2.90	0.80	5.61
G1000.10	0.60	0.20	4.67
G1000.20	0.30	0.00	4.91
U500.05	39.10	1.00	7.41
U500.10	26.20	0.50	12.05
U500.20	8.60	0.20	14.10
U500.40	10.40	0.40	49.36
U1000.05	37.90	0.80	7.81
U1000.10	28.50	0.60	12.47
U1000.20	11.60	0.10	22.19
U1000.40	9.10	0.40	35.88

Assuming that q, the number of vertices in the clusters, is small, u_{cluster} can be expected to match s if KL is successful in moving the cluster back. However, if KL fails to return the perturbed cluster to its original subset, the cut size of the partition may increase. Repeating this experiment, we derive P_{cluster}, as an approximation of the probability that the cut size of u_{cluster} is larger than that of s.

For comparison, we perturbed q vertices randomly selected within a locally optimized partition, by moving them to the other subset. We call this partition t_{random}. We apply the KL algorithm to t_{random} and then obtain a new locally optimum partition u_{random}. Repeating this experiment, we derive P_{random}, as an approximation of the probability that the cut size of u_{random} is larger than that of s.

Table 1 shows the values of P_{cluster} and P_{random} for 16 benchmark graphs. We see that P_{cluster} is always larger than P_{random}, as we would expect. We notice that the gap between P_{cluster} and P_{random} is larger on the geometric graphs (U$n.d$) than on the random graphs (G$n.d$).

4. Cluster-Handling Heuristic

Cluster analysis is a well-known problem for which plenty of algorithms exist, many of which require a lot of computation. The insight that motivates our heuristic is that the application of a local optimization process, such as IIP, to a randomly partitioned graph creates a modified partition in which clusters tend to be wholly allocated to one subset or another (and are then difficult to move, as we have already observed). A single partition of this sort is of little help in identifying clusters, because the clusters are not separated at all within each subset; but if we create many random partitions and optimize

Table 2: Truth table for Fact 1.

p	q	$p \rightarrow q$	$I(p)$	$I(q)$
True	True	True	1	1
True	False	False	1	0
False	True	True	0	1
False	False	True	0	0

Table 3: Truth table for Fact 2.

p	q	$I(p)$	$I(q)$	$p \vee q$	$I(p \vee q)$	$I(p) + I(q)$
True	True	1	1	True	1	2
True	False	1	0	True	1	1
False	True	0	1	True	1	1
False	False	0	0	False	0	0

them, we can reasonably infer that vertices that find themselves in the same subset in most of these partitions belong to the same cluster. We can make this inference in a structured way using the "genic distance" metric that we will introduce. This approach to cluster analysis may seem indirect, but it is efficient in the context of an evolutionary approach to graph partitioning, because the set of partitions required for finding clusters using genic distance is also the population which we must create to be evolved by our MA.

One way of dealing with clusters is to devise a local optimization heuristic that can identify clusters [18, 19, 38]. However, this prevents us from building on previous studies of IIP algorithms.

Our approach is to add an additional heuristic to our MA, which finds and moves clusters. The heuristic identifies clusters in the population of partitions which have already been optimized locally. It selects clusters with higher gains and moves them. IIP local optimization is then applied again.

4.1. Cluster Detection. Let $I(\cdot)$ be an *indicator function*, that is, $I(\text{true}) = 1$ and $I(\text{false}) = 0$. Then, we can trivially establish the following.

Fact 1. If $p \rightarrow q$ is true, then $I(p) \leq I(q)$.

Proof. From Table 2. □

Fact 2. $I(p \vee q) \leq I(p) + I(q)$.

Proof. From Table 3. □

We now define a metric called *genic distance*, which measure the extent to which two vertices connected by an edge can be considered to belong to the same cluster. We denote the genic distance of an edge $\{v_i, v_j\}$ within a population $\{p_1, p_2, \ldots, p_m\}$ of locally optimized partitions as $d_g(v_i, v_j)$, which can be expressed as follows:

$$d_g\left(v_i, v_j\right) := \sum_{l=1}^{m} I\left(p_l\left(g_i\right) \neq p_l\left(g_j\right)\right), \tag{3}$$

where g_i and g_j are the genes corresponding to v_i and v_j, respectively. The value of gene g_i (i.e., the partition number that vertex v_i belongs to) in the lth individual is $p_l(g_i)$. For convenience, we assume that each vertex has an edge that connects it to itself, so that $\{v, v\} \in E$ for each vertex v. Then, the following proposition holds.

Proposition 1. *For each population, d_g becomes a pseudometric on V.*

Proof. Since $I(\cdot) \geq 0$, $d_g(v_i, v_j) = \sum_{l=1}^{m} I(p_l(g_i) \neq p_l(g_j)) \geq 0$ for each $\{v_i, v_j\} \in E$. It is enough to show the following three conditions.

(i) $d_g(v_i, v_i) = 0$:

$$\begin{aligned} d_g\left(v_i, v_i\right) &= \sum_{l=1}^{m} I\left(p_l\left(g_i\right) \neq p_l\left(g_i\right)\right) \\ &= \sum_{l=1}^{m} I\left(\text{false}\right) \\ &= 0. \end{aligned} \tag{4}$$

(ii) *Symmetry*. Let $\{v_i, v_j\}$ be in E:

$$\begin{aligned} d_g\left(v_i, v_j\right) &= \sum_{l=1}^{m} I\left(p_l\left(g_i\right) \neq p_l\left(g_j\right)\right) \\ &= \sum_{l=1}^{m} I\left(p_l\left(g_j\right) \neq p_l\left(g_i\right)\right) \\ &= d_g\left(v_j, v_i\right). \end{aligned} \tag{5}$$

(iii) *Triangle Inequality*. Consider each group of three edges $\{v_i, v_j\}, \{v_i, v_k\}, \{v_k, v_j\} \in E$. If $p_l(g_i) = p_l(g_k)$ and $p_l(g_k) = p_l(g_j)$, then $p_l(g_i) = p_l(g_j)$ for each l. By contraposition, if $p_l(g_i) \neq p_l(g_j)$, then $p_l(g_i) \neq p_l(g_k)$ or $p_l(g_k) \neq p_l(g_j)$.

For each l,

$$\begin{aligned} &I\left(p_l\left(g_i\right) \neq p_l\left(g_j\right)\right) \\ &\leq I\left(p_l\left(g_i\right) \neq p_l\left(g_k\right) \vee p_l\left(g_k\right) \neq p_l\left(g_j\right)\right) \quad (\because \text{Fact 1}) \\ &\leq I\left(p_l\left(g_i\right) \neq p_l\left(g_k\right)\right) + I\left(p_l\left(g_k\right) \neq p_l\left(g_j\right)\right) \quad (\because \text{Fact 2}). \end{aligned} \tag{6}$$

By summing the above inequalities for all l,

$$\begin{aligned} &\sum_{l=1}^{m} I\left(p_l\left(g_i\right) \neq p_l\left(g_j\right)\right) \\ &\leq \sum_{l=1}^{m} \left(I\left(p_l\left(g_i\right) \neq p_l\left(g_k\right)\right) + I\left(p_l\left(g_k\right) \neq p_l\left(g_j\right)\right)\right) \\ &= \sum_{l=1}^{m} I\left(p_l\left(g_i\right) \neq p_l\left(g_k\right)\right) + \sum_{l=1}^{m} I\left(p_l\left(g_k\right) \neq p_l\left(g_j\right)\right). \end{aligned} \tag{7}$$

Therefore we have

$$d_g(v_i, v_j) \le d_g(v_i, v_k) + d_g(v_k, v_j). \quad (8)$$

That is, d_g satisfies the triangle inequality. □

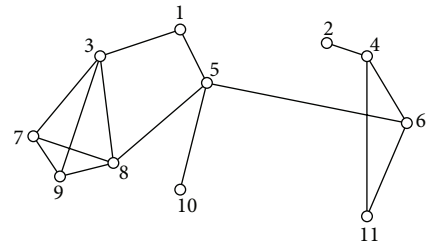
Proposition 1 suggests that the measure d_g is reasonable. A pseudometric space is a generalization of a metric space, in which points need not be distinguishable; thus it is possible that $d_g(v_i, v_j) = 0$ for some edge $\{v_i, v_j\}$, with distinct vertices $v_i \neq v_j$.

Our heuristic detects clusters by collecting a number of local optima and computes genic distances for all the edges in each graph. This takes $O(m|E|)$ time, but this cost is negligible since this computation is a preprocess performed before the MA runs. The heuristic temporarily eliminates edges with genic distances that are greater than a threshold value θ. We set θ to be the smallest value that satisfies

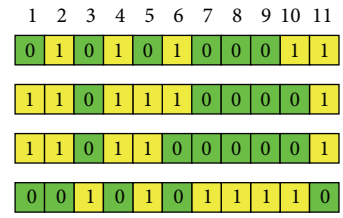
$$\left|\left\{e \in E : d_g(e) \le \theta\right\}\right| \le 0.1\,|V|. \quad (9)$$

Each remaining connected component containing more than three vertices is considered to be a *cluster*.

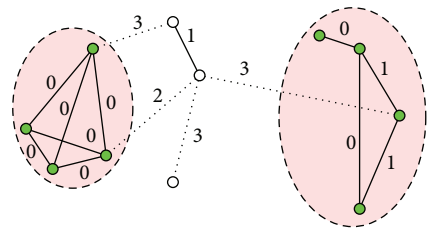
Figure 2 shows how our heuristic detects clusters. Figure 2(a) shows an example graph with 11 vertices and 15 edges. Four individuals, corresponding to locally optimized partitions, from the population are shown in Figure 2(b). In Figure 2(c), each edge is labeled with its genic distance. If the threshold value of genic distance is 1, then the edges with larger genic distances, indicated by dotted line, are eliminated. Then four connected components remain: $\{v_3, v_7, v_8, v_9\}$, $\{v_2, v_4, v_6, v_{11}\}$, $\{v_1, v_5\}$, and $\{v_{10}\}$. The last two of these connected components are considered too small to be clusters. Thus clusters $\{v_3, v_7, v_8, v_9\}$ and $\{v_2, v_4, v_6, v_{11}\}$, shaded in Figure 2(c), remain as candidates for moving.



(a) Example graph



(b) Four locally optimized partitions in the population (subsets 0 and 1)



(c) Two detected clusters

Figure 2: An example of cluster detection.

4.2. Cluster-Moving Scheme. Our heuristic improves the offspring of each generation after crossover by moving the clusters that were detected using the technique described in the previous subsection. To select the clusters to be moved and their target subsets, we introduce a measure called *cluster gain*, such that $\mathrm{cg}(x, a)$ is the reduction in the cut size of the partition when all the vertices in cluster x are moved to the subset a. For example, moving the cluster in Figure 1(a) to the other subset in the partition is associated with a cluster gain of 2.

This cluster-moving scheme, described in Algorithm 1, is applied to each individual generated by crossover, which is a partition that may be unbalanced. However, cut size and cluster gain are well defined on unbalanced partitions. Our scheme does not consider moving every cluster in every partition, because we found that making all clusters movable causes the premature convergence of the MA. Thus, our heuristic selects N clusters at random as candidates for moving. In our experiments, we set N to 5. We compute the cluster gain that results from moving each candidate cluster to each of the other $k - 1$ subsets. The candidate cluster x and destination subset a with the highest cluster gain are selected. Assume that $\mathrm{cg}(x, a)$ is positive, all the vertices in cluster x are moved to subset a, and cluster x is removed from the set of candidate clusters. This process is repeated until no candidates remain, or no move yields a positive cluster gain. No attempt is made to balance the partition during cluster-moving; this is performed later.

5. Memetic Search Framework

CMPA (cluster-moving memetic partitioning algorithm) is a memetic algorithm that we have designed for graph partitioning. In this MA, an individual is a k-way partition. Each gene in an individual corresponds to a vertex and has a value between 0 and $k - 1$, which indicates the subset to which the vertex belongs; that is, the ith gene $g_i = j \Leftrightarrow v_i \in P_{j+1}$. It is a steady-state MA, meaning that there is only one offspring from population in each generation. Crossover is followed by a cluster-moving step and then local optimization.

Algorithm 2 shows the processes that make up CMPA, which we now describe in detail.

(i) *Initialization*. When the MA starts, m individuals (i.e., partitions) are created at random. Then each individual is improved by local optimization. We set m to be 30 for bipartitioning and 50 for multiway partitioning (with $k = 8$ or $k = 32$).

```
Select clusters: S ← N clusters selected at random;
do {
   Calculate cg(x, a) for all x ∈ S, 1 ≤ a ≤ k;
   cg* ← max_{x,a} cg(x, a);
   (x̄, ā) ← arg max_{x,a} cg(x, a);
   if cg* > 0 then {
      Move cluster x̄ to partition ā;
      S ← S \ {x̄};
   }
} until (S = ∅ or cg* ≤ 0)
```

Algorithm 1: Our cluster-moving scheme.

```
Create initial population of fixed size;
Apply local optimization to each member of population;
Calculate genic distance from population;
Find clusters and store their information;
do {
   Select parent_1 and parent_2 from population;
   Normalization(parent_1, parent_2);
   offspring ← Crossover(parent_1, parent_2);
   Cluster-moving(offspring);
   Local-optimization(offspring);
   Replace(population, offspring);
} until (stopping condition);
return the best solution;
```

Algorithm 2: The process in CMPA.

(ii) *Selection*. We used the roulette-wheel-based proportional selection. The probability that the best individual is chosen was set to four times the probability that the worst is chosen. The fitness value f_i of the ith individual is expressed as $(c_w - c_i) + (c_w - c_b)/3$, where c_b, c_w, and c_i are the cut sizes of the partitions corresponding to the best, the worst, and the ith individual, respectively.

(iii) *Normalization*. Laszewski [31] first used normalization to improve the performance of GA and its variants have been suggested in [23, 26, 41, 42]. The parent individuals are normalized before crossover following Laszewski [31, 33]. The subset of one parent which shares the largest number of vertices with subsets of the other parent is selected, and that subset is numbered 0. This process is repeated, incrementing the index, until all subsets are numbered.

(iv) *Crossover and Cluster Moving*. We used a standard five-point crossover. After crossover, the cluster-handling heuristic described in Section 4.2 is applied to the individual. At this point, individuals usually correspond to unbalanced partitions. We select a random location in the individual and adjust the values of the genes, which are the subsets to which the corresponding vertices belong, to the right of this location (in a typical circular string) until the partition is balanced. This is effectively a mutation effect, and no further mutation was introduced.

(v) *Local Optimization*. KL [7] was used for the bisection problems and EFM (extended FM) [21] was used for multiway partitioning ($k = 8$ or $k = 32$).

(vi) *Replacement*. We used a replacement scheme due to Bui and Moon [10]. If an offspring is better than its closer parent, the MA replaces that parent. If it is better than its other parent, then that parent is replaced. Otherwise it replaces the worst individual in the population.

(vii) *Stopping Condition*. This is based on consecutive failures to replace an individual's parents. Termination is triggered by consecutive failures: 30 in bipartitioning and 50 in multiway partitioning.

6. Experimental Results

We conducted experiments on 2-way, 8-way, and 32-way partitioning. Table 4 shows the performance of MA combined with KL (denoted KL-MA) and CMPA on bipartitioning. Table 5 shows the performance of the genetic extended FM algorithm (GEFM) [21], one of the most effective approaches, and CMPA on 8-way partitioning, and Table 6 gives the results for 32-way partitioning. CMPA uses a cluster-handling

Table 4: Comparison of KL-MA and CMPA on bipartitioning.

Graph	Best known[1]	KL-MA			CMPA		
		Best[2]	Ave[2]	CPU[3] (Gen[4])	Best[2]	Ave[2]	CPU[3] (Gen[4])
G500.2.5	49	49	51.34	0.32 (595)	49	**51.22**	0.34 (558)
G500.05	218	218	218.65	0.59 (700)	218	**218.45**	0.64 (682)
G500.10	626	626	627.55	1.02 (681)	626	**627.46**	1.10 (672)
G500.20	1744	1744	1745.83	1.93 (694)	1744	**1745.62**	2.09 (687)
G1000.2.5	93	93	97.26	0.96 (745)	93	**97.08**	1.04 (722)
G1000.05	445	445	451.50	2.07 (884)	445	**450.92**	2.22 (860)
G1000.10	1362	1362	1366.68	4.07 (969)	1362	**1366.18**	4.45 (977)
G1000.20	3382	3382	3385.61	7.42 (971)	3382	**3385.26**	8.13 (987)
U500.05	2	2	4.96	0.55 (793)	2	**3.94**	0.66 (803)
U500.10	26	26	26.31	0.45 (459)	26	**26.04**	0.57 (465)
U500.20	178	178	178.06	0.42 (300)	178	**178.00**	0.53 (271)
U500.40	412	412	**412.00**	0.34 (190)	412	**412.00**	0.37 (102)
U1000.05	1	1	11.67	1.71 (1077)	1	**8.81**	2.12 (1068)
U1000.10	39	39	46.27	1.51 (645)	39	**44.15**	1.95 (653)
U1000.20	222	222	222.19	1.13 (347)	222	**222.07**	1.51 (327)
U1000.40	737	737	**737.00**	1.20 (243)	737	**737.00**	1.77 (205)

[1]Best result from the literature.
[2]Best and average results from 1,000 runs.
[3]CPU seconds on a Pentium IV 2.8 GHz.
[4]Average number of generations over 1,000 runs.

Table 5: Comparison of GEFM and CMPA on 8-way partitioning.

Graph	GEFM [21]			CMPA		
	Best[1]	Ave[1]	CPU[2] (Gen[3])	Best[1]	Ave[1]	CPU[2] (Gen[3])
G500.2.5	111	115.21	42.03 (2090)	111	**114.51**	40.25 (2091)
G500.05	465	468.16	85.65 (2314)	465	**467.63**	72.96 (2239)
G500.10	1254	1259.53	144.54 (2347)	1254	**1258.47**	135.04 (2297)
G500.20	3348	3354.80	293.18 (2437)	3348	**3353.71**	285.09 (2317)
G1000.2.5	212	**216.30**	146.76 (3043)	211	216.57	148.42 (3023)
G1000.05	930	**938.11**	247.20 (3148)	931	939.26	234.67 (3069)
G1000.10	2714	2726.33	408.96 (3146)	2711	**2725.52**	416.98 (3083)
G1000.20	6525	**6536.55**	791.29 (3303)	6520	6538.35	781.67 (3141)
U500.05	16	18.80	70.87 (2629)	16	**17.22**	55.89 (2137)
U500.10	143	145.12	98.75 (2277)	143	**144.26**	90.50 (1984)
U500.20	612	614.57	121.22 (1690)	611	**612.93**	97.53 (1483)
U500.40	1867	1872.60	157.94 (1396)	1867	**1871.73**	129.62 (1204)
U1000.05	21	33.66	180.35 (3638)	20	**28.53**	209.53 (3172)
U1000.10	176	183.70	243.08 (3105)	176	**182.30**	257.02 (2857)
U1000.20	812	814.05	231.08 (1824)	812	**813.00**	232.93 (1756)
U1000.40	2562	2563.05	249.46 (1353)	2562	**2562.85**	273.51 (1238)

[1]Best and average results from 100 runs.
[2]CPU seconds on a Pentium IV 2.8 GHz.
[3]Average number of generations over 100 runs.

heuristic but KL-MA and GEFM do not. We performed 1,000 runs for bipartitioning and 100 runs for 8-way and 32-way partitioning. All the programs were written in the C language and compiled using GNU's *gcc* compiler. It was run on a Pentium IV 2.8 GHz computer with Linux. In the tables, the bold-faced numbers indicate the lower of the average cut sizes obtained by the two algorithms. CMPA outperformed KL-MA and GEFM on most graphs, with comparable running times.

CMPA underperformed on some random graphs, which may be due to the weak cluster structures in these graphs. CMPA's average performance was better on all the geometric

Table 6: Comparison of GEFM and CMPA on 32-way partitioning.

Graph	Best known[1]	GEFM [21]			CMPA		
		Best[2]	Ave[2]	CPU[3] (Gen[4])	Best[2]	Ave[2]	CPU[3] (Gen[4])
G500.2.5	177	177	181.69	78.98 (1491)	177	**180.82**	74.30 (1382)
G500.05	623	624	630.47	162.26 (2197)	623	**629.92**	159.18 (2035)
G500.10	1573	1574	1581.94	300.83 (2364)	1572*	**1579.72**	355.73 (2332)
G500.20	4034	4037	4045.06	681.53 (2565)	4035	**4042.81**	673.19 (2479)
G1000.2.5	312	313	320.99	335.79 (2798)	313	**320.25**	356.44 (2648)
G1000.05	1217	1208	1218.72	698.67 (3133)	1205*	**1216.89**	738.06 (3203)
G1000.10	3360	3353*	3367.31	1171.48 (3294)	3355	**3366.08**	1323.59 (3267)
G1000.20	7818	7817	**7829.81**	2012.78 (3475)	7815*	7830.48	2363.69 (3342)
U500.05	109	112	116.39	138.29 (1640)	109	**113.18**	121.15 (1595)
U500.10	523	531	537.75	225.52 (1512)	526	**531.84**	166.26 (1222)
U500.20	1825	1831	1841.39	285.07 (1344)	1823*	**1831.88**	299.47 (1197)
U500.40	5328	5364	5380.01	561.74 (1381)	5348	**5369.83**	523.15 (1331)
U1000.05	117	118	126.02	451.03 (3137)	115*	**123.49**	464.35 (2967)
U1000.10	577	576	583.26	599.01 (2599)	571*	**578.16**	743.50 (2603)
U1000.20	2367	2375	2396.33	798.77 (1963)	2373	**2388.93**	930.43 (1848)
U1000.40	7329	7399	7417.49	1421.78 (1634)	7382	**7407.18**	1493.10 (1573)

[1]Best known values [20–22, 24, 33].
[2]Best and average results from 100 runs. Asterisked numbers are new best values.
[3]CPU seconds on a Pentium IV 2.8 GHz.
[4]Average number of generations from 100 runs.

graphs. This suggests that effective cluster handling is more important on the geometric graphs, as we suggested in Section 3.1.

The local optimization algorithm is much more expensive than the cluster-handling heuristic; thus, CMPA does not take much longer to run than KL-MA or GEFM. On average, CMPA required 14.14% more time than KL-MA for the bipartitioning problems, and 2.02% more than GEFM in 32-way partitioning. CMPA ran 5.52% faster in 8-way partitioning.

7. Concluding Remarks

We devised a cluster-moving heuristic and combined it with a memetic algorithm (MA) for graph partitioning. Experiments on 2-way, 8-way, and 32-way partitioning showed that this heuristic significantly improved the performance of the MA, especially on the 32-way partitioning.

The method of moving clusters that we have introduced addresses a significant weakness in standard IIP algorithms. The idea of adding an operation to complement a local optimization algorithm might be used to address other deficiencies in MAs.

Our method of cluster detection is based on a measure that we call genic distance, which is designed to reflect the extent to which two vertices connected by an edge belong to the same cluster. Instead of computing genic distances once in an initialization phase, an MA could recompute these distances as it progresses: this might improve the accuracy of cluster detection, at some cost in time. We believe that genic distance might also be useful in solving other problems involving clusters, such as the maximum clique problem.

Table 7: Real-world benchmark graphs.

Graph	Number of vertices	Number of edges
nasa4704	4704	50026
bcspwr09	1723	2394
bcsstk13	2003	40940
DEBR12	4096	8189

Appendix

Results on Real-World Graphs

We also tested on four real-world benchmark graphs used in [11, 15, 43]. The sizes of the graphs are given in Table 7. We conducted experiments on 2-way and 8-way partitioning. We performed 100 runs for bipartitioning and 50 runs for 8-way partitioning. It was run on an Intel Core i7 3.6 GHz computer with Linux. Table 8 compares CMPA with KL-MA on bipartitioning, and Table 9 compares CMPA with GEFM on 8-way partitioning. In the tables, the bold-faced numbers indicate the lower of the average cut sizes obtained by the two algorithms. Similarly to the results in Section 6, CMPA overall performed better than the others, with comparable running times.

Disclosure

A preliminary version of this paper appeared in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2007 (p. 1520).

Table 8: Comparison of KL-MA and CMPA on bipartitioning.

Graph	Best known[1]	KL-MA			CMPA		
		Best[2]	Ave[2]	CPU[3] (Gen[4])	Best[2]	Ave[2]	CPU[3] (Gen[4])
nasa4704	1292	1292	**1292.00**	4.15 (407)	1292	**1292.00**	3.82 (347)
bcspwr09	9	9	11.31	0.61 (1010)	9	**10.60**	0.81 (1003)
bcsstk13	2355	2355	**2355.00**	1.17 (330)	2355	**2355.00**	1.46 (282)
DEBR12	548	548	548.44	9.91 (765)	548	**548.20**	10.32 (794)

[1]Best result from the literature.
[2]Best and average results from 100 runs.
[3]CPU seconds on Intel Core i7 3.6 GHz.
[4]Average number of generations over 100 runs.

Table 9: Comparison of GEFM and CMPA on 8-way partitioning.

Graph	GEFM [21]			CMPA		
	Best[1]	Ave[1]	CPU[2] (Gen[3])	Best[1]	Ave[1]	CPU[2] (Gen[3])
nasa4704	3898	3903.66	575.69 (2235)	3896	**3902.66**	591.51 (2255)
bcspwr09	53	**57.08**	78.80 (2947)	54	57.74	85.16 (2925)
bcsstk13	8911	8939.10	281.19 (1427)	8919	**8936.14**	301.54 (1476)
DEBR12	1248	1260.16	315.97 (3969)	1248	**1259.90**	353.83 (4002)

[1]Best and average results from 50 runs.
[2]CPU seconds on Intel Core i7 3.6 GHz.
[3]Average number of generations over 50 runs.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported by the ICT R&D program of MSIP/IITP, Korea (10045253, The development of SDN/OpenFlow-Based Enterprise Network Controller Technology) project. The authors would like to thank Professor Byung-Ro Moon and Dr. Yongjoo Song for their valuable suggestions for improving this paper. The authors also thank Jong-Pil Kim for providing the source code of GEFM [21].

References

[1] Y. Kanizo, D. Hay, and I. Keslassy, "Palette: distributing tables in software-defined networks," in *Proceedings of the IEEE International Conference on Computer Communications*, pp. 545–549, April 2013.

[2] P. Wette, M. Draxler, and A. Schwabe, "MaxiNet: distributed emulation of software-defined networks," in *Proceedings of the IFIP Networking Conference*, pp. 1–9, June 2014.

[3] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, 1979.

[4] T. N. Bui and C. Jones, "Finding good approximate vertex and edge partitions is NP-hard," *Information Processing Letters*, vol. 42, no. 3, pp. 153–159, 1992.

[5] Y.-H. Kim, "Improved implementation choices for iterative improvement partitioning algorithms on circuits," in *Proceedings of the International Conference on Computer Design*, pp. 30–34, July 2008.

[6] Y. Yoon and Y.-H. Kim, "New bucket managements in iterative improvement partitioning algorithms," *Applied Mathematics & Information Sciences*, vol. 7, no. 2, pp. 529–532, 2013.

[7] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell System Technical Journal*, vol. 49, no. 2, pp. 291–307, 1970.

[8] C. Fiduccia and R. Mattheyses, "A Linear-time heuristic for improving network partitions," in *Proceedings of the Conference on Design Automation*, pp. 175–181, June 1982.

[9] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon, "Optimization by simulated annealing. An experimental evaluation. Part I. Graph partitioning," *Operations Research*, vol. 37, no. 6, pp. 865–892, 1989.

[10] T. N. Bui and B. R. Moon, "Genetic algorithm and graph partitioning," *IEEE Transactions on Computers*, vol. 45, no. 7, pp. 841–855, 1996.

[11] R. Battiti and A. A. Bertossi, "Greedy, prohibition, and reactive heuristics for graph partitioning," *IEEE Transactions on Computers*, vol. 48, no. 4, pp. 361–385, 1999.

[12] E. Rolland, H. Pirkul, and F. Glover, "Tabu search for graph partitioning," *Annals of Operations Research*, vol. 63, no. 2, pp. 209–232, 1996.

[13] T. N. Bui and L. C. Strite, "An ant system algorithm for graph bisection," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 43–51, July 2002.

[14] I. Hwang, Y.-H. Kim, and B.-R. Moon, "Multi-attractor gene reordering for graph bisection," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1209–1215, July 2006.

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

[16] P. Merz and B. Freisleben, "Fitness landscapes, memetic algorithms, and greedy operators for graph bipartitioning," *Evolutionary Computation*, vol. 8, no. 1, pp. 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*, vol. 31, no. 3, pp. 157–192, 2002.

[18] 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*, pp. 90–95, May 1998.

[19] Y. Yoon and Y.-H. Kim, "Vertex ordering, clustering, and their application to graph partitioning," *Applied Mathematics & Information Sciences*, vol. 8, no. 1, pp. 135–138, 2014.

[20] S.-J. Kang and B.-R. Moon, "A hybrid genetic algorithm for multiway graph partitioning," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 159–166, July 2000.

[21] J.-P. Kim and B.-R. Moon, "A hybrid genetic search for multiway graph partitioning based on direct partitioning," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 408–415, July 2001.

[22] Y.-H. Kim, A. Moraglio, Y. Yoon, and B.-R. Moon, "Geometric crossover for multiway graph partitioning," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1217–1224, July 2006.

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

[24] A. Moraglio, Y.-H. Kim, Y. Yoon, B.-R. Moon, and R. Poli, "Geometric crossover for permutations with repetitions: application to graph partitioning," in *Proceedings of the PPSN Workshop on Evolutionary Algorithms—Bridging Theory and Practice*, September 2006.

[25] A. J. Soper, C. Walshaw, and M. Cross, "A combined evolutionary search and multilevel optimisation approach to graph-partitioning," *Journal of Global Optimization*, vol. 29, no. 2, pp. 225–241, 2004.

[26] Y. Yoon, Y.-H. Kim, A. Moraglio, and B.-R. Moon, "Quotient geometric crossovers and redundant encodings," *Theoretical Computer Science*, vol. 425, pp. 4–16, 2012.

[27] J. Kim, I. Hwang, Y.-H. Kim, and B.-R. Moon, "Genetic approaches for graph partitioning: a survey," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 473–480, July 2011.

[28] S. Dutt and W. Deng, "A probability-based approach to VLSI circuit partitioning," in *Proceedings of the Design Automation Conference*, pp. 100–105, June 1996.

[29] B. Krishnamurthy, "An improved min-cut algorithm for partitioning VLSI networks," *IEEE Transactions on Computers*, vol. 33, no. 5, pp. 438–446, 1984.

[30] L. A. Sanchis, "Multiple-way network partitioning," *IEEE Transactions on Computers*, vol. 38, no. 1, pp. 62–81, 1989.

[31] G. Laszewski, "Intelligent structural operators for the k-way graph partitioning problem," in *Proceedings of the International Conference on Genetic Algorithms*, pp. 45–52, July 1991.

[32] H. Mühlenbein, "Parallel genetic algorithms in combinatorial optimization," in *Computer Science and Operations Research*, O. Balci, R. Sharda, and S. Zenios, Eds., pp. 441–453, Pergamon, Oxford, UK, 1992.

[33] S.-S. Choi and B.-R. Moon, "Normalization in genetic algorithms," in *Proceedings of the Genetic and Evolutionary Computation Conference*, vol. 2723 of *Lecture Notes in Computer Science*, pp. 862–873, 2003.

[34] S.-H. Kim, Y.-H. Kim, and B.-R. Moon, "A hybrid genetic algorithm for the max cut problem," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 416–423, July 2001.

[35] Y.-H. Kim, Y.-K. Kwon, and B.-R. Moon, "Problem-independent schema synthesis for genetic algorithms," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1112–1122, July 2003.

[36] K. Seo, S. Hyun, and Y.-H. Kim, "An edge-set representation based on spanning tree for searching cut space," *IEEE Transactions on Evolutionary Computation*, 2014.

[37] Y.-H. Kim, *Combinatorial optimization based on effective local search, genetic algorithms, and problem space analyses [Ph.D. thesis]*, Seoul National University, Seoul, Republic of Korea, 2005.

[38] Y.-H. Kim, "An enzyme-inspired approach to surmount barriers in graph bisection," in *Proceedings of the International Conference on Computational Science and Its Applications*, vol. 5072 of *Lecture Notes in Computer Science*, pp. 841–851, 2008.

[39] S. Dutt and W. Deng, "VLSI circuit partitioning by cluster-removal using iterative improvement techniques," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 194–200, November 1996.

[40] S. Dutt and W. Deng, "Cluster-aware iterative improvement techniques for partitioning large VLSI circuits," *ACM Transactions on Design Automation of Electronic Systems*, vol. 7, no. 1, pp. 91–121, 2002.

[41] Y. Yoon and Y.-H. Kim, "An efficient genetic algorithm for maximum coverage deployment in wireless sensor networks," *IEEE Transactions on Cybernetics*, vol. 43, no. 5, pp. 1473–1483, 2013.

[42] Y. Yoon, Y.-H. Kim, A. Moraglio, and B.-R. Moon, "A theoretical and empirical study on unbiased boundary-extended crossover for real-valued representation," *Information Sciences*, vol. 183, no. 1, pp. 48–65, 2012.

[43] B. Monien and R. Diekmann, "A local graph partitioning heuristic meeting bisection bounds," in *Proceedings of the SIAM Conference on Parallel Processing for Scientific Computing*, March 1997.