Hindawi Publishing Corporation
Discrete Dynamics in Nature and Society
Volume 2015, Article ID 162050, 8 pages
http://dx.doi.org/10.1155/2015/162050

Research Article

Accounting for Recent Changes of Gain in Dealing with Ties in Iterative Methods for Circuit Partitioning

Yong-Hyuk Kim[1] and Yourim Yoon[2]

[1]*Department of Computer Science & Engineering, Kwangwoon University, 20 Kwangwoon-ro, Nowon-gu, Seoul 139-701, Republic of Korea*
[2]*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 14 May 2015; Accepted 9 September 2015

Academic Editor: Alicia Cordero

In iterative methods for partitioning circuits, there is often a choice among several modules which will all produce the largest available reduction in cut size if they are moved between subsets in the partition. This choice, which is usually made by popping modules off a stack, has been shown to have a considerable impact on performance. By considering the most recent change in the potential reduction in cut size associated with moving each module between subsets, the performance of this LIFO (last-in first-out) approach can be significantly improved.

1. Introduction

Let $C = (M, N)$ be a circuit, where M is a set of modules and N is a set of nets, each of which connects two or more modules. A balanced bipartition divides the circuit into two disjoint subsets, each containing the same number of modules. Finding a balanced bipartition in which as many nets as possible connect modules in the same partition is an important problem, with applications that include the design automated of VLSI chips and multichip boards.

Many heuristics for circuit bipartitioning have been proposed [1–4], among which iterative improvement algorithms are perhaps the most successful. An iterative improvement algorithm can be deployed as a freestanding heuristic, as a framework for further refinement, or as a local optimizer forming part of a hybrid with metaheuristic methods [5–7]. These techniques rely on the quality of the iterative improvement algorithm. All such algorithms must start with an initial partition and move successive modules between subsets, using greedy decisions designed to reduce the number of nets connecting the subsets. The Kernighan-Lin algorithm (KL) [8, 9] is an example of a technique which swaps pairs of modules, and the Fiduccia-Mattheyses algorithm (FM) [10] is a faster version which moves one module at a time; there are several variants [3, 4, 11, 12].

Hagen et al. [13] identified the way in which an iterative improvement algorithm deals with the situation in which several modules have an equal claim to be moved between subsets as a significant denominator of its performance. Their experiments suggested that stack-based (LIFO) management of module moves reduced the number of connections between subsets most rapidly. In this paper, we build on this work by introducing an improved way of choosing candidate moves into an implementation of the FM algorithm for bipartitioning. We show that the proposed strategy outperforms existing strategies.

2. The Fiduccia-Mattheyses Iterative Method

The FM algorithm [10] starts with two randomly chosen subsets. It moves a module at a time from one subset to the other in an attempt to minimize the number of nets connecting both subsets, which is the cut size. Each module m is associated with a gain $g(m)$ which represents the reduction in cut size which would result from moving it to the other subset. Thus, $g(m) = |\{n \in I(m)\}| - |\{n' \in C(m)\}|$, where $C(m)$ is

```
{M1, M2} ← randomly chosen subsets such that M1 ∪ M2 = M and M1 ∩ M2 = ∅;
repeat
   compute gain g(m) for each module m ∈ M;
   M^L ← ∅; // for maintaining locked modules
   for i ← 1 to |M|
      if |M1 − M^L| = |M2 − M^L| then
         choose module mi ∈ M − M^L such that g(mi) is maximal; // module selection
      else if |M1 − M^L| > |M2 − M^L| then
         choose module mi ∈ M1 − M^L such that g(mi) is maximal; // module selection
      else
         choose module mi ∈ M2 − M^L such that g(mi) is maximal; // module selection
      M^L ← M^L ∪ {mi};
      for each module m ∈ M − M^L adjacent to module mi
         adjust gain g(m) if it is affected by moving module mi; // module update
      choose k ∈ {0, 1, ..., |M|} to maximize Σ_{i=1}^{k} g(mi);
      move all modules in the subset {m1, m2, ..., mk} to their opposite sides; // updating M1, M2
until there is no reduction in cut size;
return the partition {M1, M2};
```

Algorithm 1: Pseudocode of the Fiduccia-Mattheyses iterative method. Note that the details of the above module selection and module update for each gain bucket management are given in Algorithms 2 and 3.

```
// LIFO management
   for each module with maximum gain // module selection
      it is removed from the top of the stack in the bucket w.r.t. maximum gain;
   for each module of which the gain is updated // module update
      it is added to the top of the stack in the bucket w.r.t. the new gain;
// FIFO management
   for each module with maximum gain // module selection
      it is removed from the tail of the queue in the bucket w.r.t. maximum gain;
   for each module of which the gain is updated // module update
      it is added to the head of the queue in the bucket w.r.t. the new gain;
```

Algorithm 2: Pseudocodes of LIFO and FIFO management of gain buckets.

the set of nets connecting m to other modules exclusively in the subset to which m currently belongs and $I(m)$ is the set of nets which connect m exclusively to modules in the other subset. The module to move must be chosen on the basis of its gain and also the balance criterion between subsets; otherwise, we are likely to obtain a trivial solution in which the module with the fewest connections occupies one subset, and all the other modules occupy the other. The FM algorithm operates in a series of passes. After a module has been moved, it is locked to prevent further movements, until the end of the pass, which occurs when all modules have been locked. At the end of the pass, the algorithm backtracks to the situation obtained after the move that produced the fewest cut size. All the modules are now unlocked again and another pass starts. This process continues until a pass produces no reduction in cut size. Algorithm 1 shows the pseudocode of the FM algorithm.

3. The Significance of Ties between Modules

Heuristics such as KL and FM involve a chain of moves, for each of which the module with the highest gain is selected. A tie occurs if more than one module has the same highest gain. Hagen et al. [13] observed several ties while running FM for circuit partitioning; and Kim and Moon [11] and Yoon and Kim [14] also noticed ties occurring while running KL for graph bipartitioning. We found ties occurring regularly when running FM for circuit partitioning, and their number is tabulated, for several test circuits, in the second column of Table 2. This suggests that the way in which ties are dealt with is not a detail of the implementation but can drive the search along different paths.

There have been a number of studies on the handling of ties in the circuit partitioning problem. Krishnamurthy [12] pointed out that the arbitrary handling of ties can cause the FM [10] algorithm to perform poorly. He introduced a *gain vector*, which is effectively a form of look-ahead, and observed an improvement in performance. However, the gain vector requires a lot of memory [3], and ties may still occur. Hagen et al. [13] observed that stacking sets of modules with the same gain yield considerably better solutions than queueing or random ordering, in both the FM and Krishnamurthy algorithms. These authors suggest that a stack allows modules with the same gain to move one after another. Thus, they

```
// VLIFO management
   for each module with maximum gain // module selection
      it is removed from the top of the stack in the bucket w.r.t. maximum gain;
   for each module of which the gain is updated, //  module update
      if the gain is increased  then
         it is added to the top of the stack in the bucket w.r.t. the new gain;
      if the gain is reduced  then
         it is added to the bottom of the stack in the bucket w.r.t. the new gain;
// VFIFO management
   for each module with maximum gain // module selection
      it is removed from the tail of the queue in the bucket w.r.t. maximum gain;
   for each module of which the gain is updated, //  module update
      if the gain is increased  then
         it is added to the head of the queue in the bucket w.r.t. the new gain;
      if the gain is reduced  then
         it is added to the tail of the queue in the bucket w.r.t. the new gain;
```

Algorithm 3: Pseudocodes of VLIFO and VFIFO management of gain buckets.

proposed a variant on Krishnamurthy's gain vector which includes locality information and found that this improves performance. We also note that there have been some studies of mechanisms in algorithms for other graph partitioning problems [11, 14].

4. Favoring the Movement of Modules Related to Recently Moved Modules

A typical iterative improvement algorithm uses the gain bucket structure illustrated in Figure 1 and Algorithm 2, in which modules with the same gain are placed in the same bucket. If more than one module is in the bucket corresponding to the maximum gain, a tie occurs.

After a module has been moved, the iterative improvement algorithm updates the gains of the modules to which it is connected. Their gains may increase, reduce, or stay the same, as shown in Figure 2. We can use these updates to influence the order in which we deal with the modules which all have the maximum gain. Existing strategies simply consider the modules to be stacked or queued within a bucket. Motivated by the conjecture that it should be advantageous to move modules related to other modules that have recently been moved, we modify the corresponding LIFO (last-in first-out) and FIFO (first-in first-out) paradigms by considering changes in gain. We call these variants, explained in Figure 3 and Algorithm 3, VLIFO and VFIFO. The difference between LIFO (or FIFO) and VLIFO (or VFIFO) lies in the way in which updated modules are added to the stack (or queue) that constitutes a bucket: modules with increased gain are pushed onto the top of the stack (or the head of the queue), modules with reduced gain are inserted at the bottom of the stack (or the tail of the queue), and modules with unchanged gain remain in the same places.

5. Experiments

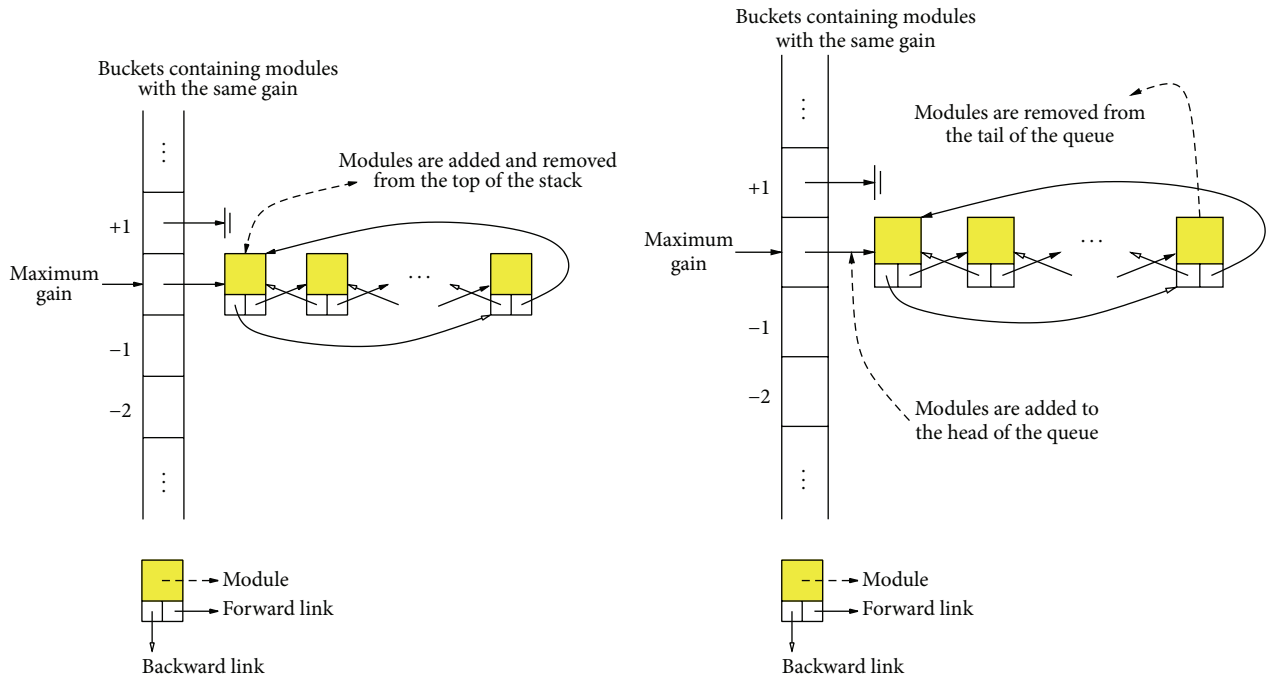
We tested the proposed algorithm on 10 benchmark circuit graphs, including seven ACM/SIGDA benchmarks. Table 1 shows the numbers of nodes, nets, and pins (connections to modules) in each circuit. In all of our experiments, we assume that the modules have unit area and constrain the partition sizes to differ by at most one module. Although a less accurate balance is usually acceptable in practical circuit partitioning, a tight constraint facilitates comparison with other partitioning techniques.

Table 1: Specification of tested circuits.

Circuit	Modules	Nets	Pins
19ks	2,844	3,282	10,547
bm1	882	903	2,910
industry2	12,637	13,419	48,158
prim1	833	902	2,908
prim2	3,014	3,029	11,219
test02	1,663	1,720	6,134
test03	1,607	1,618	5,807
test04	1,515	1,658	5,975
test05	2,595	2,750	10,076
test06	1,752	1,641	6,638

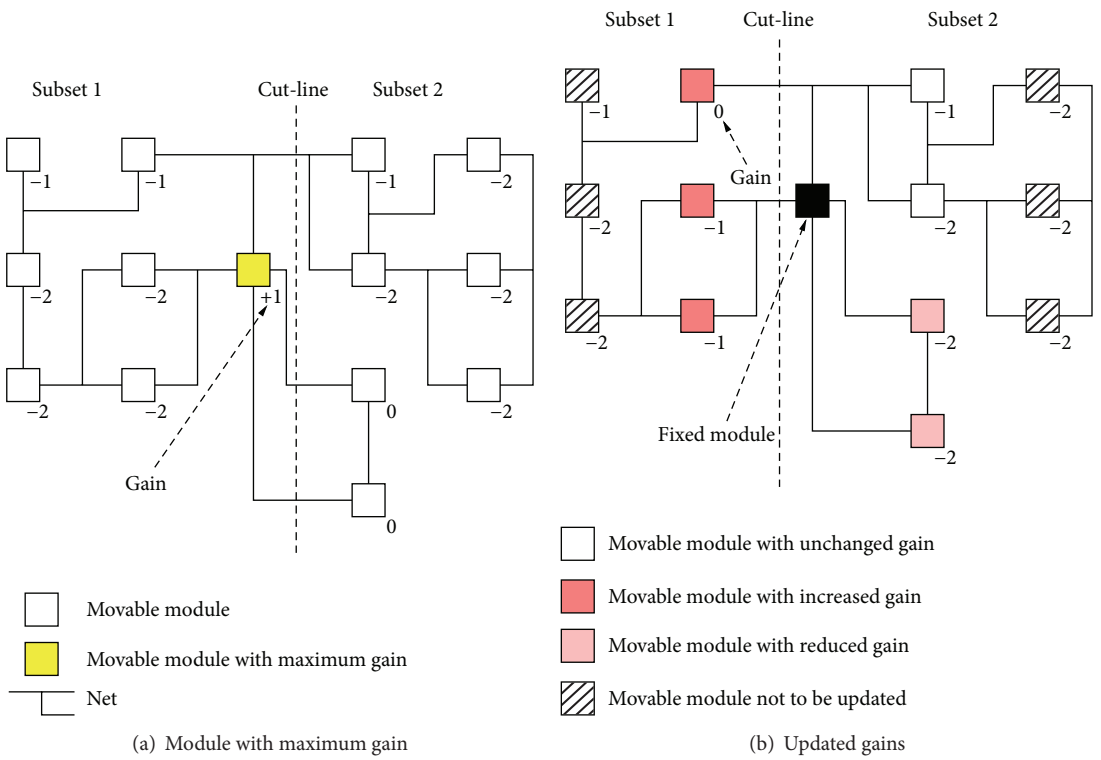
Table 2 shows the performance of five ways of dealing with ties: VFIFO, FIFO, Random, LIFO, and VLIFO. All the methods apart from Random took similar times to run; Random was slower. In line with results reported previously [13], LIFO significantly outperformed FIFO and Random on all the circuits; and Random performed better than FIFO. VLIFO gave the best results, and VFIFO gave the worst.

These results suggest that links between consecutively moved modules affect performance. We investigated this further by measuring the proportion of modules selected for moving which have just had their gain values reduced or increased (see Table 3 and Figure 4). The results, tabulated in Table 3, confirm that performance is strongly linked to the proportion of modules selected which have just had their gain increased.



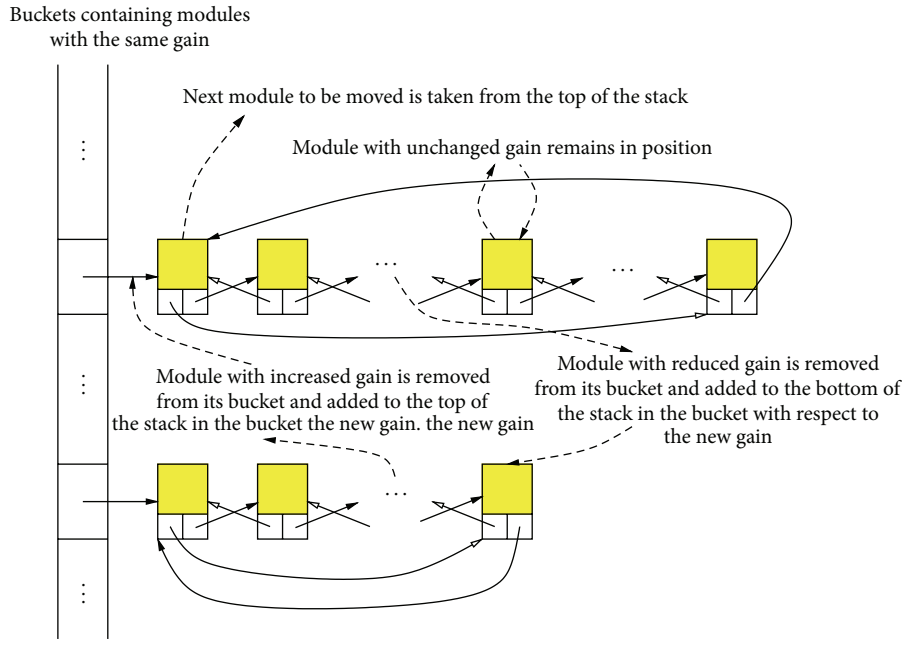
(a) LIFO: buckets are stacks (b) FIFO: buckets are queues

Figure 1: LIFO and FIFO management of gain buckets. In our implementation, circular doubly linked lists are used for efficient bucket management.

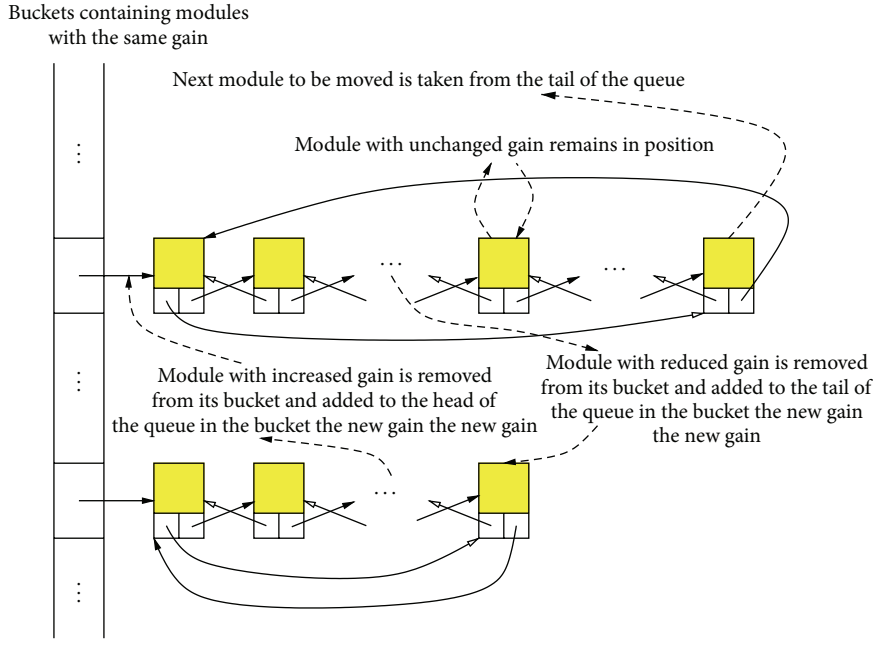


(a) Module with maximum gain (b) Updated gains

Figure 2: A movable module with the maximum gain is moved across the cut-line that separates the subsets and becomes fixed. The gains of other movable modules are then updated.



(a) VLIFO: buckets are modified stacks



(b) VFIFO: buckets are modified queues

Figure 3: VLIFO and VFIFO management of gain buckets.

6. Concluding Remarks

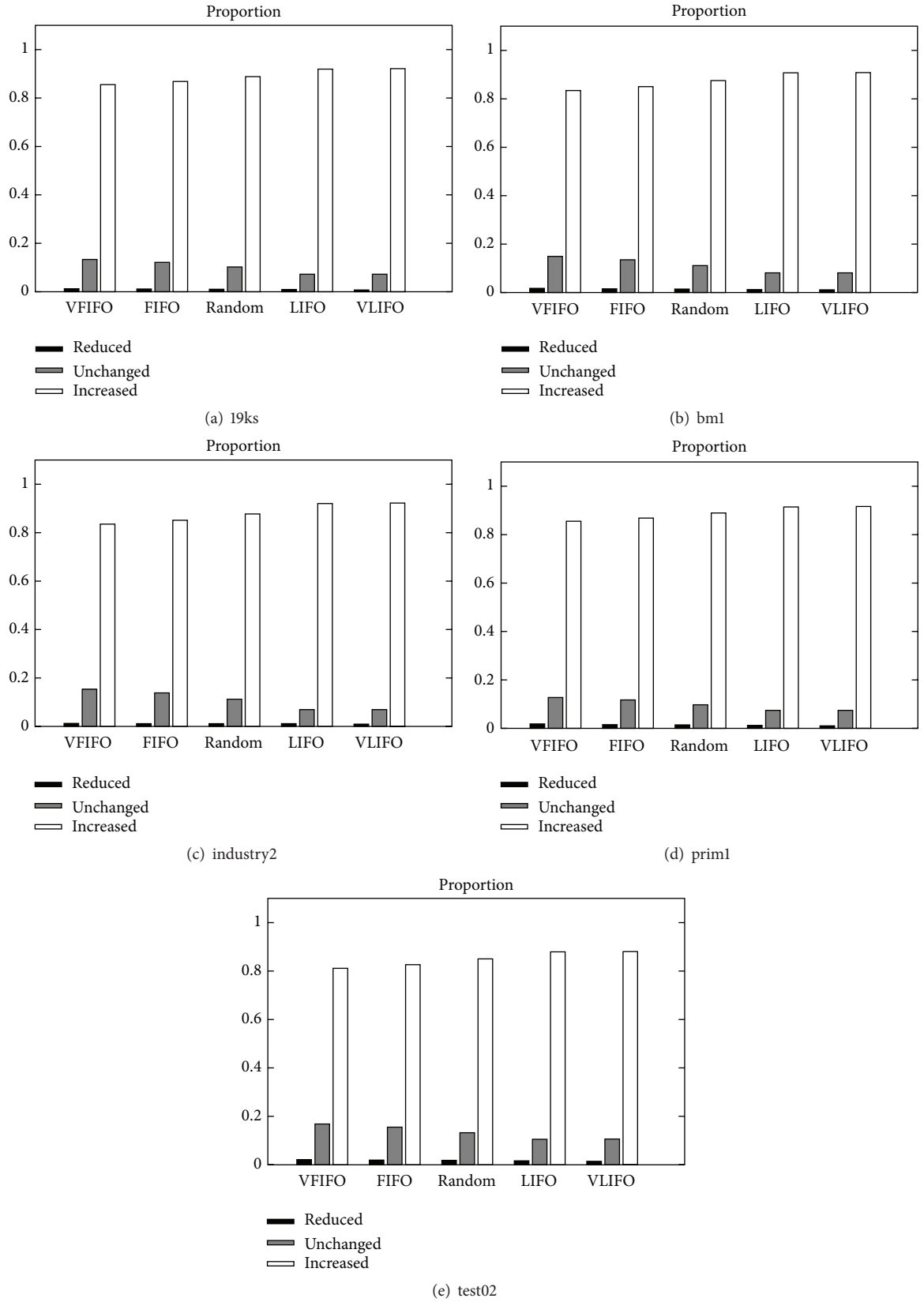
We have described an improved way of dealing with ties when modules are moved within a circuit partitioning algorithm based on iterative improvement algorithm. We modified the LIFO strategy, usually considered to be the best, by considering the most recent change in gain for each module. The application of this technique to multiway partitioning [15] is a promising avenue for future study.

Disclosure

A preliminary version of this paper appeared in the *Proceedings of the International Conference on Computer Design*, pp. 30–34, 2008.

Conflict of Interests

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



(a) 19ks

(b) bm1

(c) industry2

(d) prim1

(e) test02

Figure 4: Proportion of modules selected for moving with gain reduced, unaffected, or increased by the previous move, for five circuits (a)–(e) (data from Table 3).

Table 2: Comparison of bipartition cut sizes (averaged over 1,000 runs).

Circuit	Ties/choices (proportion)	VFIFO	FIFO	Random	LIFO	VLIFO
19ks	384.5/712.0 (54.0%)	375.3	372.2	334.3	198.0	**195.6**
bm1	118.3/221.5 (53.4%)	126.3	124.8	108.9	80.7	**79.9**
industry2	2978.9/3160.3 (94.3%)	1801.2	1750.1	1601.5	837.3	**797.5**
prim1	80.0/209.3 (38.2%)	127.0	126.0	110.9	83.7	**82.9**
prim2	623.9/754.5 (82.7%)	515.9	507.1	454.6	303.5	**300.7**
test02	211.6/416.8 (50.8%)	252.5	250.6	241.7	180.2	**178.7**
test03	191.4/402.8 (47.5%)	207.7	207.0	184.5	124.5	**123.2**
test04	132.7/379.8 (34.9%)	225.0	221.6	207.0	144.6	**142.6**
test05	290.4/649.8 (44.7%)	353.1	351.0	336.4	189.8	**189.7**
test06	172.7/439.0 (39.3%)	210.1	209.6	198.8	**94.7**	95.2

Table 3: Proportion of modules selected which have just had their gain reduced, or increased, for the five choice strategies and five test circuits (average of 1,000 runs).

Circuit	Most recent change of gain	VFIFO	FIFO	Random	LIFO	VLIFO
19ks	−	0.012	0.011	0.010	0.009	0.007
	0	0.133	0.121	0.102	0.072	0.072
	+	0.855	0.868	0.888	0.919	0.921
bm1	−	0.017	0.015	0.014	0.012	0.011
	0	0.149	0.135	0.111	0.081	0.081
	+	0.834	0.850	0.875	0.907	0.908
industry2	−	0.012	0.011	0.011	0.011	0.009
	0	0.153	0.138	0.112	0.069	0.069
	+	0.835	0.851	0.877	0.920	0.922
prim1	−	0.018	0.015	0.014	0.012	0.010
	0	0.127	0.117	0.097	0.074	0.074
	+	0.855	0.868	0.889	0.914	0.916
test02	−	0.021	0.019	0.018	0.016	0.014
	0	0.168	0.155	0.132	0.105	0.106
	+	0.811	0.826	0.850	0.879	0.880

Acknowledgments

This research was supported by the Gachon University research fund of 2015 (GCU-2015-0030). The author would like to thank Professor Byung-Ro Moon for his helpful comments and suggestions, which improved the quality of this paper.

References

[1] C. J. Alpert, J.-H. Huang, and A. B. Kahng, "Multilevel circuit partitioning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 8, pp. 655–667, 1998.

[2] C. J. Alpert and A. B. Kahng, "Recent directions in netlist partitioning: a survey," *Integration, the VLSI Journal*, vol. 19, no. 1-2, pp. 1–81, 1995.

[3] S. Dutt and W. Deng, "Probability-based approaches to VLSI circuit partitioning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 5, pp. 534–549, 2000.

[4] 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.

[5] T. N. Bui and B.-R. Moon, "GRCA: a hybrid genetic algorithm for circuit ratio-cut partitioning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 3, pp. 193–204, 1998.

[6] J.-P. Kim, Y.-H. Kim, and B.-R. Moon, "A hybrid genetic approach for circuit bipartitioning," in *Genetic and Evolutionary Computation—GECCO 2004: Genetic and Evolutionary Computation Conference, Seattle, WA, USA, June 26–30, 2004. Proceedings, Part II*, vol. 3103 of *Lecture Notes in Computer Science*, pp. 1054–1064, Springer, Berlin, Germany, 2004.

[7] S. K. Lodha and D. Bhatia, "Bipartitioning circuits using TABU search," in *Proceedings of the 11th Annual IEEE International ASIC Conference*, pp. 223–227, Rochester, NY, USA, September 1998.

[8] 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.

[9] D. G. Schweikert and B. W. Kernighan, "A proper model for the partitioning of electrical circuits," in *Proceedings of the Proceedings of the 9th Design Automation Workshop (DAC '72)*, pp. 57–62, June 1972.

[10] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proceedings of the 19th ACM/IEEE Conference on Design Automation*, pp. 175–181, IEEE, Las Vegas, Nev, USA, June 1982.

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

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

[13] L. W. Hagen, D. J.-H. Huang, and A. B. Kahng, "On implementation choices for iterative improvement partitioning algorithms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 10, pp. 1199–1205, 1997.

[14] 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.

[15] 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.