Parallel Tabu Search for Graph Multi-Partitioning Problem

Taichi Kaji

Otaru University of Commerce
3-5-21, Midori, Otaru, 047-8501, Japan
kaji@res.otaru-uc.ac.jp

ABSTRACT

Tabu Search is a general heuristic method for combinatorial optimization problem, which has been successfully applied to several types of difficult combinatorial optimization problem. We present two efficient parallel implementations of Tabu Search Algorithm for graph multi-partitioning problem. The one idea has been taken to parallelize neighbourhood space and achieved a linear speedup. Another main approach for getting high quality of solutions has been also proposed. The algorithm produces independent partial problem using simple agent and execute the multi thread procedure in which several processes explore independently the search space of partial problem. It produces cooperatively better solution from partial solutions obtained by parallel thread with reduced computational time. Finally, as the algorithm is well suited for parallel computation, an implementation on a computational environment using MPI library is described. Numerical results and speedups obtained show the efficiency of the parallel algorithm.

Keywords : Parallel Algorithm, Graph-partitioning Problem, Tabu Search, Agent, Simulated Annealing

1 INTRODUCTION

Tabu Search and Simulated Annealing are novel techniques for solving combinatorial optimization problem, which have been successfully applied to several types of difficult optimization problem. We implement basic tabu search and basic simulated annealing algorithm, and examine here the performance of the both basic type of meta-heuristic algorithms for graph multi-partitioning problem. It appears that the quality of solution obtained by both algorithms yield approximately similar results. However, We found that the tabu search algorithm required more computational time than simulated annealing, and there is tendency for the solutions obtained using basic type of meta-heuristic to be trapped in bad near optimum. Therefore, the increasing availability of parallel machines offers an interesting opportunity to explore the possibility of new tabu search algorithms [2].

In this respect, we present two efficient parallel implementations of Tabu Search Algorithm for graph multi-partitioning problem. The one idea has been taken to parallelize neighbour move and achieved a linear speedup. Another main approach, which is Multi-thread parallelism, for getting high quality of solutions has been also proposed. Multi-thread parallelism has been implemented using a framework in which a number of independent threads operate on each different partial solution space, to improve the quality of solutions. But, the problem with this kind of parallelism by iterative search is that it strongly limits the movement possibilities between different subproblems and thus generally induces a loss of quality of the global solution. We overcome this difficulty by attractive searching based on tabu search,

the extension of searching by super node move, and diversification of producing subproblems by generator agents. It produces cooperatively better solution from partial solutions obtained by parallel thread with reduced computational time. Finally, as the algorithms are well suited for parallel computation, an implementation on a computational environment using MPI library [4] is described. Numerical results and speedups obtained show the efficiency of the parallel algorithm.

The parallel heuristic procedure discussed in this paper, designed to "solve" (in the sense of approximating the optimum) the graph multi-portioning problem, is mathematically unexciting, but has performed remarkably well, both from the point of view of the computational effort involved, and from that of the quality of the solutions obtained on a variety of test problem.

2 GRAPH MULTI-PARTITIONING PROBLEM

The graph-partitioning problem is typical optimization problem that is known as *NP*-complete and well used as benchmark in order to make the comparison with other heuristic approaches. It is finding the minimum cost partition of the nodes of a graph into subsets of a given size. The graph-partitioning problem arises in many different guises in operations research. The examples are planning the VLSI circuit and assignment of locations.

In this paper, we are concerned with the multi-partitioning problem such that the total cost of edge in the cut set is minimized. This problem is complexity but multipurpose in comparison with bi-partitioning problem examined until now. To explain in detail, we give mathematical notation. Let $D(V,E)$ be a graph where V is a set of n nodes and E is a set of edges. Graph partition is defined as set partition $\{V_1, V_2,\ldots,V_\kappa\}$ such that block size, which is number of nodes for each block, is fixed size. The graph multi-

partitioning problem can be formulated mathematically as follows.

$$\begin{aligned} &min \quad \sum_{i\in V_\kappa, j\in V_l, \forall \kappa<l\in\{1,2,\ldots,\kappa\}} e_{ij} \\ &subject\ to \quad |V_i|=M_i, \quad i=1,\ldots,\kappa, \\ &\qquad\qquad \sum_{i\in\{1,\ldots,\kappa\}} M_i=n, \end{aligned} \tag{1}$$

where e_{ij} is the edge cost and M_i is a block size that is given positive number. The number of subsets, κ is the size of the partition which is a fixed number.

3 SEQUENTIAL BASIC META-HEURISTICS FOUNDATIONS

3.1 Tabu Search Algorithm

Tabu search [3] is a meta-heuristic that guides a local heuristic search procedure to explore the solution space beyond local optimality. Widespread successes in practical applications of optimization have spurred a rapid growth of tabu search in the past few years. The basis for the tabu search may be described as follows. Tabu search begins in the same way as ordinary local search, proceeding iteratively from one point (solution) to another until a chosen termination criterion is satisfied. Each $x\in X$ has an associated neighbourhood $N(x)\subset X$, where X is solution space of given problem, and each solution $x'\in N(x)$ is reached from x by an operation called a *move*. The process in which the tabu search method seeks to transcend local optimality is based on an evaluation function that chooses the highest evaluation move in terms of objective function and a tabu restriction. The tabu restriction employs a strategy of modifying $N(x)$ as the search progresses, effectively replacing it by another neighbourhood $N(\alpha,x)=N(x)-\alpha$, where α is called a *tabu list* consisting s elements of solutions visiting during the recent past, to

avoid cycling. But, tabu list memory is designed to exert a more subtle effect on the search through the use of attributive memory, which records information about solution attributes that change in moving from one solution to another. A key aspects of tabu search is the use of tabu list structures and neighbourhood structures to organize the way in which the space is explored.

We show basic Tabu Search algorithm by representing neighborhood structure and tabu memory structure for this problem. First we explain neighborhood structure for graph multi-partitioning problem, that is based on exchange of nodes. We consider a solution $x=(V_1, V_2,\ldots,V_\kappa)$, where V_i is the subset of partitioned nodes of graph. Each solution x has a following neighborhood $N(x)$.

$$N(x)=\{x'=(V_1, V_2,\ldots,V_s',\ldots,V_w',\ldots,V_\kappa) : \forall s, \forall w, \forall i, \forall j\ [V_s'=(V_s\setminus\{i\})\cup\{i\}, \quad V_w'=(V_w\setminus\{j\})\cup\{i\}, i\in V_s, j\in V_w, s\neq w]\}. \tag{2}$$

Next, we consider a tabu list structure. A data structure for tabu list will be used to store a partial range of solution attributes rather than complete visited solutions. For this problem, the attribute can consist of nodes that are shifted by moves executed. When a node v of block V_i is shifted to another block V_j, we store v as the attribute in the tabu list to avoid returning v to V_i. In some applications, the tabu search is made significantly strongly by to do including longer-term memory. Thus we also use longer-term memory. When node v is shifted, we increase longer-term memory element LTM[v] by 1. The evaluation function for exchanging the nodes v_i and v_j is modified using longer-term memory, that is, incremental cost + (LTM[v_i] + LTM[v_j]) $\times$ $BIAS/2$, where $BIAS$ is a parameter.

3.2 Simulated Annealing Algorithm

Simulated annealing algorithm [1] provide also iterated hill-climbing mechanism like tabu search and local search. Moves that worsen the current solution by an amount ΔE are accepted with probability $e^{-\Delta E/T}$. T is a control parameter analogous to temperature in the annealing of physical system. In an optimization framework, we anneal a solution by performing standard iterative improvement steps, but now accept some worsened solutions according to this temperature rule. Central to any annealing algorithm is the need to evaluate many moves. During initial annealing, T is large; in this hot regime, most uphill moves are accepted. As iterative improvement proceeds, the temperature T is slowly reduced. Near the end of annealing, T is small; in this cold regime, few uphill moves are accepted. The intent of the relatively high temperature phase of the search is to discover the gross features of the search space. Successively lower temperatures identify more and more detail while solutions become more and more localized. Ultimately, simulated annealing yields a very good solution on a very good mode of the search space.

For this problem, we use the same neighborhood structure as tabu search in the basic simulated annealing algorithm. New solution can be generated by choosing two nodes $i \in V_\kappa, j \in V_l$ for all κ and $l \neq \kappa$, and exchanging this two nodes between block V_κ and V_l. The difference in cost can be calculated incrementally from the following expression,

$$\Delta f = \sum_{r \in V_\kappa} e_{ir} + \sum_{s \in V_l} e_{js} - \sum_{r \in V_l} e_{ir} - \sum_{s \in V_\kappa} e_{js} + 2e_{ij}. \tag{3}$$

The decision to accept new solutions is based on the above mentioned basic acceptance criterion.

3.3 Computational Results for Basic Sequential Meta-Heuristic Algorithms

In this section, we evaluate the performance of basic tabu search and basic simulated annealing algorithm for graph multi-partitioning problem. The algorithm used here were coded in C++ and implemented on Intel Celeron Processor/400MHz. There are several parameters in the algorithm that have to be adjusted. First, let us examine the effect for the most important parameter, the tabu list size *tabulength*, for tabu search. The way to obtain a good tabu list size for a given problem is simply to observe the quality of solutions when size is varied. We therefore solved a randomly generated test problems for graphs with 500 nodes. We found that better solutions are obtained when $tabulength = 20$, and therefore propose that *tabulength* be set to values 20. Similarly, the *BIAS* used in longerterm memory is 0.20, by which better solutions are obtained. Next, we consider the parameter's values of the simulated annealing in order to obtain good solutions in a reasonable amount of computational time. The initial $tmp(T)$ is 15 and the stop $tmp(T)$ is 0.01, since, in spite of extending the above range of *tmp*, we did not obtain a significant improvement in the quality of the solutions. We found that $phi = 0.97$, which is cooling schedule parameter, gives an acceptable balance between solution quality and computational speed. It appears that initial $r = 100$ and $tau = 1.03$, which control the speed of convergence, yield good compromises between the quality of the solutions obtained and the time required.

We assess the performance of basic tabu search and simulated annealing using above-mentioned parameters. We solved eight partitioning test problems for graphs with the number of nodes ranging from 100 to 1000, which is randomly generated. The number of edges for the graphs is 10% of the number of nodes. Table 1 shows the cost and CPU time for tabu search and the average cost, the best cost and worst cost, and CPU time, obtained

after 10 repetitions of the simulated annealing. We obtained same quality of solutions in both algorithms. However, we found that the time required for tabu search algorithm is longer than that required for the simulated annealing algorithm.

Table 1 Computational results for basic tabu search and simulated annealing

Program Size	Tabu Search		Simulated Annealing			
	cost	time(s)	mean	best	worst	time(s)
100	3812.0	7.25	3704.8	3687.0	3734.0	29.49
200	16335.0	93.66	16269.6	16238.0	16342.0	75.41
300	37710.0	423.36	37794.0	37744.0	37879.0	147.51
400	68843.0	1357.88	68968.8	68770.0	69104.0	230.50
500	108055.0	2210.26	108425.6	108302.0	108526.0	307.12
600	157788.0	4456.19	157643.2	157549.0	157880.0	385.80
700	216037.0	5797.71	216348.2	216155.0	216730.0	464.98
800	284427.0	7385.28	284325.4	284153.0	284828.0	548.88
900	362989.0	9205.00	362609.4	362369.0	363012.0	628.17
1000	448997.0	10832.89	447511.0	447367.0	447840.0	700.38

4 PARALLEL TABU SEARCH

In Tabu Search techniques, different types of parallelism may be distinguished. We only consider here two general classes according to the method used for parallelism moves [2].

The First type of parallelism we consider consists in searching the next move concurrently. The Second type can be done by partitioning the problem itself into several independent subproblems. These algorithms are very well suited for implementation on a MIMD parallel mechanism. The parallel algorithms have been implemented on a network by MPI library [4].

4.1 Parallel Neighbour Move Algorithm

First, the search of the next move to perform can be parallelized. This

will generally require the partition of the set of feasible moves in P subsets. Each of these subsets is assigned to one process which computes its best move. The overall best of these P moves is then determined and applied to the current solution. That is, we divide the set of neighbourhood $N(x)$ into $N_1(x), N_2(x),\ldots,N_p(x)$, such that $N_1(x)\cup N_2(x)\cup\ldots\cup N_p(x)=N(x)$, $N_i(x)\cap N_j(x)=\phi$, *for* $i\neq j$, where P is the number of processors. Each partitioned neighbourhood is assigned to each processors, which computes best solution, x_i for $N_i(x)$ for $i=1, 2,\ldots,P$. The obtained best neighbour solution of each processor is mutually communicated between all processors together. Then we determine a best solution in $N(x)$ that is $min\{x_1,x_2,\ldots,x_p\}$, and applied to the current solution. The synchronization is required at each iteration step. Normally, this technique requires extensive communication since the synchronization is required at each step. However, the basic tabu search for this problem has broad space of neighbourhood, and this searching for the space of neighbourhood is very time-consuming. Therefore, it is only worth applying to this problem in which the search of the best move is relatively complex and time-consuming. We estimate that this proposed parallel algorithm, called parallel neighbour move algorithm, allows us to reduce considerably the computational time.

4.2 Parallel Agent Passing Tabu Search Algorithm

As mention later (section 4.3), Parallel neighbour move algorithm allows us to reduce considerably the computational time. However, the quality of solutions by this algorithm is as same as ones by basic algorithm. Therefore, Multi-thread parallelism has been implemented using a framework in which a number of independent threads operate on each different partial solution space, to improve the quality of solutions. But, the problem with this kind of parallelism by iterative search is that it strongly limits the movement

possibilities between different subproblems and thus generally induces a loss of quality of the global solution. We overcome this difficulty by attractive searching based on tabu search, the extension of searching by super node move, and diversification of producing subproblems by generator agents. Particularly, the agent on computers contributes to producing better solution. He has simple ability, producing subproblem, moving from processors to another, and learning strategy and state of each solution assigned on processors. The parallel algorithm based on these ideas is called parallel agent passing tabu search algorithm.

Intensification Strategy

First, We would like to consider the parallel attractive searching thread using tabu search. The type of parallelism we consider consists in performing several moves concurrently. This can be done by partitioning the problem itself into several independent subproblems. Each of the processes can then apply moves to its assigned subproblem, independently of those made in other subproblems. The strategy used here is to divide the current solution in P independent partial solutions, $x_i=(V_l,V_m)$, which are constructed by two mutually prime block where $V_l \cup V_m = V_{s_1}$, $V_l \cap V_m = \phi$, such that $V_{s_1} \cup V_{s_2} \cup \ldots \cup V_{s_p} = V$, $V_{s_i} \cap V_{s_j} = \phi$. The partition between these two prime block is then optimized independently one each sub problem. We apply the tabu search to this optimized process to intensify optimizing subproblem, which converge rapidly as the characteristic of tabu search. It is obvious that the cost thus optimized can only be less than that of the original solution. If the structure of this initial solution is not too bad then it is likely that the solution obtained will be a very good one.

Finally, the entire procedure is clearly suited for parallel computing, as optimizations are done completely independently on each procedure, and can

thus be done concurrently. The global solution is then obtained by combining the subsolutions. This method needs no communication or synchronization except for its initialization and for grouping the subsolutions at the end of thread.

Super Node Move

Furthermore, we introduce the extension of searching by super node move, adding this strategy in the end of steps of thread. Super node move correspond to the displacement of a complete portion of the block from one point to another. This choice is motivated by noting that different good solutions of a same instance generally contain a great proportion of identical. We consider enlarging the search, which can not move by only swapping of two nodes, using this super node move. To explain this super node move in detail, we consider two blocks (V_s, V_t), *such that, node* $i, j \in V_s$ and $l, m \in V_t$. Let $\delta(i, l \mid V_s, V_t)$ be the difference in cost when node i and l is exchanged. The difference in cost when super node (i, j) and (l, m) is exchanged is represented as $\delta(i, j; l, m \mid V_s, V_t)$. If following condition is satisfied,

$$\exists j \in V_x; e_{ij} > 2 \cdot \max\Big(\sum_{u \in V_t, u \neq l} e_{iu}, \sum_{u \in V_s, u \neq i} e_{lu}\Big). \tag{4}$$

Then $\delta(i, l \mid V_s, V_t) > 0$. Therefore, there is little probability of the adapting this neighbour move. However, if we use super node move in the case of following conditions;

$$\sum_{u \in V_s, u \neq j} e_{iu} + \sum_{u \in V_s, u \neq i} e_{ju} < \sum_{u \in V_t, u \neq l} e_{iu} + \sum_{u \in V_t, u \neq m, l} e_{j,u}, \tag{5}$$

$$\sum_{u \in V_t, u \neq m} e_{lu} + \sum_{u \in V_t, u \neq l} e_{mu} < \sum_{u \in V_s, u \neq i} e_{lu} + \sum_{u \in V_s, u \neq i, j} e_{mu}, \tag{6}$$

$$e_{im} = e_{lj} = 0, \tag{7}$$

Then, we have $\delta(i, j; l, m \mid V_s, V_t) > 0$, and can take advantageous neighbour

move.

Generator Agent

However, it can not show the real ability only using above-mentioned strategies, because it strongly limits the movement possibilities between different subproblems. Therefore, we introduce the agent who manages producing subproblem to diversify. The subproblems is generated by simple agent, called generator, with following simple characteristics:

- It makes the rounds of all processors and learns the information of the partial solution in each processor. (This agent has some memory).
- It produces independent partial solution mutually considering the partial problem of another processors, which can combine without contradiction.
- It can have various strategy in generating partial problem and movement between each processors.

This agent with simple ability manages producing sub problems passing through processors. We let the agent have the strategy which produce various partial problem randomly, to give the system great diversity. That is, we have a number of combinations of partial solution, and can take opportunity, which overcome to limits the movement possibilities between different subproblems. Moreover, by this agent, we can construct parallel computing system environment, which not depend on each of computer ability (CPU speed). It can implement non synchronize parallel computing algorithm easily, which is most flexible parallel algorithm.

4．3 Computational Results for Parallel Tabu Search Algorithm

First, we evaluate the performance of the parallel neighbour move algorithm, using eight partitioning test problem with the number of nodes ranging from 100 to 1000. Fig.1 illustrates the trace of CPU time required

for two processors and four processors by parallel neighbour move algorithm and for basic sequential tabu search algorithm. The results show that parallel neighbour move algorithm reduces computational time lineally as the number of processors. It should be noted that this type of parallelism for tabu search, in spite of simple parallel algorithm, allows us to reduce considerably the CPU time.

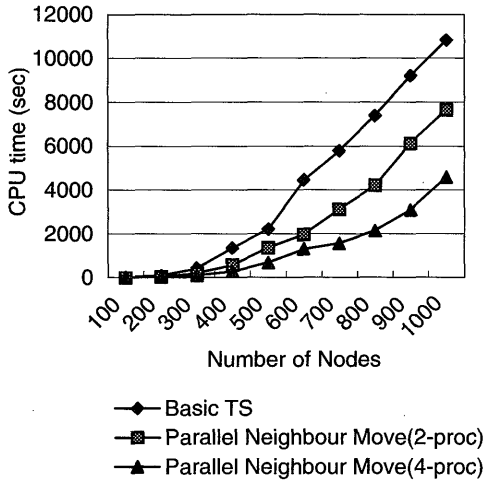

Fig.1 CPU times for parallel neighbour move algorithm

Next, parallel agent passing tabu search algorithm has been tested with the number of four processors for the eight partitioning test problem ranging from 100 to 1000 nodes. For each size and configuration, the procedure has been applied on ten times. In our tests, we always took *inner loop* = 5, which is step of thred, and *tabulenth* = 2, which seem to yield approximately good solutions. Table 2 shows the average cost, the best cost, and the worst cost obtained after 10 repetitions of the parallel agent passing tabu search algorithm and simulated annealing method and the cost for the basic tabu search method. The computational results show that the parallel

Table 2 Computational results for parallel agent passing tabu search in comparison with basic meta-heuristics

Program Size	Parallel Agent Passing TS			Simulated Annealing			Basic TS
	mean	best	worst	mean	best	worst	
100	3787.2	3774.0	3801.0	3704.8	3687.0	3734.0	3812.0
200	16270.0	16219.0	16334.0	16269.6	16238.0	16342.0	16335.0
300	37645.0	37583.0	37706.0	37794.0	37744.0	37879.0	37710.0
400	68653.6	68520.0	68761.0	68968.8	68770.0	69104.0	68843.0
500	107607.6	107427.0	107761.0	108425.6	108302.0	108526.0	108055.0
600	156887.2	156746.0	156988.0	157643.2	157549.0	157880.0	157788.0
700	215295.8	215183.0	215356.0	216348.2	216155.0	216730.0	216037.0
800	283407.8	283134.0	283778.0	284325.4	284153.0	284828.0	284427.0
900	361363.8	361204.0	361571.0	362609.4	362369.0	363012.0	362989.0
1000	446172.0	445922.0	446351.0	447511.0	447367.0	447840.0	448997.0

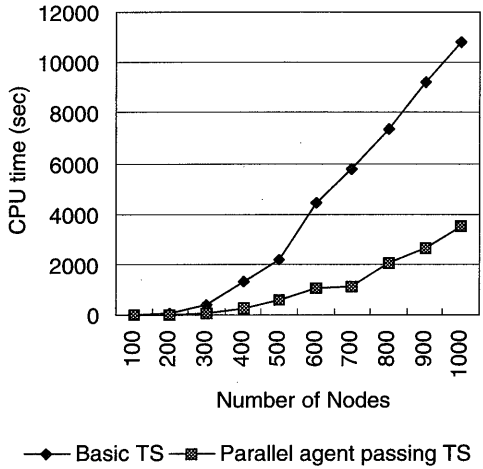

Fig.2 CPU times for parallel agent passing TS

agent passing tabu search algorithm givens a better solution than the simulated annealing algorithm and basic tabu search as a matter of course. In Fig.2, we also found that the computational time required for the parallel agent passing tabu search algorithm is approximately 1/3 of that required for the

basic tabu search algorithm. This parallelism is efficient in the improvement for quality of solutions and computational time required.

5 CONCLUSION

In this paper, we have proposed an approximation algorithms based on the parallel framework to solve graph multi-partitioning problem. First approach of the parallel neighbour move algorithm gained linear speedup in spite of easy idea, and showed the effectiveness as parallel algorithm. Another parallel agent passing algorithm, which is multi thread type using agent, gives better solution than the simulated annealing and basic tabu search and achieve speedup. This can be optimized attractively by partitioning the problem itself into several independent subproblems. It should be emphasized that we have used simple agent for diversification in managing to produce subproblem, which is optimized independently, and it has showed the importance of the diversification concept of tabu search. We have further shown the importance of extension of searching by super node move, which is displacement of a complete portion of the node from one block to another. We illustrate here how tabu search can be used for guiding an optimization process at a high level rather than at every basic step of the search. The results presented for the graph multi-partitioning problem that this method can efficiently be used for obtaining near optimal solutions for wide variety of partitioning problem.

In the foreseeable future, parallel computers will certainly grow fast in power and will become an interesting vehicle in the solving of NP-hard problems. High-level tabu search seems to offer a good framework for developing parallel heuristic search procedures.

REFERENCE

[1] Aarts, E., Korst, J. (1989) Simulated Annealing and Boltzmann Machines, John Wiley&Sons.

[2] Fiechter, C.-N. (1994) A parallel tabu search algorithm for large traveling salesman problem, Discrete Applied Mathematics 51, 243-267.

[3] Glover, F. and Laguna, M. (1997) Tabu Search, Kluwer Academic Publishers.

[4] Gropp, W., Lusk, E., and Skjellum, A. (1999) Using MPI, The MIT Press.