Applied Mathematics & Information Sciences
An International Journal

New Bucket Managements in Iterative Improvement Partitioning Algorithms

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

[1]Context Awareness Team, Future IT R&D Lab., LG Electronics, Seoul 137-130, Republic of Korea
[2]Department of Computer Science and Engineering, Kwangwoon University, Seoul 139-701, Republic of Korea

Received: Aug. 9, 2012; Revised Nov. 5, 2012; Accepted Dec. 1, 2012
Published online: 1 Mar. 2013

Abstract: In iterative improvement partitioning algorithms, tie-breaking in the choice of maximum-gain vertices has a great impact on the performance. We propose a new tie-breaking strategy in an iterative improvement algorithm for graph partitioning. The proposed method is simple but nevertheless performed better than other traditional techniques.

Keywords: graph partitioning, iterative improvement partitioning algorithm, bucket management, tie-breaking.

1. Introduction

Let $G = (V,E)$ be an unweighted undirected graph, where V is the set of n vertices and E is the set of e edges. A bisection $\{C_1,C_2\}$ of the graph G satisfies $C_1,C_2 \subset V$, $C_1 \cup C_2 = V$, $C_1 \cap C_2 = \phi$, and $||C_1| - |C_2|| \leq 1$. The cut size of $\{C_1,C_2\}$ is $|\{(v,w) \in E : v \in C_1, w \in C_2\}|$. The graph bisection problem is the problem of finding a bisection with minimum cut size.

A number of heuristics for graph bisection have been proposed. The iterative improvement algorithms are perhaps the most basic among these heuristics. An iterative improvement algorithm is used as a heuristic in itself, as a framework for further refinement, or as a local optimization engine in hybridization with metaheuristic methods. Thus, having a good, basic iterative improvement algorithm is crucial. An iterative improvement algorithm starts with an initial bisection and iteratively moves the vertices to the other side based on greedy decisions. The Kernighan-Lin algorithm (KL) [5] is a representative iterative improvement heuristic using pair swaps and the Fiduccia-Mattheyses algorithm (FM) [2] is its algorithmic speedup using single vertex moves. Their variants are discussed in [1], [7], [8], [9], and so on.

Recently, Hagen *et al.* [3] reported notable results concerning implementation choices for tie-breaking in an iterative improvement algorithm for hypergraph partitioning. They confirmed by experiment the effectiveness of stack-based management (LIFO strategy) of vertices to obtain the biggest gain for movement. In this paper, we go further than their study and introduce a new tie-breaking method in the KL algorithm for graph partitioning. We show that our tie-breaking strategy is superior to traditional tie-breaking strategies.

2. Importance of Tie-Breaking

In a traditional iterative improvement heuristic like KL or FM, a chain of moves are performed; for each move, a vertex with the highest gain is selected. When choosing a vertex, if more than one vertex has the same highest gain, a *tie* occurs. Hagen *et al.* [3] observed that many ties occur during a run of FM in hypergraph partitioning. We also observed a consistent phenomenon during runs of KL in the graph bisection problem. Figure 1 shows the average number of ties (from 1,000 runs) over the iterations in the first pass of KL with a graph of 1,000 vertices (G1000.2.5). The second column of Table 1 shows the overall average number of ties for each graph. This shows that quite a large number of ties occur in vertex selection, revealing the importance of tie-breaking since two different tie-breaking strategies can drive the search along totally different paths.

There have been a number of studies on tie-breaking mechanisms in the hypergraph partitioning problem. Kr-

* Corresponding author: e-mail: yhdfly@kw.ac.kr

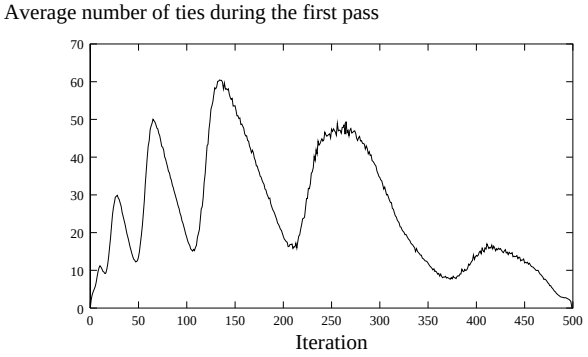

Figure 1 Average number of ties during the first pass of KL for the graph G1000.2.5

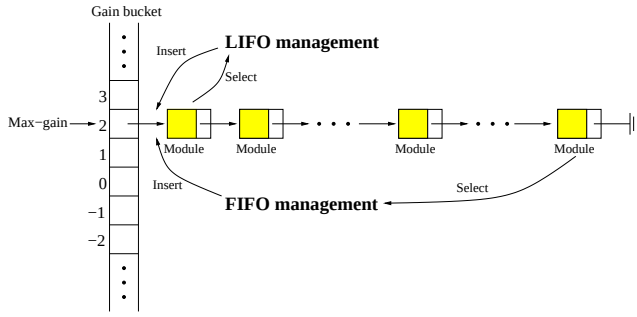

Figure 2 Illustration of the gain bucket data structure [6]

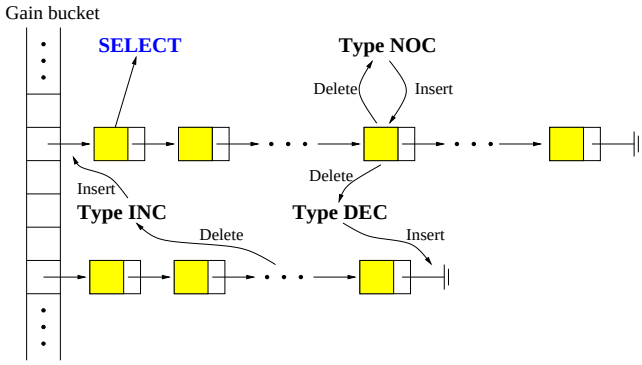

Figure 3 LIFO* bucket management in gain updates

ishnamurthy [9] pointed out that the lack of a good tie-breaking rule in the highest gain bucket causes the FM [2] algorithm to make bad choices. For tie-breaking, he introduced a *gain vector* with look-ahead and observed a performance improvement. However, his *look-ahead* algorithm requires large amounts of memory [1] and cannot be applied to the general graph partitioning problem since the gain vector is only for hypergraphs. Even with his gain vector, ties may still occur. Hagen *et al.* [3] observed that the LIFO management of gain buckets yields considerably better solutions than FIFO and Random bucket managements, for both the FM and Krishnamurthy algorithms. According to them, a possible explanation for the superiority of LIFO is that, in LIFO management, clustered vertices will tend to move sequentially. Based on this hypothesis, they proposed an alternative gain vector which includes locality information, and their variants of Krishnamurthy improved the performance.

3. A New Tie-Breaking Method

A typical iterative improvement algorithm uses the gain bucket structure as given in Figure 2 [6]. The vertices with the same gain are managed in the same bucket. Since each bucket is implemented by a linked list, we use the terms *bucket* and *bucket list* interchangeably. In this data structure, if more than one vertex is in the max-gain bucket, a tie occurs.

In an iterative improvement algorithm, after each vertex is moved, the gains of its adjacent vertices are updated. There are three types of gain updates in the KL algorithm: gain increases (type A), no change (type B), and gain decreases (type C). Previous tie-breaking methods do not consider this classification. So, the LIFO strategy, the traditional champion, inserts every updated node at the head of the corresponding bucket list and selects the node at the head of the bucket list. We change this according to our classification. We denote our variants of LIFO and FIFO by LIFO* and FIFO*, respectively. In LIFO*, the means by which the algorithm selects the vertices is the same as in LIFO. The difference between LIFO and LIFO* lies in the vertex insertion. Each vertex of type A is inserted at the head of the bucket. Each vertex of type C is inserted at the tail of the bucket. There is no change for the vertices of type B. In FIFO*, the selection is the same as that of FIFO and the insertion is the same as that of LIFO*. Figure 3 and Figure 4 show the proposed bucket management in gain updates. Our idea reflects the conjecture that it might be advantageous to move vertices having a close relationship with recently moved vertices.

4. Comparison of Tie-Breaking Strategies

We conducted tests on the eight graphs that were used in [4], [7], [8], [10]. The different classes of graphs are briefly described below.

– G$n.d$: A random graph on n vertices, where an edge is placed between any two vertices with probability p independent of all other edges. The probability p is chosen so that the expected vertex degree, $p(n-1)$, is d.
– 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. There is an edge

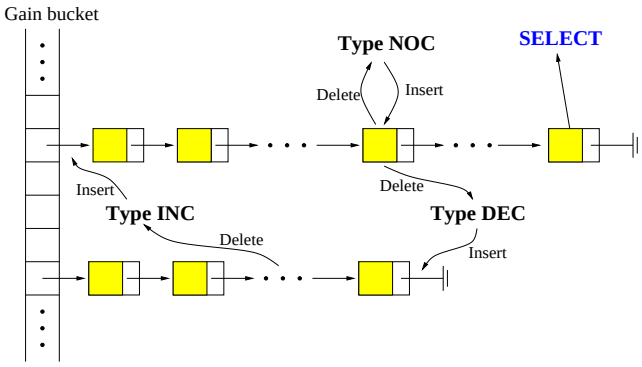

Figure 4 FIFO* bucket management in gain updates

between two vertices if their Euclidean distance is t or less, where $d = n\pi t^2$ is the expected vertex degree.
– cat.n: A caterpillar graph on n vertices, with each vertex having six legs. It is constructed by starting with a straight line (called the spine), where each vertex has degree two except the outermost vertices. Each vertex on the spine is then connected to six new vertices, the legs of the caterpillar. rcat.n is a caterpillar graph with n vertices, where each vertex on the spine has $\sqrt{n}$ legs. All caterpillar graphs have an optimal cut size of 1.
– grid$n.b$: A grid graph on n vertices and whose optimal cut size is known to be b. w-grid$n.b$ denotes the same grid but the boundaries are wrapped around.

Table 1 shows the performance of the five tie-breaking rules: FIFO*, FIFO, Random, LIFO, and LIFO*. The names reflect the ways they manage the highest-gain bucket list. Four of the methods with the exception of Random took a comparable amount of time; Random was slower than the others. Similarly to hypergraph partitioning [3], LIFO significantly outperformed FIFO and Random for most graphs except for the caterpillar graphs. For the caterpillar graphs, FIFO led LIFO. Overall, Random was dominated by LIFO but performed better than FIFO.

LIFO* was the clear winner among all the rules. By contrast, FIFO* was overall dominated by the others. This result implies that the locality of vertices plays an important role in the performance. To confirm this, for the five rules, we measured the rate of selected vertices in the max-gain buckets according to their latest updated status (Table 2). Obviously by definition, the sum of three rates is always 1. We performed 3,000 runs for each result of Table 2. One can observe that the performance is strongly dependent on the rate of type A. The larger the rate of type A, the better the performance. We saw the opposite phenomenon *only* in a caterpillar graph (rcat.5114). Interestingly enough, only the graph rcat.5114 showed differing performance trends related to tie-breaking rules. Overall, one can see that the performance of the five tie-breaking strategies correlates with their corresponding rate values. This is a possible explanation for the performance difference among the different tie-breaking rules.

5. Summary

In this paper, we described an improved tie-breaking strategy for an iterative improvement partitioning algorithm. We improved the traditional champion, namely the LIFO strategy, by considering the types of gain updates. Application to the VLSI circuit partitioning problem remains as a promising avenue for future study.

Acknowledgement

The present Research has been conducted by the Research Grant of Kwangwoon University in 2013. This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(2012-0001855).

References

[1] S. Dutt and W. Deng. A probability-based approach to VLSI circuit partitioning. In *Proceedings of the Design Automation Conference*, pages 100–105, June (1996).
[2] C. Fiduccia and R. Mattheyses. A linear time heuristics for improving network partitions. In *Proceedings of the 19th ACM/IEEE Design Automation Conference*, pages 175–181, (1982).
[3] L. Hagen, J. H. Huang, and A. B. Kahng. On implementation choices for iterative improvement partitioning algorithms. *IEEE Transactions on Computer-Aided Design*, **16**(10):1199–1205, (1997).
[4] 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*, pages 1209–1215, (2006).
[5] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, **49**:291–307, Feb. (1970).
[6] Y.-H. Kim. Improved implementation choices for iterative improvement partitioning algorithms on circuits. In *Proceedings of the International Conference on Computer Design*, pages 30–34, (2008).
[7] Y.-H. Kim and B.-R. Moon. A hybrid genetic search for graph partitioning based on lock gain. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 167–174, (2000).
[8] Y.-H. Kim and B.-R. Moon. Lock-gain based graph partitioning. *Journal of Heuristics*, **10**(1):37–57, (2004).
[9] B. Krishnamurthy. An improved min-cut algorithm for partitioning VLSI networks. *IEEE Transactions on Computers*, **C-33**:438–446, (1984).

Table 1 Comparison of Bisection Cut Sizes

Graph	Ties†	FIFO*‡	FIFO‡	Random‡	LIFO‡	LIFO*‡
G1000.05	13.25	516.82	516.77	508.86	501.24	**500.64**
G1000.20	3.88	3495.41	3495.30	3489.39	3482.95	**3482.55**
U2000.05	10.93	221.94	221.13	191.07	161.44	**160.25**
U5000.10	6.24	1178.72	1173.66	1072.81	959.89	**951.78**
cat.5252	552.73	256.26	250.85	247.46	254.15	**245.78**
rcat.5114	747.82	**166.49**	167.74	177.99	183.46	183.73
grid5000.50	16.66	88.23	88.10	74.30	**56.18**	56.75
w-grid5000.100	25.58	176.04	164.91	132.59	123.37	**123.33**

† Average over 500 runs.
‡ Average over 3,000 runs.

Table 2 Selection Rate

Graph	Type	FIFO*	FIFO	Random	LIFO	LIFO*
rcat.5114	A	0.363	0.362	0.355	0.354	0.354
	B	0.523	0.524	0.523	0.523	0.523
	C	0.114	0.114	0.122	0.123	0.123
Average of the other graphs	A	0.840	0.841	0.855	0.867	0.868
	B	0.136	0.137	0.127	0.113	0.114
	C	0.023	0.022	0.018	0.019	0.018

Average over 3,000 runs.

[10] K. Seo, S. Hyun, and Y.-H. Kim. A spanning tree-based encoding of the MAX CUT problem for evolutionary search. In *Proceedings of the 12th International Conference on Parallel Problem Solving From Nature - Lecture Notes in Computer Science*, volume **7491**, pages 510–518, (2012).

Yourim Yoon received the B.E. degree in computer engineering and the Ph.D. degree in computer science and engineering from Seoul National University, Seoul, Korea, in 2003 and 2012, respectively. Since March 2012, she has been a senior research engineer at Future IT R&D Lab. in LG Electronics, Seoul, Korea. Her research interests include optimization theory, machine learning, combinatorial optimization, evolutionary computation, discrete mathematics, operations research, smart grid, and sensor networks. She served as a reviewer for BIC-TA 2007, BMIC 2011, IEEE TEC, and TIIS.

Yong-Hyuk Kim received the B.S. degree in computer science and the M.S. and Ph.D. degrees in computer science and engineering from Seoul National University (SNU), Seoul, Korea, in 1999, 2001, and 2005, respectively. From March 2005 to February 2007, he was a Postdoctoral Scholar in SNU and also a research staff member at the Inter-University Semiconductor Research Center in SNU. Since March 2007, he has been a professor at Department of Computer Science and Engineering in Kwangwoon University, Seoul, Korea. His research interests include algorithm design/analysis, discrete mathematics, optimization theory, combinatorial optimization, evolutionary computation, operations research, data/web mining, and sensor networks. Dr. Kim has served as an Editor of TIIS journal in 2010-2013, a Committee Member of GECCO 2005-2006 & IEEE CEC 2009-2012, and a reviewer for journals (IS, TKDE, TPDS, TC, TSE, TEC) of IEEE since 2003.