An Effective Multi-level Algorithm Based on Simulated Annealing for Bisecting Graph

Lingyu Sun[1] and Ming Leng[2]

[1] Department of Computer Science,
Jinggangshan College, Ji'an, PR China 343009
[2] School of Computer Engineering and Science,
Shanghai University, Shanghai, PR China 200072
sunlingyu@jgsu.edu.cn,lengming@shu.edu.cn

Abstract. Partitioning is a fundamental problem in diverse fields of study such as knowledge discovery, data mining, image segmentation and grouping. The min-cut bipartitioning problem is a fundamental graph partitioning problem and is NP-Complete. In this paper, we present an effective multi-level algorithm based on simulated annealing for bisecting graph. The success of our algorithm relies on exploiting both the simulated annealing procedure and the concept of the graph core. Our experimental evaluations on 18 different graphs show that our algorithm produces encouraging solutions compared with those produced by MeTiS that is a state-of-the-art partitioner in the literature.

1 Introduction

Partitioning is a fundamental problem with extensive applications to many areas using a graph model, including VLSI design [1], knowledge discovery [2], data mining [3],[4], image segmentation and grouping [5],[6]. For example, inspired by spectral graph theory, Shi and Malik [6] formulate visual grouping as a graph partitioning problem. The nodes of the graph are image pixels. The edges between two nodes correspond to the strength with which these two nodes belong to one group. In image segmentation, the weights on the edges of the graph corresponds to how much two pixels agree in brightness, color, etc. Intuitively, the criterion for partitioning the graph will be to minimize the sum of weights of connections across the groups and maximize the sum of weights of connections within the groups. The *min-cut bipartitioning problem* is a fundamental partitioning problem and is NP-Complete [7]. The survey by Alpert and Kahng [1] provides a detailed description and comparison of various such schemes which can be classified as *move-based* approaches, *geometric representations*, *combinatorial* formulations, and *clustering* approaches.

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. The KL algorithm is an iterative improvement algorithm that consists of making several improvement passes. It starts with an initial bipartitioning and tries to improve it by every pass. A pass consists of the identification of two subsets of vertices, one from each part such that can lead to an improved partitioning if the vertices in the two subsets switch sides. Fiduccia and Mattheyses (FM) [9] proposed a fast heuristic algorithm for bisecting a weighted graph by introducing the concept of cell *gain* into the KL algorithm. These algorithms belong to the class of *move-based* approaches in which the solution is built iteratively from an initial solution by applying a move or transformation to the current solution. Move-based approaches are the most frequently combined with stochastic hill-descending algorithms such as those based on Tabu Search[10],[11], Genetic Algorithms [12], Neural Networks [13], Ant Colony Optimization[14], Particle Swarm Optimization[15], Swarm Intelligence[16] etc., which allow movements towards solutions worse than the current one in order to escape from local minima.

As the problem sizes reach new levels of complexity, a new class of graph partitioning algorithms have been developed that are based on the multi-level paradigm. The multi-level graph partitioning schemes consist of three phases [17],[18],[19]. 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 project successively 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 multi-level algorithm which integrates a new simulated annealing-based refinement approach and an effective matching-based coarsening scheme. Our work is motivated by the multi-level refined mixed simulated annealing and tabu search algorithm(MLrMSATS) of Gil which can be considered as a hybrid heuristic with additional elements of a tabu search in a simulated annealing algorithm for refining the partitioning in [20] and Karypis who introduces the concept of the graph *core* for coarsening the graph in [19] and supplies **MeTiS** [17], 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 [21]. Our comparative experiments show that our algorithm produces excellent 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 and describes the notation used throughout the paper. Section 3 describes the motivation behind our algorithm. Section 4 presents an effective multi-level simulated annealing refinement algorithm. Section 5 experimentally evaluates our 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 $S(A) = \sum_{v \in A} S(v)$ denotes 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 partitioning 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 different sides. 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. Naturally, vertex v belongs at the 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 b, the *min-cut bipartitioning problem* seeks a solution $P = \{V^1, V^2\}$ that minimizes *cut(P)* subject to $(1\text{-}b)S(V)/2 \leq S(V^1), S(V^2) \leq (1+b)S(V)/2$. A *bipartitioning* is *bisection* if b is as small as possible. The task of minimizing *cut(P)* can be considered as the *objective* and the requirement that solution P will be of the same size can be considered as the *constraint*.

3 Motivation

Simulated annealing belongs to the probabilistic and iterative class of algorithms. It is a combinatorial optimization technique that is analogous to the annealing process used for metals [22]. The metal is heated to a very high temperature, so the atoms gain enough energy to break chemical bonds and become free to move. The metal is then carefully cooled down so that its atoms crystallize into high ordered state. In simulated annealing, the combinatorial optimization cost function is analogous to the energy $E(s)$ of a system in state s which must be minimized to achieve a stable system.

The main idea of simulated annealing is as follows: Starting from an initial configuration, different configurations of the system states are generated at random. A *perturbation* of a system state consists of reconfiguring the system from its current state to a next state within a neighborhood of the solution space. The change in energy cost between the two configurations is determined and used to compute the *probability* p of the system moving from the present state to the next. The *probability* p is given by $\exp(-\frac{\triangle E}{T})$, where $\triangle E$ is the increase in the energy cost and T is the temperature of the system. If $\triangle E$ is negative, then the change in state is always accepted. If not, then a random number r between 0 and 1 is generated and the new state of the system is accepted if $r \leq p$, else the system is returned to its original state. Initially, the temperature is high meaning that a large number of *perturbations* are accepted. The temperature is reduced gradually according to a *cooling schedule*, while allowing the system to reach equilibrium at each temperature through the cooling process.

In [23], Gil proposed the refinement of mixed simulated annealing and tabu search algorithm(RMSATS) that allows the search process to escape from local minima by the simulated annealing procedure, while simultaneously the occurrence of cycles is prevented by a simple tabu search strategy. At each iteration of

RMSATS, the hybrid heuristic strategy is used to obtain a new partitioning $\overline{s}$ in the neighbourhood, $N(s)$, of the current partitioning s through moving vertex v to the other side of the partitioning s. Every feasible partitioning, $\overline{s} \in N(s)$, is evaluated according to the cost function $c(\overline{s})$ to be optimized, thus determining a change in the value of the cost function, $c(\overline{s}) - c(s)$. The problem with local search techniques and hill climbing is that the searching may stop at local optimum. In order to overcome this drawback and reach the global optimum, RMSATS must sometimes accept the worse partitioning to jump out from a local optimum. Therefore, admissible moves are applied to the current partitioning allowing transitions that increase the cost function as in simulated annealing. When a move increasing the cost function is accepted, the reverse move should be forbidden during some iterations in order to avoid cycling, as in tabu search. In [20], Gil presents the MLrMSATS approach that is enhancement of the RMSATS algorithm with the multi-level paradigm and uses the RMSATS algorithm during the *uncoarsening and refinement phase* to improve the quality of the finer graph $G_l(V_l,E_l)$ partitioning $P_{G_l}=\{V_l^1, V_l^2\}$ which is projected from the partitioning $P_{G_{l+1}}=\{V_{l+1}^1, V_{l+1}^2\}$of the coarser graph $G_{l+1}(V_{l+1},E_{l+1})$.

In this paper, we present a new multi-level simulated annealing refinement algorithm(MLSAR) that combines the simulated annealing procedure with a boundary refinement policy. It has distinguishing features which are different from the MLrMSATS algorithm. First, MLSAR introduces the conception of *move-direction* to maintain the balance constraint of a new partitioning $\overline{s}$. Second, MLSAR defines $c(\overline{s})=cut(\overline{s})$ and exploits the concept of *gain* to fast the computation of $c(\overline{s}) - c(s)$ that is computed by $ED(v)$-$ID(v)$, where the vertex v is chosen to move to the other side of the partitioning s. MLSAR also uses two buckets with the last-in first-out (LIFO) scheme to fast storage and update the gains of boundary vertices of two sides and facilitate retrieval the highest-gain vertex. Finally, MLSAR doesn't select vertex v to move at random in boundary vertices as in MLrMSATS, but always chooses to move a highest-gain vertex v from the larger side of the partitioning. It is important for simulated annealing to strengthen its effectiveness and achieve significant speedups for high quality solutions with well-designed heuristics and properly move generation strategy.

In [17], Karypis presents the sorted heavy-edge matching (SHEM) algorithm that identifies and collapses together groups of vertices that are highly connected. Firstly, SHEM sorts the vertices of the graph ascendingly based on the *degree* of the vertices. Next, the vertices are visited in this order and SHEM matches the vertex v with unmatched vertex u such that the weight of the edge $W(v,u)$ is maximum over all incident edges. In [19], Amine and Karypis introduce the concept of the graph *core* for coarsening the *power-law* graphs. In [11], Leng and Yu present the core-sorted heavy-edge matching (CSHEM) algorithm that combines the concept of the graph *core* with the SHEM scheme. Firstly, CSHEM sorts the vertices of the graph descendingly based on the *core* number of the vertices by the algorithm in [24]. Next, the vertices are visited in this order and CSHEM matches the vertex v with its unmatched neighboring vertex whose edge-weight is maximum.

In our multi-level algorithm, we adopt the MLSAR algorithm during the *refinement phase* and an effective matching-based coarsening scheme during the *coarsening phase* that uses the CSHEM algorithm on the original graph and the SHEM algorithm on the coarser graphs. The pseudocode of our multi-level algorithm is shown in Algorithm 1.

Algorithm 1 (our multi-level algorithm)

```
INPUT: original graph $G(V,E)$
OUTPUT: the partitioning $P_G$ of graph $G$
/*coarsening phase*/
$l = 0$
$G_l(V_l,E_l)=G(V,E)$
$G_{l+1}(V_{l+1},E_{l+1})$=CSHEM($G_l(V_l,E_l)$)
While ( $|V_{l+1}| > 20$) do
    $l = l + 1$
    $G_{l+1}(V_{l+1},E_{l+1})$=SHEM($G_l(V_l,E_l)$)
End While
/*initial partitioning phase*/
$P_{G_l}$=GGGP($G_l$)
/*refinement phase*/
While ( $l \geq 1$ ) do
    $P'_{G_l}$=MLSAR($G_l$,$P_{G_l}$)
    Project $P'_{G_l}$ to $P_{G_{l-1}}$;
    $l = l - 1$
End While
$P_G$=MLSAR($G_l$,$P_{G_l}$)
Return $P_G$
```

4 An Effective Multi-level Simulated Annealing Refinement Algorithm

Informally, the MLSAR algorithm works as follows: At cycle zero, an initialization phase takes place during which the initial partitioning Q is projected from the partitioning $P_{G_{l+1}}$ of the coarser graph G_{l+1}, the Markov chain length L is set to be the number of vertices of the current level graph G_l, the internal and external degrees of all vertices are computed and etc. The main structure of MLSAR consists of a nested loop. The outer loop detects the frozen condition by an appropriate termination criterion whether the current temperature T_k is less than final temperature; the inner loop determines whether a thermal equilibrium at temperature T_k is reached by using the following criterions: The number of attempted moves exceeds L, or the bucket of the start side of the *move-direction* is empty. In the inner loop of the MLSAR algorithm, a neighbor of the current partitioning P is generated by selecting the vertex v with the highest gain from the larger side of the partitioning P and performing the move according to the following rule: The move is certainly accepted if it improves $cut(P)$, or probabilistically accepted according to a random number uniformly distributed on the interval [0,1]. In the latter case, if the acceptance test is negative then no move is performed, and the current partitioning P is left unchanged. The pseudocode of MLSAR is shown in Algorithm 2. The cycles counter is denoted by k and L represents the Markov chain length. Let *Best* be the best partitioning seen so far and P be the current partitioning. At cycle k, T_k represents the current temperature and the counter of neighbors sampled is denoted by L_k.

Algorithm 2 (MLSAR)

```
    INPUT: initial bipartitioning $Q$,balance constraint $b$,attenuation rate $\alpha$
           initial temperature $T_i$,final temperature $T_f$
  OUTPUT: the best partitioning $Best$, cut of the best partitioning $cut(Best)$
  MLSAR(
  /*Initialization*/
  $k = 0$
  $T_k = T_i$
  Set current parition $P = Q$;
  Set the best parition $Best = Q$;
  Set Markov chain length $L$=$|V|$;
  For every vertex $v$ in $G = (V, E)$ do
    $ID_v = \sum_{(v,u)\in E \wedge P[v]=P[u]} W(v,u)$
    $ED_v = \sum_{(v,u)\in E \wedge P[v]\neq P[u]} W(v,u)$
    Store $v$ in boundary hash-table if and only if $ED_v > 0$;
  End For
  /*Main loop*/
  While $T_k \geq T_f$ do
   $L_k$=1
   Compute the gains of boundary vertices of two sides;
   Insert the gains of boundary vertices of two sides in buckets respectively;
   While $L_k \leq L$ do
     Decide the move-direction of the current move;
     If (the bucket of the start side of the move-direction is empty) then
       Break;
     Else
       Select the vertex $v$ with the highest gain in the bucket;
       Designate the vertex $v$ as tabu status by inserting $v$ in tabu list;
       If $(random(0,1) \leq min(1, exp(\frac{(ED_v - ID_v)\times|boundary\ hash\text{-}table|}{2\times cut(Q)\times T_k})))$ then
         $L_k$=$L_k$+1
         Update $P$ by moving the vertex $v$ to the other side;
         original cut Minus its original gain as the cut of new partition $P$;
         Update the internal and external degrees of its neighboring vertices;
         Update the gains of its neighboring vertices in two buckets;
         Update boundary status of its neighboring vertices in boundary hash-table;
```

```
        If (the cut is minimum and satisfies balance constraint $b$) then
          Best=P
          Record roll back point;
          Record new cut minumum;
        End If /* cut is minimum*/
      End If /* $r \leq p$*/
    End If /* the bucket is empty*/
  End While /* thermal equilibrium $L_k \leq L$*/
  Roll back to minumum cut point by undoing all moves and updating the
              internal and external degrees and boundary hash-table;
  Empty the tabu list and two buckets;
  $T_{(k+1)} = \alpha \times T_k$
  $k = k + 1$
End While /* frozen criterion $T_k \geq T_f$*/
Return Best and cut(Best)
```

The MLSAR algorithm uses a tabu list, which is a short-term memory of moves that are forbidden to execute, to avoid cycling near local optimum and to enable moves towards worse solutions, as in the MLrMSATS algorithm. In the terminology of tabu search [25], the MLSAR strategy is a simple form of tabu restriction without aspiration criterion whose prohibition period is fixed at $|V_l|$. Because the MLSAR algorithm aggressively selects the best admissible vertex based on the tabu restriction, 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 MLSAR by using the bucket with the LIFO scheme, as tabu search memory structure. The *internal* and *external degrees* of all vertices, as complementary tabu search memory structures, help MLSAR to facilitate computation of vertex *gain* and judgement of boundary vertex. We also use a *boundary hash-table*, as another complementary tabu search memory structure, to store the boundary vertices whose *external degree* is greater than zero.

During each iteration of MLSAR, the *internal* and *external degrees* and *gains* of all vertices are kept consistent with respect to the current partitioning P. This can be done by updating the *degrees* and *gains* of the vertex v's neighboring vertices. Of course, the *boundary hash-table* might change as the current partitioning P changes. For example, due to a move in an other boundary vertex, a boundary vertex would no longer be such a boundary vertex and should be removed from the *boundary hash-table*. Furthermore, a no-boundary vertex can become such a vertex if it is connected to a boundary vertex which is moved to the other side and should be inserted in the *boundary hash-table*.

5 Experimental Results

We use the 18 graphs in our experiments that are converted from the hypergraphs of the ISPD98 benchmark suite [21] and range from 12,752 to 210,613 vertices.

Table 1. 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

Each benchmark comes with 3 files, a .net file, a .are file and a .netD file. Each hyperedge is a subset of two or more vertices in hypergraph and is stored in .net file. We convert hyperedges 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 doesn'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 format of **MeTiS** [17]. The characteristics of these graphs are shown in Table 1.

We implement the MLSAR algorithm in ANSI C and integrate it with the leading edge partitioner **MeTiS**. In the evaluation of our multi-level algorithm, we must make sure that the results produced by our algorithm can be easily compared against those produced by **MeTiS**. We use the same balance constraint b and random seed in every comparison. In the scheme choices of three phases offered by **MeTiS**, we use the SHEM algorithm during the *coarsening phase*, the greedy graph growing partition algorithm during the *initial partitioning phase* that consistently finds smaller edge-cuts than other algorithms, the boundary KL (BKL) refinement algorithm during the *uncoarsening and refinement phase* because BKL can produce smaller edge-cuts when coupled with the SHEM algorithm. These measures are sufficient to guarantee that our experimental evaluations are not biased in any way.

The quality of partitions is 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 results, two measures are obtained in twenty runs whose random seed is different to each other. For all

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

benchmark	vertices	edges	Metis(α)		our algorithm(β)		ratio(β:α)	
			MinCut	AveCut	MinCut	AveCut	MinCut	AveCut
ibm01	12752	109183	517	1091	354	575	0.685	0.527
ibm02	19601	343409	4268	11076	4208	6858	0.986	0.619
ibm03	23136	206069	10190	12353	6941	8650	0.681	0.700
ibm04	27507	220423	2273	5716	2075	3542	0.913	0.620
ibm05	29347	349676	12093	15058	8300	10222	0.686	0.679
ibm06	32498	321308	7408	13586	3525	8667	**0.476**	0.638
ibm07	45926	373328	3219	4140	2599	3403	0.807	0.822
ibm08	51309	732550	11980	38180	11226	16788	0.937	**0.440**
ibm09	53395	478777	2888	4772	2890	3375	1.001	0.707
ibm10	69429	707969	10066	17747	5717	8917	0.568	0.502
ibm11	70558	508442	2452	5095	2376	3446	0.969	0.676
ibm12	71076	748371	12911	27691	11638	16132	0.901	0.583
ibm13	84199	744500	6395	13469	4768	7670	0.746	0.569
ibm14	147605	1125147	8142	12903	8203	9950	**1.007**	0.771
ibm15	161570	1751474	22525	46187	14505	32700	0.644	0.708
ibm16	183484	1923995	11534	22156	9939	17172	0.862	0.775
ibm17	185495	2235716	16146	26202	14251	17126	0.883	0.654
ibm18	210613	2221860	15470	20018	15430	18248	0.997	**0.912**
average							**0.819**	**0.661**

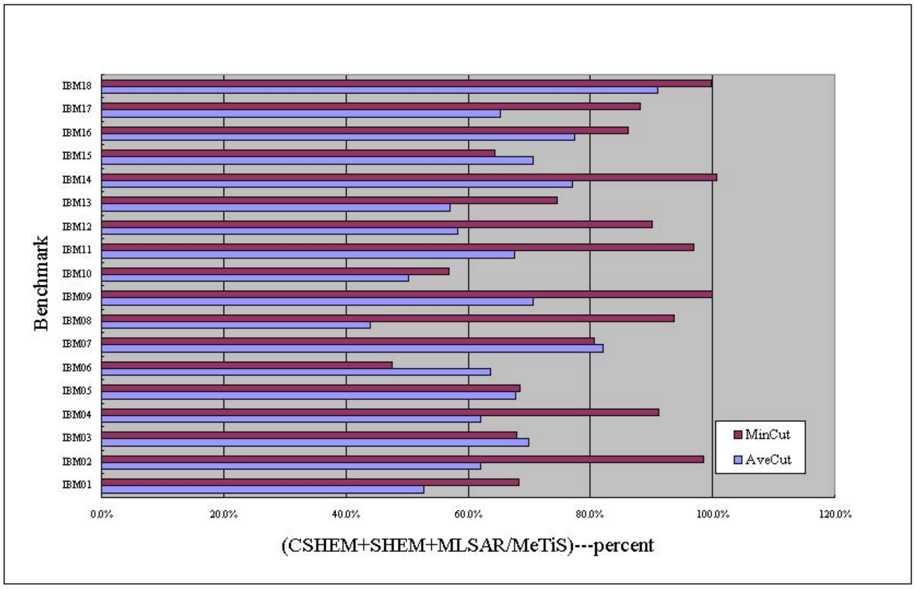

Fig. 1. The MinCut and AveCut comparisons of two algorithms on 18 graphs

experiments, we use a 49-51 *bipartitioning* balance constraint by setting b to 0.02. Furthermore, we adopt the experimentally determined optimal set of parameters values for MLSAR, α=0.9, T_i=10.0, T_f=0.01.

Table 2 presents *min-cut bipartitioning* results allowing up to 2% deviation from exact bisection and Fig. 1 illustrates the MinCut and AveCut comparisons of two algorithms on 18 graphs. As expected, our algorithm reduces the AveCut by 8.8% to 56.0% and reaches 33.9% average AveCut improvement. Although our algorithm produces partitioning whose MinCut is up to 0.7% worse than that of **MeTiS** on two benchmarks, we still obtain 18.1% average MinCut improvement and between -0.7% and 52.4% 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 four hours.

6 Conclusions

In this paper, we have presented an effective multi-level algorithm based on simulated annealing. The success of our algorithm relies on exploiting both the simulated annealing procedure and the concept of the graph core. We obtain excellent *bipartitioning* results 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. For example, we note that adopting the CSHEM algorithm alone leads to poorer experimental results than the combination of CSHEM with SHEM. We need to find the reason behind it and develop a better matching-based coarsening scheme coupled with MLSAR. In the MinCut evaluation of benchmark ibm09 and ibm14, our algorithm is 0.7% worse than **MeTiS**. Therefore, the second question is to guarantee find good approximate solutions by setting optimal set of parameters values for MLSAR.

Acknowledgments

This work was supported by the international cooperation project of Ministry of Science and Technology of PR China, grant No. CB 7-2-01, and by “SEC E-Institute: Shanghai High Institutions Grid” project. Meanwhile, the authors would like to thank professor Karypis of university of Minnesota for supplying source code of **MeTiS**. The authors also would like to thank Alpert of IBM Austin research laboratory for supplying the ISPD98 benchmark suite.

References

1. Alpert, C.J., Kahng, A.B.: Recent directions in netlist partitioning. Integration, the VLSI Journal 19, 1–81 (1995)
2. Hsu, W.H., Anvil, L.S.: Self-organizing systems for knowledge discovery in large databases. In: International Joint Conference on Neural Networks, pp. 2480–2485 (1999)

3. Zha, H., He, X., Ding, C., Simon, H., Gu, M.: Bipartite graph partitioning and data clustering. In: Proc. ACM Conf. Information and Knowledge Management, pp. 25–32 (2001)
4. Ding, C., He, X., Zha, H., Gu, M., Simon, H.: A Min-Max cut algorithm for graph partitioning and data clustering. In: Proc. IEEE Conf. Data Mining, pp. 107–114 (2001)
5. Shi, J., Malik, J.: Normalized cuts and image segmentation. In: Proc. IEEE Conf. Computer Vision and Pattern Recognition, pp. 731–737 (1997)
6. Shi, J., Malik, J.: Normalized cuts and image segmentation. IEEE Trans. Pattern Anal. Mach. Intell, 888–905 (2000)
7. Garey, M.R., Johnson, D.S.: Computers and intractability: A guide to the theory of NP-completeness. WH Freeman, New York (1979)
8. Kernighan, B.W., Lin, S.: An efficient heuristic procedure for partitioning graphs. Bell System Technical Journal 49, 291–307 (1970)
9. Fiduccia, C., Mattheyses, R.: A linear-time heuristics for improving network partitions. In: Proc. 19th Design Automation Conf. pp. 175–181 (1982)
10. Leng, M., Yu, S., Chen, Y.: An effective refinement algorithm based on multi-level paradigm for graph bipartitioning. In: The IFIP TC5 International Conference on Knowledge Enterprise. IFIP Series, pp. 294–303. Springer (2006)
11. Leng, M., Yu, S.: An effective multi-level algorithm for bisecting graph. In: The 2nd International Conference on Advanced Data Mining and Applications. LNCS/LNAI, pp. 493–500. Springer, Heidelberg (2006)
12. Żola, J., Wyrzykowski, R.: Application of genetic algorithm for mesh partitioning. In: Proc. Workshop on Parallel Numerics, pp. 209–217 (2000)
13. Bahreininejad, A., Topping, B.H.V., Khan, A.I.: Finite element mesh partitioning using neural networks. Advances in Engineering Software, 103–115 (1996)
14. Leng, M., Yu, S.: An effective multi-level algorithm based on ant colony optimization for bisecting graph. In: The 11th PacificAsia Conference on Knowledge Discovery and Data Mining. LNCS/LNAI, pp. 138–149. Springer, Heidelberg (2007)
15. Sun, L., Leng, M., Yu, S.: A new multi-level algorithm based on particle swarm optimization for bisecting graph. In: The 3rd International Conference on Advanced Data Mining and Applications. LNCS/LNAI, Springer, Heidelberg (2007)
16. Sun, L., Leng, M.: An effective refinement algorithm based on swarm intelligence for graph bipartitioning. In: The International symposium on combinatorics, algorithms, probabilistic and experimental methodologies. LNCS/LNAI, Springer, Heidelberg (2007)
17. 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)
18. Selvakkumaran, N., Karypis, G.: Multi-objective hypergraph partitioning algorithms for cut and maximum subdomain degree minimization. IEEE Trans. Computer Aided Design 25, 504–517 (2006)
19. Amine, A.B., Karypis, G.: Multi-level algorithms for partitioning power-law Graphs. Technical Report, Department of Computer Science, University of Minnesota (2005), Available on the WWW at URL `http://www.cs.umn.edu/~metis`
20. Gil, C., Ortega, J., Montoya, M.G.: Parallel heuristic search in multilevel graph partitioning. In: Proc. 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing, pp. 88–95 (2004)

21. Alpert, C.J.: The ISPD98 circuit benchmark suite. In: Proc. Intel Symposium of Physical Design, pp. 80–85 (1998)
22. Aarts, E., Korst, J.: Simulated annealing and boltzmann machines. A Stochastic Approach to Combinatorial Optimization and Neural Computing. John Wiley and Sons, New York (1990)
23. Gil, C., Ortega, J., Montoya, M.G., Basnos, R.: A mixed heuristic for circuit partitioning. Computational Optimization and Applications 23, 321–340 (2002)
24. Batagelj, V., Zaversnik, M.: Generalized cores. Journal of the ACM, 1–8 (2002)
25. Glover, F., Manuel, L.: Tabu search: Modern heuristic techniques for combinatorial problems, pp. 70–150. Blackwell Scientific Publications, Oxford (1993)