The 7th International Symposium on Operations Research and Its Applications (ISORA'08)
Lijiang, China, October 31–Novemver 3, 2008
Copyright © 2008 ORSC & APORC, pp. 10–17

Problem Solving by General Purpose Solvers

Toshihide Ibaraki[*]

Department of Informatics
Kwansei Gakuin University
Sanda, Japan 669-1337

Abstract To solve problems abundant in real world applications, we have been proposing an approach of using general purpose solvers, since we cannot afford to develop special purpose algorithms for all individual problems. For this purpose, we have developed solvers for some standard problems such as CSP (constraint satisfaction problem), RCPSP (resource constrained project scheduling problem), GAP (generalized assignment problem) and VRP (vehicle routing problem), among others. These solvers have been successfully applied to many applications. In this talk, we report our recent experience of ITC2007 (International Timetabling Competition), in which we used our general purpose CSP solver to handle the presented benchmarks.

Keywords Combinatorial optimization; constraint satisfaction; local search; metaheuristics; timetabling; ITC2007 competition.

1 Introduction

There are many types of problems we have to solve in real world applications. We propose to take an approach of using general purpose solvers, since we cannot afford to develop special purpose algorithms for all individual problems, considering that available manpower and time are very limited. Linear programming (LP) and integer programming (IP) are powerful solvers currently used for such purposes. Software packages of LP and IP are commercially available and they have been very successfully applied to many practical problems. We consider, however, that other types of solvers are also necessary to cover wider area of applications, in particular those in the area of combinatorial optimization.

It is known that most combinatorial optimization problems of practical interest are NP-hard. Mathematically this says that, if we have an algorithm for an NP-hard problem B, then an instance of any problem A in class NP can be formulated as a problem instance of B, and thus can be solved by applying an algorithm developed for B. Roughly speaking, class NP is almost equal to the class of all combinatorial problems of practical interest. The implication of NP-hardness is therefore twofold.

(a) An algorithm for an NP-hard problem can play the role of general purpose problem solver, as it can solve all problems in class NP.

(b) Any NP-hard problem is (believed to be) computationally intractable.

[*]ibaraki@kwansei.ac.jp

Property (a) has been an impetus to developing a general purpose solver such as integer programming (IP). However, due to property (b), its success is doomed to be limited.

One of the possible avenues to overcome the computational barrier of NP-hardness is to aim at obtaining good approximate solutions instead of exact optimal solutions. In practice, exact optimal solutions are rarely required, and good suboptimal solutions are sufficient for practical purposes.

The theory of NP-hardness as described above may suggest that an approximate algorithm for any NP-hard problem B can serve as a general purpose problem solver. Unfortunately this is not the case, because the reductions to such problem B do not usually preserve the metric of the objective values. That is, assuming that an instance I_A of problem A is formulated as an instance I_B of problem B, a good approximate solution to I_B may not provide a good approximate solution if it is viewed as a solution to I_A. Therefore the formulations to problem B should be as natural as possible so that they preserve the distance of approximate solutions to the exact optimal solutions.

As a typical standard problem B, we may consider IP. The above observation says that the formulation as an IP problem may not be natural, depending on the type of combinatorial optimization problem at hand, even though the formulation itself is always possible in theory. Such inadequacy may be evidenced by a large number of integer variables and constraints required to formulate the given problem. For example, it is commonly known that some types of scheduling problems require excessively many variables and constraints. Similar observation holds if the problem contains very complicated combinatorial conditions.

This suggests that we need a number of standard problems, together with their solvers, in order to cover different types of combinatorial optimization problems. Given a list of standard problems, the user is asked to choose one to formulate his/her problem, and to apply its solver to obtain a good approximate solution efficiently.

A challenge here is to define those standard problems to be included in the list, and to develop effective solvers for them. So far we have constructed our solvers for the following standard problems:

1. constraint satisfaction problem (CSP) [4, 6],
2. resource constrained project scheduling problem (RCPSP) [5],
3. generalized assignment problem (GAP) [10, 7, 9, 8],
4. vehicle routing problem (VRP) [3, 2]

among others.

As algorithms for these solvers of standard problems, we rely on local search and metaheuristics, which are recently receiving intensive attention. In the following sections, after giving a brief introduction of local search and metaheuristics, we describe some details of our CSP solver. We then report some computational experience of our solvers applied to real world problems, putting emphasis on our recent challenge to ITC2007 (International Timetabling Competition).

2 Algorithms for Solvers

The solvers for the above standard problems must be efficient so that large scale problems 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 improving solution).

Metaheuristics: Algorithms in metaheuristics 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 solutions 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 randomly 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 probability is controlled by a parameter called *temperature* in SA, to diversify the search in the initial phase and then concentrate the search to a 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 criteria are used to reflect the computational history such as when best solutions have been improved during iterations of I-III.

We have developed so far several solvers for standard problems such as CSP, RCPSP, GAP and VRP, following the framework of metaheuristics. Some are based on TS and others are based on ILS.

3 Applications of General Purpose Solvers

Our solvers have been applied to many academic and real world problems. Applications range from solving standard benchmarks to evaluate the performance of solvers in respective categories, to handling real world problems provided from industrial users. They include production scheduling in factories of such industries as ship building and heavy industry, timetabling in high schools and universities, nurse scheduling in hospitals, and labor shift scheduling in various companies. In some cases, our solvers are included in the schedulers for some companies and are being used on the daily basis.

The solvers for CSP and RCPSP are now included in the optimization software package offered by Mathematical Systems Inc., and are applied to solve various real world problems of the customers.

Here we report our recent experience of applying our CSP solver to timetabling problems, presented as ITC2007 (International Timetabling Competition, `http://www.cs.qub.ac.uk/itc2007/`). We first describe in the next section about CSP and its solver, and then explain some details of ITC2007 together with our results.

4 CSP and Its Solver

4.1 Problem Definition

The constraint satisfaction problem CSP is formally defined by (V,D,C): V is a finite set of variables $\{X_1,X_2,\cdots,X_n\}$, $D = D\times D\times\cdots\times D$ is the corresponding domains of n variables, and C is a finite set of constraints $\{C_1,C_2,\cdots,C_r\}$, where each constraint is defined by

$$C_l(X_{l_1},X_{l_2},\cdots,X_{l_{t_l}}) \subseteq D,$$

that is, a set of all the legal t_l-tuples on variables $X_{l_1},\cdots,X_{l_{t_l}}$. A *feasible* solution of an instance of CSP is an assignment of exactly one value in D to each variable $X_i, i = 1,2,\cdots,n$, such that all constraints $C_1,C_2,\cdots,C_r$ are satisfied.

Let us introduce a *value-variable* x_{ij} for each variable-value pair:

$$x_{ij} = \begin{cases} 1, & \text{if variable } X_i \text{ takes value } j\ (\in D), \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

In general, an instance of CSP is defined by the following constraints:

$$\begin{array}{ll} \sum_{i,j} a_{hij}x_{ij} \le b_h, & h = 1,2,\cdots m', \\ \text{ad-hoc constraints } h, & h = m'+1,\cdots,m, \\ x_{ij} = 0 \text{ or } 1, & i = 1,2,\cdots,n, \quad j \in D. \end{array} \tag{2}$$

CSP differs from IP in that it allows ad hoc constraints, which may be given in any form (e.g., inequality constraints containing quadratic forms and all-different constraint stating that the values of variables in a given set V' are all different). Because of this flexibility, the formulations by CSP are usually much more compact than those by IP.

CSP permits two types of constraints; hard and soft. It seeks a solution that minimizes the total penalty of soft constraints among those satisfying all hard constraints.

4.2 Tabu Search Algorithm

Our algorithm for CSP is based on tabu search [1]. It conducts the search within the space

$$X^* = \left\{ x \;\middle|\; \sum_{j \in D} x_{ij} = 1, \quad i = 1, 2, \cdots, n \right\}. \tag{3}$$

Starting with an initial solution $x \in X^*$, we repeat modifying x so that the weighted sum of violated constraints may decrease. For this, we introduce the *penalty function* $p(x)$ defined by

$$p(x) = \sum_{h=1}^{m} w_h p_h(x), \tag{4}$$

where

$$p_h(x) = \begin{cases} \max\left(\sum_{i,j} a_{hij} x_{ij} - b_h, 0\right), & \text{if the } h\text{-th constraint is } \sum_{i,j} a_{hij} x_{ij} \le b_h, \\ \text{the amount of violation (nonnegative number),} & \\ & \text{if the } h\text{-th constraint is defined differently,} \end{cases}$$

and $w_h > 0$ is a weight given to the h-th constraint.

Function p satisfies $p(x) = 0$ if and only if x is a feasible solution of the given CSP instance. In other words, we treat CSP as a minimization problem:

$$\begin{array}{ll} \text{minimize} & p(x) \\ \text{subject to} & x \in X^*. \end{array} \tag{5}$$

The local search in our algorithm is based on the shift neighborhood defined by

$$N(x) = \left\{ x(x_{ij} \leftarrow 1) \;\middle|\; x_{ij} = 0,\ i = 1, 2, \cdots, n,\ j \in D \right\}, \tag{6}$$

where $x(x_{ij} \leftarrow 1)$ denotes the solution obtained from x by changing x_{ij} from 0 to 1 (which then incurs the change $x_{ij'} \leftarrow 0$ for the value-variable $x_{ij'}$ that currently satisfies $x_{ij'} = 1$, as a result of the constraint $x \in X^*$).

The local search is controlled by a tabu list T defined as follows. In each iteration k, we prepare a list $H^{(k)}$ that keeps the variables $X^{(l)}$ of the moves made in the past t iterations, $l = k-1, k-2, \cdots, k-t$; i.e., $H^{(k)} = \{X^{(k-1)}, X^{(k-2)}, \cdots, X^{(k-t)}\}$, and define T as the set of solutions obtained from the current solution $x^{(k)}$ by the moves that use some variables in $H^{(k)}$. Here, t is a parameter that describes how much of the past we should remember and is called the *tabu tenure*.

In addition to the standard ingredients of tabu search, our algorithm has the following features [4, 6].

(1) Diversification by long term memory.
(2) Reduction of the size of neighborhood by considering only the variables pertaining to the currently violated constraints.
(3) Automatic control mechanism of tabu tenure t.
(4) Introduction of swap neighborhood, which exchange assignments of two variables.
(5) Incorporation of objective functions to the penalty function in order to handle optimization problems.

5 ITC2007 Challenge

After presenting a brief introduction of ITC2007 and three tracks of timetabling problems, we report the result of our challenge.

5.1 Description of ITC2007

This is the second competition following the first one held in 2002, and is sponsored by PATAT (Conference on the Practice and Theory of Automated Timetabling) and WATT (Working Group on Automated Timetabling). It offers the following three tracks of timetabling problems with different features. Each track contains a set of benchmark instances, which are open to participants. Participants solve these instances on their machines using a specified CPU time, and submit their results to the organizers, based on which five finalists in each track are selected. Then the organizers collect executable codes from the finalists and test them on hidden datasets, before announcing the finalists orderings.

(1) **Examination timetabling**: We are given lists of examinations, students and rooms, as well as an enrollment table (describing the correspondence between students and examinations), sizes of examinations, capacities of rooms, and time durations required by examinations. We are also given all the scheduled time slots during the examination period, whose lengths differ, e.g., one hour, one and a half hours, two hours and so forth. We are asked to assign all examinations to time slots and rooms so that each student can take all examinations he/she enrolled, and the room of every examination can accommodate all enrolled students (hard constraints). In some cases, more than one examination may be assigned to the same room in the same slot, unless the total number of the enrolled students exceeds the capacity. There are various soft constraints (which incurs penalties if violated) concerning the adequacy of the assignment. We are asked to satisfy all hard constraints and to minimize the total penalty on soft constraints. The presented problem instances have 200$\sim$1000 examinations, 5000$\sim$16000 students, 20$\sim$80 slots and 1$\sim$50 rooms.

(2) **Post enrolment based course timetabling**: This is a typical timetabling problem encountered in universities. After student enrolment, the timetable is constructed in such a way that all students can attend the events on which they are enrolled. Similarly to the above problem, we are given lists of lectures, students and rooms, as well as an enrollment table, and sizes of lectures and rooms. In this case, all slots have the same time duration, and every day has the same number of slots. We are asked to construct a timetable for one week (typically, 5 (days) $\times$ 9 (slots) = 45 total slots), such that all hard constraints are satisfied. Hard constraints include that all students can take all the enrolled lectures, some specified subsets of lectures (e.g., those taught by the same professor) must be assigned to different slots, room capacities must be observed, assigned rooms must have necessary features required by the lectures and so forth. Soft constraints, on the other hand, states that every student wants to avoid the last slot of each day, to avoid two successive slots, three successive slots, etc., to avoid only one lecture in a day (i.e., 0 or more than 1 is preferable) and so forth. Penalties are incurred by the number of students who are given such undesirable slots. The problem instances have 300$\sim$1000 students, 200$\sim$400 lectures and 10$\sim$20 rooms.

(3) **Curriculum based course timetabling**: This problem consists of the weekly scheduling of the lectures, where conflicts between courses are set according to the curricula published by the university. The number of students in each curriculum is known. The details are similar to (2) but soft constraints are defined on the basis of curricula (e.g., the numbers of lectures should be evenly distributed between days, for each curriculum). The problem instances have 150$\sim$450 lectures, 5$\sim$20 rooms and 25$\sim$45 slots.

5.2 Our Results

We attacked all the above tracks of problems by our general purpose CSP solver. Our efforts are focused on how to formulate those problems into CSP instances, since the numbers of variables and constraints may significantly differ depending upon the adopted formulations. We tested various formulations for each track before determining the final ones. In all the formulations, we used 0-1 variables x_{ij} to represent whether lecture i is assigned to slot j or not, and y_{ik} to represent whether lecture i is assigned to room k or not. Additional variables are sometimes introduced to represent complicated constraints, while efforts are made to keep the number of such additional variables minimum.

Final results of ITC2007 can be found in the web page http://www.cs.qub.ac.uk/itc2007/winner/finalorder.htm. Our challenge ended with the third place, second place and third place for the above three tracks, respectively. We are quite happy with these, because our CSP solver could prove to be very effective in all tracks even compared with other solvers specially designed for timetabling problems. The approach of using general purpose solvers appears promising in the sense that the same solver can cover a wide spectrum of problems simply by changing formulations slightly.

6 Conclusion

We outlined our developments of general purpose solvers, all of which are based on metaheuristics. Each of these solvers can handle various problems within its scope, and is reasonably efficient as evidenced by our challenge to ITC2007. Putting these solvers together, therefore, we may expect that quite a wide range of problems of practical importance can now be accessed, and useful solutions can be obtained.

Acknowledgments: All of the metaheuristic codes for the standard problems mentioned in Section 1 have been developed with the cooperation of M. Yagiura of Nagoya University, K. Nonobe of Hosei University, and other colleagues and students in our group at Kyoto University. The challenge to ITC2007 was made jointly with M. Atsuta of Kwansei Gakuin University (currently with NS Solutions Co.) and K. Nonobe.

References

[1] GLOVER, F., Tabu search - Part I, *ORSA Journal on Computing* (1989) **1**, 1190-206.

[2] HASHIMOTO, H., IBARAKI, T., IMAHORI S., AND YAGIURA, M., The vehicle routing problem with flexible time windows and traveling times, *Discrete Applied Mathematics* (2006) **154**, 2271-2290.

[3] IBARAKI, T., IMAHORI, S., KUBO, M., MASUDA, T., UNO, T., AND YAGIURA, M., Effective local search algorithms for routing and scheduling problems with general time window constraints, *Transportation Science* (2005) **39**, 206-232.

[4] NONOBE, K. AND IBARAKI, T., A tabu search approach to the constraint satisfaction problem as a general problem solver, *European J. Operational Research* (1998) **106**, 599–623.

[5] NONOBE, K. AND IBARAKI, T., Formulation and tabu search algorithm for the resource constrained project scheduling problem, *Essays and Surveys in Metaheuristics* (MIC'99), edited by Ribeiro, C. C., and Hansen, P., Kluwer Academic Publishers, Boston, 557-588, 2002.

[6] NONOBE, K. AND IBARAKI, T., An improved tabu search method for the weighted constraint satisfaction problem, *INFOR* (2001) **39**, 131-151.

[7] YAGIURA, M., IBARAKI, T. AND GLOVER, F., An ejection chain approach for the generalized assignment problem, *INFORMS Journal on Computing* (2004) **16**, 133-151.

[8] YAGIURA, M., IBARAKI, T. AND GLOVER, F., A path relinking approach for the generalized assignment problem, *European J. of Operational Research* (2005) **169**, 548-569.

[9] YAGIURA, M., IWASAKI, S., IBARAKI, T., AND GLOVER, F., A very large-scale neighborhood search algorithm for the multi-resource generalized assignment problem, *Discrete Optimization* (2004) **1**, 87-98.

[10] YAGIURA, M., YAMAGUCHI, T. AND IBARAKI, T., A variable depth search algorithm with branching search for the generalized assignment problem, *Optimization Methods and Software* (1998) **10**, 419–441.