SOLVING LARGE SCALE SET PARTITIONING PROBLEM TO OPTIMALITY IN PARALLEL

Tran Van Hoai
University of Technology, VNU-HCM

***ABSTRACT**: Various practical applications can be modeled as a set partitioning (SP) problem. Instead of modeling the problem as an assignment model, in which variables correspond to a mapping of demands to resources, all possibilities of assignment are generated explicitly or implicitly in a systematic way. Then, a solution method to the generated SP problem is to choose the best subset of them to cover all demands. The obstacle is that the SP problem is NP-Hard. This paper presents a research to computationally solve the problem on parallel computers. The parallelism is performed on a sequential branch-and-cut based solver which employs advanced methods and techniques to the problem. Computational results solving solve large scale instances generated from different practical applications on a cluster of workstations show that optimality* can be reached within a reasonable computation time.

1.INTRODUCTION

Set partitioning is one of the most widely used models in practical applications. The formulation is as follows.

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax = e \\ & x \in \{0,1\}^n, \end{aligned} \tag{1}$$

where A is an $m \times n$ matrix of zeros and ones, c is an arbitrary n-vector which presents the preference of choosing a variable. Note that the right hand side e is an m-vector of 1's.

Let $M = \{1, ..., m\}$ be the row index set and $N = \{1, ..., n\}$ be the column index set of Eq. 1. For each column A_j, let $M^j = \left\{i \in M : A_{ij} = 1\right\}$ and associate the set with a cost c_j. We also denote by $N^i = \left\{j \in N : A_{ij} = 1\right\}$. With these notations, we can interpret a set partitioning problem as finding a minimum cost family of subsets $M^j, j \in N$ which is a partition of M. In reality, it is not common that every subset of M belongs to the optimal partition. This relates to the fact that "application" constraints seems to be embedded into the definition of the set $\left\{M^j : j \in N\right\}$.

In order to know how the SP model can be used in applications, we consider a scheduling problem defined as follows. Given

- a finite set M,
- a constraint set C defining a family P of "feasible subsets" of M, and
- a cost associated with each member of P,

find a minimum cost collection of members of P, which is a partition of M. A general two-step framework below is often used to solve the problem.

- Step 1: Generate explicitly all feasible subsets of M according to the constraint set C (i.e., generate P). An approximation approach could be performed by creating a subset $P' \subseteq P$ so that the probability of an optimal solution being contained in P' is sufficiently high.
- Step 2: Create a SP problem in which P (P', for approximations) defines the set of columns. Solving the SP problem is equivalent to solving the original scheduling problem.

Applying the idea above, the SP problem is used to model many practical applications: bus crew scheduling, airline crew scheduling, vehicle routing, circuit design, facility location problems, timetabling, etc.

However, the SP problem is NP-hard [1] which has been researched through several decades. Different aspects of the problem have been investigated aiming to solving practical application effectively. Following the same target, this paper presents an effort to speed up the solution process of the problem.

The paper is organized as follows. Section 2 gives a survey of practical efforts to solve the SP problems. Computational aspects to solve the problem by a branch-and-cut approach is presented in Section 3, in which features of a sequential SP solver are studied and a transformation of the solver to parallel version is described right after. Experiments and computational results are showed in Section 4. The final section closes the paper with some concluding remarks and open topics for future research.

2.RELATED WORKS

Solving large scale SP problems is always a challenge to theoretical and practical researchers. In a well-known set of integer programming problems MIPLIB 2003, there are 27 problems (totally 60 problems in MIPLIB 2003) which have set partitioning constraints in which 3 of them also exist in an older test set MIPLIB 1998. This means the SP problem is computationally hard and still requires sophisticated solution methods.

In the scope of this paper, all aspects of the SP problem and sequential methods cannot be covered. Readers interested in theoretical view of the problem can read [2]. Another good survey for the research before 1980s is in [3]. Recent successful sequential computations to solve large scale SP problems are presented in [4, 5, 2]. More discussions on heuristics used for set covering/partitioning problems can be found in [6]. One of the most significant approaches to large scale combinatorial optimization nowadays is to apply parallel computing to solution methods. In this paper, only aspects of parallel computing to SP problems is considered.

One of the most successful works in this area is presented in [7]. The authors apply both domain decomposition and functional decomposition in a LP-based branch-and-bound algorithm. Their parallel SP solver has a controller to manage the overall algorithm and assign jobs (row/column removal, primal heuristics, cutting plane generation) to workers. The authors experiment their ideas on a cluster of 16 processors to solve SP problems coming from different practical applications. Computation shows some good results in speedup, but it is skeptical that the computation can be scale up easily to a larger number of processors.

Another way of using all computing resource of parallel computers is to perform a parallel search on branch-and-bound tree. This exploitation is often applied to general mixed integer programming solvers. An, innovative research belongs to the projects BCP and SYMPHONY [8]. The scalability of parallel computation is one of the main targets in these projects. Although the basic model is still master-slave, the hierarchical design is employed to partition computation into small groups with less inter-group communication. However, software developed in these projects are for general mixed integer programming problems. The efficiency of them cannot be highly expected.

The main contribution of this paper is to employ advanced techniques to solve SP problems and perform a parallelization which highly guarantees a good performance for large computations.

3.THE PARALLEL SET PARTITIONING SOLVER

3.1.Branch-and-cut components

In order to solve a problem efficiently, techniques to be employed should be problem-specific. To do that, a sequential SP solver is implemented with strong techniques and methods working well for SP problems. In following sections, components of the solver are described. Note that, the parallel version is then transformed from the sequential one by using a parallel library, called parallel ABACUS [9], for combinatorial optimization.

3.1.1.Branch-and-bound

In this paper, a thorough discuss of how to branch a branch-and-bound node and how to select the next open node will not be given. Many basic points in this are addressed in textbooks [6, 10]. One of the best surveys on search strategies is in [11].

The branching method used in the solver is suggested in [12]. This method is quite complicated, intending to obtain two targets simultaneously: x_j^* is close to 1/2 and $|c_j|$ is large. In order to find j, the method firstly computes two values:

$$L = \max_{j \in F} \{x_j^* : x_j^* \leq 1/2\}, U = \min_{j \in F} \{x_j^* : x_j^* \geq 1/2\}.$$

Now, with a given parameter $\alpha \in [0,1]$ (by default, $\alpha = 0,25$), choose $j = \arg\max_{j \in F} \{|c_j| : (1-\alpha)L \leq x_j^* \leq U + \alpha(1-U)\}$. The implementation also applies strong branching because it is quite useful for degenerate problems like SP problems.

3.1.2.Preprocessing

The constraint matrix of the SP problem is so special and can be preprocessed efficiently by advanced techniques. The sequential solver implements the following techniques:

- Duplicated columns: It is quite clear that if $M^k = M^l$ and $c_k \geq c_l$, the column k can be removed without any change to the problem.
- Dominated rows: For two rows $k, l \in M$, if $N^k \subseteq N^l$, we can set all $\{x_j : j \in N^l N^k\}$ to 0 and remove the row l.
- Row cliques: If a column k is neighbors to every column l of a row, x_k can be removed. (Two columns k,l are neighbor to each other if $M^k \cap M^l = \phi$.)
- Row singleton: If a row has only one nonzero coefficient, the associated variable can be set to one.

Moreover, throughout the tree search the reduced cost fixing is also applied to fix and set variables using (global and local) lower bounds and the global upper bound.

3.1.3.Cutting plane generation

Although general cutting planes, such as the Chvátal-Gomory cutting planes, the lift-and-project cutting planes, have been revisited and introduced good performance, they are locally valid. Generating and storing them for large trees can lead memory overload. Instead, the implementation only generates globally valid cutting planes: clique inequalities, odd cycle inequalities and odd wheel inequalities.

- Clique inequalities: Clique inequality is proved to be a facet of the polytope. However, separating clique inequalities is a difficult problem ([13]). Therefore, only fast methods of generating the same style of inequalities on complete subgraph are studied.
- *Row-lifting* [14]: The main idea of the method is starting from a small complete subgraph of fractional variables which can be obtained easily from the constraints of the model. Then, other fractional and zero-valued variables are lifted into the starting set.
- *Greedy heuristic [*2]: In this method, the starting variable set of the valid inequality comprises only one fractional variable. The method will then try to include other fractional variables into the set. Similar to the row-lifting, zero-valued variables will be included into the set. However, they are computed in a different way. We find all variables neighbor to the fractional set. In order to guarantee that they create a complete subgraph, the set of all these zero-valued variables will be intersected with the rows to find out the maximum cardinality set. Finally, they will be inserted into the fractional set to make the final inequality.
- *Recursive Smallest First* [15]: This method aims at solving the following recursive equation:

$$\max_{Q \text{ clique in } G} \sum_{k\in Q} x_k^* = \max\left\{ x_j^* + \max_{Q \text{ clique in } G[N(j)]} \sum_{k\in Q} x_k^*, \max_{Q \text{ clique in } G-j} \sum_{k\in Q} x_k^* \right\}, \tag{2}$$

where G[N(j)] is the graph induced from the node set N(j). The efficiency of solving the equation depends much on the way of choosing the variable j. One is to choose the variable which has the smallest number of incident edges in the associated intersection graph.

- Odd cycle inequalities: The inequalities can be separated in polynomial time by an so-called GLS algorithm ([13]). The main idea of the algorithm is to transform the separation problem to finding the shortest path on a new graph which is generated from the intersection graph of the fractional variables. Let G=(V,E) be this intersection graph. We construct the new bipartite graph G_B as follows: the node set of G_B consists of two copies V' and V" of V"; an edge u'v" is in G_B if uv is in G.

We easily realize that a path P_u from u' to u" in G_B corresponds to an odd cycle C_u in G. The weight $v_{u'v''}$ is assigned with $1-x_u^*-x_v^*$. By doing so, we have

$$w(C_u) = \sum_{u'v''\in P_u} (1-x_u^*-x_v^*)$$
$$= |C_u| - 2\sum_{u\in C_u} x^* u$$

Therefore, we have the following equivalences:

$$w(C_u) < 1 |C_u| - 2\sum_{u\in C_u} x_u^* < 1 \sum_{u\in C_u} x_u^* > \frac{|C_u|-1}{2}$$

This proves that an odd cycle having the weight less than 1 corresponds to a violated odd cycle inequality. The most violated odd cycle can be found by solving the shortest path problem on the associated graph G_B. In the implementation of the algorithm, when labeling the neighboring nodes, it is only necessary to consider labels whose distances are less than 1. Note that, with the weight assignment above, $0 < w_{u'v''} < 1$.

- Odd wheel inequalities: They are lifted from the odd cycle inequalities.

3.1.4.Primal heuristics

One of the difficult problems associated with the branch-and-bound approach is that the number of nodes grows drastically. This leads not only to memory overload, but also to a rather time consuming computation. In that case, a "good" feasible solution is quite important to fathom nodes which cannot give a better solution. Two LP-based primal heuristics will be discussed in this section and used in the branch-and-cut code.

- Dive-and-fix: The idea of this method is to solve the linear relaxation of an integer problem and fix some fractional variables to suitable bounds. Certainly, if we fix variables which are nearer to integer points, there are more possibilities that the remaining problem is integer feasible.
- Near-integer-fix: It can be said that the second heuristic is also a variant of the dive-and-fix heuristic. Instead of simply fixing fractional variables nearest to integer points, the second method employs a more complicated technique of choosing variables to be fixed. A main difference between these two variants is that the near-integer-fix heuristic will not fix a given number of variables. It will fix all fractional variables whose integer distances fall below a given number.
- Small set partitioning problems: Working on a small portion of the variable set is an easier task and a way to find primal bound for the branch-and-bound framework. The heuristic presented in this section aims at solving a set partitioning problem containing a subset of variables. Columns which produce the small problem are:
- Variables currently being in the basis of the last linear relaxation,
- Non-zero valued variables,
- Zero valued variables. These variables are added to the model in order to guarantee the ratio between the number of variables and the number of constraints approximately equal to a given value. Furthermore, all constraints should be covered equally by the chosen variables.

There are still many heuristics suggested for solving set partitioning problems. Some of them are: dual heuristics [4], the dual cost perturbation heuristic [5], small set partitioning heuristic [7]. More discussions on heuristics used for set covering/partitioning problems can be found in [6].

3.2.Parallelization of the sequential solver

Very specific points will not be presented in implementation which can distract the attention of readers. One can read [9] for more details on the design of parallel ABACUS. The process of parallelizing the sequential code is quite simple. Following the instructions in [9, 16], we can transform any sequential combinatorial optimization code to a related parallel version with a little effort.

Finally, after being compiled and linked with the parallel ABACUS on a parallel computer supporting MPI, a parallel solver is ready to perform computations. It is an advantage of the parallel ABACUS. The expense of designing a parallel code from a sequential one is quite small. Furthermore, the resulting code can be used on many parallel computers. The computational results of the parallel set partitioning solver will be studied in the next section.

4.PRELIMINARY COMPUTATIONAL RESULTS

The interesting aspects of computational results should be described firstly. The last three column in Table 1 are dedicated to the computation time of the solver. They are "%Par" the parallel library time, "%Idle" the idle time, and "%CPU" the CPU time of parallel computation. There should be a remark here on how to collect these quantities. The measurement is only started after an integer program formulation has been read into memory. Moreover, they will not be shown in any time unit. Instead, they are viewed as a percentage of the total computation time which is shown in the column "ttotal". This way of presentation is expected to have a better view of the solver's computation. The overhead of the library is quite interesting to observe, especially with the help of columns "%Par" and "%Idle". It is quite obvious that when the idle time increases, the number of calls to communication functions also increases. This leads to a high CPU time within the parallel library. However, the overall computation time will not be influenced.

The name of test instances is in the first column which is followed by the column "#proc" showing the number of processors involved in computation. Column "B&B" presents the total number of branch-and-bound nodes. Due to the anomaly phenomenon in parallel search, these numbers can change significantly with different numbers of processors involved.

The settings for the SP solver are configured as follows.

- Linear programming solver: CLP
- Cutting plane generation: cliques and odd cycle inequalities,
- Branching: Padberg-Rinaldi with α =0.25; strong branching whose the number of candidates is 10,
- Node selection strategy: best first search,
- Preprocessing: all methods supported by the sequential runs,
- Heuristics: dive-and-fix and near-int-fix.

The parallel system for our computation is Supernode II which is the cluster of 64 dual processor computers. The network is Gigabit Ethernet to connect all computing nodes. The processor type is Intel Xeon 2.4 GHz and each computer has 1 Gbyte RAM.

Test instances used in our experiments come from two practical applications. Instances "air01", "air04", and "nw04" are models of the crew pairing problem of airlines companies. In order to find an optimal set of pairings, the original problem is transformed to a SP problem in a similar way mentioned in Section 1. In a same way, the capacitated vehicle routing problem can be solved as a SP problem. That is the instance "eil33.1".

Table 1. Computational results for large scale SP instances

Name	#proc	B&B	%Par	%Idle	%CPU	ttotal
aa01	1	281	0.00	0.00	97.28	0:26:04
	2	216	8.31	13.44	88.29	0:14:34
	4	192	24.72	27.56	71.44	0:08:12
	8	264	46.62	53.28	54.75	0:06:35
	16	219	67.30	79.31	39.96	0:06:26
	32	277	66.40	88.90	31.69	0:07:28

aa04	1	239	0.00	0.00	97.99	0:28:47
	2	465	3.95	5.44	93.11	0:21:11
	4	332	9.63	12.86	85.43	0:09:04
	8	495	27.23	27.94	69.43	0:06:50
	16	537	54.19	59.88	47.94	0:05:18
	32	696	65.49	75.21	38.77	0:05:27
nw04	1	331	0.00	0.00	96.15	0:37:39
	2	167	9.54	30.93	63.01	0:18:23
	4	312	35.82	49.90	59.51	0:14:31
	8	235	33.79	78.69	37.75	0:16:26
	16	212	40.54	90.06	32.22	0:16:32
	32	326	44.86	94.91	26.59	0:32:05
eil33.1	1	4999	0.00	0.00	98.50	1:38:30
	2	4972	2.07	3.41	96.03	0:46:52
	4	5202	6.30	8.26	91.24	0:24:21
	8	5702	15.29	17.94	82.98	0:14:36
	16	8881	41.80	39.00	60.65	0:11:15
	32	8079	55.46	65.25	48.34	0:08:51

With the default settings, we obtain a quite good performance of computational results in Table 1. The most important target of the design has been achieved concerning the reduction of the computation time in most cases. With a sufficient number of processors, the total computation time is reduced for all test instances. If the idle percentage is not so high, the speedup is quite good. However, bad efficiency is received for instances with few branch-and-bound nodes, such as in "aa01", "aa04", "nw04" although there are still time reductions.

Now, we will try to investigate in more details why the poor performance happens when using many processors. Since the parallel design is based on the idea of polling through MPI communication objects to check their completions, the overhead of the parallel ABACUS library naturally increases if there are very few open nodes globally which are not satisfied by many idle processors. This often occurs in the beginning or at the end of a run. The tree search for the problem "nw04" is visualized in Figure 1.a as an example. This results in bad utilization of all processors. Columns "%Par" and "%Idle" in Table 1 show the increase of the parallel overhead and the idle time when increasing the number of processors. From the table, we imply that most of the parallel overhead is due to the idleness of processors. With this weakness, one way to improve performance for test instances with small number of branch-and-bound nodes is to use a different approach of parallelization.

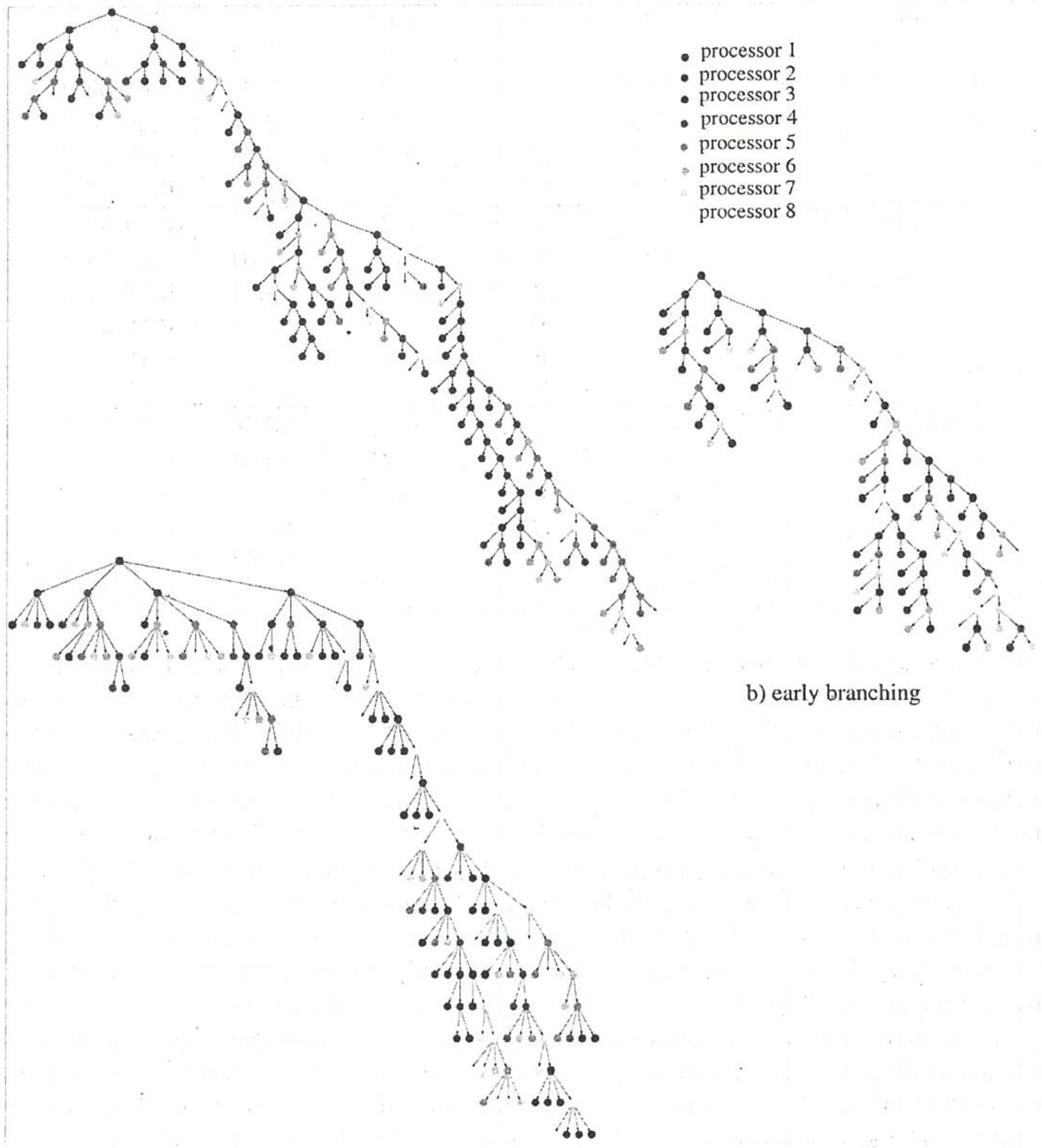

Figure 1. Search trees of "nw04" using 8 processors

In order to cope with bad behavior as shown in the first tree of Figure 1, we employ some balancing techniques to reduce the idleness of processors. The first one is early branching which forces a processor to stop its separation immediately when realizing there are not enough nodes for idle processors. As seen in the computation using the default settings, processors are often idle because there are not enough problems for them. It is expected that the early branching technique will do a better job in the same situation. However, stopping the cutting plane generation very early is possibly not a good choice because doing that makes a branch-and-cut code only execute a simple branch-and-bound algorithm. But we can receive a benefit by reducing the idle time of processors. It is intuitively clear that there are usually more numbers of branch-and-bound nodes in comparison with the previous default computation.

The second balancing method in examination is the multi-branching method which will generate enough nodes for idle processors. The third tree in Figure 1 shows all processors are to be utilized at any stage of the computation. Even though there are very few open nodes at some time, the multi-branching method also prefers to creating jobs for idle processors. Remember that, if we branch on many variables at a time, the feasible region of the child nodes is stricter in comparison with branching on one variable only. The reason is quite obvious as we fix many variables at a time. Then, the multi-branching helps to reduce the tree depth a lot (e.g., the depth of the third tree is half of that of the first tree for "nw04"). The number of branch-and-bound nodes is usually increased with respect to the number of processors.

5.CONCLUSION

Solving large scale SP instances to optimality requires specific-problem methods, techniques and even power computing resource. The paper shows a study to develop an efficient parallel SP solver which has been tested on several large instances from various practical applications. The computational results show that the solver is scalable for large problems in terms of the number of branch-and-bound nodes. With a suitable number of processors, solving a set of well-known large scale test instances in parallel, the solver reduces significantly the running time in the default settings. Researchers now have a tool to study large combinatorial optimization problems model in SP formulation. The study also presents several potential possibilities in future. One of them in aspects of parallel computing is to upgrade efficiently the solver to larger computing environment, such as Grid. The memory management for large branch-and-bound search should be studied in order to solve some SP instances ("ds", "t1717") to optimality.

ACKNOWLEDGMENT: The author would like to thank for the support and supervision of professor Hans Georg Bock and professor Gerhard Reinelt during his PhD study which mainly contributes the study presented in this paper.

GIẢI TỐI ƯU BÀI TOÁN PHÂN HOẠCH TẬP HỢP KÍCH THƯỚC LỚN DÙNG TÍNH TOÁN SONG SONG

Trần Văn Hoài
Trường Đại học Bách khoa, ĐHQG-HCM

ABSTRACT: *Nhiều ứng dụng thực tế trong những lĩnh vực khác nhau có thể được mô hình dưới dạng bài toán phân hoạch tập hợp. Thay vì thể hiện ứng dụng dưới dạng mô hình gán (assignment model), dạng các biến tương ứng với những ánh xạ từ yêu cầu đến tài nguyên, phương pháp được chọn là tạo ra tất cả những khả năng gán có thể một cách tường minh hoặc không tường minh trong một trình tự có hệ thống. Sau đó, cách tìm nghiệm được quy về bài toán tìm một tập con tốt nhất của chúng mà phủ tất cả những yêu cầu. Một trở ngại chính của bài toán phân hoạch tập hợp là nó thuộc nhóm NP-hard. Vì thế bài báo này sẽ giới thiệu một cách tiếp cận dùng máy tính song song để giải quyết trở ngại này. Một thư viện tuần tự sử dụng những phương pháp và kỹ thuật nâng cao trong thuật toán dựa trên nhánh-và-cắt được chuyển sang dạng song song. Kết quả tính toán trên những dữ liệu từ các ứng dụng thực*

tế khác nhau cho thấy nghiệm tối ưu của chúng có thể đạt được trong một thời gian tính toán hợp lý nếu sử dụng máy tính song song.

REFERENCES

[1]. Garey, M., Johnson, D.: *Computers and Intractability: A Guide to the Theory of NP-completeness.* Freeman (1979)

[2]. Borndörfer, R.: *Aspects of Set Packing, Partitioning, and Covering.* PhD thesis, Technischen Universität Berlin (1998)

[3]. Balas, E., Padberg, M.: S*et Partitioning: A Survey.* SIAM Review 18(4) (1976) 710–760

[4]. Fisher, M., Kedia, P.: *Optimal Solution of Set Covering/Partitioning Problems Using Dual Heuristics.* Management Science 36 (1990) 674–688

[5]. Wedelin, D.: *An Algorithm for Large Scale 0-1 Integer Programming with Applications to Airline Crew Scheduling.* Annals of Operations Research 57 (1995) 283–301

[6]. Nemhauser, G.L., Wolsey, *L.A.: Integer and Combinatorial Optimization.* John Wiley & Sons, New York (1988)

[7]. Linderoth, J.T., Lee, E.K., Savelsbergh, M.W.P.: *A Parallel, Linear Programming-based Heuristic for Large-Scale Set Partitioning Problems.* INFORMS Journal on Computing 13(3) (2001) 191–209

[8]. Ralphs, *T.K.: SYMPHONY 3.0 user's manual* (2002)

[9]. Tran, V.H.: S*olving Large Scale Crew Pairing Problems.* PhD thesis, University of Heidelberg (2005)

[10]. Wolsey, L.A.: *Integer Programming.* John Wiley, New York (1998)

[11]. Linderoth, J.T., Savelsbergh, M.W.P.: *A Computational Study of Branch-and-Bound Search Strategies for Mixed Integer Programming.* INFORMS Journal on Computing 11 (1999) 173–187

[12]. Padberg, M., Rinaldi, G.: *A Branch-and-Cut Algorithm for the Resolution of Large Scale Symmetric Traveling Saleman Problems.* SIAM Review 33(1) (1991) 60–100

[13]. Grötschel, M., Lovász, L., Schrijver, A.: *Geometric Algorithms and Combinatorial Optimization.* Springer, Berlin (1988)

[14]. Hoffman, K., Padberg, M.: *Solving Airline Crew Scheduling Problems by Branch-and-Cut.* Management Science 39 (1993) 657–682

[15]. Carraghan, R., Pardalos, P.M.: *An Exact Algorithm for the Maximum Clique Problem.* Operations Research Letters 9 (1990) 375–382

[16]. Böhm, M.: *Parallel ABACUS - Introduction and Tutorial.* Technical report, University of Cologne (1999)