A Hybrid Genetic Algorithm for Multiway Graph Partitioning

So-Jin Kang
Motorola Korea Software Center
Kumha B/D, 41-2, Chungdam-Dong
Kangnam-Gu, Seoul, 135-766 Korea
Sojin.Kang@motorola.com

Byung-Ro Moon
School of Computer Science & Engineering
Seoul National University
Shilim-dong, Kwanak-gu, Seoul, 151-742 Korea
moon@cs.snu.ac.kr

Abstract

A hybrid genetic algorithm for multiway graph partitioning is proposed. The algorithm includes an efficient local optimization heuristic. Starting at an initial solution, the heuristic iteratively improves the solution using *cyclic* movements of vertices. The suggested heuristic performed well in itself and as a local optimization engine in the hybrid genetic algorithm. Combined with the framework of hybrid genetic algorithms, it showed significant performance improvement.

1 Introduction

Given a graph $G = (V, E)$ on n vertices, where V represents the set of vertices and E represents the set of edges, *k-way partitioning* is grouping the vertex set V into k disjoint subsets. In a k-way partition, the total number of edges whose end points belong to different subsets is called the *cut size*. The k-way partitioning problem is to find a k-way partition with minimal cut size. A k-way partition is said to be *strictly balanced* if the difference of cardinalities between the largest subset and the smallest subset is at most one. In graph partitioning, strict or rough balance is required in most cases. If we have a good algorithm for strictly balanced k-way partitioning, roughly balanced k-way partitioning needs only slight modification in most cases. Two-way partition is called *bisection* or *bipartition*. If there is no balance requirement, graph bipartitioning problem can be solved to optimality in polynomial time [14]. Hereafter, unless otherwise noted, graph partitioning means strictly balanced partitioning. Graph partitioning problems arise in diverse areas such as parallel computing, sparse matrix factorization, network partitioning, and VLSI circuit placement.

K-way partitioning is one of the most well-known NP-hard problems [16]. It is known that the graph bisection problem is NP-hard even for bipartite graphs [16] and that even finding good approximation solutions for general graphs or planar graphs is also NP-hard [5][1]. These results make it necessary to sacrifice optimality and look for approximation algorithms within polynomial time budgets. Among such methods are Kernighan-Lin algorithm (KL) [19], simulated annealing (SA) [20] [18], genetic algorithms (GA) [26] [9] [21] [10] [17] [7] [29], tabu search (TS) [25] [12] [3], and large-step Markov chain [23] [15].

KL is a group migration heuristic which starts with an initial bisection and iteratively swaps pairs of vertices to decrease the cut size. Owing to its simplicity and efficiency, it has been one of the most popular algorithms for the last 30 years. For k-way partitioning, a well-known method is the recursive KL which partitions the vertex set to smaller and smaller halves by repeatedly applying KL [28] [7] [27]. Another is the pairwise KL which was suggested in the original paper for KL [19]. Starting at an initial k-way partition, it repeats the process that chooses two out of k subsets and applies KL on the union of the two subsets. They are described in Section 2. There are also various sophisticated methods such as multi-level gain [28], geometric embedding [1], solving by transforming into a max-cut problem [22], relaxed locking [11], and primal-dual algorithm [30].

Recursive KL and pairwise KL are known to be useful in multiway partitioning. However, they have some drawbacks. Recursive KL focuses on the current partitioning without considering the connection inside each subset; subsequent partitionings may have larger cut size than are reasonable. Pairwise KL has a fixed se-

[1]For special classes of graphs such as trees and planar graphs with $O(\log n)$ optimal cut size, exact polynomial time algorithms exist [8].

quence of subset pairs for KL bisections. It thus restricts the movement paths of vertices. It is particularly weak when the number of subsets are considerably large. We devised a heuristic that attempts direct k-way partitioning and allows more freedom to vertex movements. Starting at an initial k-way partition, the heuristic improves on it by *cyclic* movements of vertices.

There have been a number of studies on graph partitioning in GA community, too [21] [13] [10] [17] [2] [6] [7] [29]. But the experimental results were typically not enough, with few exceptions, to verify the effectiveness of GA approaches. Bui and Moon [7] suggested genetic algorithms for graph partitioning with extensive experimental results and showed its superiority to SA and the multi-start KL. However, their experiments were largely for graph bisection. In multiway partitioning, they combined the recursive KL and the pairwise KL with the genetic search and showed simple experimental results for 4-way graph partitioning. As we will be showing later, the frameworks of the recursive KL and the pairwise KL turn out to be not so effective for multiway graph partitioning. As we devised a new multiway partitioning heuristic, we combined it with the genetic search. The effectiveness of the genetic search is examined in Section 5

The remainder of this paper is organized as follows; Section 2 presents KL and two existing KL-based k-way partitioning methods. Section 3 provides our new local heuristic for k-way partitioning. A hybrid GA which is the combination of GA and our local heuristic is followed in Section 4. Lastly we show experimental results in Section 5 and make conclusions in Section 6.

2 Preliminaries

In this section, we describe the KL algorithm and two existing KL-based k-way partitioning algorithms.

2.1 Kernighan-Lin Algorithm (KL)

The Kernighan-Lin algorithm [19] is a local optimization heuristic for graph bisection. Starting with an initial bisection, it continues swapping equal-sized subsets of the bisection to create a new bisection. It proceeds repeatedly until no improvement can be obtained.

Let (A, B) be a bisection of $G = (V, E)$. For vertices $a \in A$ and $b \in B$, denote by $g(a, b)$ the reduction in the cut size of the bisection when the two vertices a and b are exchanged. Denote by g_v the cut-size reduction when vertex v moves to the opposite side. The related equation 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 $(a,\ b)$ that 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. In this way, a sequence of pairs $(a_1,\ b_1), \ldots, (a_{n/2-1},\ b_{n/2-1})$ are selected $(a_i \in A,\ b_i \in B,\ i = 1, \ldots, n/2 - 1)$. The algorithm then 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. This is a pass of KL. With the bisection acquired after the exchange of X and Y, KL repeats the above pass until no more improvement is possible.

2.2 Recursive KL (RKL)

Recursive KL (RKL) is a simple extension of bisection to multiway partition. It hierarchically divides a given graph into 2, 4, 8, …subsets by KL bisection algorithm. This is a representative and popular approach for k-way partitioning [28] [7] [27]. In this approach, one first makes two equal-sized subsets, then divides each of them independently into two subsets, and so on till k subsets are created.

2.3 Pairwise KL (PKL)

A potential problem in RKL is that the first partitioning tries to minimize the cut size between the first two subsets without considering the connections inside each subset. This may cause high cut size in subsequent partitions.

An alternative is to start with k subsets and directly improves on them. This algorithm starts with k equal-sized initial subsets. A pair of subsets among them are selected and KL is applied to reduce the cut size between the two subsets. The process continues until a round of all possible combinations ($\binom{k}{2}$ cases) do not produce any improvement.

3 Cyclic K-way Partitioning Algorithm (CP)

Two k-way partitioning algorithms described in Section 2 use the KL bisection algorithm as an engine. Since RKL tries to minimize the cut size of the current bipartitioning, the cut size of subsequent bipartitionings can be hurt. In case of PKL, if it is desirable that a vertex in subset i eventually moves to subset j, it cannot directly move from subset i to subset j unless they are directly paired; the vertex has to pass through a number of other subsets according to the given sequence of pairs. If the vertex fails to move in at least one of the pairwise KL bisections, the vertex cannot eventually propagate to the subset j. The fixed sequence can be a barrier in the space search. The common drawback of RKL and PKL is that k-way partitioning is accomplished by a sequence of bisections with "local scope." We suggest a direct k-way partitioning with "more global scope."

In KL bipartitioning, there is only one gain g_v for each vertex because a vertex can move only to the opposite side. On the other hand, in this k-way direct partitioning, there exist $k-1$ candidate subsets for each vertex to move; we have $k-1$ gains for each vertex. Denote by $g_v[i]$ the gain by moving vertex v to subset i. For ease of implementation, $g_v[i]$ is set to $-\infty$ if vertex v currently belongs to subset i since it is not an actual moving. We make cycles of vertex movements based on these gains.

The Cyclic Partitioning algorithm (CP) is given in Figure 1. It starts with an initial k-way partition. With the initial solution, a number of passes are performed until the stop condition is satisfied. In a pass (a call of MoveCycles), a sequence of cyclic movements are tried. Figure 2 shows an example sequence of movements. In Figure 2(a), a sequence of vertices are selected from subsets 3, 2, 0, 6, 0, 6, and 5 until the movements make a cycle. Then the gain sums of partial cycles are compared and the cycle with the max gain sum is selected. Figure 2(b) shows the first three partial cycles among six candidate cycles as a result of Figure 2(a). The nomenclature for the algorithm MoveCycles is as follows:

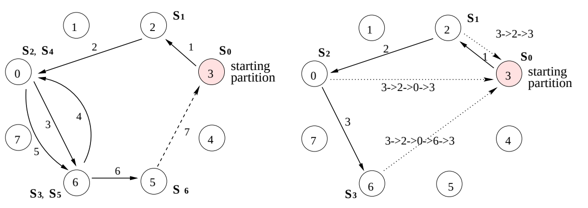
- V_i : subset i, $\bigcup_{1\le i\le k} V_i = V$
- S_i : the subset to which i^{th} vertex on a cycle of movements belongs
- P_v : the subset to which vertex v moves
- $g_v[i]$: the gain of moving vertex v to subset i
- g_v : the maximum among $g_v[i]$'s for all $i = 1,\dots,k$
- G_{ij} : the maximum among $g_v[j]$'s for all v's in V_i
- G_i : the maximum among G_{ij}'s for all $j = 1,\dots,k$
- ξ_i : the gain of i^{th} movement in a cycle
- τ_i : the gain of the movement from subset S_i to the starting subset (for closing a partial cycle)
- Ξ_j : the gainsum of cycle j

g_v denotes the largest gain that can be obtained by moving vertex v to any subset to which v does not belong. G_i is the largest gain among all vertices in subset i. It can be represented equivalently by $G_i =$

CP(G, P)
  // P: a given initial partition;
  repeat {
    $P \leftarrow$ **MoveCycles**(G, P);
  } **until** (stop condition)

MoveCycles(G, P)
for each $v \in V$ {
  $\forall i = 1, 2, \dots, k$, calculate $g_v[i]$;
  $g_v \leftarrow \max_{1\le i\le k} g_v[i]$;
}
for $i \leftarrow 1$ **to** k {
  for $j \leftarrow 1$ **to** k
    $G_{ij} \leftarrow \max_{v\in V_i} g_v[j]$;
  $G_i \leftarrow \max_{1\le j\le k} G_{ij}$;
}
$j \leftarrow 0$;
do {
  Choose a starting subset S_0 s.t. $\forall j = 1, 2, \dots, k$, $G_{S_0} \ge G_j$;
  $i \leftarrow 0$;
  do {
    Choose a vertex $v \in S_i$ s.t. $g_v \ge g_u\ \forall u \in V_{S_i}$;
    $P_v \leftarrow a \in \{1, 2, \dots, k\}$ that maximizes $g_v[a]$;
    Move v to subset P_v and lock v;
    $\xi_i \leftarrow G_{S_i}$; $\tau_i \leftarrow G_{S_i S_0}$;
    i++;
    $S_i \leftarrow P_v$;
    Adjust g's and G's that are affected by v's movement;
  } **while** $(S_i \ne S_0)$;
  Find $l \in \{1, \dots, i-1\}$ that maximizes $(\sum_{a=0}^{l-1} \xi_a) + \tau_l$;
  Undo the sequence of movements from S_l to S_0 in the above;
  Undo the sequence of adjustments from S_l to S_0 in the above;
  Move the maxgain vertex $v \in V_{S_l}$ to S_0;
  Adjust g's and G's that are affected by the movement of v from S_l to S_0;
  $\Xi_j \leftarrow (\sum_{a=0}^{l-1} \xi_a) + \tau_l$;
  j++;
} **while** ($\exists$ at most one subset containing any unlocked vertex);
Find $m \in \{0, 1, \dots, j-1\}$ that maximizes $\sum_{a=0}^{m} \Xi_a$;
Undo the movements after the cycle m;

Figure 1: Cyclic Partitioning Algorithm (CP)



(a) A Cyclic Movement (b) Subcycles

Figure 2: An Example of 8-way Cyclic Movements

```
Create initial population of fixed size p;
do {
    Select parent1 and parent2 from population;
    Normalization(parent1, parent2);
    offspring ← crossover(parent1, parent2);
    CP(G, offspring);
    if suited (offspring)
    then replace(population, offspring);
} until (stopping condition);
return the best answer;
```

Figure 3: The Genetic Cyclic Multiway Partitioning Algorithm (GCMA)

$\max_{i \leq j \leq k} G_{ij}$ or $G_i = \max_{v \in V_i} g_v$.

In the algorithm of Figure 1, lines 1 through 4 compute the gains for all vertices. Lines 5 through 9 calculate the gains related to each subset. Each loop of lines 11 through 30 creates a unit cycle of movements. As shown in Figure 2(a), a starting subset S_0 looks into the next direction with the help of G_{S_0} and then decides $S_1, S_2, \ldots, S_i$ according to $G_{S_1}, G_{S_2}, \ldots, G_{S_i}$, respectively, until S_i is the same as S_0. After this process, we find a partial cycle that maximizes the gainsum (line 23). The gainsum of this cycle is stored (line 28). A number of such cycles are produced as a result of lines 11 through 30. Finally, a prefix in the sequence of cycles is selected to maximize the total gain (line 31).

In KL, swapping a pair of vertices is the minimum unit of change. On the other hand, a cycle of vertex movements is the minimum unit of change in CP. There is no restriction in the length of cycles. An obvious upper bound for the length is $|V| - \lfloor \frac{|V|}{k} \rfloor + 1$ due to the locking.

4 Genetic Cyclic Multiway Partitioning Algorithm (GCMA)

A genetic algorithm starts with a set of initial solutions (*chromosome*) which are called *population*. This population evolves into different population for a number of iterations. At the end the algorithm returns the best member of the population as the solution to the problem.

We combined CP with the genetic search. Figure 3 shows the structure of our hybrid GA for the multiway graph partitioning problem. We call this algorithm Genetic Cyclic Multiway graph partitioning Algorithm (GCMA). Every solution is represented by a linear chromosome. We use a k-ary string for each chromosome to represent a k-way partition. For example, if the i^{th} vertex belongs to subset j, the value of the i^{th} gene is j. We set the population size as 50. For parent selection, we used the proportional roulette-wheel selection. The probability that the best chromosome is chosen was given four times higher than the probablity that the worst chromosome is chosen.

```
Normalization(parent1, parent2)
        //n : |V|
        //count[k][k] : k is the number of subsets(partitions)
        //map[k]
    for i ← 0 to k
        for j ← 0 to k
            count[i][j] ← 0;
    for i ← 0 to n
        count[parent1[i]][parent2[i]]++;
    for i ← 0 to k {
        Find p and q that count[p][q] is maximum;
        for j ← 0 to k
            count[p][j] ← −∞,    count[j][q] ← −∞;
        map[q] ← p;
    }
    for i ← 0 to n
        parent2[i] ← map[parent2[i]];
```

Figure 4: The Normalization Step Between Parents

We normalized the parents before crossover. Figure 4 shows the process of normalization. It was used in [21] and helps maintain consistency between the two parents. Table 1 shows the effect of normalization. We tested on a set of 10 graphs. For 8-way and 32-way partitioning, the solution quality was significantly improved, due to the normalization. We used five-point crossover operators. After crossover, chromosomes are usually not balanced. We start at a random point on the chromosome and adjust the gene values to the right until the balance is satisfied. This has some mutation effect, so we do not add any specific mutation. The offspring replaces the closer parent in Hamming distance (only if it is better than the closer parent) and, if not, the other parent is replaced if the offspring is better. If not again, the worst in the population is replaced.

5 Experimental Results

In this experiment, we used 40 benchmark graphs which were devised by [4] [18] [7]. Their sizes range from 100 to 5,252. They have been widely used for benchmarking graph partitioning [24] [29] [3] although largely used for bisection. We tested on 8-way and 32-way partitionings. The C language was used on a Pentium III 450 MHz computer with Linux operating system.

We first examine the performance of the suggested lo-

Table 1: Cut sizes of Experiments Without and With Normalization

Graph	8-way				32-way			
	Ordinary[1]		**Normailzed**[2]		**Ordinary**[1]		**Normalized**[2]	
	Best[3]	Average[4]	Best[3]	Average[4]	Best[5]	Average[6]	Best[5]	Average[6]
G500.04	3355	3369.21	3354	3364.13	4054	4069.58	4043	4057.20
G1000.0025	227	251.16	220	233.24	343	367.64	326	345.59
U500.40	1865	1866.50	1865	1866.01	5335	5348.10	5328	5338.24
U1000.05	71	114.36	57	83.96	153	198.66	130	147.29
reg500.20	131	144.32	128	133.77	240	244.90	128	133.77
reg5000.0	1107	1181.11	1096	1174.61	1750	1854.30	1096	1174.61
cat.5252	224	351.68	204	314.44	381	558.36	377	552.77
rcat.134	27	27.11	27	27.00	91	91.00	91	91.00
grid5000.50	250	254.74	250	250.20	673	700.06	659	676.60
w-grid100.20	60	60.00	60	60.00	128	128.00	128	128.00

1. No normalization between the two parents
2. Processed with normalization before crossover[21]
3. The best cut size of 100 runs
4. The average cut size of 100 runs
5. The best cut size of 50 runs
6. The average cut size of 50 runs

cal optimization heuristic itself against two traditional heuristics mentioned before; then, we show the experimental results of the hybrid GA with the new heuristic. However, since the hybrid GA uses the local optimization heuristic as an engine, it is obvious that the hybrid GA would perform better than the local optimization heuristic. We thus examine the effectiveness of genetic search by comparison with a random multi-start local optimization with comparable time budgets.

Table 2 shows the results on 8-way partitioning. The three algorithms described in Section 2 and Section 3 are compared. The statistics are from 1,000 independent runs; so the average results are very stable. The bold-faced numbers indicate the best among them. RKL was visibly faster than the other two at the cost of low quality. On the other hand, with respect to quality, PKL and CP were preferable. Their performance was different from graph to graph. For random graphs (G*.*) and caterpillar graphs (cat.*, rcat.*), CP outperformed RKL and PKL. For geometric graphs (U*.*) and all regular graphs (reg*.*), PKL outperformed the others. RKL performed best for grid graphs (grid*.*, w-grid*.*). Table 3 shows the results on 32-way partitioning. The results are a bit different from those of 8-way partitioning. Most notably CP's relative performance was improved. CP outperformed the others for 28 graphs out of 40. On the other hand, PKL's performance was sharply weakened.

The numbers of graphs that RKL performed best among them in 8-way and 32-way partitionings were 8 and 7, respectively. In case of PKL, they were 16 and 5, respectively. In case of CP, they were 16 and 28, respectively. CP spent visibly more time than the others. However, the others were not comparable with CP even when similar time budgets were provided (by giving more trials).

By combining CP with genetic search, its results were significantly improved. However, the hybrid GA (GCMA) took 135 times more time than a single CP run on the average. It is not clear how critical is the genetic search to the performance improvement. We examined this by comparing GCMA with a multi-start CP which runs CP on 135 random initial solutions and returns the best. The experimental results are shown in Table 4. One can observe that, given comparable time budgets, the genetic search is significantly better than the multi-start CP. For 8-way partitioning, the best and the average are from 100 runs and, for 32-way partitioning, they are from 50 runs. Thus, each of the best in multi-start CP is from 13,500 and 6,750 runs of CP. GCMA significantly outperformed multi-start CP in both 8-way partitioning and 32-way partitioning. We may try multi-start RKL or multi-start PKL. But their performance will be clearly not comparable even with multi-start CP.

6 Conclusions

We proposed a hybrid genetic algorithm for multiway graph partitioning and provided extensive experimental results using over 4 million CPU seconds. In order to design a good hybrid GA, we devised a new local optimization heuristic. The most notable feature of the algorithm is that it utilizes cyclic movements of vertices. By attempting direct k-way partitioning and allowing more freedom to vertex movements, it improved the solution quality.

The genetic search turned out to be critical for the performance. The comparison between multi-start CP and GCMA showed the superiority of our genetic al-

Table 2: The Results of 8-way Partitioning Over 1,000 Runs

Graph	RKL			PKL			CP		
	Best	Average	CPU†	Best	Average	CPU†	Best	Average	CPU†
G500.005	**131**	145.52	0.02	**131**	**143.64**	0.10	133	148.00	0.20
G500.01	507	526.92	0.02	502	524.72	0.13	**491**	**516.88**	0.32
G500.02	1312	1343.74	0.04	1312	1341.46	0.22	**1275**	**1318.29**	0.55
G500.04	3447	3487.03	0.08	3438	3478.97	0.40	**3396**	**3436.50**	1.17
G1000.0025	254	278.67	0.05	**253**	**274.13**	0.23	266	290.97	0.36
G1000.005	1030	1063.67	0.06	1017	1050.46	0.35	**1004**	**1036.81**	0.60
G1000.01	2869	2913.03	0.10	2852	2901.38	0.57	**2807**	**2856.86**	1.08
G1000.02	6737	6802.69	0.19	6697	6782.33	1.04	**6646**	**6709.38**	2.02
U500.05	54	84.22	0.03	**31**	**64.32**	0.16	58	95.61	0.21
U500.10	174	242.72	0.04	**156**	**209.73**	0.26	185	272.75	0.40
U500.20	629	736.44	0.07	**621**	**720.18**	0.45	622	784.02	0.91
U500.40	1957	2109.33	0.13	1875	2051.54	0.75	**1868**	**1961.05**	1.95
U1000.05	**97**	163.13	0.06	**97**	**131.42**	0.35	152	202.65	0.32
U1000.10	247	398.02	0.11	**226**	**346.12**	0.63	378	524.19	0.61
U1000.20	862	1021.81	0.19	**832**	**1013.02**	1.17	861	1302.31	1.47
U1000.40	2586	**2872.22**	0.32	2592	2931.31	2.11	**2565**	3025.45	3.79
reg500.0	141	160.38	0.02	**131**	**149.02**	0.12	144	170.06	0.28
reg500.12	148	169.48	0.02	138	**157.55**	0.11	**136**	174.00	0.25
reg500.16	148	172.57	0.02	**139**	**159.76**	0.12	154	176.19	0.25
reg500.20	152	179.82	0.02	**149**	**166.79**	0.11	154	178.50	0.27
reg5000.0	1502	1601.01	0.37	**1368**	**1431.07**	2.50	1751	1851.47	2.82
reg5000.4	1528	1615.95	0.39	**1375**	**1438.80**	2.44	1752	1851.77	2.71
reg5000.8	1521	1631.84	0.43	**1378**	**1445.56**	2.40	1706	1853.48	2.86
reg5000.16	1527	1645.82	0.48	**1365**	**1455.66**	2.45	1744	1849.65	2.74
cat.352	35	51.02	0.01	31	43.55	0.05	**28**	**41.13**	0.22
cat.702	70	95.63	0.01	65	85.01	0.09	**60**	**75.06**	0.32
cat.1052	102	135.18	0.02	92	122.36	0.14	**79**	**104.96**	0.43
cat.5252	488	581.46	0.14	524	611.74	0.88	**430**	**510.81**	1.43
rcat.134	**27**	33.09	0.00	**27**	31.31	0.01	**27**	**30.76**	0.13
rcat.554	15	88.28	0.01	17	77.83	0.06	**13**	**60.37**	0.41
rcat.994	26	148.11	0.01	28	160.71	0.11	**25**	**135.93**	0.43
rcat.5114	**135**	**543.14**	0.11	345	866.91	0.67	196	566.25	1.30
grid100.10	42	44.94	0.00	41	48.58	0.02	**40**	**43.04**	0.08
grid500.21	90	**97.71**	0.01	94	113.27	0.13	**87**	110.43	0.27
grid1000.20	**114**	**124.41**	0.04	128	156.87	0.32	116	167.91	0.45
grid5000.50	**250**	**280.16**	0.27	283	362.22	2.75	256	620.58	2.79
w-grid100.20	**60**	64.77	0.00	61	67.25	0.02	**60**	**61.81**	0.08
w-grid500.42	135	**143.30**	0.01	139	153.84	0.13	**132**	149.80	0.24
w-grid1000.40	**175**	**201.58**	0.03	185	214.05	0.32	183	231.33	0.42
w-grid5000.100	**400**	**448.68**	0.25	426	488.11	2.79	419	725.73	2.88

† CPU seconds on Pentium III 450 MHz

gorithm. Since there have been few experimental results of multi-way graph partitioning on the benchmark graphs, the results in this paper may be used as a basis for further experimental competitions. We should also note that large-step Markov chain [23] [15] and tabu search [25] [12] [3] are known to have effective search capabilities and have been successful for a number of difficult problems. Combining CP with these approaches are left for further study.

Acknowledgements

This work was partly supported by Motorola Korea and Brain Korea 21 Project.

References

[1] C. J. Alpert and A. B. Kahng. Multiway partitioning via geometric embeddings, orderings, and dynamic programming. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 14(11):1342–1358, 1995.

[2] M. Anil, M Kishan, K. M. Chilukuri, and R. Sanjay. Optimization using replicators. In *International Conference on Genetic Algorithms*, pages 209–216, 1995.

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

[4] 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.

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

[6] T. N. Bui and B. R. Moon. On multi-dimensional encoding/crossover. In *International Conference on Genetic Algorithms*, pages 49–55, 1995.

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

[8] T. N. Bui and A. Peck. Partitioning planar graphs. *SIAM J. Comp.*, 21(2):203–215, 1992.

Table 3: The Results of 32-way Partitioning Over 1,000 Runs

Graph	**RKL**			**PKL**			**CP**		
	Best	Average	CPU†	Best	Average	CPU†	Best	Average	CPU†
G500.005	200	212.76	0.03	202	218.58	0.37	**196**	**211.28**	1.55
G500.01	676	695.10	0.04	690	706.99	0.47	**654**	**675.34**	2.87
G500.02	1667	1687.77	0.05	1674	1699.87	0.61	**1617**	**1639.73**	5.89
G500.04	4181	4207.29	0.10	4177	4211.63	0.92	**4084**	**4117.70**	15.20
G1000.0025	**366**	**383.08**	0.07	366	385.10	0.91	368	392.18	2.21
G1000.005	1335	1361.88	0.09	1349	1378.36	1.16	**1291**	**1324.13**	4.12
G1000.01	3569	3599.51	0.14	3570	3618.87	1.57	**3470**	**3507.04**	8.45
G1000.02	8127	8164.21	0.25	8123	8176.54	2.25	**7942**	**8004.31**	17.49
U500.05	135	168.34	0.05	**127**	**150.52**	0.68	134	163.40	2.41
U500.10	564	625.76	0.07	**552**	**591.84**	0.92	553	597.84	4.17
U500.20	1912	1997.52	0.11	1878	1960.92	1.30	**1842**	**1907.00**	12.48
U500.40	5394	5498.45	0.19	5385	5531.51	1.81	**5346**	**5489.48**	48.83
U1000.05	199	249.74	0.11	**165**	**199.85**	1.57	216	272.64	3.26
U1000.10	651	782.66	0.18	**634**	**714.93**	2.56	659	795.46	7.80
U1000.20	2511	2693.37	0.30	2512	2684.59	4.02	**2476**	**2628.23**	23.92
U1000.40	7630	7891.14	0.50	7564	7858.76	6.34	**7374**	**7652.98**	64.94
reg500.0	246	260.46	0.03	258	273.08	0.45	**232**	**249.23**	1.72
reg500.12	245	258.83	0.04	252	268.33	0.46	**231**	**246.16**	1.60
reg500.16	245	260.50	0.04	245	269.35	0.46	**232**	**248.01**	1.57
reg500.20	251	266.74	0.04	260	275.21	0.45	**240**	**253.36**	1.58
reg5000.0	2179	2243.29	0.56	2056	2114.34	8.74	**2036**	**2101.29**	16.55
reg5000.4	2184	2248.46	0.59	2051	2119.55	8.74	**2032**	**2102.67**	16.55
reg5000.8	2187	2250.72	0.62	2062	2123.90	8.65	**2023**	**2105.70**	16.89
reg5000.16	2186	2252.22	0.69	2054	2121.53	8.62	**2037**	**2104.19**	16.81
cat.352	91	101.19	0.01	**86**	**95.94**	0.20	90	100.84	1.45
cat.702	116	146.06	0.03	**88**	125.83	0.52	90	**104.61**	5.26
cat.1052	169	199.94	0.04	**144**	184.02	0.73	152	**168.66**	6.23
cat.5252	717	812.60	0.20	715	861.21	4.22	**575**	**645.56**	18.95
rcat.134	**91**	**92.56**	0.00	**91**	95.01	0.04	**91**	93.38	0.29
rcat.554	170	190.91	0.02	**159**	182.56	0.28	**159**	**162.08**	5.00
rcat.994	73	220.01	0.03	32	188.70	0.70	**31**	**32.88**	8.72
rcat.5114	688	1041.41	0.20	870	1341.51	3.23	**605**	**943.45**	45.71
grid100.10	122	133.72	0.01	165	172.90	0.03	**108**	**108.95**	0.23
grid500.21	232	**248.43**	0.04	251	275.57	0.58	**226**	251.38	3.00
grid1000.20	**318**	**336.47**	0.09	363	399.03	1.39	337	387.35	5.22
grid5000.50	**660**	**732.12**	0.77	884	978.64	11.24	952	1233.53	11.09
w-grid100.20	141	154.54	0.01	184	192.13	0.03	**128**	**128.56**	0.24
w-grid500.42	276	293.43	0.04	296	317.02	0.57	**272**	**292.15**	2.90
w-grid1000.40	**387**	**413.53**	0.09	433	462.90	1.38	398	449.53	4.60
w-grid5000.100	**819**	**898.95**	0.75	1026	1114.88	11.19	1043	1372.60	10.53

† CPU seconds on Pentium III 450 MHz

[9] J. P. Cohoon, W. N. Martin, and D. S. Richards. A multi-population genetic algorithm for solving the k-part on hyper-cubes. In *Fourth International Conference on Genetic Algorithms*, pages 244–248, 1991.

[10] R. Collins and D. Jefferson. Selection in massively parallel genetic algorithms. In *Fourth International Conference on Genetic Algorithms*, pages 249–256, 1991.

[11] A. Dasdan and V. Aykanat. Two novel multiway circuit partitioning algorithms using relaxed locking. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 16(2):169–178, 1997.

[12] M. Dell'Amico and F. Maffioli. A new tabu search approach to the 0-1 equicut problem. In *Meta-Heuristics 1995: The State of the Art*, pages 361–377. Kluwer Academic Publishers, 1996.

[13] R. J. Donald and A. B. Mark. Solving partitioning problems with genetic algorithms. In *International Conference on Genetic Algorithms*, pages 442–489, 1991.

[14] L. R. Ford and D. R. Fulkerson. *Flows in Networks.* Priceton University Press, 1962.

[15] A. S. Fukunaga, J. H. Huang, and A. B. Kahng. On clustered kick moves for iterated-descent netlist partitioning. In *IEEE International Symposium on Circuits and Systems*, volume 4, pages 496–499, 1996.

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

[17] M. Harpal, M. Kishan, M. Chilukuri, and R. Sanjay. Genetic algorithms for graph partitioning and incremental graph partitioning. In *IEEE Proceedings of the Supercomputing*, pages 449–457, 1994.

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

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

[20] S. Kirkpatrick, Gelatt C. D. Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

Table 4: The Results of Multi-Start CP and GCMA

Graph	8-way						32-way					
	Multi-Start CP			GCMA			Multi-Start CP			GCMA		
	Best[1]	Average[2]	CPU[3]	Best	Average	CPU[3]	Best[4]	Average[5]	CPU[3]	Best	Average	CPU[3]
G500.005	130	135.50	26.13	116	124.80	18.75	195	198.82	204.20	181	187.04	148.70
G500.01	489	497.40	42.51	468	477.14	39.78	652	659.14	381.19	631	642.02	305.42
G500.02	1283	1293.53	71.44	1257	1268.45	75.09	1608	1619.56	784.48	1587	1594.98	744.53
G500.04	3384	3401.18	152.58	3354	3364.13	198.52	4076	4092.24	2030.90	4043	4057.20	2035.89
G1000.0025	262	271.60	47.07	220	233.24	51.34	371	375.30	294.87	326	345.59	268.77
G1000.005	997	1007.08	79.13	942	959.09	86.16	1288	1297.46	548.70	1222	1246.87	571.08
G1000.01	2801	2816.53	140.62	2729	2746.65	185.89	3454	3471.18	1147.36	3370	3400.74	1474.57
G1000.02	6632	6651.13	258.36	6531	6562.20	403.43	7949	7962.84	2355.47	7839	7862.33	3878.28
U500.05	53	69.24	28.47	22	33.35	38.99	130	137.12	316.93	115	120.76	286.51
U500.10	163	196.58	54.50	143	149.58	64.58	550	559.84	561.43	532	541.40	541.93
U500.20	613	633.60	126.22	611	613.08	124.37	1831	1849.64	1657.32	1825	1832.66	1351.17
U500.40	1867	1870.42	261.24	1865	1866.01	150.17	5337	5362.40	6385.35	5328	5338.24	4983.42
U1000.05	140	157.80	44.46	57	83.96	74.37	211	228.52	442.98	130	147.29	810.77
U1000.10	349	399.12	86.19	187	217.53	136.55	653	692.90	1029.54	577	589.17	1762.44
U1000.20	855	963.94	197.05	812	822.35	240.63	2441	2481.28	3140.80	2367	2394.17	4107.66
U1000.40	2564	2574.94	485.23	2562	2562.23	470.18	7370	7410.90	8688.90	7329	7343.08	7494.34
reg500.0	136	143.90	35.77	116	123.47	28.42	230	234.82	232.10	222	228.96	145.48
reg500.12	144	152.08	34.60	118	124.17	34.34	228	232.42	215.74	222	226.90	125.14
reg500.16	147	154.92	34.16	122	125.91	34.48	231	234.90	209.71	222	229.00	131.22
reg500.20	151	158.92	33.97	128	133.77	32.39	236	240.38	206.29	231	235.84	128.38
reg5000.0	1690	1760.94	368.35	1096	1174.61	586.90	2031	2049.12	2286.98	1720	1845.36	2748.89
reg5000.4	1735	1770.84	370.50	1093	1162.79	632.17	2032	2051.94	2296.16	1725	1845.10	2903.34
reg5000.8	1675	1766.82	368.55	1098	1179.79	597.55	2027	2053.36	2302.16	1737	1837.87	2842.56
reg5000.16	1721	1767.26	367.89	1077	1134.26	682.06	2025	2052.20	2305.64	1693	1802.83	2998.63
cat.352	29	32.85	30.08	19	25.27	26.78	90	93.30	192.26	85	86.58	168.55
cat.702	58	62.00	42.32	31	49.71	38.72	90	94.10	712.55	60	79.71	845.03
cat.1052	83	86.90	57.21	50	73.46	46.79	148	154.16	832.70	101	141.36	1046.32
cat.5252	429	447.40	198.31	204	314.44	265.66	563	589.24	2573.53	377	552.77	2905.65
rcat.134	27	27.23	18.33	27	27.00	7.18	91	91.00	41.33	91	91.00	9.32
rcat.554	14	16.57	56.41	9	13.23	31.13	159	159.00	675.14	159	159.24	275.33
rcat.994	26	54.87	58.73	15	19.51	42.35	31	31.72	1190.67	31	31.08	262.78
rcat.5114	124	227.02	178.98	45	51.73	144.32	495	612.42	6223.60	489	491.38	8936.98
grid100.10	40	40.00	11.11	40	40.00	3.89	108	108.00	30.30	108	108.02	9.88
grid500.21	87	88.58	36.67	86	86.60	21.52	225	230.22	402.18	220	223.10	234.93
grid1000.20	114	120.57	59.59	114	114.02	40.65	327	340.14	701.70	314	316.52	502.79
grid5000.50	311	361.08	381.02	250	250.20	316.21	918	992.18	1506.38	659	676.60	2065.60
w-grid100.20	60	60.00	10.77	60	60.00	3.40	128	128.00	31.37	128	128.00	9.87
w-grid500.42	132	133.43	33.33	131	132.49	17.43	268	273.38	393.29	266	270.16	195.11
w-grid1000.40	180	187.08	57.39	176	179.92	42.29	392	407.48	635.82	384	387.48	404.96
w-grid5000.100	415	486.95	388.53	400	400.95	328.71	1069	1144.46	1471.00	820	840.68	1854.84

1. The best of 13,500 runs of CP
2. Average of 100 runs, each of which is the best of 135 runs of CP
3. CPU seconds on Pentium III 450 MHz
4. The best of 6,750 runs of CP
5. Average of 50 runs, each of which is the best of 135 runs of CP

[21] G. Laszewski. Intelligent structural operators for the k-way graph partitioning problem. In *Fourth International Conference on Genetic Algorithms*, pages 45–52, 1991.

[22] C. H. Lee, M. Kim, and C. I. Park. An efficient k-way graph partitioning algorithm for task allocation in parallel computing systems. In *1st International Conference on System Integration*, pages 748–751, 1990.

[23] O. Martin, S. W. Otto, and E. W. Felten. Large-step markov chains for the traveling salesman problem. *Complex Systems*, 5:299–326, 1991.

[24] B. R. Moon and C. K. Kim. A two-dimensional embedding of graphs for genetic algorithms. In *International Conference on Genetic Algorithms*, pages 204–211, 1997.

[25] E. Rolland, H. Pirkul, and F. Glover. A tabu search for graph partitioning. *Annals of Operations Research*, 63, 1996.

[26] Y. Saab and V. Rao. Stochastic evolution: A fast effective heuristic for some genetic layout problems. In *27th IEEE/ACM Design Automation Conference*, pages 26–31, 1990.

[27] P. Sadayappan, F. Ercal, and J. Ramanujam. Partitionig graphs on message-passing machines by pairwise mincut. *Information Sciences*, pages 223–237, 1998.

[28] L. A. Sanchis. Multiple-way network partitioning. *IEEE Trans. on Computers*, 38(1):62–81, 1989.

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

[30] C. W. Yeh, T. Y. Lin, and C. K. Cheng. A general purpose multiple way partitioning algorithm. In *29th IEEE/ACM Design Automation Conference*, pages 421–426, 1991.