AN EFFECTIVE REFINEMENT ALGORITHM BASED ON MULTILEVEL PARADIGM FOR GRAPH BIPARTITIONING

Ming Leng, Songnian Yu, Yang Chen
School of Computer Engineering and Science,Shanghai University, Shanghai, PR China.
Email:lmsly@263.net

Abstract: The min-cut bipartitioning problem is a fundamental partitioning problem and is NP-Complete. It is also NP-Hard to find good approximate solutions for this problem. In this paper, we present a new effective refinement algorithm based on multilevel paradigm for graph bipartitioning. The success of our algorithm relies on exploiting both new Tabu search strategy and boundary refinement policy. Our experimental evaluations on 18 different graphs show that our algorithm produces excellent solutions compared with those produced by MeTiS that is a state-of-the-art partitioner in the literature.

Key words: min-cut, graph bipartitioning, multilevel paradigm, Tabu search

1. INTRODUCTION

Partitioning is a fundamental problem with extensive applications to many areas, including VLSI design [1], information retrieval [2], parallel processing [3], computational grids [4] and data mining [5]. The *min-cut bipartitioning* problem is a fundamental partitioning problem and is NP-Complete [6]. It is also NP-Hard [7] to find good approximate solutions for this problem. Because of its importance, the problem has attracted a considerable amount of research interest and a variety of algorithms have been developed over the last thirty years [8],[9]. The survey by Alpert and Kahng [1] provides a detailed description and comparison of various such

Please use the following format when citing this chapter:
Leng, Ming, Yu, Songnian, Chen, Yang, 2006, in International Federation for Information Processing (IFIP), Volume 207, Knowledge Enterprise: Intelligent Strategies In Product Design, Manufacturing, and Management, eds. K. Wang, Kovacs G., Wozny M., Fang M., (Boston: Springer), pp. 294-303.

schemes which include *move-based* approaches, *geometric representations*, *combinatorial* formulations, and *clustering* approaches.

As problem sizes reach new levels of complexity recently, a new class of graph partitioning algorithms have been developed that are based on the multilevel paradigm. The multilevel graph partitioning schemes include three phases [10], [11],[12], [13]. The *coarsening phase* is to reduce the size of the graph by collapsing vertex and edge until its size is smaller than a given threshold. The *initial partitioning phase* is to compute initial partition of the coarsest graph. The *uncoarsening phase* is to successively project the partition of the smaller graph back to the next level finer graph while applying an iterative refinement algorithm. In this paper, we present a new effective refinement algorithm based on multilevel paradigm which is integrated with Tabu search [14] for refining the partition. Our work is motivated by the multilevel partitioners of Saab [13] who promotes locked vertex to free by introducing two buckets for per side of the partition and Karypis [10], [11],[12] who proposes a boundary refinement algorithm and supplies **MeTiS** [10], distributed as open source software package for partitioning unstructured graphs. We test our algorithm on 18 graphs that are converted from the hypergraphs of the ISPD98 benchmark suite [15]. Our experiments show that our algorithm produces partitions that are better than those produced by **MeTiS** in a reasonable time.

The rest of the paper is organized as follows. Section 2 provides some definitions, describes the notation that is used throughout the paper and the *min-cut bipartitioning problem*. Section 3 briefly describes the motivation behind our algorithm. Section 4 presents an effective refinement algorithm based on multilevel paradigm for graph *bipartitioning*. Section 5 experimentally evaluates the algorithm and compares it with **MeTiS**. Finally, Section 6 provides some concluding remarks and indicates the directions for further research.

2. MATHEMATICAL DESCRIPTION

A graph $G=(V,E)$ consists of a set of vertices V and a set of edges E such that each edge is a subset of two vertices in V. Throughout this paper, n and m denote the number of vertices and edges respectively. The vertices are numbered from *1* to n and each vertex $v \in V$ has an integer weight $s(v)$. The edges are numbered from *1* to m and each edge $e \in E$ has an integer weight $w(e)$. A decomposition of a graph V into two disjoint subsets V_1 and V_2, such that $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \varnothing$, is called a *bipartitioning* of V. Let

$S(A)=\sum_{v\in A}S(v)$ denote the size of a subset $A\subseteq V$. Let $ID(v)$ be denoted as v's *internal degree* and is equal to the sum of the edge-weights of the adjacent vertices of v that are in the same side of the partition as v, and v's *external degree* denoted by $ED(v)$ is equal to the sum of edge-weights of the adjacent vertices of v that are in the different side of the partition. The *cut* of a *bipartitioning* $P=\{V_1,V_2\}$ is the sum of weights of edges which contain two vertices in V_1 and V_2 respectively, such that $cut(P)=\sum_{e\cap V_1\neq\varnothing,e\cap V_2\neq\varnothing}W(e)$. Naturally, vertex v belongs at boundary if and only if $ED(v)>0$ and the *cut* of P is also equal to $0.5\sum_{v\in V}ED(v)$.

Given a balance constraint *r*, the *min-cut bipartitioning problem* seeks a solution $P=\{V_1,V_2\}$ that minimizes $cut(P)$ subject to $(1-r)S(V)/2\le S(V_1),S(V_2)\le(1+r)S(V)/2$. A *bipartitioning* is *bisection* if *r* is as small as possible. If the edges are ignored, the problem is the set partitioning problem which is NP-Complete [6]. Therefore, unless all vertices have unit sizes, it is NP-Hard to find just a bisection of arbitrary cost of graph. Most existing partitioning algorithms are heuristics in nature and they seek to obtain reasonably good solutions in a reasonable amount of time. Kernighan and Lin (KL) [8] proposed a heuristic algorithm for partitioning graphs, which requires $O(n^2\log(n))$ computation time. The KL algorithm is an iterative improvement algorithm that consists of making several improvement passes. It starts with an initial bipartition $\{A,B\}$ and tries to improve it by every pass. A pass consists of the identification of two subsets of vertices $A'\subset A$ and $B'\subset B$ of equal size such that can lead to an improved partition if the vertices in the two subsets switch sides, that is to say, $\{(A-A')\cup B',(B-B')\cup A'\}$ is a better bipartition than $\{A,B\}$. Fiduccia and Mattheyses (FM) [9] proposed a fast heuristic algorithm for bisecting a weighted graph by introducing the concept of cell *gain* into KL algorithm. The previously unmoved vertex $v\in A$ (or $v\in B$) with highest *gain* that computed by $cut(\{A,B\})-cut(\{A-v,B+v\})$ is moved from A to B.

3. MOTIVATION

Tabu search has its antecedents in methods designed to cross boundaries of feasibility or local optimality standardly treated as barriers, and to systematically impose and release constraints to permit exploration of otherwise forbidden regions. Tabu search is a "higher level" heuristic procedure for solving optimization problems, designed to guide to other methods to escape the trap of local optimality [14].

During each pass of KL and FM, they have the same restriction that each vertex can move only once and the restriction may prevent the exploration of certain promising region of the search space. In the terminology of Tabu Search, the KL and FM strategy is a simple form of Tabu search without aspiration criterion whose prohibition period is fixed at n. In [13], Saab introduces the concept of *forward move step* and *restore balance step* for refining the partition and adopts aspiration criterion to allow a locked vertex to become free in the same passes of ALG2 algorithm. As a consequence of allowing locked vertices to move in an iterative pass, ALG2 algorithm can explore a certain promising regions of the search space. However, Saab limits the vertices *move-direction* of two kinds of steps in A→B and B→A respectively and is obliged to adopt two different aspiration criterions for two kinds of steps respectively to avoid cycling issue.

In both FM and ALG2 refinement algorithms, we have to assume that all vertices are free and insert the *gain* of all vertices in free bucket. However, most of *gain* computations are wasted since most of vertices moved by FM and ALG2 refinement algorithms are boundary vertices that straddle two sides of the partition. As opposed to the non-boundary refinement algorithms, the cost of performing multiple passes of the boundary refinement algorithms is small since only boundary vertices are inserted into the bucket as needed and no work is wasted. In [11], the boundary KL (BKL) refinement algorithm presented by Karypis swaps only boundary vertices and is much faster variation of the KL algorithm.

In this paper, we present a boundary Tabu search (BTS) refinement algorithm that combines the Tabu search theory with boundary refinement policy. It has three distinguishing features which are different from ALG2 algorithm. First, we initially insert into the free bucket the gains for only boundary vertices. Second, we remove the above limitation by introducing the conception of *step-status* and *move-direction*. Finally, we derive aspiration criterion from lots of experiments that is simpler than that of ALG2 algorithm.

4. BTS: A NEW EFFECTIVE REFINEMENT ALGORITHM

BTS introduces the conception of *move-direction* and *step-status* that consists of both *forward-move* and *restore-balance*. Given an input partition Q and balance tolerance t, if the input partition Q satisfies the balance tolerance t, current *step-status* is *forward-move* and we initially choose to move a vertex from the side whose bucket contains a highest *gain* vertex

among all boundary vertices. Otherwise current *step-status* is *restore-balance* and we choose to move a vertex from the larger side of the partition. As BTS enters into the next step, if new partition satisfies the balance tolerance *t*, current *step-status* is *forward-move* and we choose the last *move-direction* to apply the current step, else current *step-status* is *restore-balance* and the *move-direction* of current step starts from the larger side of the partition. The strategy of BTS eliminates the limitation of *move-direction* of steps and its goal is to increase the chances of vertices that are closely connected migrating together from one side of the partition to the other by allowing moving sequences of vertices at one pass from one side of the partition to the other, although the highest *gain* maybe negative in a sequences of steps.

BTS uses free and tabu buckets to fast storage and retrieval the gains of free and tabu vertices respectively. At the beginning of a pass, all vertices are free and the internal and external degrees of all vertices are computed and two free buckets are inserted the gains of boundary vertices of two sides respectively that are computed by $ED(v)-ID(v)$. After we move a vertex v, the gains of itself and its neighboring vertices should be changed. First, we must lock the vertex v by deleting its original *gain* from bucket and insert its new *gain* (negative value of original *gain*) into the tabu bucket of the other side. Second, we update the gains of the neighboring vertices of vertex v. If any of these neighboring vertices becomes a boundary vertex due to the move of vertex v , we insert its *gain* into the free bucket of side in which it locates. If any of these neighboring vertices becomes a non-boundary vertex due to the move of vertex v, we delete its original *gain* from bucket that maybe free or tabu. If any of these neighboring vertices is already a boundary free vertex, we only update its *gain* in free bucket. If any of these neighboring vertices is a boundary locked vertex, we must delete its original *gain* from tabu bucket and insert its new *gain* into the free bucket of side in which it locates. In the terminology of Tabu Search, the tabu restriction forbids moving vertices which are designated as tabu status and the prohibition period of tabu vertex can be changed dynamically and decided by the above promotion rule. The purpose of promotion rule is to increase their chances of following neighbors to the other side of the partition and to allow the chance for the movement of a cluster from one side of the partition to the other.

The remains problem in BTS is how to select a vertex to move in current step. The vertex to move must be selected from free bucket or tabu bucket of the side that is start point of the *move-direction* of current step. When the current *step-status* is *forward-move*, the next vertex to move is the highest *gain* vertex in the free bucket if it is not empty. Otherwise, the highest *gain* vertex in tabu bucket is chosen to move. If the current *step-status* is *restore-*

balance, the next vertex to move is the highest *gain* vertex in both free and tabu buckets. It is not possible to run out of moves as long as tolerance t satisfies $0 < t < 1$. The choice rule of Tabu search is to select the highest *gain* vertex in free bucket and the aspiration criterion we have selected to override the tabu restriction is simple criterion that allows tabu vertex as candidate of vertex to move in the current step if current *step-status* is *restore-balance* or the free bucket is empty.

Table 1. The pseudocode of the BTS algorithm

```
BTS (initial partition Q, balance tolerance t, Total Moves k){
1  BEST=P;
2  COUNTER=0;
3  compute the internal and external degrees of all vertices;
4  compute the gains of boundary vertices of two sides;
5  insert the gains of boundary vertices in free bucket respectively;
6  while COUNTER <= k do {
7     decide the step-status and move-direction of the current step;
8     select the vertex to move by choice rule and aspiration criterion;
9     move the vertex and lock it;
10    original cut minus its original gain as the cut of new partition;
11    update the internal and external degrees of its neighboring vertices;
12    update the gains of its neighboring vertices by promotion rule;
13    if (the cut is minimum and satisfies balance constraints) then
14       BEST=P;
15    end if
16    COUNTER = COUNTER +1;
17  }end while
18 }
```

The pseudocode of the BTS algorithm is given in Table. 1. The while loop (lines 6-17) of BTS is iterated as long as improvements can be made and it is necessary in BTS that setting an upper limit on the parameter k. Because Tabu search aggressively selects the best admissible vertex based on the tabu restriction and aspiration criterion, it must examine and compare a number of boundary vertices by the bucket that allows to storage, retrieval and update the gains of vertices very quickly. It is important to obtain the efficiency of BTS by using the bucket with the last-in first-out (LIFO) scheme, as Tabu search memory structure, can enforce the "locality" in the choice of vertices to move. The internal and external degrees of all vertices, as complementary Tabu search memory structures, help BTS to facilitate computation of vertex *gain* and judgement of boundary vertex.

5. EXPERIMENTAL RESULTS

We use the 18 graphs in our experiments that are converted from the

hypergraphs of the ISPD98 benchmark suite [15] and rang from 13,000 to 210,000 vertices. Each benchmark comes with 3 files, a .net file, a .are file and a .netD file. We convert hyperedges of .netD file into edges by the rule that every subset of two vertices in hyperedge can be seemed as edge. We create the edge with unit weight if the edge that connects two vertices didn't exist, else add unit weight to the weight of the edge. Next, we get the weights of vertices from .are file. Finally, we store 18 edge-weighted and vertex-weighted graphs in graph format of **MeTiS** [10]. The characteristics of these graphs are shown in Table 2.

Table 2. The characteristics of 18 graphs to evaluate our algorithm

benchmark	vertices	hyperedges	edges
ibm01	12752	14111	109183
ibm02	19601	19584	343409
ibm03	23136	27401	206069
ibm04	27507	31970	220423
ibm05	29347	28446	349676
ibm06	32498	34826	321308
ibm07	45926	48117	373328
ibm08	51309	50513	732550
ibm09	53395	60902	478777
ibm10	69429	75196	707969
ibm11	70558	81454	508442
ibm12	71076	77240	748371
ibm13	84199	99666	744500
ibm14	147605	152772	1125147
ibm15	161570	186608	1751474
ibm16	183484	190048	1923995
ibm17	185495	189581	2235716
ibm18	210613	201920	2221860

We implement the BTS refinement algorithm in ANSI C and integrate it with the leading edge partitioner **MeTiS**. In the evaluation of BTS refinement algorithm, we must make sure that the results produced by BTS refinement algorithm can be easily compared against those produced by **MeTiS**. First, we use the same balance constraint r and random seed in every comparison. Second, we select the sorted heavy-edge matching (SHEM) algorithm during *coarsening phase* because of its consistently good behavior in **MeTiS**. Third, we adopt the greedy graph growing partition (GGGP) algorithm during *initial partitioning phase* that consistently finds smaller edge-cuts than others algorithms. Finally, we select the BKL algorithm to compare with BTS algorithm during *uncoarsening and refinement phase* because BKL can produce smaller edge-cuts when coupled with SHEM algorithm. These measures are sufficient to guarantee that our experimental evaluations are not biased in any way.

The quality of partitions produced by our BTS refinement algorithm and those produced by **MeTiS** are evaluated by looking at two different quality measures, which are the minimum *cut* (Mincut) and the average *cut* (AveCut). To ensure the statistical significance of our experimental result, two measures are obtained in twenty runs whose random seed is different with each other. For all experiments, we use a 49-51 *bipartitioning* balance constraint by setting r to 0.02. Next, we pass the same *initial partition*, result of *initial partitioning* phase, to BTS and BKL algorithms in every comparison that guarantees a fair evaluation. Furthermore, we set the number of vertices of the current level graph as the value of parameter k and 5% as the value of parameter t.

Table 3. Min-cut bipartitioning results with up to 2% deviation from exact bisection

benchmark	Metis		BTS		BTS/Metis	
	Mincut	AveCut	Mincut	AveCut	Mincut	AveCut
ibm01	517	1091	506	1081	0.979	0.991
ibm02	4268	11076	4184	8410	0.980	0.759
ibm03	10190	12353	7548	10299	0.741	0.834
ibm04	2273	5716	2336	2789	1.028	0.488
ibm05	12093	15058	12074	14597	0.998	0.969
ibm06	7408	13586	2470	11692	0.333	0.861
ibm07	3219	4140	2867	3449	0.891	0.833
ibm08	11980	38180	12845	18470	1.072	0.484
ibm09	2888	4772	2917	3982	1.010	0.834
ibm10	10066	17747	6359	9743	0.632	0.549
ibm11	2452	5095	2406	3593	0.981	0.705
ibm12	12911	27691	12383	19301	0.959	0.697
ibm13	6395	13469	4582	8702	0.716	0.646
ibm14	8142	12903	7608	10359	0.934	0.803
ibm15	22525	46187	12112	36590	0.538	0.792
ibm16	11534	22156	10111	14787	0.877	0.667
ibm17	16146	26202	14792	20620	0.916	0.787
ibm18	15470	20018	14903	19199	0.963	0.959
average	---	---	---	---	0.863	0.758

Table 3 presents *min-cut bipartitioning* results allowing up to 2% deviation from exact bisection and Fig. 1 illustrates the Mincut and AveCut comparisons of two refinement algorithms on 18 graphs. As expected, the BTS refinement algorithm reduces the AveCut by 1% to 52% and reaches 24% average AveCut improvement. Although the BTS refinement algorithm produces partition whose Mincut is up to 7.2% worse than that of **MeTiS** on some benchmarks, we still obtain 14% average Mincut improvement and between -7% and 67% improvement in Mincut. All evaluations that twenty runs of two algorithms on 18 graphs are run on an 1800MHz AMD Athlon2200 with 512M memory and can be done in half an hour.

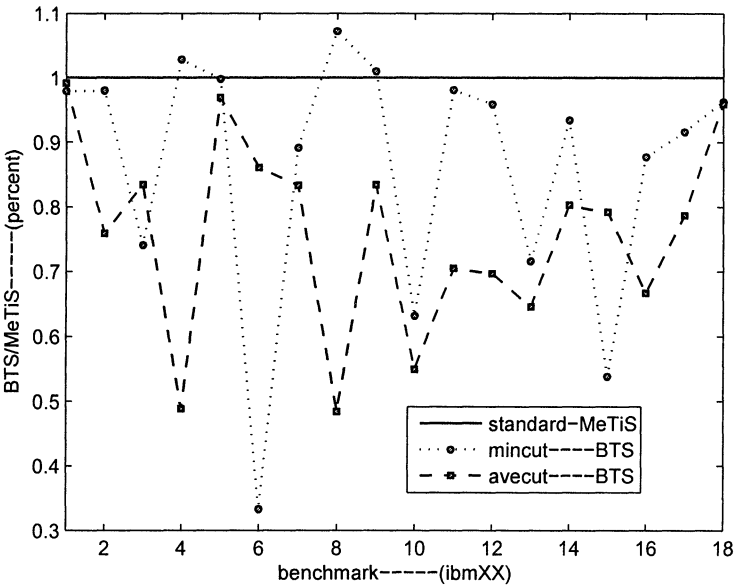

Figure 1. The Mincut and AveCut comparisons of two refinement algorithms on 18 graphs

6. CONCLUSIONS

In this paper, we have presented a new refinement algorithm based on multilevel paradigm. The success of our BTS algorithm relies on exploiting both new Tabu search strategy and boundary refinement policy. We obtain excellent *bipartitioning* result compared with those produced by **MeTiS**. Although it has the ability to find cuts that are lower than the result of **MeTiS** in a reasonable time, there are several ways in which this algorithm can be improved. We raise three questions about possible improvement below. The first question is how to find an optimal value for balance tolerance t. The second question, how to find an optimal value for k, is similar with Saab's question about r [13] because we observe that larger values of k lead to better partitions at the expense of a proportional increase in running time and the improvement in the quality of partitions in not linearly related to k. In the Mincut evaluation of benchmark ibm04, ibm08 and ibm09, the BTS refinement algorithm is 2.8%, 7.2%, 1% worse than **MeTiS** respectively. This shows that a good initial partition has been lost in the *uncoarsening phase* before it can yield a good bisection for the original

graph. Therefore, the third question is how to guarantee find good approximate solutions by setting appropriate value for k and t.

7. ACKNOWLEDGMENTS

This work was supported by the Science Foundation of Shanghai Municipal Commission of Science and Technology, grant No. 00JC14052, and by "SEC E-Institute: Shanghai High Institutions Grid" project.

8. REFERENCES

1. Alpert, C.J., Kahng, A.B.: Recent Directions in Netlist Partitioning. Integration, *the VLSI Journal,* Vol. 19 (1995) 1-18
2. Zha, H., He, X., Ding, C., Simon, H., Gu, M.: Bipartite graph partitioning and data clustering. *Proc. ACM Conf Information and Knowledge Management* (2001)
3. Hendrickson, B., Leland, R.: An Improved Spectral Graph Partitioning Algorithm for Mapping Parallel Computations. *SIAM Journal on Scientific Computing,* Vol. 16 (1995) 452-469
4. Wanschoor, R., Aubanel, E.: Mesh Partitioning for Computational Grids. *Proc. 2nd Annual Conf on Communication Networks and Services Research* (2004)
5. Ding, C., Xiaofeng, H., Hongyuan, Z., Ming, G., Simon, H.: A Min-Max Cut Algorithm for Graph Partitioning and Data Clustering. *Proc. IEEE Conf Data Mining* (2001) 107-114
6. Garey, M.R., Johnson, *D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness.* WH Freeman New York (1979)
7. Bui, T., Leland, C.: Finding Good Approximate Vertex and Edge Partitions Is NP-Hard. *Information Processing Letters*, Vol. 42 (1992) 153-159
8. Kernighan, B.W., Lin, S.: An Efficient Heuristic Procedure for Partitioning Graphs. *Bell System Technical Journal,* Vol. 49 (1970) 291-307
9. Fiduccia, C., Mattheyses, R.: A Linear-Time Heuristics for Improving Network Partitions. *Proc. 19th Design Automation Conf* (1982) 175-181
10. Karypis, G., Kumar, V.: MeTiS 4.0: Unstructured Graphs partitioning and sparse matrix ordering system. *Technical Report, Department of Computer Science, University of Minnesota* (1998) Available on the WWW at URL http://www.cs.umn.edu/~metis
11. Karypis, G., Kumar, V.: A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing,* Vol. 20 (1999) 12.
12. Selvakkumaran, N., Karypis, G.: Multi-Objective Hypergraph Partitioning Algorithms for Cut and Maximum Subdomain Degree Minimization. *IEEE Trans. Computer Aided Design,* Vol. 25 (2006) 504-517
13. Saab, Y.G.: An Effective Multilevel Algorithm for Bisecting Graphs and Hypergraphs. *IEEE Trans. Computers,* Vol. 53 (2004) 641-653
14. Glover, F., Manuel, L.: *Tabu search: Modern heuristic Techniques for Combinatorial Problems.* Blackwell Scientific Publications Oxford (1993) 70-150
15. Alpert, C.J.: The ISPD98 Circuit benchmark suite. *Proc. Intel Symposium of Physical Design* (1998) 80-85