The Japan Society of Mechanical Engineers

International Symposium on Scheduling 2004
May 24-26, 2004, Awaji-Yumebutai, Japan
K1

BUILDING GENERAL SOLVERS FOR SCHEDULING PROBLEMS

Toshihide IBARAKI
Graduate School of Informatics,
Kyoto University,
Kyoto, Japan 606-8501
ibaraki@i.kyoto-u.ac.jp

Abstract

We describe our attempts to build general solvers that cover a large portion of schduling problems encountered in real world applications. For this, we select a list of standard problems, and develop their solvers which are based on local search and metaheuristics. As standard problems, we have chosen so far RCPSP (recource constrained project scheduling problem), CSP (constraint satisfaction problem), VRP (vehicle routing problem), SCP (set covering problem) and others. In this paper, we outline definitions of these problems, algorithmic contents of solvers, and some computational results.
Keywords: General solvers, scheduling problems, local search, metaheuristics, standard problems.

1. Real World Scheduling Problems

Scheduling problems are abundant in real world applications. They take quite different forms, sizes and complexities. We give below four types of scheduling problems, as examples which can be commonly found in various areas of social and industrial activities.

Machine scheduling: There are n jobs to be processed on m machines, each of which handles only one job at a time. Other resources such as operators, machine tools and associated materials also have to be assigned when a job is processed on a machine. The objectitive is to find a time schedule of all jobs on m machines, which minimizes a certain measure such as the time to complete all jobs, while satisfying the resource and technological constraints.

Workforce scheduling: A typical example in this type of scheduling is the nurse scheduling problem in a hospital, where the work time table of n nurses is constructed over a specified time horizon, e.g., one month. There are three shifts, day, evening and night, in one day, and each shift in each day requires a given number of nurses. Also from the side of each nurse, there are a number of constraints, e.g., at least one day-off every week, one day-off after consecutive night shifts, no three consecutive night shifts, no single isolated night shift, no four consecutive day shifts and so forth. The total numbers of day, evening and night shifts of each nurse in each month, respectively, have to be kept within given upper and lower bounds. There are usually other constraints particular to each hospital. Then find a shift assignment of all nurses that minimizes the number of violations of given constraints.

Parcel delivery problem: There are vehicles to deliver parcels to customors, starting from a post office. Each parcel has its weight and each vehicle can carry some number of parcels within its capacity. It is asked to determine the set of customers which each vehicle serves, together with the travel route of each vehicle, so that the total travel distance of all vehicles is mimimized.

Crew scheduling problem: As a typical example, we consider the assignment of pilots to all flights scheduled in an airline company, where each flight must have at least one pilot. A seqence of consecutive flights attended by a pilot is called a leg, if it satisfies the safety and other regulations. Then we are asked to find a minimum number of legs, which together cover all flights.

Although all these are called scheduling problems, they have very different mathematical structures, which we need to exploit in order to produce effective algorithms. As a result, it is usually necessary to invest some amount of manpower and time to develop such solvers. Furthermore, as a characteristic of combinatorial optimization problems, such algorithms may exhibit quite different performance if some new constraints are added, or some new features are introduced to the problem, necessitating the development of an entirely new algorithm in some cases.

In view of these, we consider it very useful if we could develop general sovers, each of which can cover a wide range of schduling problems of similar types. Such solvers must be efficient, flexible, robust and easy to use.

NII-Electronic Library Service

The Japan Society of Mechanical Engineers

2. NP-Hardness and General Solvers

An important theoretical achievement in complexity theory is the concept of NP-completeness and NP-hardness. The NP is a class of problems that includes most of the combinatorial problems encountered in applications. Some problems in class NP can be shown to be NP-complete or NP-hard. (There are slight differences in the definitions of NP-completeness and NP-hardness, but we use only terminology NP-hard hereafter for simplicity sake.) For example, problems SAT (satisfiability) and IP (integer programming) are NP-hard.

Every NP-hard problem A has an important property that

any problem in class NP *can be reduced to* A,

implying that an algorithm for A can solve any problem B in class NP. In other words, the algorithm for A can be regarded as a general solver for all problems in NP. This is a positive side of NP-hardness.

Such a general solver is difficult to construct, however, because of the following negative side of NP-hardness.

NP-*hard problems are computationally intractable (not solvable in polynomial time).*

(To be more precise, this statement holds only if the famous N≠NP conjecture is true.) Therefore, it is not possible to build a general solver that works efficiently for all problems.

An approach to overcome this difficulty may be using an efficient approximate algorithm for an appropriate NP-hard problem A. As good approximate solutions are sufficient for most of the practical purposes, this approach appears quite meaningful. However, there are the following two obstacles.

- The problem size may explode in the process of reducing a given problem instance to a problem instance of A.
- A good approximate solution to A may not be a good approximate solution of the original instance, due to the distortion of the distance to optimal solution incurred during reduction process.

To avoid this, we have to allow only "natural" reductions, which are free of the above two defects.

Our conclusion from these observations is that we have to prepare a list of standard NP-hard problems, instead of a single problem that represent all problelms in NP. Based on this, we have defined several standard problems so far and developed effective approximate algorithms for them. This scheme is illustrated in Figure 1.

3. List of Standard Problems

Among possible standard problems for scheduling, we selected the following list so far. Each standard problem is selected from the view point that it is general enough to cover a wide range of important scheduling problems, flexible enough to allow various additional constraints and objective functions, and still has mathematical structures that make the development of efficient algorithms possible.

1. *Integer programming* (IP)
2. *Constraint satisfaction problem* (CSP)
3. *Resource constrained project scheduling problem* (RCPSP)
4. *Vehicle routing problem* (VRP)
5. *Set covering problem* (SCP)

IP solvers have been studied as the most representable general solvers for combinatorial problems including scheduling problems. A few commercial software packages are already available and appear very powerful. For this reason, we have not tried to develop IP solvers. Although IP is very general, there still remain many problems which are not appropriate for IP. Problems CSP, RCPSP and VRP can handle scheduling problems with some special structures that make direct application of IP solvers rather difficult. Problem SCP may be considered as a special case of IP, but can be solved more efficiently by exploiting its structure.

CSP: In principle, CSP allows any type of constraints, and tries to find a solution that satisfies all constraints. If no such solution exists, CSP finds a solution with a minimum violation of constraints. The violation of constraints is usually measured by the sum of weights given to the violated constraints. In our implementation, we treat linear equality and inequality constraints under 0-1 variables as standard constraints (like IP approach), as well as non-equal ($\neq$) constraints. Quadratic constraints of 0-1 variables can also be handled. Any domain of each variable consisting of finite values can be transformed into domains of 0-1 variables by introducing value variables (which are 0-1 valued). By adding constraints oracles (prepared by users), any constraints can be incorporated into the solver.

RCPSP: This problem asks to determine start times of n activities, where each activity consumes some amounts of resources, such as machines, tools, man-power, energy, budget, raw materials and so forth, and the available amounts of those resources (which are time-dependant) are given as constraints. The resources are classified into renewable and nonrenewable. To process each activity, one mode from a given set of modes can be chosen, where each mode has its own process time and resource consumption. Other aspects such as precedence constraints between activities, setup times and various types of objective functions can also be taken into account. If there is no schedule that satisfies all constraints, it is asked to output a solution with the minimum constraint violation, as in CSP.

VRP: There are m vehicles and n customers, where a good of weight w_j must be delivered to customer j and the capacity of each vehicle is Q. It is asked to assign those customers to vehicles, so that the capacity constraint is satisfied for each vehicle, and then to determine the route of each vehicle. Each customer may have its time window constraint

NII-Electronic Library Service

The Japan Society of Mechanical Engineers

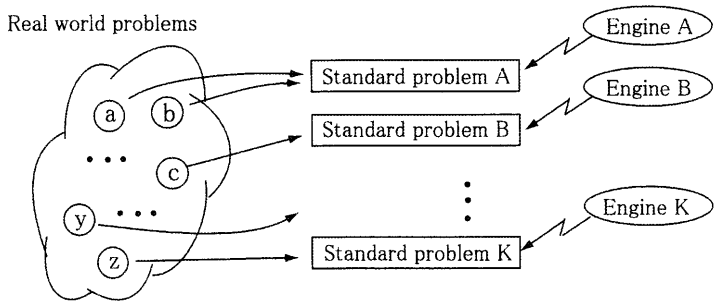

Fig. 1 General solvers via standard problems

that specifies the time slots in which the good can be delivered. In such a case, not only the route of each vehicle but also the start time of serving the customer have to be determined. A typical objective function to minimize is the total length of all vehicles d_{sum}. A typical example of VRP is the parcel delivery problem described previously.

SCP: Given a family of subsets $S_j, j = 1,2,\ldots,n$ of a ground set $S = \{1,2,\ldots,m\}$, SCP asks to find a subfamily $S_{j_l}, l = 1,2,\ldots,k$ such that the sum of weights given to S_{j_l} is minimized under the constraint that union of the selected subfamily covers S. This problem has wide applications since certain types of vehicle routing problems and crew scheduling problems can be formulated as SCP.

4. Algorithmic Aspects of Solvers

The solvers for the above standard problems must be efficient so that large scale instances arising in practice can be handled, flexible so that constraints and objective functions particular to applications can be included, and robust against small structural changes in the problem. A key question here is whether algorithms with such characteristics exist or not. We believe that the framework of metaheuristics based on *local search* (LS) is the one for such purposes. Metaheuristics include as special cases such algorithms as *genetic algorithms* (GA), *evolutionary computation* (EC), *simulated annealing* (SA), *tabu search* (TS), *iterated local search* (ILS) and others.

Local search: LS starts from an appropriate initial solution x, and repeats the operation of moving to a better solution x' (i.e., $x := x'$) in its neighborhood $N(x)$ if such a solution exists. If there is no better solution in the neighborhood $N(x)$, solution x is called *locally optimal* and LS halts there.

The performance of LS depends on how the solution space and the neighborhood are defined, and how other details are implemented such as construction of initial solutions and the order of searching the solutions in the neighborhood, and when to move to a new solution (e.g., the best solution in $N(x)$ or the first improved solution found).

Metaheuristics: Algorithms in metaheuristics use LS as their important ingredient, and repeat the processes of generating an initial solution and its improvement by LS in the following manner.

> METAHEURISTICS
> I (Initial solution): Generate an initial solution x.
> II (LS): Improve x by applying (generalized) LS.
> III (iteration): If the stopping criterion holds, halt after outputting the best solution found so far. Otherwise, return to I.

To generate initial solutions in I, it is common that the computational history by then is taken into consideration. For example, a certain number of good solusions are maintained during computation, and initial solutions are generated by combining them in some manner. In GA, offspring is generated from a selected pair of good solutions by a *crossover* operation. In ILS, initial solutions are generated by ramdomly modifying the best solution in the pool.

The generalized LS in II for example permits the randomized search in $N(x)$ and the move to a worse solution with certain probability. The probality is controlled by a parameter called *temperature* in SA, to diversify the search in the initial phase and then concentrate the search to the promising area found in the initial phase. In TS, the move in II is always done to the best solution in $N(x)$ even if it is worse than x. In this case, to prevent cycling of solutions, a *tabu list* of solutions is prepared and the moves to tabu solutions are prohibited, where tabu list usually contains a certain number of most recently visited solutions or a set of features of such solutions.

The stopping criterion in III can be very simple, e.g., it stops if a specified time limit of computation is over. In other cases, more sophisticated criterion may be used to consider the computational history such as when best solutions have been improved during iterations of I$\sim$III.

Detailed description on metaheuristics can be found for example in (Yagiura and Ibaraki, 2001).

We have developed solvers for the above standard problems CSP, RCPSP, VRP and SCP, following the framework of metaheuristics. Some are based on TS and others are based on ILS. For details of these solvers, please see the references (Nonobe and Ibaraki, 2001; Nonobe and Ibaraki, 2002; Ibaraki *et al.*, forthcoming; Yagiura, Kishida and Ibaraki, submitted).

NII-Electronic Library Service

The Japan Society of Mechanical Engineers

5. Modeling as Standard Problems

Most important factors pertaining to the success of our approach are perhaps how to select an appropriate standard problem from the list, and how to model the given problem instance into a compact instance of the standard problem. This is not an easy task. It requires a deep insight into the relation between structure of the given problem and behavior of the selected solver.

As an example, consider the parcel delivery problem given in Section 1, which can be naturally modelled as VRP. As noted in the definition of VRP, the problem may also have time window constraints. The problem may then be generalized by allowing pick-up and delivery situation, i.e., vehicles accept orders of delivering parcels from some customers to other customers, where the pick-up and delivery of a percel has to be done in the same route (without taking it back to post office). The resulting problem is no longer appropriate for the standard problem VRP, because the constraint of pick-up and delivery orders is not easily handled. One of the natural approaches in this case is to use SCP. We first construct a number of cadidate routes of vehicles by selecting appropriate sets of pick-up and delivery orders within the capacity constraint, and then choose a minimum number of routes from them so that all orders can be covered. The last problem can be easily formulated as SCP.

The SCP approach for the pick-up and delivery problem may also be supported from an algorithmic consideration of the local search for VRP. A solution in the local search of VRP is a set of routes that together covers all customers. We search an improved solution in the neighborhood by exchanging some subroutes in different routes in a systematic manner. Hoever, under the constraint of pick-up and delivery orders, such exchanges tend to destroy its fesibility, and finding feasible solutions in the neighborhood becomes very difficult. This appears to indicate that the local search for VRP is not effective.

We need this kind of careful consideration whenever solvers are used in applications. However, due to space limitaion, we omit further discussion here.

6. Implementation of Solvers and Their Computational Results for VRP and SCP

Extensive computational experiments of the solvers for the above standard problems have been conducted and can be found in the associated papers (Nonobe and Ibaraki, 2001; Nonobe and Ibaraki, 2002; Ibaraki *et al.*, forthcoming; Yagiura, Kishida and Ibaraki, submitted). Here, for two problems VRP and SCP, we explain some implementation details of solvers, and report a highlight of their computational results.

VRP: Our solver can handle the VRP with capacity and time window constraints. A constraint is called *hard* if it must be satisfied and is called *soft* if it can be violated. The amount of violation of soft constraints is usually penalized and added to the objective function. In our foumulation, time window and capacity constraints are both considered soft.

It should also be emphasized that the time window constraints we consider are very general in the sense that one or more time slots can be assigned to each customer. That is, the corresponding penalty function can be non-convex and discontinuous as long as it is piecewise linear. In this case, after fixing the order of customers for a vehicle to visit, we must determine the optimal start times of services at all customers so that the total time penalty of the vehicle is minimized. We solve this sub-problem by dynamic programming.

Let n_k be the number of customers assigned to vehicle k, and δ_k be the total number of linear pieces in the penalty functions for those customers. Note that δ_k is considered as the input size of the penalty functions of n_k customers, where $\delta_k = O(n_k)$ holds in many cases. The time complexity of our dynamic programming is $O(n_k\delta_k)$ if it is solved from scratch. We also show that the optimal time penalty of each solution in the neighborhood of the current solution can be evaluated in $O(\sum_{k\in M'}\delta_k)$ time from the information of the current solution, where M' is the set of indices of vehicles which the neighborhood operation involves.

The essential part of the solver, i.e., assigning customers to vehicles and determining the route of each vehicle, is based on local search (LS). In the literature, three types of neighborhoods, called the cross exchange, 2-opt* and Or-opt neighborhoods, have been widely used. In addition to these standard neighborhoods, we use a new type of neighborhood called the *cyclic exchange neighborhood*. This is defined to be the set of solutions obtainable by cyclically exchanging two or more paths of length at most L^{cyclic} (a parameter). As the size of this neighborhood grows exponentially with the input size, we propose an efficient heuritsic algorithm based on the *improvement graph*.

We use iterated local search (ILS) and adaptive multi-start local search (AMLS) in our framework of metaheuristics. ILS generates initial solutions for LS by perturbing good solutions obtained in the search by then. On the other hand, AMLS keeps a set P of good solutions found in the previous search, and generates initial solutions by combining parts of the solutions in P.

We report here some results on Solomon's benchmark instances (Solomon, 1987). The number of customers in each instance is 100, and their locations are distributed in the square $[0,100]^2$ of the plane. The distances between customers are measured by Euclidean distance, and the traveling times are proportional to the corresponding distances. Each customer i (including the depot) has a single time window $[w_i^{\text{r}}, w_i^{\text{d}}]$, an amount of requirement q_i and a service time u_i. All vehicles k have a fixed capacity Q. Both time window and capacity constraints are considered hard. For these instances, the number of vehicles m is also a decision variable, and the objective is to find a solution σ with the minimum $(m, d_{\text{sum}}(\sigma))$ in the lexicographical order.

These benchmark instances consist of six different sets called C1, C2, R1, R2, RC1 and RC2, respectively. Loca-

NII-Electronic Library Service

The Japan Society of Mechanical Engineers

Table 1 Comparison of the solution quality on Solomon's VRP instances

problem class		ILS 2000s	ILS 15000s	AMLS 2000s	AMLS 15000s	G&H (2002)	BBB (2003)	Br (2003)	BVH (2001)	H&G (2003)
C1	MNV	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00
	MTD	828.38	828.38	828.38	828.38	828.63	828.48	828.38	828.38	828.38
C2	MNV	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00
	MTD	589.86	589.86	589.86	589.86	590.33	589.93	589.86	589.86	589.86
R1	MNV	12.00	11.92	11.92	11.92	12.00	11.92	11.92	11.92	11.92
	MTD	1215.83	1214.26	1220.02	1217.40	1217.57	1221.10	1222.12	1213.25	1212.73
R2	MNV	2.73	2.73	2.73	2.73	2.73	2.73	2.73	2.73	2.73
	MTD	978.84	967.03	961.64	959.11	961.29	975.43	975.12	966.37	955.03
RC1	MNV	11.50	11.50	11.63	11.50	11.50	11.50	11.50	11.50	11.50
	MTD	1385.89	1385.42	1378.72	1391.03	1395.13	1389.89	1389.58	1384.22	1386.44
RC2	MNV	3.25	3.25	3.25	3.25	3.25	3.25	3.25	3.25	3.25
	MTD	1147.38	1131.24	1132.17	1122.79	1139.37	1159.37	1128.38	1141.24	1123.17
All	CNV	406	405	406	405	406	405	405	405	405
	CTD	57798	57516	57480	57444	57641	57952	57710	57567	57309

tions of customers are clustered in groups in type C, uniformly distributed in type R, and the two types are mixed in type RC. Furthermore, for instances of type 1, the time window is narrow at the depot, and hence only a small number of customers can be served by one vehicle (meaning that relatively many vehicles are required). Conversely, for instances of type 2, the time window is wide, and hence many customers can be served by one vehicle. Each type consists of from 8 to 11 instances.

The results in Table 1 compare the best solutions obtained by algorithms ILS and AMLS with other existing methods. In the table,

"MNV" represents the mean number of vehicles,
"MTD" represents the mean total distance,
"CNV" represents the cumulative number of vehicles, and
"CTD" represents the cumulative total distance.

Column "ILS 2000s" is the result of ILS, where the time limit for each instance is 2000 seconds. The meaning of columns ILS 15000s, AMLS 2000s and AMLS 15000s are similar. Other columns are taken from the following references: "G&H (2002)" is the result by algorithm HM4C in (Gehring and Homberger, 2002), "BBB (2003)" is the result by (Berger, Barkaoui and Bräysy, forthcoming), "Br (2003)" is the result by algorithm RVNS(2) in (Bräysy, forthcoming), "BVH (2001)" is the result by (Bent and Van Hentenryck, 2001), and "H&G (2003)" is the result by (Homberger and Gehring, forthcoming).

The average computational time of algorithms G&H (2002), BBB (2003), Br (2003), and BVH (2001) for each instance are roughly estimated as 800, 6000, 1300, 500, and 14000 seconds, respectively, if they were run on our computer. Computational time of H&G (2003) is not clearly stated in (Homberger and Gehring, forthcoming).

The solution quality of ILS and AMLS with the time limit of 2000 seconds are competitive with G&H (2002), but slightly worse than BBB (2003), Br (2003), BVH (2001), and H&G (2003). If much longer computational time, 15000 seconds, is allowed, both ILS and AMLS exhibit better quality than BBB (2003), Br (2003), and BVH (2001). Note that the computational time of our algorithms is roughly equivalent to algorithm BVH (2001).

These results are significant, since our algorithms are very general and not tailored to the VRP of Solomon's instances.

SCP: The set covering problem (SCP) can be formulated as the following special case of integer programming:

$$\begin{aligned} \text{Minimize} \quad & \sum_{j \in J} c_j x_j \\ \text{subject to} \quad & \sum_{j \in J} a_{ij} x_j \geq 1, \quad i \in M \\ & x_j \in \{0,1\}, \quad j \in N, \end{aligned}$$

where

$$a_{ij} = 1 \ (\text{if } i \in S_j), \ \ 0 \ (\text{otherwise}).$$

It is understood that variable x_j equals 1 if subset S_j is chosen, and 0 otherwise.

The SCP has been intensively studied, and various codes are available. Our code is based on the iterated local search and has the following features.

(1) The use of large neighborhood called the *3-flip neighborhood*, which is the set of solutions obtainable from the current solution by flipping up to three elements. As the size of the 3-flip neighborhood is $O(n^3)$, the neighborhood search becomes expensive if naively implemented. To overcome this, we employ an efficient implementation that greatly reduces the number of candidates in the neighborhood without sacrificing the solution quality.

(2) It is allowed for the search to visit the infeasible region, and the *strategic oscillation* technique is incorporated.

(3) The size reduction of the problem by using the information from the Lagrangean relaxation of SCP is added, which turned out to be effective in solving very large instances.

NII-Electronic Library Service

The Japan Society of Mechanical Engineers

Table 2. Computational results of SCP algorithms.

instances	m	n	density	LB	3-FNLS	CNS	CFT
RAIL507	507	63009	1.2%	173	*174	*174	*174
RAIL516	516	47311	1.3%	182	*182	*182	*182
RAIL582	582	55515	1.2%	210	*211	*211	*211
RAIL2536	2536	1081841	0.4%	685	*690	692	691
RAIL2586	2586	920683	0.4%	936	*945	951	947
RAIL4248	4248	1092610	0.2%	1053	*1064	1070	1065
RAIL4872	4872	968672	0.2%	1509	*1528	1534	1534

* The best results obtained by these algorithms.

Table 2 shows some computational results for benchmark instances known as RAIL, which come from the crew assignment problem in Italian railway systems. In Table 2, "density" denotes the density of nonzero elements in matrix $\{a_{ij}\}$, "LB" the lower bound of the optimum value obtained by Lagrangean relaxation, "3-FNLS" the best objective values computed by our code, "CNS" the best values of (Ceria, Nobili and Sassano, 1998) and "CFT" the best values of (Caprara, Fischetti and Toth, 1999). We note that CFT is known as one of the best SCP codes. For large instances, our code 3-FNLS could get better solutions than those obtained by CFT, by spending more computatinal time (about 5 times) (Yagiura, Kishida and Ibaraki, submitted).

Acknowledgement

Most of this research, particularly the devolopment of solvers, has been done in collaboration with Mutsunori Yagiura and Koji Nonobe of Kyoto University, and graduate students in our group, which is gratefully appreciated. This work was partially supported by a Grant-in-Aid from the Ministry of Education, Culture, Sports, Science and Technology of Japan, and the 21 Century COE project "Informatics Research Center for Development of Knowledge Society Infrastructure" of Kyoto University.

References

Bent, R. and P. Van Hentenryck. (2001) A two-stage hybrid local search for the vehicle routing problem with time windows, Technical Report, CS-01-06, Dept. of Computer Science, Brown University, Providence, RI.

Berger, J., M. Barkaoui and O. Bräysy. (Forthcoming) A route-directed hybrid genetic approach for the vehicle routing problem with time windows, *INFOR*.

Bräysy, O. (Forthcoming) A reactive variable neighborhood search for the vehicle routing problem with time windows," *INFORMS Journal on Computing*.

Caprara, A., M. Fischetti, and P. Toth. (1999) A heuristic method for the set covering problem, *Operations Research*, Vol. 47, pp. 730-743.

Ceria, S., P. Nobili and A. Sassano. (1998) A Lagrangean-based heuristic for large-scale set covering problems, *Mathematical Programming*, Vol. 81, pp. 215-228.

Gehring, H. and J. Homberger. (2002) Parallelization of a two-phase metaheuristic for routing problems with time windows, *Journal of Heuristics* Vol. 8, pp. 251–276.

Homberger, J. and H. Gehring. (Forthcoming) A two-phase hybrid metaheuristic for the vehicle routing problem with time windows, *European Journal of Operational Research*, to appear.

Ibaraki, T., M. Kubo, T. Masuda, T. Uno and M. Yagiura. (Forthcoming) Effective local search algorithms for the vehicle routing problem with general time window constraints, *Transportation Science*.

Nonobe, K. and T. Ibaraki. (2001) An improved tabu search method for the weighted constraint satisfaction problem, *INFOR*, Vol. 39, pp. 131-151.

Nonobe, K. and T. Ibaraki. (2002) Formulation and tabu search algorithm for the resource constrained project scheduling problem, in *Essays and Surveys in Metaheuristics* (MIC'99), edited by C. C. Rebeiro and P. Hansen, Kluwer Academic Publishers, pp. 557-588.

Solomon, M. M. (1987) The vehicle routing and scheduling problems with time window constraints, *Operations Research*, Vol. 35, pp. 254–265.

Yagiura, M. and T. Ibaraki. (2001) *Combinatorial Optimization – Centering around Metaheuristics –*, (in Japanese), Asakura-Shoten, Tokyo.

Yagiura, M., M. Kishida and T. Ibaraki. (Submitted) A 3-flip neighborhood local search for the set covering problem.

NII-Electronic Library Service