数理解析研究所講究録
981巻 1997年 11-21

Tabu Search Approach to CSP (Constraint Satisfaction Problem) as a General Problem Solver

CSP(制約充足問題) による組合せ問題に対する汎用アルゴリズム

Koji Nonobe (野々部 宏司)
Department of Applied Mathematics and Physics
Graduate School of Engineering Kyoto University Kyoto, Japan 606
nonobe@kuamp.kyoto-u.ac.jp

Toshihide Ibaraki (茨木 俊秀)
Department of Applied Mathematics and Physics
Graduate School of Engineering Kyoto University Kyoto, Japan 606
ibaraki@kuamp.kyoto-u.ac.jp

Abstract

As it is known that various combinatorial problems can be naturally formulated into CSP (constraint satisfaction problem), we develop in this paper a tabu search based algorithm for CSP in the hope of using it as a general problem solver. With our code, we solved various problems selected from wide applications, e.g., graph coloring, block design, generalized assignment, set covering, timetabling and nurse scheduling. The obtained results appear to be competitive with those of existing algorithms specially developed for the respective problem domains. Among these results, we report in this paper some computational results of graph coloring, timetabling, and nurse scheduling.

1 Introduction

The *constraint satisfaction problem* (CSP) asks to find an assignment to its variables such that all given constraints are satisfied (i.e., feasible). It is known that CSP is able to formalize a large variety of combinatorial problems of practical importance, such as SAT (satisfiability), graph coloring, graph partitioning, set covering, scheduling, generalized assignment, timetabling and many others. Therefore, an algorithm for the CSP can solve all such problems, and hence can be regarded as a *general problem solver.*

CSP has been studied mainly in the field of artificial intelligence [4, 5, 6, 12, 15]. There are two main approaches to solving CSP; "constructive" backtracking approaches and "repair" (i.e., iterative improvement or hill climbing) approaches. In the early literature, most of the efforts were directed to backtracking methods, in which we start from a partial assignment which satisfies all constraints, and enlarge the assignment step by step until all variables are assigned feasible values. Various techniques of problem reduction and constraint propagation have been developed to enhance backtracking procedure [4, 5, 6, 15]. In contrast, in repair type approaches, an initial solution which is an assignment to

all variables (but is not feasible) is gradually repaired in order to reduce the infeasibility until all constraints are satisfied. Basic tools used for this procedure is local search and its variants. Recently, this approach has been found to be very effective, particularly for large scale problems [10, 12, 14, 15].

It should be noted at this point that a repair type approach may fail to find a feasible solution even if one exists, and cannot tell the fact if there exists no feasible solution. However, since CSP is NP-complete (implying its intractability if exact solutions are needed), approximate or heuristic methods which can find feasible solutions (or those satisfying most of the constraints) in reasonable computation time are of great practical value. For such purposes, the repair type approach is better suited because assignments to all variables, which satisfy most of the constraints, are usually available even if the computation is cut off before normal termination.

This paper also pursues the same direction, and use the tabu search as a basic wheel to carry out the repairing process. Tabu search is a metaheuristic method that has been proved successful in solving many practical optimization problems [7, 8]. It consists of such building blocks as neighborhood, short term memory (tabu list), long term memory, aspiration criteria, and criteria for diversification and/or strategic oscillation.

To evaluate the performance of our approach, we tested various combinatorial problems such as graph coloring, block designs, generalized assignment, set covering, timetabling, and nurse scheduling. The tabu search based CSP algorithm developed in this paper turned out to be reasonably effective to all these problems, even though it is still a preliminary version, indicating that the combination of CSP and tabu search is a right choice as a general problem solver of this kind. In this paper, we report some computational results of graph coloring, timetabling, and nurse scheduling.

2 Formulation of CSP

The CSP is formally defined by $(V, \boldsymbol{D}, \boldsymbol{C})$ [4, 12, 15]: V is a finite set of variables $\{X_1, X_2, \cdots, X_n\}$, $\boldsymbol{D}$ is a set of the corresponding domains $\{D_1, D_2, \cdots, D_n\}$, where D_i is a finite domain of variable X_i, and $\boldsymbol{C}$ is a finite set of constraints $\{C_1, C_2, \cdots, C_r\}$, where each constraint is defined by

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

that is, a set of all the legal t_l-tuples on variables $X_{l_1}, \cdots, X_{l_{t_l}}$. Throughout this paper, we assume for simplicity that

$$D_1 = \cdots = D_n = D,$$

and that $|D| = d$. 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. In this paper, we are mainly interested in finding one feasible solution (or a solution satisfying most of the constraints), even if there may be many feasible solutions.

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

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

There are $\sum_{i=1}^{n} |D_i| = dn$ value-variables. With these value-variables, an assignment to all variables X_i can be represented by a dn dimensional 0-1 vector $\boldsymbol{x} = (x_{ij} \mid i = 1, 2, \cdots, n, \ j \in D)$. For example,

if $V = \{X_1, X_2, X_3, X_4\}$ and $D = \{1, 2, 3\}$, an assignment $(X_1 \leftarrow 2, X_2 \leftarrow 1, X_3 \leftarrow 1, X_4 \leftarrow 3)$ is represented by a 0-1 vector $\boldsymbol{x} = (010|100|100|001)$. In this formulation, we always assume the constraints,

$$\sum_{j \in D} x_{ij} = 1, \quad i = 1, 2, \cdots, n, \tag{2}$$

so that all variables X_i are respectively assigned exactly one value. We shall always consider constraints (2) implicitly in our tabu search algorithm (i.e, the search is restricted only to these solutions satisfying (2)).

At this point, we note that each constraint C_l can be described in different ways. In this paper, we take a view that the basic tool to represent a constraint C_l is a set of linear inequalities, since any constraint C_l can always be represented in this way, and we know from the experience in operations research that such formulations are usually very efficient to represent various constraints arising in practical problems. For example, the constraint that at most (resp., at least) K variables X_i in set $V'(\subseteq V)$ must take a value j is given by

$$\sum_{X_i \in V'} x_{ij} \leq K \quad \left(\text{resp.}, \sum_{X_i \in V'} -x_{ij} \leq -K \right), \qquad j \in D,$$

the constraint that any two variables X_{i_1}, X_{i_2} in set $V''(\subseteq V)$ must take different values is given by

$$\sum_{X_i \in V''} x_{ij} \leq 1, \qquad j \in D,$$

and the constraint that the total cost must be less than c_0, when the cost coefficient of x_{ij} is c_{ij}, is given by

$$\sum_{i,j} c_{ij} x_{ij} \leq c_0.$$

In this paper, we assume that an instance of CSP is described by the following inequalities of 0-1 variables:

$$\begin{aligned} &\sum_{i,j} a_{hij} x_{ij} \leq b_h, \qquad h = 1, 2, \cdots m, \\ &x_{ij} = 0 \text{ or } 1, \qquad i = 1, 2, \cdots, n, \quad j \in D, \end{aligned} \tag{3}$$

in addition to constraint (2) of the value-variables. Our goal is to find a feasible 0-1 vector $\boldsymbol{x} \in \{0, 1\}^{dn}$ that satisfies all these constraints.

3 Basic implementation of tabu search

During the entire search process, we assume that constraint (2) is always satisfied; i.e., we consider only the following search space X^* of value-variables:

$$X^* = \left\{ \boldsymbol{x} \;\middle|\; \sum_{j \in D} x_{ij} = 1, \quad i = 1, 2, \cdots, n \right\}. \tag{4}$$

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

$$p(\boldsymbol{x}) = \sum_{h=1}^{m} w_h p_h(\boldsymbol{x}), \tag{5}$$

where

$$p_h(\boldsymbol{x}) = \max\left(\sum_{i,j} a_{hij} x_{ij} - b_h, 0\right),$$

and $w_h > 0$ is a weight given to constraint $\sum_{i,j} a_{hij} x_{ij} \leq b_h$.

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

$$\begin{aligned} &\text{minimize} \quad p(\boldsymbol{x}) \\ &\text{subject to} \quad \boldsymbol{x} \in X^*. \end{aligned} \tag{6}$$

Determination of appropriate weights w_h of inequalities is an important issue, because the performance of the search highly depends on them. Furthermore, when the optimal value is not 0 (i.e. the CSP has no feasible solution), we may want to find an acceptable solution that violates only a small number of less important inequalities. In this case, the resulting solution is obviously influenced by such weights, and we may have to tune them carefully so that satisfactory results may be reached.

We next describe the neighborhood $N(\boldsymbol{x})$ used in our implementation of tabu search. Let $\boldsymbol{x}(x_{ij} \leftarrow 1)$ denote the solution obtained from $\boldsymbol{x}$ 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 $\boldsymbol{x} \in X^*$). $N(\boldsymbol{x})$ denotes the set of solutions obtained by such modifications, i.e.

$$N(\boldsymbol{x}) = \left\{ \boldsymbol{x}(x_{ij} \leftarrow 1) \;\middle|\; x_{ij} = 0,\ i = 1, 2, \cdots, n,\ j \in D \right\}, \tag{7}$$

where $|N(\boldsymbol{x})| = (d-1)n$ holds. In each iteration k of tabu search, the current solution $\boldsymbol{x}^{(k)}$ is replaced by $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)}(x_{i^*j^*} \leftarrow 1)$, which is the best solution in $N(\boldsymbol{x}^{(k)}) \setminus T$ in the sense that

$$p\left(\boldsymbol{x}^{(k)}(x_{i^*j^*} \leftarrow 1)\right) \leq p\left(\boldsymbol{x}^{(k)}(x_{ij} \leftarrow 1)\right) \quad \text{for all } \boldsymbol{x}^{(k)}(x_{ij} \leftarrow 1) \in N(\boldsymbol{x}^{(k)}) \setminus T. \tag{8}$$

In this case, we say that $\boldsymbol{x}^{(k+1)}$ is obtained from $\boldsymbol{x}^{(k)}$ by a *move* $x_{i^*j^*} \leftarrow 1$.

The above set T in (8) is called a *tabu list* (or short term memory). Let $t = |T|$, which is a program parameter called *tabu tenure*. Tabu list T stores t solutions obtainable from the current solution $\boldsymbol{x}^{(k)}$ by reversing the moves chosen in the most recent t iterations. The idea of T is to suppress a move that brings the current solution $x^{(k)}$ back to a solution that was recently tested, whereby preventing a cycling over a small set of solutions. Although not explicitly stated in the above, we incorporate an additional rule in our implementation that if $x_{ij} \leftarrow 1$ is the move used in the k-th iteration, no move $x_{ij'} \leftarrow 1$ is allowed in the next iteration for the same i.

As a standard equipment of tabu search, we also maintain an array LTM of long term memory, where $LTM(x_{ij})$ records the number of times the move $x_{ij} \leftarrow 1$ has been chosen. To include the effect of LTM in the choice of moves, we modify the selection rule (8) as follows: Choose $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)}(x_{i^*j^*} \leftarrow 1) \in N(\boldsymbol{x}^{(k)}) \setminus T$ such that

$$p\left(\boldsymbol{x}^{(k)}(x_{i^*j^*} \leftarrow 1)\right) + \alpha LTM(x_{i^*j^*}) \leq p\left(\boldsymbol{x}^{(k)}(x_{ij} \leftarrow 1)\right) + \alpha LTM(x_{ij}), \quad \text{for all } \boldsymbol{x}^{(k)}(x_{ij} \leftarrow 1) \in N(\boldsymbol{x}^{(k)}) \setminus T, \tag{9}$$

where $\alpha \geq 0$ is also a program parameter. The incorporation of LTM makes the selection of $x_{ij} \leftarrow 1$ difficult if the move $x_{ij} \leftarrow 1$ has been used frequently in the past iterations.

The *aspiration criteria* is also incorporated in our code; i.e., if a tested solution $\boldsymbol{x}^{(k)}(x_{ij} \leftarrow 1)$ is better than the currently best solution found so far (in the sense of having a smaller penalty), it is selected as $\boldsymbol{x}^{(k+1)}$ even if it belongs to T.

Before proceeding further, we briefly mention some implementation issues of the above algorithm. First, since the $m \times dn$ coefficient matrix $A = (a_{hij})$ of (3) is usually sparse for large-scale applications, we keep only nonzero elements in the form of linked lists, in which pointers are constructed so that the nonzero coefficients can be retrieved efficiently both column-wise and row-wise. Using this data structure, the difference

$$\Delta p(\boldsymbol{x}^{(k)}(x_{ij} \leftarrow 1)) = p(\boldsymbol{x}^{(k)}(x_{ij} \leftarrow 1)) - p(\boldsymbol{x}^{(k)}) \tag{10}$$

can be computed only by looking at relevant nonzero coefficients, and, from this information, the move $x_{i^*j^*} \leftarrow 1$ of (9) can be efficiently determined.

To make the references to T efficient, instead of keeping t solutions in T explicitly, we implement T in the form of an array such that component $T(x_{ij})$ keeps the iteration number k at which the move $x_{ij} \leftarrow 0$ has occurred most recently. In this way, the condition whether $\boldsymbol{x}(x_{ij} \leftarrow 1) \in T$ or not can be checked in constant time.

All the algorithms in this paper are coded in C, and run on Sun SPARC classic 4/15 (59.1MIPS, 4.6MFLOPS).

Graph coloring problem. We first apply the above tabu search algorithm to the *graph coloring problem* (GCP). GCP is the problem to assign a color to each vertex of a given undirected graph $G(V, E)$, so that every pair of adjacent vertices gets different colors. To handle GCP in the framework of CSP, we consider GCP as a decision problem that asks whether k colors are sufficient or not. The GCP can be formulated as CSP by introducing variables X_i corresponding to vertices $i = 1, 2, \cdots, n$, domain $D = \{1, 2, \cdots, k\}$ representing possible colors, and value-variables x_{ij} such that

$$x_{ij} = \begin{cases} 1, & \text{if vertex } i \text{ gets color } j, \\ 0, & \text{otherwise.} \end{cases}$$

Then constraint $C_l(X_{i_1}, X_{i_2})$ is prepared for each edge $(i_1, i_2) \in E$ by

$$x_{i_1 j} + x_{i_2 j} \leq 1, \qquad j \in D.$$

Therefore, in this formulation, there are $k|V| = kn$ value-variables and $m = k|E|$ inequalities (in addition to those in X^* of (4)).

To see the performance of our tabu search algorithm, some of the DIMACS benchmarks are tested by solving each problem instance ten times. In each run, we set the maximum computational time as 600 seconds. If a feasible solution is found before 600 seconds, it is considered a *success*; otherwise a *failure*. The computational results are shown in Table 1, where k is set to the number of colors obtained by the approximate graph coloring algorithm developed by Goto [9], and t is a tabu tenure $|T|$. For all

instances, a program parameter α is set to 0.01. This approximate algorithm performs very well and usually gives better k than other well known heuristic coloring algorithms such as DSATUR [1] and COSINE [11], though it usually consumes more computation time (it is an $O(n^3)$ time algorithm). The column "success rate" denotes the number of success runs (out of 10), and column "CPU time" denotes the average CPU time (in seconds) of one run. It is observed that our CSP algorithm is about 1.5 - 9 times slower than the above heuristic graph coloring algorithm [9]. However, in view of that our algorithm is general purpose and use many value-variables for its formulation (the program of Goto [9] uses only n variables), the results may be considered encouraging because our algorithm could find solutions of the same quality with high probability (except for a few instances).

Table 1: Computational results for DIMACS benchmarks of the graph coloring problem (GCP).

instances	$\|V\|$	$\|E\|$	optimum[1]	k	t	success rate	CPU time
mulsol.i.1.col	197	3925	—	49	20	10/10	35.0
mulsol.i.2.col	188	3885	—	31	20	10/10	18.4
mulsol.i.3.col	184	3916	—	31	20	10/10	18.4
mulsol.i.4.col	185	3946	—	31	20	10/10	18.6
mulsol.i.5.col	186	3973	—	31	20	10/10	18.8
zeroin.i.1.col	211	4100	—	49	20	10/10	36.6
zeroin.i.2.col	211	3541	—	30	20	10/10	16.8
zeroin.i.3.col	206	3540	—	30	20	10/10	16.6
le450_5a.col	450	5714	5	5	30	9/10	277.8
le450_5b.col	450	5734	5	5	30	4/10	415.2
le450_5c.col	450	9803	5	5	30	10/10	73.3
le450_5d.col	450	9757	5	5	30	10/10	102.2
le450_15a.col	450	8168	15	17	30	10/10	161.3
le450_15b.col	450	8169	15	17	30	10/10	161.7
le450_15c.col	450	16680	15	19	30	0/10	600
le450_15d.col	450	16750	15	19	30	0/10	600
le450_25a.col	450	8260	25	25	30	10/10	81.7
le450_25b.col	450	8263	25	25	30	10/10	70.2
le450_25c.col	450	17343	25	28	30	1/10	583.8
le450_25d.col	450	17425	25	29	30	6/10	393.6

1. The minimum number of colors needed; '—' means that the optimal value is not known.

4 Problems from real applications

As emphasized in the introduction, our CSP code is designed as a general purpose problem solver. From this view point, it is important to test it against a wide variety of problems encountered in

practical applications. We are currently collecting problem data from real world applications. Here, we report the results of two case studies on timetabling at a high school and nurse scheduling in a hospital.

Timetabling. The data is taken from a real high school located in the suburb of Tokyo. This high school has 30 classes, 60 teachers, 30 classrooms and 3 rooms designed for some special subjects (such as painting and cooking). There are a total of 780 lectures, each of which is defined by its subject, class and teacher, and 13 meetings, each of which is defined by a set of teachers to attend. All of these lectures and meetings must be assigned to 34 periods in a week (6 periods a day from Monday through Friday, and 4 periods in Saturday morning) under various constraints to be described shortly. To formulate this problem as CSP, we introduce $n = 780 + 13 = 793$ variables X_i and domain $D = \{1, 2, \cdots, 34\}$ corresponding to the periods in a week; i.e., there are $793 \times 34 = 26962$ value-variables x_{ij}.

There are a large number of constraints to be satisfied. Some of them are common to all timetabling problems, but others reflect special requirements from subjects, teachers and classes. We list here part of such constraints:

- Each lecture or meeting is assigned to exactly one period.
- Each class receives at most one lecture in each period.
- Each teacher has at most one lecture or meeting in each period.
- Each special room receives at most one lecture of the corresponding subject in each period.
- For each class, each teacher and each special room, there may be some periods which have already been scheduled, or which must not be assigned for some reason. (In other words, the corresponding value-variables are fixed beforehand.)
- For each teacher, the number of lectures and meetings in a day must not exceed a given upper bound (which is typically 4).
- Some lectures must be assigned to successive periods on the same day.
- Some lectures of similar subjects must not be scheduled on the same day.

These constraints are described by 12747 linear inequalities.

We ran our tabu search algorithm with 50 minutes upper bound on computational time, using $t = 15$ and $\alpha = 0.01$, and could find a solution that satisfies all constraints except one. By analyzing the violated constraint and the related ones, however, we could show that it is not possible to satisfy all constraints. Hence the solution found by our tabu search turned out to be best possible.

There are a number of papers on timetabling such as [2, 3, 16]. Among these, [3] contains computational results on large scale instances with similar side constraints, which are also taken from an existing high school. Although it is difficult to make precise comparison, our CSP algorithm appears to be as good as this special purpose algorithm in [3].

Nurse scheduling. The nurse scheduling problem is also discussed in various papers (e.g., [13]). In this problem, each nurse must be assigned every day one of the five alternatives: day shift, evening

shift, night shift, scheduled meeting and day off. Our example, which is based on the real data in a hospital in Tokyo, has 25 nurses. To make a schedule of these nurses on 30 days (one month), CSP formulation has $25 \times 30 = 750$ variables X_i and domain $D = \{\text{day shift}, \ldots, \text{day off}\}$ with $d = 5$; hence $750 \times 5 = 3750$ value-variables x_{ij}. This problem also has many constraints, some of which are quite complicated. We list here some representative ones.

- The nurses are divided into two teams A and B, where team A has 13 nurses and team B has 12 nurses. Among 25 nurses, 6 in A and 5 in B are regarded as leaders, respectively.
- In each shift of each day, the predetermined numbers of nurses and leaders (which vary depending on the days) must be selected evenly from the two teams (more precisely at least $\min\{4, \lfloor k/2 \rfloor\}$ members must be selected from A and B respectively, if the total size is k). Typically, a day shift consists of 10 nurses, among which 4 nurses are leaders, an evening shift requires 4 nurses including 2 leaders, and a night shift requires 3 nurses including 2 leaders.
- Each nurse may have some days which have already been scheduled (e.g., scheduled meetings and days off).
- The numbers of day shifts, evening shifts, night shifts, scheduled meetings and days off assigned to each nurse during 30 days must be within their lower and upper bounds, respectively.
- Each nurse must get day shift and day off at least once a week, respectively. The pattern of (day off, duty, day off) should be avoided.
- The following severe patterns must be avoided: (i) 3 successive night shifts, (ii) 4 successive evening shifts, (iii) 5 successive day shifts, (iv) day shift, evening shift or meeting after a night shift, (v) day shift or meeting after an evening shift.
- An isolated night shift must be avoided (i.e., we need two successive night shifts). After a series of night shifts is over, the next night shift should be avoided for at least 6 days.

The resulting formulation has 9731 inequalities to describe the above and other constraints. It turns out, however, that there are some conflicting constraints which make a solution satisfying all constraints impossible to exist. Therefore, we are asked to achieve certain compromise among constraints, by taking into account the fact that some constraints are not very rigid and can be ignored if there is other compensation. In our computation, we set our target to minimize the number of day shifts which do not have the required number of leaders, under the condition that all the important constraints are satisfied (but some of the constraints of less importance may be violated). To achieve this, we introduced weights w_h of (5) to all inequalities and manually controlled them so that the target can be achieved smoothly.

We ran our CSP algorithm several times, consuming 30 minutes each time, and could find a number of acceptable solutions, where we used $t = 35$ and $\alpha = 0.001$. An example is illustrated in Figure 1. In this schedule, the numbers of leaders on day shifts are not sufficient for 6 days, the numbers of nurses on day shifts are not sufficient for 2 days, and there are the following additional violations:

- Nurse 1 does not have the required interval of 6 days between the night shift on day 19 and the night shift on day 25.

- Nurse 2 has an isolated night shift on day 8.
- Nurse 2 has a pattern of three successive night shifts from day 20 to day 22.
- Nurse 17 has a day shift after the night shift on day 10.

5 Conclusion and discussion

We developed in this paper a tabu search algorithm for CSP in order to use it as a general problem solver. Judging from the computational results for the problems chosen from a wide variety of applications, our CSP algorithm appears to be competitive with special purpose algorithms developed in the respective domains.

However, the algorithm contains some program parameters, such as tabu tenure t, weights α of the long term memory, and weights w_h to constraints, all of which must be adjusted beforehand (usually by preliminary experiment) to attain its maximum efficiency. To make our algorithm more easily usable, it would be important to establish a guide line for adjusting such parameters, or preferably, to develop some automatic mechanism for that purpose.

As our project is still on-going, we plan to increase its power and applicability by further elaboration. Toward this goal, the following points may deserve consideration:

1. More sophisticated elements of tabu search, such as strategic oscillation, path relinking and target analysis, may be worthwhile to include.
2. Incorporation of neighborhoods other than $N(\boldsymbol{x})$ may also be useful. Flexible choices of neighborhood structures will enlarge the applicability of our algorithm, but at the same time necessitate establishment of a rule of choosing an appropriate neighborhood from many possibilities.
3. Most of the problems tested in this paper are combinatorial in the sense that all the coefficients a_{hij} in constraints (3) are 0 or ± 1. However, our algorithm may not be very effective for the problems with general a_{hij}. To overcome this, it appears necessary to exploit mathematical programming techniques (e.g., linear programming) in order to extract global structural information of the entire feasible region and to use it in the search process.
4. To make the system easily accessible from many users, it should be equipped with a friendly interface so that the user can easily input his/her problem in the CSP form and execute the CSP algorithm effectively. Development of such an interface is an important future project.

Acknowledgments

We appreciate M. Watanabe and K. Kaneko of C & C Research Laboratories, NEC Corporation for the data of timetabling in Section 4, and A. Ikegami of Seikei University for the data of nurse scheduling also in Section 4. This research was partially supported by the Scientific Grant in Aid by the Ministry of Education, Science and Culture of Japan.

		days																												
	nurses	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
	1*	/	−	−	=	=	/	−	=	≡	≡	/	−	−	−	−	/	−	≡	≡	/	/	−	=	/	≡	≡	/	−	=	/
	2*	≡	/	/	−	−	=	=	≡	/	/	−	−	+	/	/	−	=	/	−	≡	≡	≡	/	−	−	=	=	/	−	−
	3*	−	/	−	≡	≡	/	/	−	−	−	−	=	=	/	≡	≡	/	−	=	=	=	/	−	−	=	/	−	−	/	/
	4*	/	=	=	/	−	−	/	−	−	=	/	/	≡	≡	/	−	−	=	/	+	−	−	−	=	/	/	−	=	≡	≡
	5*	−	−	−	−	/	≡	≡	/	−	−	=	/	−	=	=	=	≡	≡	/	−	−	/	/	/	/	−	≡	≡	/	=
	6*	=	≡	≡	/	/	−	−	/	=	=	≡	≡	/	−	−	/	/	−	−	−	/	=	≡	≡	/	−	−	/	−	−
A	7	/	/	−	−	/	−	−	−	=	/	−	−	=	/	−	−	/	−	−	≡	≡	/	−	−	−	−	=	=	/	/
	8	=	=	/	−	−	−	=	/	−	+	/	−	−	≡	≡	/	−	=	=	/	−	−	−	=	/	/	/	−	−	/
	9	/	−	=	/	/	−	≡	≡	/	−	−	/	/	−	=	=	=	/	−	−	−	=	/	−	−	−	/	−	−	−
	10	+	/	/	−	−	=	/	−	≡	≡	/	/	−	=	/	+	−	−	−	=	/	−	=	/	−	−	−	/	≡	≡
	11	−	−	/	≡	≡	/	−	=	/	/	−	−	/	−	−	+	/	−	−	−	/	−	−	−	=	=	/	−	=	/
	12	/	−	−	−	−	/	/	/	/	−	=	=	/	−	−	+	−	≡	≡	/	−	−	/	−	−	−	−	/	−	−
	13	−	−	−	=	=	/	+	−	/	/	/	≡	≡	/	−	−	−	−	/	−	=	≡	≡	/	/	/	−	−	−	=
	14*	−	−	=	/	−	=	/	−	−	+	/	/	−	≡	≡	/	/	−	=	=	=	/	−	≡	≡	/	/	−	−	−
	15*	−	≡	≡	/	/	/	/	−	−	−	≡	≡	/	/	−	=	/	−	−	−	=	=	/	−	≡	≡	/	=	=
	16*	/	/	−	−	=	≡	≡	/	−	=	=	/	+	−	/	−	−	−	/	/	+	≡	≡	/	−	−	=	/	−	−
	17*	≡	/	/	−	−	−	/	=	=	≡	≡	−	+	/	−	=	/	≡	≡	/	/	−	−	−	/	−	−	=	/	≡
	18*	/	−	−	=	/	−	=	≡	≡	/	/	−	=	=	/	≡	≡	/	−	+	/	−	−	−	=	=	/	≡	≡	/
B	19	=	/	−	−	/	−	+	=	=	≡	≡	/	−	−	−	+	/	−	−	/	−	−	/	/	/	/	−	=	/	−
	20	−	=	=	/	/	/	−	−	/	−	−	=	=	/	−	−	−	/	/	−	−	/	−	=	=	≡	≡	/	−	+
	21	/	−	−	=	=	≡	≡	/	−	=	/	−	−	/	/	−	=	/	−	=	/	−	−	≡	≡	/	−	−	/	+
	22	+	≡	≡	/	−	=	=	/	/	/	−	−	/	−	−	/	−	=	/	−	=	/	−	−	−	=	=	/	−	+
	23	−	−	/	/	−	−	−	−	/	/	−	−	/	=	≡	≡	/	−	=	/	−	=	=	/	−	−	≡	≡	/	+
	24	≡	/	/	−	−	−	−	/	−	−	=	/	/	−	=	=	≡	≡	/	−	=	/	−	−	/	/	/	−	=	=
	25	=	=	/	≡	≡	/	−	−	/	−	−	=	/	−	=	/	−	=	/	≡	≡	/	−	=	/	−	−	−	/	−
	−	8	10	10	10	10	10	8	10	9	8	10	10	7	10	10	8	10	10	10	9	9	10	13	10	9	10	10	10	10	8
	=	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	5	4	4	4	4	4	4	4	4	4
total	≡	3	3	3	3	3	3	4	3	3	4	3	3	3	3	4	3	3	5	3	3	3	3	3	3	3	3	4	3	3	3
	+	2	0	0	0	0	0	2	0	0	2	0	0	3	0	0	4	0	0	0	2	1	0	0	0	0	0	0	0	0	4
	/	8	8	8	8	8	8	7	8	9	7	8	8	8	8	7	6	8	6	8	7	7	8	5	8	9	8	7	8	8	6

−: day shift
=: evening shift
≡: night shift
+: meeting
/: day off

*: leader

Figure 1: A schedule of 25 nurses on 30 days.

References

[1] Brélaz, D., "New methods to color the vertices of a graph", *Communications of the ACM* 22 (1979) 251-256.

[2] Carter, M.W., Laporte, G., and Lee, S.Y., "Examination timetabling: algorithmic strategies and applications", *Journal of the Operational Research Society* 47 (1996) 373-383.

[3] Costa, D., "A tabu search algorithm for computing an operational timetable", *European Journal of Operational Research* 76 (1994) 98-110.

[4] Decher, R., and Pearl, J., "Network-based heuristics for constraint-satisfaction problems", *Artificial Intelligence* 34 (1988) 1-38.

[5] Freuder, E.C., "A sufficient condition for backtrack-free search", *Journal of the Association for computing Machinery* 29 (1982) 24-32.

[6] Freuder, E.C., "Complexity of K-tree structured constraint satisfaction problems", *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI)* (1990) 4-9.

[7] Glover, F., "Tabu search - Part I", *ORSA Journal on Computing* 1 (1989) 190-206.

[8] Glover, F., "Tabu search fundamentals and uses", Technical Report, University of Colorado (1995).

[9] Goto, M., "An approximate algorithm for the graph coloring problem", Graduation thesis, Department of Applied Mathematics and Physics, Kyoto University (1993).

[10] Gu, J., "Local search for satisfiability (SAT) problem", *IEEE Transactions on Systems, Man, and Cybernetics* 23 (1993) 1108-1129.

[11] Hertz, A., "COSINE: A new graph coloring algorithm", *Operations Research Letters* 10 (1991) 411-415.

[12] Minton, S., Johnston, M.D., Philips, A.B., and Laird, P., "Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems", *Artificial Intelligence* 58 (1992) 166-205.

[13] Randhawa, S.U., and Sitompul, D, "A heuristic-based computerized nurse scheduling system", *Computers and Operations Research* 20 (1993) 837-844.

[14] Selman, B., Levesque, H., and Mitchell, D., "A new method for solving hard satisfiability problems", *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI)* (1992) 440-446.

[15] Tsang, E., *Foundations of constraint satisfaction*, Academic Press, London, 1993.

[16] Wright, M., "School timetabling using heuristic search", *Journal of the Operational Research Society* 47 (1996) 347-357.