A Parallelization of State-of-the-Art Graph Bisection Algorithms on the Grid

Nan Dun,† Kenjiro Taura† and Akinori Yonezawa†

We have parallelized Reactive Randomized Tabu Search (RRTS) graph partition algorithm and adapted it to the grid environment. Speedup is achieved by parallelizing scoring phase of RRTS. We also introduced two techniques: multilevel scoring and parallel initialization. By using multilevel scoring, tabu lengths are highly tuned to adapt large graphs. And by parallel initialization, dispersed computing efforts are centralized to output more consistent graph partitions. Experiments show that our approach significantly outperformed state-of-the-art heuristics by partitioning large graphs with higher quality and efficiency.

1. Introduction

Given a graph $G = (V, E)$ with V as its vertices set and E as edge set. The *k-partition* problem is defined as follows: To partition V into k subsets $V_1, V_2, \ldots, V_k$, such that:

$V_i \cap V_j = \phi$ for $i \neq j$

$|V_i| = |V|/k$ for $\bigcup V_i = V$

$C = \{(v_i, v_j) | (v_i, v_j) \in E, v_i \in V_i, v_j \in V_j\}$ and $|C|$, or *edgecut* of partitions, is minimized. When $k = 2$, we come on to *bipartition*, or *bisection* problem, the basic case of graph partition.

Graph partition problem has been extensively investigated over years. A k-partition problem is NP-complete[1], so is a bisection problem. To find approximate solution is also NP-hard[2].

The importance of graph partition lies not only in mathematics, such as sparse matrix-vector multiplication and Gaussian elimination, but also in many practical applications, such as mesh distributing, load balancing, VLSI and large network designs.

2. Related Works

Before presenting our work, we briefly introduce several graph partition algortihms that are popularly used nowadays.

2.1 Kernighan-Lin Heuristic (KL)

The *Kernighan-Lin heuristic*[3] start from an initial partition bisected by randomly selecting or other algorithms. Then KL searches pairs of vertices that will yield smaller edgecut if they were swapped. In KL, the reduction of edgecut is indicated by *gain* of a vertex when it is moved from partition to the other, as follows,

Given a bisected graph V and its partitions V_1, V_2.

gain of moving: For $v_i \in V_1$,

$$g(v_i) = \sum_{v_j \in V_1} \omega(v_i, v_j) - \sum_{v_k \in V_2} \omega(v_i, v_k) \quad (1)$$

where, for $v_i, v_j \in V$,

$$\omega(v_i, v_j) = \begin{cases} 0 & (v_i, v_j) \notin E \\ 1 & (v_i, v_j) \in E \end{cases} \quad (2)$$

gain of swapping: For $v_i \in V_1, v_j \in V_2$,

$$g(v_i, v_j) = 2\omega(v_i, v_j) + g(v_i) + g(v_j) \quad (3)$$

This algorithm iteratively searches through all vertices by computing and comparing theirs gains to maximize reduction of edgecut. It stops when no pair of vertices can satisfy edgecut-reducing criteria. Native KL[3] costs $O(|E|log|E|)$ time for each iteration. And we refer to one improved version[4], leading to a less complexity as $O(|E|)$.

2.2 Region Growing Partition (RGP)

Region Growing Partition (RGP) stands for a class of heuristics to initialize partitions of original graphs. Intuitively, RGP starts to grow a region from a vertex in breath-first-search way and stops until bisections reach the equal size.

Greedy Graph Growing Partitioning (GGGP) algorithm[5] improves partitioning performance by using the same idea of *gain* as in KL. Instead of growing one region, *Min-Max Greedy Partitioning* (MMGP) algorithm[7] initials two partitions from two seeds by repeatedly adding vetices to them. RGP algorithms are sensitive to the choice of beginning vertex[7]. In our experiments, comparing to GGGP, MMGP not

† The University of Tokyo

only produces better bisections in less running time, but also shows less dependency to initial seeds.

2.3 Tabu Search (TS)

The *Tabu Search heuristic* is also known as *prohibition-based* search algorithm. As in KL algorithm, Tabu Search incrementally refines initial bisections by swapping pairs of vertices to decrease edgecut. However, unlike KL, in which previously swapped vertices are locked, TS prevents swapped vertices from moving again only for a period of time. And this prohibition period is referred as *Tabu Length*, denoted by a fraction of the number of vertices, as follows,

$$L_{tabu} = f_{tabu} \times |V| \tag{4}$$

where $f_{tabu} \in [0.01, 0.02, \cdots, 0.25]$. Experience suggests that these 25 candidates are sufficient for most graphs[7].

The motivation of temporary prohibiting in TS is to exceed *local minimum* cases that will not lead to any improvement by current local search. Experiments also shows that graphs in different structure require differenet appropriate tabu lengths to yield good bisections. More specific, denser random graphs tend to prefer smaller tabu lengths, while denser geometric graphs tend to prefer larger tabu lengths[7]. And we further point out that graphs having uniform distribution of *vertex degree* tend to have unique appropriate tabu length, where the vertex degree is defined as follows,

degree of vertex: For any $v \in V$,

$$d(v) = \sum_{v' \in V} \omega(v, v'). \tag{5}$$

Accordingly, choosing right tabu length has a crucial effect on partition quality.

Now we present three variants[7] of TS in their developing sequence. They all start from an initialized graph by MMG heuristic.

Fixed Tabu Search (FTS) iteratively applies passes of tabu search on bisection with a fixed tabu length. Since we are not able to tell what tabu length will lead to a better bisection before we obtaining any results, tabu lengths should be tuned by programmers.

Randomized Tabu Search (RTS) performs TS with several randomly choosed tabu lengths, and finally returns the best result found in overall runs. Similarily, there is no assurance of proper tabu length fitting current graph.

Reactive Randomized Tabu Search (RRTS) first evaluates how each possible tabu length fits current graph by a scoring routine, and high-scored (most appropriate) tabu length is employed to perform deeper search. Thus, RRTS is adaptive to graphs in various structures.

2.4 Multilevel Partition

The multilevel partition technique[5] is mainly used to reduce the partitioning time of large-scale graphs. It coarsens a graph down to a few hundreds vertices first, bisects this shrunk small graph, and then uncoarsens partitionss to original graph. During each stage of uncoarsening, previously collapsed edges are released, and local searches are tried to refine partition. METIS[12] is a high-quality and high-speed graph partition library of using this scheme.

3. Evaluation of Heuristics

In this section, we check different aspects of graph bisection in order to demonstrate some issues need to be addressed.

3.1 Quality vs. Time

In our first experiement, we evaluated each heuristic with two criteria in practical usage, partition quality and partitioning time, over a set of "real-world" benchmark[5][7][13]. Since RRTS and METIS are more advanced heuristics evolving from other elementary ones and they perform almost the same for small graphs, we primarily compare them for relative larger problems ($|V| > 10,000$) on a single machine with 2.53GHz P4 processor, 1GB of memory, and 120GB 5400rpm disk. As a follow up of this comparison, we tested native FTS with those best-scored tabu lengths obtained by former RRTS runs. The results are as **Table 1** shows. METIS is referred as its serial version 4.0.1[12]. RRTS100 indicates 100 iterations of heuristic, while FTS10000 means 10000 iterations. Time is measured in seconds and we set timeout as 3600 seconds.

Although running fastest, due to limited free degrees during coarsen-uncoarsen stages, MEITS fell behind his competitor RRTS in bisection quality. However, RRTS suffers from the longest running time, since its adaptive feature requires more effort to score each tabu length. And from FTS10000 column, we learn

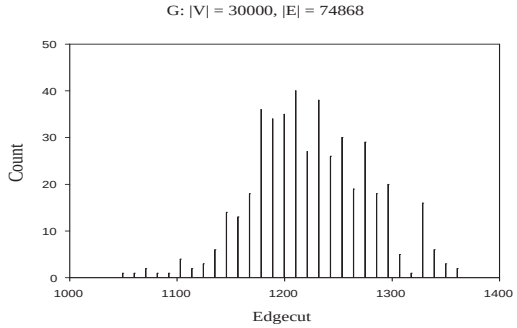

Fig. 1 Distribution of edgecuts obtained by running RRTS on 400-node grid

that if we know proper tabu length in advance, FTS can lead to high-quality partitions in a much more efficient way.

3.2 Chance to Reach the Best

Our second experiment simply repeats identical heuristic independently on the same graph for multiple runs, with a purpose of illustrate how best a bisection can be obtained and how often the best one occurs. To do it more efficiently, we had concurrently run indepent instantces of heuristc, say RRTS, on a 400-nodes grid[15]. The distribution of edgecuts appears in **Fig. 1**. A peaked distribution remains similar even when different graphs or heuristics being applied. Detailed analysis of this phenomenon can be found in Schreiber and Martin's paper[8], which suggests an issue that the incidence of best edgecut only shares a small fraction of whole trials.

4. Basic Design

4.1 Design Goals

Our design aimed at taking advantage of computing capability of girds to produce ever best partitions for a variety of large-scale graphs and overcoming two main issues mentioned in 3.

Table 1 A Comparison of METIS, RRTS and FTS on large-scale graphs

	METIS		RRTS100		FTS10000	
	cut	time	cut	time	cut	time
G_1	**130**	0.01	**130**	168.11	**130**	1.22
G_2	366	0.07	**353**	696.49	354	13.85
G_3	311	0.10	311	935.56	**306**	32.85
G_4	6337	0.04	**6257**	353.45	6316	3.77
G_5	950	0.17	timeout		**929**	31.55

Graph	$\|V\|$	$\|E\|$	f_{tabu}^{best}
G_1:fe_4elt	11143	32818	0.02
G_2:fe_pwt	36519	144794	0.02
G_3:fe_body	45087	163734	0.02
G_4:mem	17758	54196	0.14
G_5:wing	62032	121544	0.01

The overall design goals are:

High-quality Partitioning

To partition graphs with high-quality not worse than state-of-the-art libraries, such as METIS and native RRTS, is basic requirement. Fortunately, the grids provide us abundant resources to achieve this computation-costing goal.

High-efficient Partitioning

The partitioning of graphs should not only cost as less time as possible, but also hold a high possibility of obtaining the best partitions. In this context, efficient partitioning suggests rapid and consistent outputs of high-quality bisections.

Grid Optimization

Since we will partition graphs on the grid environment. It is reasonable to bring other particular factors into our consideration, such as cost of communication, coordination among all nodes, fault tolerance, and user operability.

4.2 Parallelization of Heuristics

Besides multilevel technique, an intuitive way to speed up algorithms is to parallelize them. But unfortunately, as to graph partition problem, parallelizing swapping based heuristics such as Kernighan-Lin is hard by using message-passing model[9], a dominant programming model in the grid environment, similar for Tabu Search. ParMETIS[12], parallel version of METIS, also merely parallelized the coarsen phase, left partition phase in serial on one node.

Recalling 3, if given a proper tabu length, FTS significantly outperforms RRTS. Based on this fact, we suggest parallelize the scoring phase in RRTS instead of partitioning phase.

4.3 Multilevel Scoring

Multilevel scoring is motivated by **Eq.** 4, in which tabu lengths are proportional to the number of vertices, irrelevant to graph scales. Original scoring routine is adaptive to classes of graphs instead of the very specific ones. Thus, finer tabu lengths are required to perform more precise searchings, especially for large graphs. are required.

In multilevel scoring, *level–1* scoring begins as usual ($f_{tabu}^{L1} \in [0.01, 0.02, \cdots, 0.25]$) and returns the best-scored tabu length, taking 0.02 as an example here. Then level–2 scoring uses this best-scored one (0.02) as median and divide its

two adjacent section into finer segments, here $f_{tabu}^{L2} \in [0.011, \cdots, 0.019] \cup [0.021, \cdots, 0.029]$. We repeat this division and stop at level–i if comparing to level–(i-1) scores, their improvement appears less than predefined threshold.

4.4 To Parallelize Starting Phase

Before looking into details in this section, we prefer to dive into an analysis of FTS behavior by investigating how individual phase contributes to whole results.

As introduced before, FTS have three phases: MMG initializing phase, tabu searching phase and KL refinement phase. We did a partitioning on graph *wing* by FTS10000 on 50 nodes, with profiling of each phase. Then we choose four best runs (column) from overall runs. as shown in **Table 2**. Column 1 is final edgecuts, while column 2 is initial paritions with smallest edgecut produced by MMG in corresponding runs. Column 3 and 4 give the maximum edgecut reduction detected in passes of TS and KL, respectively. From Table 2, it is obvious that MMG actually contributes more than TS and KL. Further, an important discovery is: better initial partition yields better final edgecuts, while reductions gained by TS and KL remain in small difference. However, on the other side, initial phase costs much less time than TS or KL search.

Since initial partitions heavily concern with the quality of final partitions, we advocate to extract initializing phase out of FTS and parallelize it individually. Also noticed that initial partitions are sensitive to seeds in MMG, it is reasonable to take advantage of grid's natural distribution to assign different seeds to different nodes in the grid environment.

5. Parallel Graph Partitioning

Our parallel partitioning employs basic master-worker model. In this framework, a master takes charge of assigning jobs and processing mid-way and final results, while workers share the computing burden of scoring and partitioning.

Table 2 Edgecut redcution by different phase of FTS10000

	MMG	TS	KL
$edgecut_{final}$	$edgecut_{init}$	$\Delta edgecut$	$\Delta edgecut$
912	1078	114	158
1032	1185	174	187
1075	1197	213	139
1079	1221	133	133
wing: $\|V\| = 62032, \|E\| = 121544$			

5.1 Parallel Multilevel Scoring Stage

This stage intends to determine proper tabu lengths for graphs. Note that there are initially 25 f_{tabu}^{L1} candidates.

Dispatching Initially, n workers are divided into 25 groups. Workers in the jth group are assigned a $f_{tabu}^{L1} = j \times 0.01, j = 1, 2, \cdots, 25$. We believe that scores evaluated by groups are more disinterested than individual nodes.

Reporting When scoring finished, each (the ith) worker returns two items to master: score of tabu length scr_j^i (belonging to group j), "best-so-far" edgecut cut_{best}^i.

Evaluating Master averages all scr_j^i obtained from group j to scr_j, and chooses the f_{tabu}^{L1} with the highest score as a global metric value f_{best}^g. Also, the overall smallest cut_{best}^i are selected to be global threshold cut_{best}^g.

Multilevel Scoring If problem is large-scale, it is necessary to perform a multilevel scoring refered in 4.3. In this case, master breaks f_{best}^g into f_{tabu}^{L2}, and repeats dispatching, reporting and evaluating to update ever better f_{best}^g and cut_{best}^g.

5.2 Parallel Partitioning Stage

In this stage, we have got f_{best}^g and cut_{best}^g from previous scoring stage.

Parallel Initializing Before any refinement, we simply dedicate all nodes to run MMG. Every local best initial partiton found by each nodes is recorded and reported back to master. Then master selects global best five ones as initial partition candidates.

Parallel Partitioning With f_{best}^g, a couple of runs of TS and KL are performed on each initial partition candidate. Eventually, cut_{best}^g is recorded and returned as final result.

6. Implementation

Our C-impelmented graph partition library includes all partition algorithms introduced in this paper. Experience shows that graph heuristics need careful implementation to achieve high performance, such as proper data structure to maintain nodes' information and fast indexing.

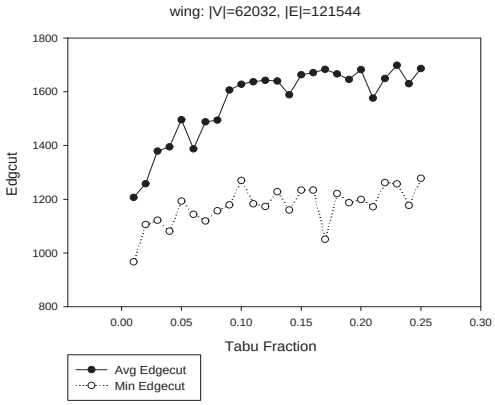

Fig. 2 FTS10000 edgecuts for different f_{tabu}^{L1}

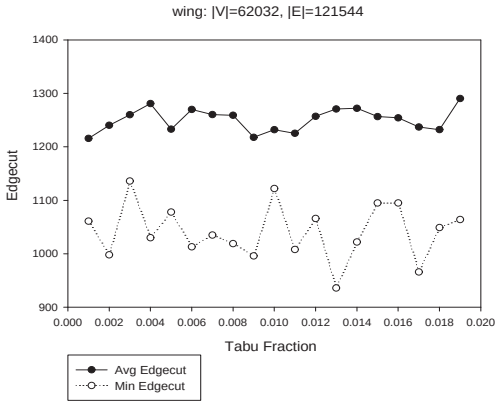

Fig. 3 FTS10000 edgecuts for different f_{tabu}^{L2}

GXP cluster shell[11] provides powerful advantage of simultaneously running numerous instances on clusters. Besides this, we significantly simplied communication implementation between master and workers by making use of GXP's master-worker command utility[14].

7. Experiements

Our experiment environment is composed of 50 computing nodes each with Xeon 2.4GHz dual processors, 2GB memory, 40GB harddisk and running Linux 2.4.20. And GXP cluster shell has been installed on all nodes.

7.1 Using Multilevel Scoring

We first made a comparison of performance between uni-level scoring and multilevel scoring on large graphs. When uni-level scoring was used, we ran FTS10000 with given f_{tabu} on all nodes and averaged all results. All 25 f_{tabu} were tried to see how they fit problems. Then in multilevel scoring, we selected the best f_{tabu} obtained in previous uni-level (equally as f_{tabu}^{L1}), and splited it into level-2 f_{tabu}^{L2} for FTS10000. Graph *wing* was used as a representative demonstration, while outcome of other large graphs remains alike.

From **Fig. 2** we can find that among 25 f_{tabu}^{L1}, both average and minimum edgecuts are reached when f_{tabu}^{L1} is 0.01 ($L_{tabu} = 620, edgecut_{best}^{g} = 967$). The edgecut will become larger with the increasing of tabu length, which characters dense random graphs.

However, in **Fig. 3** the curve tends to be flat. Overall average edgecuts have been dominated under 1300, and minimum one was obtained when f_{tabu}^{L2} was 0.013 ($L_{tabu} = 806, edgecut_{best}^{g} = 936$), instead of 0.01.

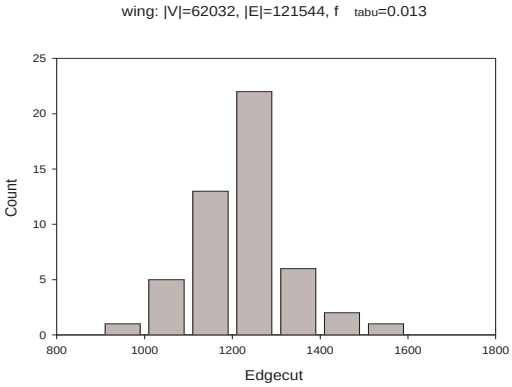

Fig. 4 Distribution of edgecuts obtained by running FTS10000 on 50 nodes

7.2 With Parallel Initialzation Phase

Here we compare results from two parallelizing schemes. One is parallel running FTS10000. The other is the parallel initializing phase mentioned in 5.2. For consistency, we still used graph *wing* as our test case and chose previously best tabu fraction, $f_{tabu}^{L2} = 0.013$.

The distribution of edgecuts in **Figure 4** have the same property as Figure 1. Few of nodes yield the best partition even average results were good.

However, situation significantly changed in **Figure 5**. During the initializing phase, we obtained four best initial partitions with edgecut 1078, 1156, 1185 and 1197. These four partitions are further sent to TS and KL refining phase and finally lead four *unique* best edgecuts 915, 985, 1000, and 987, correspondingly. This result not only demonstrats our judgement of initial-final relationship, but also suggests that

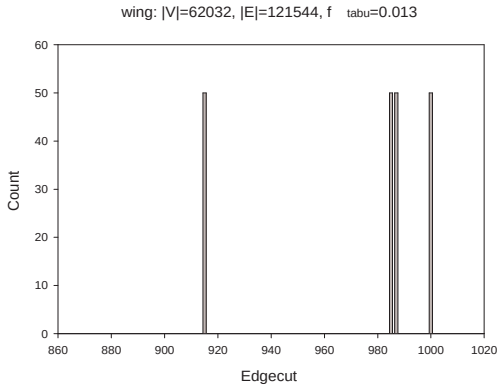

Fig. 5 Distribution of edgecuts obtained by feeding "best-so-far" initial partitions

once we have obtained a good initial partition, we are able to assure it will lead to better results. Further, rigorous line-distribution in Figure 5 also implies refinement could be done on only one node without losing quality. Therefore, more computation effort should be devoted to initilizing phase to find better initial partitions, especially by using computing capability of grids.

8. Conclusions

In this paper, we have introduced two schemes to parallelize RRTS graph partition algorithms: multilevel scoring and parallel initializing. Experiements on "real-world" benchmark illustrated that our approach significantly outperformed state-of-the-art graph partition heuristics by partitioning large graphs with higher quality and efficiency.

References

1) M.R. Garey and D.S. Johnson.: Computers and Intractability, A guide to the theory of NP Completeness, *W.H. Freeman and Company*, New York, (1979).
2) T. N. Bui and C. Jones.: Finding good approximate vertex and edge partitions is NP-hard. *Information Processing Letters*, Vol. 42, No. 3, pp.153–159 (1992).
3) B. Kernighan and S. Lin.: An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell System Technical Journal*, Vol. 49, No. 2, pp. 291–207 (Feb. 1970).
4) C.M. Fiduccia and R.M. Mattheyses.: A linear-time heuristic for improving network partitions. *In Proceedings of the nineteenth design automation conference*, pp. 175–181, (1982).
5) G. Karypis and V. Kumar.: A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, Vol.20, No.1, pp.359–392, (1998).
6) F.Glover.: Tabu Search—Part I. *ORSA J. Computing*, Vol.1, No.3, pp.190–260, (1989).
7) R. Battiti and A. Bertossi.: Greedy, Prohibition, and Reactive Heuristics for GraphPartitioning, *IEEE Transactions on Computers*, Vol.48, No.4, pp.361–385, (1999).
8) G.R. Schreiber and O.C. Martin.: Cut Size Statistics of Graph Bisection Heuristics, *manuscript in submission to SIAM J. Optimization*, (1997).
9) J.R. Gilbert and E. Zmijewski.: A parallel graph partitioning algorithm for a message-passing multiprocessor. *International Journal of Parallel Programming*, Vol.16, No.6, pp.427–449, (1987).
10) R. Battiti, A. Bertossi and Cappelletti.: Multilevel Reactive Tabu Search for Graph Partitioning. *Preprint UTM 554, Dip. Mat., Univ. Trento, Italy*, (1999).
11) K. Taura.: GXP: An Interactive Shell for the Grid Environment. *International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems*, Vol.00, pp. 59–67, (2004).
12) METIS: http://glaros.dtc.umn.edu/gkhome/views/metis/
13) Inter Tools: http://rtm.science.unitn.it/intertools/graph-partitioning/benchmark.html
14) GXP: http://www.logos.ic.i.u-tokyo.ac.jp/phoenix/gxp_quick_man.shtml
15) GCF: http://www.hpcc.jp/sacsis/2006/grid-challenge