A Hybrid Genetic Approach for Circuit Bipartitioning

Jong-Pil Kim[1], Yong-Hyuk Kim[2], and Byung-Ro Moon[2]

[1] Electronics and Telecommunications Research Institute
161 Gajeong-dong, Yuseong-gu, Daejeon, 305-350 Korea
kimjp@etri.re.kr
[2] School of Computer Science & Engineering, Seoul National University
Shillim-dong, Kwanak-gu, Seoul, 151-742 Korea
{yhdfly, moon}@soar.snu.ac.kr

Abstract. We propose a hybrid genetic algorithm for partitioning a VLSI circuit graph into two disjoint graphs of minimum cut size. The algorithm includes a local optimization heuristic which is a modification of Fiduccia-Matheyses algorithm. Using well-known benchmarks (including ACM/SIGDA benchmarks), the combination of genetic algorithm and the local heuristic performed better than hMetis, a representative circuit partitioning algorithm.

1 Introduction

Hypergraph partitioning is an important problem and has many applications including design automation of VLSI chips and multi-chip systems. A hypergraph $H(V, E)$ consists of a set of nodes V and a set of nets E. Each *net* $e \in E$ is a subset of two or more nodes in V. The nodes associated with a net are called *pins*. A bisection of H is dividing the set V into two disjoint subsets V_1 and V_2. The cut size $c(V_1, V_2)$ is defined as

$$c(V_1, V_2) \;=\; |\{\; e \in E \mid e \cap V_1 \neq \phi \;\text{ and }\; e \cap V_2 \neq \phi \;\}|.$$

In other words, the cut size is the number of nets whose end points are in different subsets. The *min-cut bisection problem* minimizes the $c(V_1, V_2)$ subject to $|V_1| = |V_2|$. Since it is NP-hard problem [16], heuristic algorithms, which yield approximate solutions in acceptable times, are generally used. These include iterative improvement methods [12] [13] [14] [20] [23], meta-heuristic methods such as simulated annealing (SA) [18] [22], large-step Markov chain (LSMC) [15] [25], tabu search (TS) [3] [11] [27], and genetic algorithm (GA) [4] [7] [8] [17] [21] [24] [28] [30]. An extensive survey of hypergraph partitioning appeared in [2].

The Kernighan-Lin algorithm (KL) [20] is a representative iterative improvement algorithm for general graphs. KL starts with a random initial two subsets and proceeds by iteratively swapping nodes. Subsequently, Schweikert and Kernighan [29] modified the algorithm for hypergraphs. Fiduccia and Mattheyses algorithm (FM) [14] is a variation of KL, which reduces the time complexity

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1054–1064, 2004.

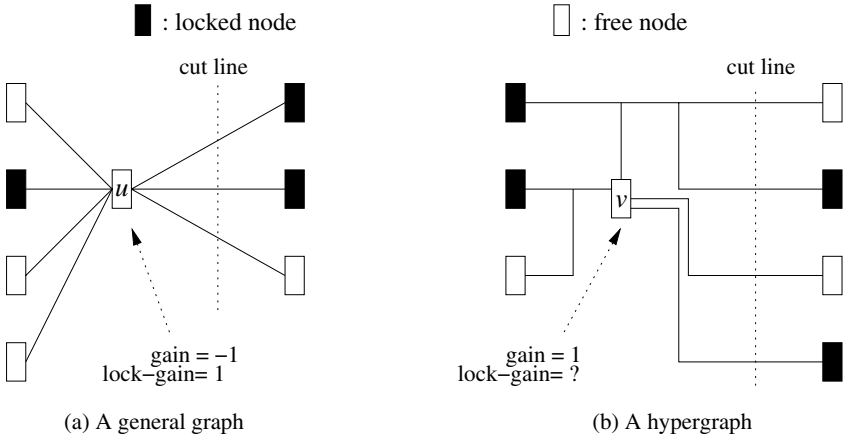


Fig. 1. Gain vs. lock gain

of the algorithm to $O(|P|)$ with effective data structures, where $|P|$ is the number of pins in a hypergraph. Krishnamurthy [23] enhanced FM by introducing the concept of level gains and obtained better results in small graphs. Dutt and Deng proposed PROP [12] which uses a probabilistic gain computation, and CLIP/CDIP [13] which uses a cluster-removal method.

Lock gain is a new measure which showed improvement over the traditional gain for general graphs [21]. In this paper, we adapt the lock gain for hypergraphs within the framework of FM [14]. We combine the lock-gain-based hypergraph partitioning heuristic with a steady-state genetic algorithm.

The remainder of this paper is organized as follows. Section 2 summarizes the FM algorithm and explains the lock gain. In Section 3, we describe the local optimization heuristic for min-cut bisection problem. Details of the genetic algorithm are provided in Section 4. We show the experimental results in Section 5 and summarize the study in Section 6.

2 Preliminaries

2.1 Fiduccia-Matheyses Algorithm

The FM algorithm [14] starts with two random initial subsets. It moves a node at a time from one subset to the other in an attempt to minimize the cut size. Each node v has a *gain* $g(v)$ which represents the cut size reduction by moving v to the opposite subset:

$$g(v) = |\{e \in I(v)\}| \; - \; |\{e' \in C(v)\}|$$

where $C(v)$ is the set of nets containing v in which all pins lie in the subset to which v belongs; $I(v)$ is the set of nets such that all the other pins except v

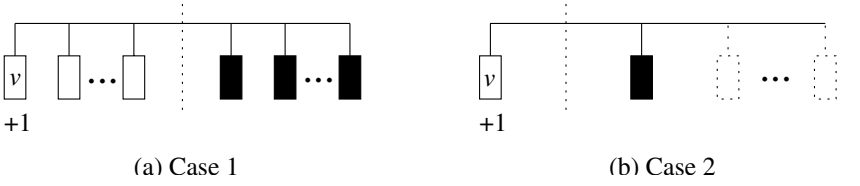


Fig. 2. Positive lock gain

belong to the opposite subset. To differentiate the gain with the lock gain of the next section, we sometimes use the term "general gain."

The node to move is chosen on the basis of its gain and the balance criterion. If we do not consider the balance criterion, all nodes would eventually migrate to one subset except a lowest-degreed node. After a node is moved, it is locked to prevent further movements. Only unlocked nodes are allowed to move. A pass stops when all nodes are locked. Then we undo the sequence of movements beyond the point that maximizes the total gain. All the nodes are now marked unlocked again and another pass starts. This process is repeated until no more improvement is obtained.

2.2 Lock Gain

The general gain of a node only reflects the local information of a node. Hence, it is hard to predict the future state of the node from the gain. FM improves an initial solution through short-sighted moves based on the gain. Kim and Moon [21] introduced lock gain as a primary measure for choosing the nodes to move. It uses the history of search more efficiently. Lock gain showed excellent performance for general graphs.

In general graphs, the lock gain $l(v)$ of a node v is defined to be the gain of the node v only with respect to the locked nodes. Let node v be in the subset V_1 without loss of generality. Formally,

$$\begin{aligned} l(v) \;=\; & |\{w \mid (v,w) \in E, \text{ and node } w \in L(V_2)\}| \\ & - |\{w' \mid (v,w') \in E, \text{ and node } w' \in L(V_1)\}| \end{aligned} \qquad (1)$$

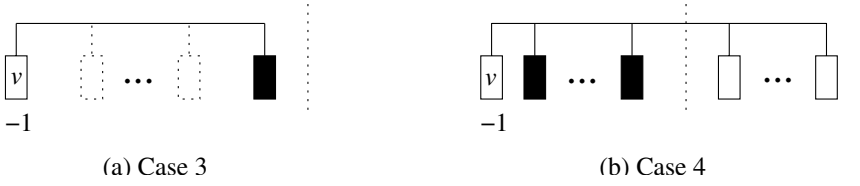
where $L(V_i)$ is the set of locked nodes in the subset V_i.

An example is given in Figure 1(a). In the figure, the gain of the node u is -1. But when only locked nodes are considered, the lock gain of the node u becomes 1.

3 Lock Gain in Hypergraphs

3.1 Lock Gain in Hypergraphs

To apply the lock gain to the hypergraph bisection, considerable modification is necessary for the lock gain calculation method. In general graphs, nets cannot



(a) Case 3 (b) Case 4

Fig. 3. Negative lock gain

have degrees greater than two. But in hypergraphs, most nets have degrees greater than two. A net may contain both locked nodes and unlocked nodes. The above term (1) does not work for this case. For example, in Figure 1(b), the gain of the node v is 1; but the lock gain of the node cannot be calculated in the same method as in Figure 1(a).

We propose a new lock gain calculation method for the hypergraph bisection. Let's define $l_e(v)$ to be the lock gain of a node v due to the net $e \in E$. $l_e(v)$ is obtained as the following.

We assume that the node v is on the left side without loss of generality. L, R, and L', R' are defined to be the numbers of nodes on the left side and on the right side and the numbers of locked nodes on the left side and on the right side, respectively.

i) Positive Lock Gain
   - Case 1 (e.g., Figure 2(a)) :

$$L > 0, \quad L' = 0, \quad R = R' > 0.$$

     If all nodes on the right side are locked and there is no locked node on the left side, then $l_e(v) = 1$.
   - Case 2 (e.g., Figure 2(b)) :

$$L = 1, \quad L' = 0, \quad R \geq R' > 0.$$

     If there exist locked nodes on the right side and there is one free node on the left side, then $l_e(v) = 1$.

ii) Negative Lock Gain
   - Case 3 (e.g., Figure 3(a)) :

$$L > L' > 0, \quad R = R' = 0.$$

     If all nodes are on the left side and at least one node is locked, then $l_e(v) = -1$.
   - Case 4 (e.g., Figure 3(b)) :

$$L - L' = 1, \quad R > 0, \quad R' = 0.$$

     If all nodes on the left side except v are locked and there is no locked node on the right side, then $l_e(v) = -1$.

```
LFM (G, V1, V2)
    // Vi: a given initial subset, V1 ∪ V2 = V and V1 ∩ V2 = φ
1.  do {
2.      compute g(v) for each v ∈ V;
3.      set l(v) to 0 for each v ∈ V;
4.      Q = φ;
5.      for i = 1 to n {
6.          if (V1 > V2)
7.              choose vi ∈ V1 − Q such that l(v) is maximal over all choices of v ∈ V1;
8.          else
9.              choose vi ∈ V2 − Q such that l(v) is maximal over all choices of v ∈ V2;
10.         Q = Q ∪ {vi};
11.         for each v ∈ V − Q adjacent to vi
12.             adjust l(v) and g(v) if affected by moving vi;
13.     }
14.     choose k ∈ {1, ..., n} to maximize ∑_{i=1}^{k} g(vi);
15.     move nodes in the subsets {v1, ..., vk} to their opposite sides;
16. } until (there is no improvement)
```

Fig. 4. LFM algorithm

iii) Zero Lock Gain
In all the other cases, $l_e(v) = 0$.

Then, the lock gain $l(v)$ of the node v is defined to be

$$l(v) = \sum_{e \in N(v)} l_e(v)$$

where $N(v)$ is a set of nets to which the node v is connected. If all nodes are locked, $l(v)$ is equal to $g(v)$.

Our lock-gain based FM algorithm is given in Figure 4. In the algorithm, lines 2 through 4 compute the gains, set lock gains to 0 for all nodes, and initialize Q (the set of locked nodes). Lines 6 through 12 choose a node, move and lock it, and adjust general gains and lock gains which are affected by the movement. In lines 14 and 15, we undo the sequence of movements beyond the point that maximizes the total gain (i.e., minimizes the cut size). We call this algorithm LFM.

3.2 Implementation

We use the bucket data structure to maintain lock gain lists. This data structure allows constant-time selection of the most attractive node and fast gain update after each move [14]. Since the lock gain is an integer in the range $[-D_{max}, D_{max}]$ where D_{max} is the maximum node degree in the graph, the bucket structure maintains the lock gain efficiently. When a tie occurs in max lock gain, general gains are used for the tie-breaking. For the efficient implementation of this tie-breaking strategy, we enlarge the number of buckets.

Kim and Moon [21] used $(2D_{max}+1)^2$ buckets for tie-breaking. In this paper, we use $(2\lceil \frac{D_{max}}{2} \rceil + 1)^2$ buckets for speed up. In an examination, we observed that

Table 1. The Comparison of Two Buckets (Using LFM)

Circuits	$(2\lceil \frac{D_{max}}{2} \rceil + 1)^2$		$(2D_{max} + 1)^2$	
	Ave[1]	CPU[2]	Ave[1]	CPU[2]
Test03	80.40	0.046	81.36	0.072
Test04	77.03	0.060	78.23	0.108
Test05	107.20	0.117	106.95	0.169
Prim1	66.17	0.009	66.17	0.009

1. The average cut size of 1,000 runs
2. CPU seconds on Pentium III 1 GHz

the number of nodes with a degree greater than $\lceil \frac{D_{max}}{2} \rceil$ is less than 20% of the entire nodes. We set the range of the gain values to $[-\lceil \frac{D_{max}}{2} \rceil, \lceil \frac{D_{max}}{2} \rceil]$. If a gain value is out of the range, we set it to one of the two bounds.

A node v with lock gain $l(v)$ and gain $g(v)$ is stored at the bucket indexed by the value $R(l(v))(2\lceil \frac{D_{max}}{2} \rceil + 1) + R(g(v))$ where

$$R(k) = \begin{cases} -\lceil \frac{D_{max}}{2} \rceil, & \text{if } k < -\lceil \frac{D_{max}}{2} \rceil \\ k, & \text{if } -\lceil \frac{D_{max}}{2} \rceil \leq k \leq \lceil \frac{D_{max}}{2} \rceil \\ \lceil \frac{D_{max}}{2} \rceil, & \text{if } k > \lceil \frac{D_{max}}{2} \rceil \ . \end{cases}$$

Table 1 compares the two strategies with $(2\lceil \frac{D_{max}}{2} \rceil + 1)^2$ and $(2D_{max} + 1)^2$ buckets. One can observe that the former is faster than the latter without notable impact on performance.

4 Genetic Algorithm

We devised a hybrid steady-state genetic algorithm for the problem. A hybrid steady-state genetic algorithm is often called a memetic algorithm [26]. Figure 5 shows the template of the GA.

- Encoding:
  A chromosome is defined to be an n-tuple $\langle c_1, c_2, \ldots, c_n \rangle$ where $c_i \in \{0, 1\}$ for $i = 1, 2, \ldots, n$. If the i^{th} node belongs to the left side, the value of the i^{th} gene is 0; otherwise, it is 1.

- Initialization:
  p chromosomes are created at random. Each chromosome undergoes a balancing process. We set the population size p to be 50.

- Selection:
  The roulette-wheel proportional selection scheme is used. The probability that the best chromosome is chosen was set four times higher than the probability that the worst chromosome is chosen.

```
create initial population of fixed size;
do {
    choose parent1 and parent2 from population;
    offspring ← crossover (parent1, parent2);
    local-improvement(offspring);        // LFM algorithm
    replace (population, offspring);
} until (stopping condition);
return the best member of the population;
```

Fig. 5. A hybrid genetic algorithm for hypergraph partitioning

– Crossover and Mutation:
We used five-point crossover. 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 explicit mutation.

– Replacement:
We combine preselection [6] and Genitor-style replacement [31]. If it is superior to the closer parent in Hamming distance, the offspring replaces the closer parent, if not, the inferior parent is replaced if the offspring is better. If not again, the worst in the population is replaced.

– Stopping condition:
If 70 percent of the population converges with the same cut size as the best solution, we stop the GA.

5 Experimental Results

5.1 Test Set and Test Environment

Before showing the experimental results, we introduce the benchmarks used in this experiment. We tested the proposed algorithm on 9 benchmark circuit graphs, including seven ACM/SIGDA benchmarks [1]. Table 2 shows the numbers of nodes, nets, and pins for each circuit graph. All cut size results are from the strict 50:50% balance criterion. Although some flexibility is allowed in the balancing in real-world circuit partitioning, this bisection model has a strong indication to the performance in other partitioning models.

We first examine the performance of the suggested local optimization heuristic (LFM) against the original FM and a well-known partitioner hMetis [19]. Metis (hMetis) is a recent representative partitioning algorithm which has been a standard for comparison in a number of papers [5] [9] [10]. Then we show the

Table 2. Specification of Circuit Graphs

Circuits	#nodes	#nets	#pins
Test02	1663	1721	6134
Test03	1607	1618	5807
Test04	1515	1658	5975
Test05	2595	2751	10076
Test06	1752	1674	6638
Prim1	833	902	2908
Prim2	3014	3029	11219
19ks	2844	3282	10547
Industry2	12142	12949	47193

experimental results of the hybrid GA. 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 the genetic search by comparison with the multi-start local optimization with comparable time budgets.

C language was used on a Pentium III 1 GHz PC with Linux operating system. It was compiled with GNU's *gcc* compiler.

5.2 Experimental Results

We first examine the performance of the local optimization heuristic (LFM) in Table 3. FM, hMetis, and LFM are compared. The statistics of the three algorithms are from 1,000 independent runs. So the average results are fairly stable. The bold-faced numbers indicate the best result among the three algorithms. LFM significantly improved the performance of the original FM at the cost of a bit more CPU time. However, LFM was not comparable to hMetis as its own.

Table 4 shows the performance of the suggested GA. The genetic algorithm (GA) significantly improved the performance of LFM. Because the GA took roughly 200 times more than a single run of LFM, it is not clear how critical the genetic search is to the performance improvement. Thus, we compared the GA with the multi-start version (LFM200) that runs LFM on 200 random initial solutions and returns the best out of them. For the same reason, hMetis200, that is a multi-start version of hMetis with 200 runs, was compared. On the average, the proposed GA performed best in six graphs among nine. The difference between LFM200 and GA is evidently owing to the genetic process.

As mentioned, hMetis outperformed LFM. It constructs a multi-level hierarchy and performs complex declustering and reclustering process called V-cycle. We want to emphasize that LFM, which simply changed the measure for finding nodes to move, obtained high quality performances and the genetic algorithm boosted up the LFM's quality so that it performed better than hMetis.

Table 3. Bipartition Cut Sizes of FM, hMetis, and LFM

Circuits	FM			LFM			hMetis		
	Ave[1]	CPU[2]	Best[3]	Ave[1]	CPU[2]	Best[3]	Ave[1]	CPU[2]	Best[3]
Test02	172.88	0.016	114	109.03	0.059	91	**101.87**	0.061	88
Test03	118.72	0.078	64	80.40	0.046	58	**63.94**	0.059	59
Test04	140.22	0.014	70	77.63	0.060	51	**58.81**	0.050	54
Test05	181.47	0.039	98	107.20	0.117	75	**80.24**	0.096	71
Test06	92.09	0.017	67	75.57	0.026	63	**68.50**	0.061	64
Prim1	80.17	0.007	54	66.17	0.009	54	**56.79**	0.032	53
Prim2	296.96	0.068	176	230.46	0.115	147	**197.00**	0.172	148
19ks	190.88	0.059	136	154.30	0.089	112	**127.52**	0.130	111
Industry2	839.25	0.475	342	311.84	0.757	199	**228.21**	1.159	180

1. The average cut size of 1,000 runs
2. CPU seconds on Pentium III 1 GHz
3. The best cut size of 1,000 runs

Table 4. Bipartition Cut Sizes of LFM200, hMetis200, and GA

Circuits	LFM200			GA			hMetis200		
	Ave[1]	CPU[2]	Best[3]	Ave[4]	CPU[2]	Best[5]	Ave[6]	CPU[2]	Best[7]
Test02	92.08	11.58	89	**88.16**	17.20	88	89.81	12.12	88
Test03	58.93	9.07	58	**58.00**	5.59	58	59.90	11.74	58
Test04	52.41	11.98	51	**51.15**	8.90	51	54.77	10.02	54
Test05	77.37	20.83	72	72.05	27.85	71	**71.33**	19.22	71
Test06	63.53	5.10	63	**63.12**	4.89	63	64.05	12.30	64
Prim1	53.82	1.86	53	53.87	1.14	53	**53.00**	6.33	53
Prim2	152.76	24.78	146	**149.02**	19.45	146	177.76	34.39	146
19ks	116.50	19.72	112	**110.22**	21.44	110	110.81	25.89	110
Industry2	210.48	130.50	195	186.97	211.25	183	**181.15**	233.35	180

1. The average cut size of 100 runs, each of which is the best of 200 runs of LFM
2. CPU seconds on Pentium III 1 GHz
3. The best cut size of 20,000 runs
4. The average cut size of 100 runs
5. The best cut size of 100 runs
6. The average cut size of 100 runs, each of which is the best of 200 runs of hMetis
7. The best cut size of 20,000 runs

6 Conclusions

We proposed a hybrid genetic algorithm for the hypergraph min-cut bisection problem. In order to design a good hybrid GA, we devised a new local optimization heuristic. The suggested local heuristic is a variation of FM algorithm that uses lock gain for the choice of moving nodes.

The hybrid genetic algorithm with the new local search heuristic synergetically achieved good performance. The experimental results showed the effectiveness of the suggested genetic algorithm.

There are stochastic methods such as tabu search [27] [11] [3] and large-step Markov chain [25] [15] that are known to have effective search capabilities. Combination of our suggested local heuristic and other stochastic methods is considered to be a promising topic for further studies.

Acknowledgments. This study was supported by Brain Korea 21 Project. The ICT at Seoul National University provided research facilities for this study.

References

1. Benchmark. http://vlsicad.cs.ucla.edu/~cheese/benchmarks.html.
2. C. J. Alpert and A. B. Kahng. Recent directions in netlist partitioning: A survey. *Integration: the VLSI Journal*, 19(1-2):1–81, 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 and B. R. Moon. Genetic algorithm and graph partitioning. *IEEE Trans. on Computers*, 45(7):841–855, 1996.
5. A. E. Caldwell, A. B. Kahng, and I. L. Markov. Improved algorithms for hypergraph bipartitioning. In *Asia and South Pacific Design Automation Conference*, pages 661–666, 2000.
6. D. Cavicchio. *Adaptive Search Using Simulated Evolution.* PhD thesis, University of Michigan, Ann Arbor, MI, 1970.
7. 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.
8. R. Collins and D. Jefferson. Selection in massively parallel genetic algorithms. In *Fourth International Conference on Genetic Algorithms*, pages 249–256, 1991.
9. J. Cong and S. K. Lim. Edge separability based circuit clustering with application to circuit partitioning. In *Asia and South Pacific Design Automation Conference*, pages 429–434, 2000.
10. J. Cong, S. K. Lim, and C. Wu. Performance driven multi-level and multiway partitioning with retiming. In *ACM/IEEE-CAS/EDAC Design Automation Conference*, pages 274–279, 2000.
11. 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.
12. S. Dutt and W. Deng. A probability-based approach to VLSI circuit partitioning. In *Proc. Design Automation Conference*, pages 100–105, 1996.
13. S. Dutt and W. Deng. VLSI circuit partitioning by cluster-removal using iterative improvement techniques. In *Proc. International Conference on Computer-Aided Design*, pages 194–200, 1996.
14. C. Fiduccia and R. Mattheyses. A linear time heuristics for improving network partitions. In *19th IEEE/ACM Design Automation Conference*, pages 175–181, 1982.
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. G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Application in VLSI domain. In *Proc. Design Automation Conference*, pages 526–529, 1997.
20. B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 49:291–307, 1970.
21. Y. H. Kim and B. R. Moon. Lock-gain based graph partitioning. *Journal of Heuristics*, 10(1):37–57, 2004.
22. S. Kirkpatrick, Gelatt C. D. Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.
23. B. Krishnamurthy. An improved min-cut algorithm for partitioning VLSI networks. *IEEE Trans. on Computers*, 33:438–446, 1984.
24. G. Laszewski. Intelligent structural operators for the k-way graph partitioning problem. In *Fourth International Conference on Genetic Algorithms*, pages 45–52, 1991.
25. O. Martin, S. W. Otto, and E. W. Felten. Large-step markov chains for the traveling salesman problem. *Complex Systems*, 5:299–326, 1991.
26. Pablo Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report C3P 826, Caltech Concurrent Computation Program, 1989.
27. E. Rolland, H. Pirkul, and F. Glover. A tabu search for graph partitioning. *Annals of Operations Research*, 63, 1996.
28. 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.
29. D. G. Schweikert and B. W. Kernighan. A proper model for the partitioning of electrical circuits. In *Proc. 9th Design Automation Workshop*, pages 57–62, 1972.
30. A. G. Steenbeek, E. Marchiori, and A. E. Eiben. Finding balanced graph bipartitions using a hybrid genetic algorithm. In *IEEE Conference on Evolutionary Computation*, pages 90–95, 1998.
31. D. Whitley and J. Kauth. Genitor: A different genetic algorithm. In *Proceedings of Rocky Mountain Conference on Artificial Intelligence*, pages 118–130, 1988.