ELSEVIER

European Journal of Operational Research 106 (1998) 599–623

EUROPEAN
JOURNAL
OF OPERATIONAL
RESEARCH

A tabu search approach to the constraint satisfaction problem as a general problem solver

Koji Nonobe [1], Toshihide Ibaraki *

Department of Applied Mathematics and Physics, Graduate School of Engineering, Kyoto University, Kyoto, Japan 606

Received 1 April 1996; received in revised form 1 March 1997

Abstract

Many combinatorial problems, including a variety of combinatorial optimization problems, can be naturally formulated as a constraint satisfaction problem (CSP). We develop in this paper a tabu search-based algorithm for the CSP as a foundation for a general problem solver. In addition to the basic components of tabu search, we develop a number of elaborations, such as an automatic control mechanism for the tabu tenure, modification of the penalty function to handle objective functions, and enlargement of the neighborhood by allowing swap operations. Computational results with our algorithm are reported for various problems selected from a wide range of applications, i.e., graph coloring, generalized assignment, set covering, timetabling and nurse scheduling. Our results appear to be competitive with those of existing algorithms specially developed for the respective problem domains. © 1998 Elsevier Science B.V. All rights reserved.

Keywords: Combinatorial problems; Constraint satisfaction problem (CSP); Meta-heuristics; Tabu search; General problem solver

1. Introduction

The *constraint satisfaction problem* (CSP) seeks an assignment of values to its variables that will satisfy all given constraints (i.e., that will yield a feasible solution). It is known that the CSP is able to formalize a large variety of combinatorial problems of practical importance, such as satisfiability (SAT), 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*. Such a general purpose algorithm would be practically useful and meaningful, if it is easy to use, and if it can find acceptable solutions in reasonable computational time for various problems in real applications.

The CSP has been studied mainly in the field of artificial intelligence [9,12,13,24,30]. There are two main approaches to solving the CSP;

* Corresponding author. E-mail: ibaraki@kuamp.kyoto-u-ac.jp.

[1] E-mail: nonobe@kuamp.kyoto-u.ac.jp.

0377-2217/98/$19.00 © 1998 Elsevier Science B.V. All rights reserved.
PII S0377-2217(97)00294-4

"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, which, starting from a partial assignment which satisfies all constraints, enlarge the assignment step by step until all variables are assigned feasible values. This type of approach is usually designed to solve given problem instances exactly. Although various techniques of problem reduction and constraint propagation have been developed to enhance the backtracking procedure [9,12,13,30], the problem sizes that can be handled by this approach are not large enough to accommodate practical problems in many cases. By 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 incorporate local search and its variants. Recently, this approach has been found to be very effective, particularly for large scale problems [17,24,29,30].

If a feasible solution is not known in advance, no method is known that is guaranteed to find a feasible solution even if one exists, or that can determine whether a feasible solution exists to be found. However, since the CSP is NP-complete (implying its intractability if exact solutions are needed), approximate or heuristic methods which can often 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 pursues the repair type approach, and uses tabu search as a basic mechanism to carry out the repairing process. Tabu search is a metaheuristic method that has proved successful in solving many practical optimization problems [15,16]. It consists of such building blocks as neighborhoods, short term memory (embodied in tabu lists), long term memory and aspiration criteria.

General purpose algorithms like ours may not have the highest performance if restricted to a given particular problem, because such performance would be attainable only by special-purpose algorithms individually designed. However, this does not deny the usefulness of general purpose algorithms, because researchers and other problem solvers (particularly those in application fields) may not be able to invest enough manpower to develop special-purpose algorithms for all problems at hand.

In order to approach the performance of special-purpose algorithms, we add various elaborations to the basic components of tabu search. For example, since the tabu tenure (= the length of a tabu list) t is an important parameter that has a crucial influence over performance but is tedious to tune for each problem under consideration, we introduced an automatic control mechanism so that t is adaptively adjusted during search. Our algorithm can also handle optimization problems with objective functions, by adding the constraint that the solutions to be searched must have better objective values than the current incumbent value; the weight of this constraint is also automatically adjusted. The local search in our algorithm is normally carried out by using the so-called shift neighborhood, but a larger neighborhood that contains some of the swap neighbors in addition to the shift neighbors is also introduced to deal with tougher problems. We also argue that special care is necessary to describe the constraints of given problems appropriately.

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

2. Formulation of CSP

The CSP is formally defined by $(\boldsymbol{V}, \boldsymbol{D}, \boldsymbol{C})$ [9,24,30]: $\boldsymbol{V}$ is a finite set of variables $\{X_1, X_2, \ldots, X_n\}$, $\boldsymbol{D}$ is a set of the corresponding domains $\{D_1, D_2, \ldots, D_n\}$, where D_i is a finite domain of variable X_i, and $\boldsymbol{C}$ is a finite set of constraints $\{C_1, C_2, \ldots, C_r\}$, where each constraint C_l is defined by

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

that is, a set of all the legal t_l-tuples on variables $X_{l_1}, \ldots, X_{l_{t_l}}$. 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 the CSP is an assignment of exactly one value in D to each variable $X_i, i = 1, 2, \ldots, n$ such that all constraints $C_1, C_2, \ldots, C_r$ 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 pair of variable X_i and its value $j \in D$, such that

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

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 $\boldsymbol{x} = (x_{ij} \mid i = 1, 2, \ldots, n,\ j \in D)$. 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,

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

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

At this point, we note that each constraint C_l can be described in different ways. We take the view that one of the basic tools 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 experience in operations research that such formulations are usually very efficient to represent various constraints arising in practical problems such as scheduling problems, assignment problems and so on.

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

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

the constraint that all variables X_i in set $V''(\subseteq V)$ must take different values is given by

$$\sum_{X_i \in V''} x_{ij} \leqslant 1, \quad j \in D \tag{3}$$

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

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

However, we shall also allow other methods of representing constraints in Section 4, and will observe that such generalization is sometimes very important, particularly for large scale problems, to keep the formulation compact and efficient. For example, in order to impose a constraint that all variables in V'' take different values, we may simply prepare a subroutine that counts how many variables violate this constraint in the current solution (instead of adding linear inequalities (3)). This flexibility of formulation is an advantage of the CSP approach, not found in other approaches such as integer programming. At the current stage, however, it is the users' responsibility to find an appropriate CSP formulation of given problems. In our current code, only linear inequalities, linear equalities, and the above not-equal-values constraints are incorporated as building blocks.

In summary, our starting assumption is that an instance of the CSP is described by the following constraints of 0-1 variables:

$$\begin{array}{ll} \sum_{i,j} a_{hij} x_{ij} \leqslant b_h, & h = 1, 2, \ldots, m', \\ \text{ad hoc constraints } h, & h = m' + 1, \ldots, m, \\ x_{ij} = 0 \text{ or } 1, & i = 1, 2, \ldots, n, \quad j \in D, \end{array} \tag{4}$$

in addition to the implicit constraint (2), where x_{ij} are the value–variables introduced in the beginning of this section. 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

3.1. Penalty function

During the search process, we assume that constraint (2) is always satisfied; i.e., we consider only the following search space $\mathscr{X}$ defined on the value–variables x_{ij}:

$$\mathscr{X} = \left\{ \boldsymbol{x} \,\middle|\, \sum_{j \in D} x_{ij} = 1,\ i = 1,2,\ldots,n \right\}. \tag{5}$$

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

$$p(\boldsymbol{x}) = \sum_{h=1}^{m} w_h p_h(\boldsymbol{x}), \tag{6}$$

where

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

and $w_h > 0$ is a weight given to each constraint.

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

$$\begin{array}{ll} \text{minimize} & p(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} \in \mathscr{X}. \end{array} \tag{7}$$

Determination of appropriate weights w_h for the constraints 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.

3.2. Neighborhood and tabu list

We next describe the *shift* 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 $\boldsymbol{x} \in \mathscr{X}$). $N(\boldsymbol{x})$ denotes the set of solutions obtained by such modifications, i.e.

$$N(\boldsymbol{x}) = \left\{ \boldsymbol{x}(x_{ij} \leftarrow 1) \,\middle|\, x_{ij} = 0,\ i = 1,2,\ldots,n,\ j \in D \right\}, \tag{8}$$

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

$$p\left(\boldsymbol{x}^{(k)}(x_{i^*j^*} \leftarrow 1)\right) \leqslant p\left(\boldsymbol{x}^{(k)}(x_{ij} \leftarrow 1)\right)$$
$$\text{for all } \boldsymbol{x}^{(k)}(x_{ij} \leftarrow 1) \in N(\boldsymbol{x}^{(k)}) \setminus T. \tag{9}$$

In this case, we say that $\boldsymbol{x}^{(k+1)}$ is obtained from $\boldsymbol{x}^{(k)}$ by a *move* $x_{i^*j^*} \leftarrow 1$. Here, the set T in Eq. (9) is a set of solutions excluded from the candidates of the next solutions, and called a *tabu list* [15], which is introduced to prevent cycling over a small set of solutions and to introduce vigor into the search by a form of short term diversification. There are various methods to define set T. A straightforward one is to keep explicitly the solutions to be excluded from serving as candidates. A more popular one is to define T by prohibiting the reverse moves of those moves which were executed recently. In our algorithm, the latter method is adopted. To be precise, in each iteration k, we prepare a list $H^{(k)}$ that keeps the attributes $a^{(l)}$ of the moves (to be defined below) made in the past lth iterations, $l = k-1, k-2, \ldots, k-t$; i.e., $H^{(k)} = \{a^{(k-1)},$

$a^{(k-2)}, \ldots, a^{(k-t)}\}$, and define T as the set of solutions obtained from the current solution $x^{(k)}$ by the moves that use some attributes in $H^{(k)}$. Here, t is a parameter that describes how much of the past we should remember and is called a *tabu tenure*. This t has to be tuned carefully, since the performance of tabu search highly depends on it. In the next section, we consider automatic adjustment of the tabu tenure.

For the attribute $a^{(l)}$, we consider the following two types, corresponding to the variable and its value, associated with the move in the lth iteration, respectively. That is, if a move in the lth iteration changes assignment $x_{i^*j'} \leftarrow 1$ to $x_{i^*j^*} \leftarrow 1$ (which incurs $x_{i^*j'} \leftarrow 0$), we set $a^{(l)} = X_{i^*}$ and $a^{(l)} = x_{i^*j'}$, respectively. In the former case, any move obtained by changing the value of variable X_{i^*} is prohibited, while in the latter case, any move $x_{i^*j'} \leftarrow 1$ is prohibited. Clearly, the tabu list T determined by the attributes in $H^{(k)}$ becomes larger if the former definition is used (except that both definitions coincide if $d = 2$). In the rest of this paper, we refer to these two as *variable-based* and *value-based* tabu lists, respectively.

An *aspiration criterion* is also incorporated in our code; i.e., if a tested solution $\boldsymbol{x}^{(k)}(x_{ij} \leftarrow 1)$ is better than the incumbent solution (i.e., the currently best solution found so far, in the sense of having the smallest penalty), it is selected as $\boldsymbol{x}^{(k+1)}$ even if it belongs to T. In our tabu search, we use a weaker aspiration criterion, which will be explained in Section 4.2 together with the automatic control of the tabu tenure.

3.3. Other considerations

Other considerations listed below are also incorporated into our CSP algorithm.

(i) *Diversification by long term memory.* For the diversification of tabu search, long term memory is often used [15,16]. In our code, we 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. We try to diversify by making the move of value–variable x_{ij} with large LTM (x_{ij}) difficult; more precisely, in the choice of the next solution, ties between the best candidate solutions with the smallest penalty value are broken by preferring one with the smallest LTM value.

(ii) *Reduction of the neighborhood.* In each iteration, the solutions in the neighborhood $N(\boldsymbol{x}^{(k)})$ of the current solution $\boldsymbol{x}^{(k)}$ are tested. Since checking all solutions in $N(\boldsymbol{x}^{(k)})$ takes a lot of computational time, however, we reduce their number by screening them heuristically as follows: Since any move $x_{ij} \leftarrow 1$ that decreases the penalty value must be related to some constraints which are currently violated, we consider only those solutions obtained by changing the values of some variables in such violated constraints.

(iii) *Generation of greedy initial solutions.* To start the tabu search approach, an initial solution must be prepared. Using a random solution is easy, but may make the early stage of the search wasteful. To remedy this, we use a greedy method which, starting from the solution such that no variable is assigned a value, chooses variables X_i in nonincreasing order of $w(X_i) = \sum_{l \in I(X_i)} w_l$, where $I(X_i)$ is the set of constraints in Eq. (4) that contain X_i. Then we assign X_i the value that increases the penalty value by the smallest amount, where ties are broken randomly. In our experiment, this amount of randomness was sufficient to generate different initial solutions in all runs. However, if more randomness is necessary, it may be a good idea to introduce further randomness as used in GRASP (greedy randomized adaptive search procedure) [10,20] or more generally in probabilistic tabu search [15,16], in assigning greedy values.

(iv) *Randomness of the tabu tenure.* If a cycling occurs in the tabu search, it is usually prevented by making the tabu tenure t larger. But, since a large t may prohibit other effective moves as well, a random choice of t is also used as an alternative method. In all the experiment in this paper, we also incorporate this method by using the effective tabu tenure t' which is randomly chosen from set $\{t-1, t, t+1\}$ in each iteration, where t is the tabu tenure determined by our control mechanism described in Section 4.

The resulting algorithm is more or less a standard implementation of tabu search, but specialized to problem CSP. 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 Eq. (4) 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 columnwise and rowwise. Using this data structure, for example, the difference

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

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

To make the references to T efficient, we implement $H^{(k)}$ by preparing the array H, in the case of a variable-based (resp., value-based) tabu list, $H(X_i)$ (resp., $H(x_{ij})$), to keep the iteration number k at which the move of attribute X_i (resp., x_{ij}) 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.

4. Adaptive control of tabu tenure

The performance of tabu search depends on various program parameters, particularly the tabu tenure t. To alleviate the task of finding appropriate values of t, without resorting to costly preliminary experiment, it is attractive from the view point of a general problem solver to incorporate an automatic mechanism of controlling t. One such approach is provided by reactive tabu search [3], in which, based on a tabu list of attributes generated by a hash function (which seeks to assign a different integer to each solution, with as little duplication of such assignments as possible), the tabu tenure is increased if some solutions (or more precisely, hash values) are visited repeatedly, while it is decreased if only new solutions (hash values) are encountered for a while. However, we take a slightly different approach in this paper because our tabu list T is defined by the attributes associated with the moves in the past iterations. It has been sometimes argued that a tabu list T defined by moves is more effective than the one that approximates the storage of past solutions explicitly, not only because it prevents cycling but also it restricts the direction of search so that a certain degree of diversification is enforced. Also, for large scale problems, the controlling method of reactive tabu search may not be strong enough because cycling rarely occurs if the solution space is large.

In our method, tabu tenure is initially set to $t := 1$. Its control mechanism consists of two kinds of rules that operate when t is increased and when t is decreased, respectively. These rules will be explained in Sections 4.1 and 4.2.

4.1. Increment rule of tabu tenure

At every iteration k, we keep two lists of attributes $H^{(k)}$ and $A^{(k)}$, where $H^{(k)}$ is used to define the tabu list T (as described before), and $A^{(k)}$ also stores recent attributes of some period which is determined by a different rule and is used to control the tabu tenure t. Initially, we set $A^{(0)} := \phi$ and add all attributes $a^{(l)}, l = 1, 2, \ldots, k-1$, to obtain $A^{(k)}$.

Rule 1 to increase t by one: The attribute $a^{(k)}$ associated with the move in the kth iteration already occurs in $A^{(k)}$ as $a^{(k')}$ (where $k' < k$), and $A^{(k)} = A^{(k')}$ holds (i.e., $A^{(l)}$ has not changed during the interval given by $l = k', k'+1, \ldots, k$).

If $a^{(k)}$ and $A^{(k)}$ satisfy this condition, we consider that the search is confined to a small area, because this implies that the indicated attributes are repeatedly chosen during recent iterations (since tabu tenure t is small). Therefore, we try to add diversification effect by increasing t (i.e., enlarging the tabu list T). In order to prevent the size of $A^{(k)}$ from increasing without limit, a rule is introduced to activate the reset operation $A^{(k)} := \phi$ if one of the following conditions is met (in which case, we consider that the current diversification process is completed).

1. The incumbent solution (the best solution obtained so far) is updated to a solution with a smaller penalty value.
2. After each increment of t, all attributes $a^{(l)}$ not in $A^{(l)}$ are *marked* (until the next reset $A^{(k)} := \phi$ occurs) if the lth move gives a worse penalty value than the current penalty value $p(\boldsymbol{x}^{(l-1)})$. Then the reset rule is activated if there is a marked attribute in $H^{(k-1)} \setminus H^{(k)}$ (i.e., the

attribute is removed from $H^{(k-1)}$ in the process of defining $H^{(k)}$) but which is not chosen in the kth iteration. (This means that putting back the move prohibited by the marked attribute does not improve the penalty value.)

In either case, we conclude that diversification is attained to a satisfactory extent. Furthermore, the complement of condition 2 is used as the second rule for increasing t.

Rule 2 to increase t by one: A marked attribute is again chosen on the kth iteration immediately after it is removed from $H^{(k-1)}$.

In some cases, the continuation of the solution process becomes impossible because all of the neighbors of the current solution are contained in the tabu list T. In this case, as an exceptional rule, the tabu tenure t is reset to 1 and $A^{(k)}$ to $A^{(k)} := \phi$. Then the search resumes by using these new $H^{(k)}$ and $A^{(k)}$ lists.

4.2. Decrement rule of tabu tenure

In tabu search, an aspiration rule is usually incorporated to allow a solution to be visited even if it is on the tabu list T. We use this aspiration rule also as a rule to decrease t, as it may indicate that the current tabu list is too large.

Rule to decrease t by one: A solution is chosen on the kth iteration by an aspiration rule.

Aspiration criterion. As the aspiration rule in our algorithm, in addition to the standard one that allows a solution in T to be chosen if it improves upon the current incumbent solution, we use the following rule. For an attribute $a^{(k')}$ in $H^{(k)}$ (i.e., $x^{(k')}$ was obtained from $x^{(k'-1)}$ by the move with attribute $a^{(k')}$, where $k' < k$ holds), let

$$\Delta p^{(k')} := p(\boldsymbol{x}^{(k')}) - p(\boldsymbol{x}^{(k'-1)}),$$
$$\Delta p' := p(\boldsymbol{x}') - p(\boldsymbol{x}^{(k)}),$$

where $\boldsymbol{x}'$ is the solution obtained by the move prohibited by applying $a^{(k')}$ to $\boldsymbol{x}^{(k)}$. We activate the aspiration rule and choose $\boldsymbol{x}'$ as $\boldsymbol{x}^{(k+1)}$ if two conditions

(i) $\Delta p^{(k')} \leqslant 0$ and $\Delta p' < 0$ or $\Delta p^{(k')} < 0$ and $\Delta p' \leqslant 0$ (i.e., both moves give improvements),

(ii) $p(\boldsymbol{x}') \leqslant p(\boldsymbol{x}^{(k')})$ (10)

hold, and $\boldsymbol{x}'$ has a smaller penalty than any other solutions in $N(\boldsymbol{x}^{(k)})$.

4.3. Experiment with graph coloring problem

We first applied our algorithm described so far to the *graph coloring problem* (GCP) to see the effects of the automatic control mechanism for tabu tenure, types of attributes (i.e., variable-based or value-based) and other factors of tabu search. Our algorithm is coded in C and runs on a Sun Ultra 2. GCP is the problem of assigning a color to each vertex of a given undirected graph $G(V, E)$, so that, for every pair (i_1, i_2) of adjacent vertices, i_1 and i_2 receive different colors. To handle the GCP in the framework of CSP, we consider the GCP as a decision problem that asks whether k colors are sufficient or not. The GCP can be formulated as a CSP by introducing variables X_i corresponding to vertices $i = 1, 2, \ldots, n$, a domain $D = \{1, 2, \ldots, k\}$ representing possible colors, and associated variables x_{ij} such that

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

To prohibit adjacent vertices from receiving the same color, one approach is to introduce a constraint $C_l(X_{i_1}, X_{i_2})$ consisting of the following linear inequalities for each edge $(i_1, i_2) \in E$:

$$x_{i_1 j} + x_{i_2 j} \leqslant 1, \qquad j \in D. \tag{11}$$

In this formulation, there are $k|V| = kn$ variables x_{ij} and $m = k|E|$ inequalities (in addition to those in $\mathscr{X}$ of Eq. (5)). An alternative method to impose the same constraint, as noted in Section 2, is to prepare a subroutine that counts the number of edges violating the color constraint. This formulation is also tested in our experiment.

The performance of our algorithm is tested against some of the DIMACS benchmarks called Leighton graphs. Although the minimum numbers of colors k of these instances are known by construction, some of them are very difficult to find, as described in [11]. We tested four types of control mechanisms for t, i.e., those using fixed $t = 10, 20, 30$, and the automatic control mechanism described in Sections 4.1 and 4.2 (which is denoted as "adaptive"), for the two attribute types variable-based and value-based (thus in total of eight cases). We execute 10 runs for each instance, where a run is terminated if a feasible solution is found, or the computational time reaches 300 s. Each run starts from a different initial solution, due to the randomness in the algorithm of generating initial solutions (see Section 3.3 (iii)). A run is considered a *success* if a feasible solution is found before 300 s; otherwise a *failure*.

Computational results for Leighton graphs are shown in Table 1, where $|V|, |E|, k$ are the numbers of vertices, edges, and the minimum number of colors, respectively. For each instance, computational results are given in the manner explained under the table. For the adaptive mechanism, the average size of the tabu tenure t is also given. Table 1 says that in both cases of variable-based and value-based, best sizes for fixed t vary according to problem instances, while the results obtained for an adaptive t are always close to the results obtained for the best fixed t values. Thus taking into account that the adaptive method does not need to tune t beforehand, it may be highly recommended for practical use.

Now, we consider how attribute types affect the performance. In most cases, the variable-based attribute is better than the value-based attribute, except that, for the cases in which feasible colorings are not found (such as le450-15a, b), the value-based attribute tends to give colorings having smaller penalties. As we shall also see later, the general tendency may be summarized as follows. For small scale problems, the variable-based attribute is not very effective, since the tabu list tends to prohibit too many solutions. For large scale instances, however, the variable-based attribute becomes more suitable, since the tabu list realized by the value-based attributes appears to be too small in order to allow sufficient diversification. However, this tendency is not always true, and the final choice of the type of attributes should be made from the experience of similar problems or by preliminary experiments. Indeed, as noted in [15,16], there can be value in defining tabu conditions based on more than one attribute type, and our experience suggests that such a combined determination of tabu status is likely to prove favorable in the present setting as well.

Next, we compare the results of applying our method with those obtained by the two algorithms hybrid and tabu search of [11], which are specially developed for graph coloring. As shown in Table 2, the minimum colorings are found by the specialized methods for all instances except two, le450-25c and d (for those instances, the results in Table 2 are only for $k = 26$ relaxed from the minimum value 25). Our algorithm also failed to solve these two instances. For other instances, however, we may say that our algorithm performs at least as well as the methods of [11], even if we take into account the difference of the computers. In particular, for such instances as le450-5a, b, 15c, and d, our algorithm could find feasible solutions in less computational time.

The coloring constraint (11) was described by a subroutine, not by linear inequalities, in the above experiment. To see the difference, Table 3 shows the average CPU time (in seconds) in the two cases in which constraint (11) is represented by linear inequalities and by a subroutine, respectively, for three instances, le450-5b, 15c, and 25a (using the variable-based adaptive method). There is no difference in the numbers of iterations, but CPU time is different. As is clear from Table 3, considerable computational time can be saved by using a subroutine that describes constraints. The effect of using such subroutines highly depends on the types of constraints involved, and preparing them requires some effort on the part of users (our code has a tool to describe the not-equal-value constraints as Eq. (11) easily). It is the users' responsibility to decide how much of the constraints will be written by inequalities and how much by subroutines.

As other benchmarks used in [11], random graphs are also tested. For these instances, optimal values are not known and k is set to the best known value $\tilde{k}$ except for some instances for which $k = \tilde{k}$ and $\tilde{k} + 1$ are used. As in the experiment for Leighton graphs, the maximum CPU time for each run is set to 300 s, and runs are repeated 10 times for each instance. Table 4 gives some of the com-

Table 1
Computational results for Leighton graphs

Instance	$\|V\|$	$\|E\|$	k	Variable-based					Value-based				
				Fixed			Adaptive		Fixed			Adaptive	
				$t = 10$	$t = 20$	$t = 30$		Ave. of t	$t = 10$	$t = 20$	$t = 30$		Ave. of t
le450_5a.col	450	5714	5	3.8	1.4	2.9	1.9	13.2	9/10	3.1	3.3	6.2	7.2
				6988.4	1928.4	5791.8	2896.9		(4.1)	6943.9	7078.2	16105.9	
le450_5b.col	450	5734	5	9/10	3.3	5.3	2.6	16.7	25.8	8.1	8.7	25.6	6.9
				(6.1)	7499.6	11925.8	4686.2		71201.9	22001.3	21534.8	80915.5	
le450_5c.col	450	9803	5	7/10	0.8	0.8	0.9	14.2	9/10	1.1	1.8	1.4	6.9
				(10.1)	921.6	1097.7	1264.5		(8.4)	1743.4	3402.6	2169.5	
le450_5d.col	450	9757	5	1.5	0.6	0.6	1.3	32.5	9/10	0.8	1.1	1.5	6.6
				2040.4	648.2	697.2	1815.3		(5.0)	959.2	1684.8	2697.3	
le450_15a.col	450	8168	15	0/10	0/10	0/10	0/10	14.7	0/10	0/10	0/10	0/10	8.1
				(5.1)	(8.8)	(11.9)	(6.3)		(8.9)	(3.6)	(5.4)	(2.0)	
le450_15b.col	450	8169	15	0/10	0/10	0/10	0/10	14.3	0/10	0/10	0/10	1/10	7.5
				(4.3)	(8.1)	(11.8)	(5.8)		(1.7)	(3.2)	(4.6)	(1.6)	
le450_15c.col	450	16680	15	0/10	110.4	48.8	81.6	20.8	0/10	1/10	8/10	0/10	16.2
				(169.3)	56430.8	32608.6	38275.2		(177.1)	(91.2)	(1.1)	(77.7)	
le450_15d.col	450	16750	15	0/10	8/10	78.5	106.1	21.1	0/10	0/10	1/10	0/10	18.4
				(176.1)	(0.2)	65036.5	70516.1		(182.3)	(123.5)	(10.0)	(79.5)	
le450_25a.col	450	8260	25	1.4	2.2	20.0	0.4	4.6	0.3	0.3	0.3	0.3	2.2
				1669.2	2872.2	19186.0	314.8		212.8	221.5	339.6	294.8	
le450_25b.col	450	8263	25	0.4	0.6	1.4	0.2	2.1	0.2	0.2	0.2	0.2	1.1
				258.4	237.9	559.0	39.9		51.1	61.6	55.8	57.8	
le450_25c.col	450	17343	25	0/10	0/10	0/10	0/10	15.4	0/10	0/10	0/10	0/10	9.3
				(17.0)	(20.0)	(24.9)	(17.7)		(18.9)	(13.9)	(16.5)	(12.5)	
le450_25d.col	450	17425	25	0/10	0/10	0/10	0/10	15.6	0/10	0/10	0/10	0/10	9.2
				(15.8)	(20.2)	(24.1)	(16.8)		(18.8)	(14.6)	(16.4)	(12.1)	

The upper row in each entry gives the average CPU time (in seconds) and the lower row gives the average number of iterations, if all 10 runs are successful. Otherwise, the upper row describes the success rate (e.g., 9/10 says that 9 runs out of 10 are successful), and the lower row (with parentheses in this case) the average number of edges violating the color constraint in the last incumbent solutions of 10 runs.

Table 2
Computational results with two special purpose algorithms [11] for Leighton graphs

Instance	k	CPU time	(s) [a]
		hybrid [b]	tabu [c]
le450_5a.col	5	49.2	303.0
le450_5b.col	5	45.3	228.4
le450_5c.col	5	15.2	3.3
le450_5d.col	5	39.0	3.3
le450_15a.col	15	1920.0	220.6
le450_15b.col	15	2464.0	191.8
le450_15c.col	15	4828.0	— [d]
le450_15d.col	15	5036.0	— [d]
le450_25a.col	25	4.0	4.0
le450_25b.col	25	3.9	3.9
le450_25c.col	26	3089.0	2756.0
le450_25d.col	26	3322.0	4258.0

[a] Using a SPARC 10 processor.
[b] Tabu search-genetic hybrid in [11].
[c] Tabu search in [11].
[d] '—' means a feasible coloring cannot be found in 24 h.

putational results by the value-based adaptive method, showing the number of vertices $|V|$, edges $|E|$, colors k, the average number of iterations and CPU time (in seconds for 10 runs), and the average tabu tenure t. For graphs with 100 vertices, all runs are successful if $k = 15$, while if $k = 14$, the feasible coloring was found for only one instance g20.col. For graphs with 300 vertices, all runs are successful except for gg2.col and gg8.col (for which nine out of 10 are successful) if we set $k = 34$. For $k = 33$, we found feasible colorings for four instances, with success rates of $\frac{1}{10}$–$\frac{4}{10}$. By spending much longer time, we could find a feasible coloring for one more instance. In [11], for the same 20 graph instances of 100 vertices, an average CPU time of 9.5 s (on a SPARC 10 processor) is required to solve all instances with $k = 15$ successfully, and for the same 10 graph instances of 300 vertices, an average CPU time of 353 s solves all instances with $k = 34$ successfully. The average minimum numbers of colors needed to color graphs of 100 vertices and 300 vertices are 14.95 and 33.5, respectively, which are the same as our results.

From the results in this section, our algorithm based on the CSP formulation appears to be competitive with the special-purpose algorithms in [11] for the graph coloring problem.

5. How to deal with objective functions

5.1. Schemes of incorporating objective functions

The CSP is a decision problem that seeks a feasible solution for a given problem instance. However, as optimization problems are often encountered in practical applications, we consider in this section how to deal with objective functions within the CSP framework. For simplicity, we only consider the minimization of a linear objective function

$$f(x) = \sum_{i,j} c_{ij} x_{ij},$$

where, the c_{ij} coefficients are given integers, though there is no theoretical difficulty in handling more general functions (e.g., nonlinear functions with real coefficients).

A direct approach to incorporate $f(x)$ into the CSP framework is to modify the objective function of Eq. (7) as follows:

$$\begin{array}{ll} \text{minimize} & q(x) = w_0 f(x) + p(x) \\ \text{subject to} & x \in \mathscr{X}, \end{array} \qquad (12)$$

where $w_0 > 0$ is a weight given to $f(x)$, and must be adjusted. If w_0 is large, it tends to search those

Table 3
Effect of the methods for representing the constraints

Instance	Number of iterations	CPU time (s)		A/B
		Linear inequalities (A)	Subroutine (B)	
le450-5b	4686.2	19.0	2.61	7.3
le450-15c	38275.2	774.4	81.64	9.5
le450-25a	314.8	12.0	0.38	31.6

Table 4
Computational results for random graphs

Instance	$\lvert V\rvert$	$\lvert E\rvert$	k	CPU time (s)	Iterations	Average of t
g1.col	100	2487	15	0.6	2011.8	3.5
g2.col	100	2487	15	0.9	2749.4	3.7
g3.col	100	2482	15	2.0	6684.1	4.0
g4.col	100	2503	15	1.1	3539.3	3.4
⋮	⋮	⋮	⋮	⋮	⋮	⋮
g18.col	100	2472	15	0.9	2964.2	3.9
g19.col	100	2527	15	2.3	7368.0	4.1
g20.col	100	2420	15	0.2	535.7	2.4
			14	5.5	19576.8	4.5
gg1.col	300	22482	34	79.4	82601.6	6.2
gg2.col	300	22569	34	(211.6) [a]	(217443.0) [a]	6.8
gg3.col	300	22393	34	78.6	82783.6	6.0
gg4.col	300	22446	34	103.4	107937.1	6.3
gg5.col	300	22360	34	81.9	85808.5	6.1
gg6.col	300	22601	34	305.1	314476.5	6.8
gg7.col	300	22327	34	70.4	73823.5	6.1
gg8.col	300	22472	34	(130.8) [a]	(135266.0) [a]	6.5
gg9.col	300	22520	34	176.6	186164.5	6.5
gg10.col	300	22543	34	256.2	262569.3	6.8

[a] Nine runs (out of 10) are successful, and these numbers are the averages of successful runs.

solutions with smaller objective values, but may be difficult to attain feasibility. On the other hand, if w_0 is small, we may get the opposite tendency.

An alternative approach is to introduce a new constraint that the value of $f(\boldsymbol{x})$ must not be greater than a given value z, i.e.,

$$f(\boldsymbol{x}) \leqslant z \tag{13}$$

and seek feasibility by adding this constraint to the set of original constraints. The right-hand side z may be initially set to a sufficiently large value or an appropriate value determined from experience with similar problem instances. The addition of the new constraint (13) is equivalent to considering

$$\begin{array}{ll} \text{minimize} & q(\boldsymbol{x}) = w_0 \max(f(\boldsymbol{x}) - z, 0) + p(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} \in \mathscr{X}, \end{array} \tag{14}$$

where $w_0 > 0$ is a weight given to constraint (13). If the objective value of Eq. (14) becomes 0, the solution $\boldsymbol{x}$ is a feasible solution whose objective value is at most z. Then we update

$$z := z' - 1,$$

where z' is the value of $f(\boldsymbol{x})$ found in the preceding computation. We repeat this iteration until an acceptable solution is found, or the preassigned computation time expires.

Now let z' denote the incumbent value, that is, the best objective value $f(\boldsymbol{x})$ among the feasible solutions found so far. The first approach (12) does not take into account the knowledge of this z', and hence the search may be wasted in the region where $f(\boldsymbol{x})$ take larger values than z'. The second approach remedies this defect, but may not have enough power to find feasible solutions with smaller objective values if the current value is already smaller than z', since the penalty term $\max(f(\boldsymbol{x}) - z, 0)$ in Eq. (14) is independent of the difference $z - f(\boldsymbol{x})$ if $f(\boldsymbol{x})$ is not larger than z.

To compromise these two approaches, we generalize Eqs. (12) and (14) by combining them in the following problem:

$$\begin{array}{ll} \text{minimize} & q(\boldsymbol{x}) = w_0\{\max(f(\boldsymbol{x}) - z, 0) + \\ & \theta \min(f(\boldsymbol{x}) - z, 0)\} + p(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} \in \mathscr{X}, \end{array} \tag{15}$$

where $0 \leqslant \theta \leqslant 1$ is a program parameter. This is equivalent to Eq. (14) if $\theta = 0$, and to Eq. (12) if

$\theta = 1$. The penalties $\max(f(\boldsymbol{x}) - z, 0) + \theta \min(f(\boldsymbol{x}) - z, 0)$ are illustrated in Fig. 1 for $\theta = 0, 0.5$ and 1, respectively.

In using the automatic control mechanism of t in Section 4 for these problems, it should be noted that only changing the role of $p(\boldsymbol{x})$ to $q(\boldsymbol{x})$ of Eq. (15) is not sufficient, because we want to adjust the weight w_0 in the course of computation, as will be described in Section 5.2. Therefore, we regard $p(\boldsymbol{x})$ and $f(\boldsymbol{x})$ as two independent penalty values in describing the control mechanism, and activate each rule (of increasing and decreasing t) if one of $p(\boldsymbol{x})$ and $f(\boldsymbol{x})$ satisfies the stated condition.

5.2. Computational experiment for problems with objective functions

5.2.1. Generalized assignment problem

To see the effect of parameters θ and w_0 in Eq. (15) to handle the objective function $f(\boldsymbol{x})$, we solve the *generalized assignment problem* (GAP). The GAP seeks a minimum cost assignment of n jobs to d agents such that each job is assigned to exactly one agent, and for each agent, the total resource requirement of the assigned jobs does not exceed its capacity. Although the version of maximizing a gain function is also discussed in the literature, we consider here only the minimization version.

Let $I = \{1, 2, \ldots, n\}$ be a set of jobs, and $J = \{1, 2, \ldots, d\}$ be a set of agents. For $i \in I$ and $j \in J$, we define c_{ij} to be the cost of assigning job i to agent j, r_{ij} to be the amount of resource required by assigning job i to agent j, and b_j to be the resource capacity of agent j. By introducing a 0-1 variable x_{ij} for each pair of job i and agent j such that

$$x_{ij} = \begin{cases} 1 & \text{if job } i \text{ is assigned to agent } j, \\ 0 & \text{otherwise.} \end{cases}$$

The GAP can be formulated as follows:

$$\text{minimize} \quad f(\boldsymbol{x}) = \sum_{i=1}^{n} \sum_{j=1}^{d} c_{ij} x_{ij}$$

$$\text{subject to} \quad \sum_{j=1}^{d} x_{ij} = 1, \; i = 1, 2, \ldots, n, \qquad (16)$$

$$\sum_{i=1}^{n} r_{ij} x_{ij} \leqslant b_j, \; j = 1, 2, \ldots, d. \qquad (17)$$

It is not difficult to see that, if we regard variables X_i to correspond to jobs i and their domain D to be given by the set of agents, this is in the form of the CSP (with an objective function) in which constraints (16) correspond to Eq. (2).

We then apply our tabu search algorithm to those GAP instances found in [6,7,26] and others, which have the following characteristics:

- d is set to 5, 8 and 10, and n/d is set to 3, 4, 5 and 6, respectively.
- The coefficients c_{ij} and r_{ij} are integers chosen from the uniform distribution $U(15, 25)$ and $U(5, 25)$, respectively, and $b_j = (0.8/d) \sum_{i \in I} r_{ij}$.

For each (n, d) combination, five instances are generated, and the total number of test instances is 60. The optimal values of these instances are known by the branch and bound procedure of [5].

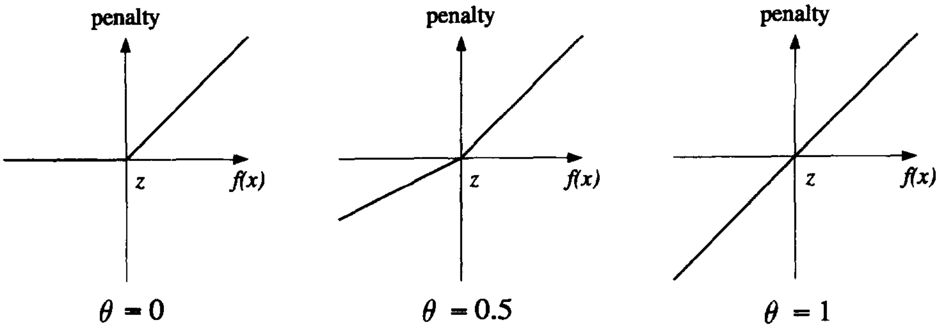

Fig. 1. The penalties for $f(x)$ in cases $\theta = 0, 0.5$ and 1.

Table 5
Computational results for a GAP instance with $n = 60$ and $d = 10$ using various θ and w_o

	$w_o = 1$	$w_o = 2$	$w_o = 4$	$w_o = 6$	$w_o = 8$	$w_o = 10$
$\theta = 0$	9/10	4.1	3.5	5.5	5.8	7.6
	(0.021%)	3069.8	2464.4	3873.9	4054.1	5285.0
$\theta = 0.5$	4.8	3.2	1.2	1.5	1.8	6.7
	3836.8	2419.0	844.9	1002.1	1217.3	4734.0
$\theta = 1$	2.8	1.1	1.1	7/10	0/10	0/10
	2202.4	797.7	744.9	(—)	(—)	(—)

Results are indicated in the same manner as Table 1.

First, we solved an instance with $n = 60$ and $d = 10$, while varying the program parameters θ and w_o in Eq. (15). We set the maximum computational time to 60 s, and run the tabu search 10 times in each case. In all cases, we use the value-based adaptive method. The results are shown in Table 5, in the same manner as in Table 1. That is, if all of 10 runs are successful, the average CPU time (in seconds) appears in the upper row and the average number of iterations required to find optimal solutions appears in the lower row; otherwise, the success rate (out of 10 runs) appears in the upper row and the average percentage deviation of the best solution from the optimal value appears in the lower row. The mark '—' means that there are runs in which no feasible solution is found (hence the incumbent value is infinity). From these results, we observe that if $\theta = 1$, the performance critically depends on w_o, and therefore it is very risky to use this θ value. Comparing $\theta = 0$ and $\theta = 0.5$, we can see that $\theta = 0.5$ has better performance than $\theta = 0$ in most cases. We also tested other instances of the GAP, and observed similar tendency.

Now we consider the effect of weight w_o. Table 6 lists the results for five instances of the GAP with $n = 60$ and $d = 10$ with different values of w_o, where θ is fixed to 0.5. With a smaller w_o, the influence of penalty function $p(\boldsymbol{x})$ in Eq. (15) becomes larger and the solutions tested in the search tend to be feasible, while the tendency is reversed with a larger w_o. From the results in Table 6, however, it is difficult to identify an appropriate value of w_o, as the best choice of w_o fluctuates from instance to instance.

5.2.2. *Automatic adjustment of weight w_o*

To overcome the difficulty of finding appropriate values of w_o, we then introduced an adaptive method to control w_o which tries to keep the rate of the tested solutions being infeasible within a specified range [LB,UB]. More precisely, w_o is

Table 6
Effect of weight w_0 under different control mechanisms for five GAP instances with $n = 60$ and $d = 10$ ($\theta = 0.5$, LB = 0.6, UB = 0.8, and $\sigma = 3.0$)

	With fixed weight w_o						With control mechanism		
	$w_o = 1$	$w_o = 2$	$w_o = 4$	$w_o = 6$	$w_o = 8$	$w_o = 10$	$w_o^{(0)} = 0.1$	$w_o^{(0)} = 1$	$w_o^{(0)} = 10$
Instance 1	3.8	1.2	0.6	0.9	0.7	0.4	1.4	1.1	1.0
	3115.8	890.3	378.0	421.3	476.5	250.7	1110.4	811.6	756.7
Instance 2	2.3	2.4	2.9	4.2	15.3	6.9	3.2	2.8	1.2
	1805.9	1768.7	2049.0	2918.1	7670.1	4706.6	2501.4	2170.6	913.3
Instance 3	4.8	3.2	1.2	1.5	1.8	6.7	2.9	1.1	1.2
	3836.8	2419.0	844.9	1002.1	1217.3	4734.0	2217.9	864.8	859.1
Instance 4	7/10	16.7	3.0	9/10	9.1	7.7	6.3	4.6	2.6
	(0.021%)	12818.1	2161.9	(0.007%)	6366.4	5469.6	5014.3	3645.2	2034.2
Instance 5	5.0	8.0	4.8	12.0	1/10	0/10	4.1	4.3	5.8
	4097.2	5921.0	3399.0	8413.4	(—)	(—)	3232.4	3389.0	4600.1

Results are indicated in the same manner as Table 1.

multiplied (resp., divided) by σ, where $\sigma > 1$ is a program parameter, if the rate of the tested solutions being infeasible in the most recent 100 iterations is smaller than LB (resp., larger than UB).

To see the effect of the range [LB, UB] and parameter σ, Fig. 2 gives the average CPU time (in seconds) for [LB,UB] = [0.2,0.4], [0.4,0.6], [0.6,0.8], [0.2,0.8] and $\sigma = 1.1$, 1.5, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, respectively. For each combination of [LB,UB] and σ, five GAP instances of Table 6 are solved; 30 runs for each instance, which consist of 10 runs for each of three initial values of w_o, $w_o^{(0)} = 0.1, 1, 10$, respectively, setting maximum computational time of each run to 60 s. It is found that using a small σ (such as 1.1 and 1.5) is not a good choice, because w_o cannot be adjusted rapidly enough, in particular, when $w_o^{(0)}$ is far from its appropriate value. However, performance of the search is not very sensitive to σ. On the other hand, parameters LB and UB have greater influence than σ, and hence must be tuned carefully. Among the ranges tested in this experiment, [LB,UB] = [0.6,0.8] appears to be the best choice. As $\sigma = 3.0$ gives the best performance for this [LB,UB], the results in Table 6 with the control mechanism of w_o are given with these values of the parameters (for three initial values $w_o^{(0)} = 0.1, 1, 10$). The good performance observed with a rather large value of σ and a narrow range of [LB,UB] may be attributed to a diversification effect realized by such a choice; the infeasibility rate during search oscillates between two sides of the center range [LB,UB]. The appropriate location of the center range [LB,UB] appears to vary according to the problem types. If it is easy to find feasible solutions $\boldsymbol{x}$ (in the sense of $p(\boldsymbol{x}) = 0$), both of LB and UB should be set to small values (e.g., [LB,UB] = [0.2,0.4] or [0.4,0.6]); otherwise they should be set to large values such as [LB,UB] = [0.6,0.8]. For this reason, in the next experiment for the set covering problem, we used [LB,UB] = [0.4,0.6].

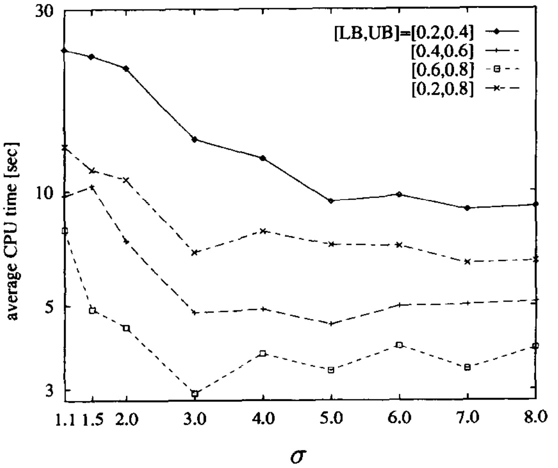

Fig. 2. Effect of parameters LB, UB, and σ for GAP instances with $n = 60$ and $d = 10$ ($\rho = 0.5$).

5.2.3. Set covering problem

The *set covering problem* (SC) is a well known combinatorial optimization problem defined as follows [1,4,2,14,22,23]:

$$\begin{aligned} &\text{minimize} && f(\boldsymbol{x}) = \boldsymbol{c}\boldsymbol{x} \\ &\text{subject to} && A\boldsymbol{x} \geqslant \boldsymbol{e} \qquad (18) \\ & && x_i = 0 \text{ or } 1, \quad i = 1, 2, \ldots, n, \end{aligned}$$

where $A = (a_{hi})$ is an $m \times n$ 0-1 matrix, which is called the incidence matrix, $\boldsymbol{c} = (c_1, c_2, \ldots, c_n)$ is a vector of cost coefficients $c_i > 0$ and $\boldsymbol{e}$ is the m-vector of ones.

Precisely speaking, the variables x_i are not "value-variables" in the sense of Eq. (1) but are variables with domain $D = \{0, 1\}$. Therefore, if Eq. (18) is to be described in the standard CSP form of Section 2, the variables x_{i1} and x_{i0} may be introduced for each x_i such that $x_{i1} = 1$ (resp., 0) if the variable $x_i = 1$ (resp., 0), and $x_{i0} = 1 - x_{i1}$ holds by constraint (2). Since these two value-variables are completely dependent on each other, however, we identify x_{i1} with x_i and do not use x_{i0} for notational simplicity.

We first applied our CSP algorithm resulting from the above experiment on the GAP to some SC instances defined from *Steiner triple systems* (STS) [1,14,23], which are known to be very difficult. These instances have characteristics that cost coefficients c_i are 1 for all i, there are exactly three ones in each row of A, and there is exactly one row i such that $a_{ij_1} = a_{ij_2} = 1$ for each pair of columns j_1 and j_2 of A. It is known that such matrices exist if and only if $n \geqslant 3$ and $n \equiv 1, 3 \pmod 6$, in which case $m = \frac{1}{6}n(n-1)$. We tested three STS problems with $n = 81, 135$ and 243, denoted by STS_{81}, STS_{135} and STS_{243}, respectively. The optimal value of STS_{81} is known to be 61, which is proved by a branch and bound procedure [23]. For STS_{135} and STS_{243}, the optimal values are not known, and the best previously known values were 104 and 202, respectively [23], but the latter was improved to 198 by our experiment. Our tabu search was run 30 times for each instance, setting the maximum computational time of each run as 300 s. In all runs, the optimal or best known solutions are obtained successfully. Table 7 gives the minimum, average, and maximum CPU times (in seconds) in upper rows, and the number of iterations in lower rows, required to find the best solutions. Here we used the variable-based adaptive method, and parameter values $\theta = 0.5, \mathrm{LB} = 0.4, \mathrm{UB} = 0.6, \sigma = 3.0$ and $w_0^{(0)} = 1$ for all runs.

However, for other types of SC instances in which cost coefficients c_i vary to a great extent, the performance of this tabu search algorithm becomes rather poor. The reason for this phenomenon is that, because the effect of the objective function in $q(\boldsymbol{x})$ of Eq. (15) is dominated by those variables x_i with large coefficients c_i, it is difficult to conduct fine tuning of the variables with small c_i coefficients so that the objective value may be slightly improved from the current level. To overcome this difficulty, we may have to develop a different idea, and we consider this to be a challenge for future work.

6. Swap neighborhood

6.1. Definitions and implementation

Many problems have constraints of type

$$\sum_{X_i \in V'} x_{ij} = a_j, \quad j \in D, \tag{19}$$

where V' is a subset of the set of all variables V, and a_j are constants; typically $a_j = 1$ for all j. Assume that a solution $\boldsymbol{x}$ of Eq. (19) satisfies $x_{i_1 j_1} = 1$ and $x_{i_2 j_2} = 1$ for some $i_1 \neq i_2$ and $j_1 \neq j_2$. Then the solution $\boldsymbol{x}'$ obtained from $\boldsymbol{x}$ by swapping $(x_{i_1 j_1} = 1,\ x_{i_2 j_2} = 1)$ into $(x_{i_1 j_2} = 1,\ x_{i_2 j_1} = 1)$ also satisfies constraint (19). Therefore, there may be a good chance of decreasing the penalty value by a swap operation. However, such a swap may be difficult to generate if we apply two moves $x_{i_1 j_2} \leftarrow 1$ and $x_{i_2 j_1} \leftarrow 1$ sequentially since the solution $\boldsymbol{x}''$ obtained from $\boldsymbol{x}$ by a single move $x_{i_1 j_2} \leftarrow 1$ or $x_{i_2 j_1} \leftarrow 1$ does not satisfy constraint (19) and hence is likely to increase the penalty value. To avoid this phenomenon, we define the *swap neighborhood*

$$N_{\mathrm{swap}}(\boldsymbol{x}) = \left\{ \boldsymbol{x}(x_{i_1 j_2} \leftarrow 1,\ x_{i_2 j_1} \leftarrow 1) \;\middle|\; \begin{array}{l} x_{i_1 j_1} = 1,\ x_{i_2 j_2} = 1, \\ i_1 \neq i_2,\ j_1 \neq j_2, \\ i_1, i_2 \in V',\ j_1, j_2 \in D \end{array} \right\} \tag{20}$$

and use the enlarged neighborhood $N(\boldsymbol{x}) \cup N_{\mathrm{swap}}(\boldsymbol{x})$ in each iteration of tabu search.

Although this enlarged neighborhood can be more effective in finding improved solutions, the time required for one iteration may increase considerably, since the size of $N_{\mathrm{swap}}(\boldsymbol{x})$ is $\mathrm{O}(|V'|^2)$, while the size of $N(\boldsymbol{x})$ is only $\mathrm{O}(d|V|)$. In actual implementation, therefore, we try to look at only a portion of $N_{\mathrm{swap}}(\boldsymbol{x})$ by employing the following rules (i) and (ii) in each iteration from $\boldsymbol{x}^{(k)}$: (i) We first check $N(\boldsymbol{x}^{(k)})$, and, only if the best move $x_{i^* j^*} \leftarrow 1$ in $N(\boldsymbol{x}^{(k)}) \setminus T$ does not improve the penalty q of Eq. (15), we move to $N_{\mathrm{swap}}(\boldsymbol{x}^{(k)})$ and check its possibility of improvement. (ii) In checking $N_{\mathrm{swap}}(\boldsymbol{x}^{(k)})$, we choose the first improved solution as $\boldsymbol{x}^{(k+1)}$ and suppress further search process. If these rules (i) and (ii) are still not sufficient to keep the computation time spent on $N_{\mathrm{swap}}(\boldsymbol{x})$ within an acceptable amount, we may restrict the use of $N_{\mathrm{swap}}(\boldsymbol{x})$ further; e.g., by em-

Table 7
Computational results for SC instances $\mathrm{STS}_{81}, \mathrm{STS}_{135}$ and STS_{243}

	Our best/best in [23]	CPU time and number of iterations		
		Min	Average	Max
STS_{81}	61/61	0.1	0.3	1.0
		14.0	204.1	788.0
STS_{135}	104/104	0.8	4.9	19.2
		215.0	1796.7	7233.0
STS_{243}	198/202	2.0	23.7	95.6
		117.0	3114.3	12983.0

ploying a smaller subset V' in Eq. (20), by restricting i_1 and i_2 in Eq. (20) only to those related to the constraints that are currently violated by $\boldsymbol{x}^{(k)}$, and by resorting to $N_{\text{swap}}(\boldsymbol{x})$ only if no improvement has been observed in a fixed number of recent iterations.

In using the swap neighborhood, the following modifications of the rules are needed to define $H^{(k)}$ and $A^{(k)}$ of Section 4.1 and aspiration criteria of Section 4.2.

- We implement a swap $(x_{i_1j_2} \leftarrow 1, x_{i_2j_1} \leftarrow 1)$ by two successive moves $x_{i_1j_2} \leftarrow 1$ and $x_{i_2j_1} \leftarrow 1$. $H^{(k)}$ and $A^{(k)}$ are updated by these moves in the same manner as in Section 4.1.
- If at least one of the two moves $x_{i_1j_2} \leftarrow 1$ and $x_{i_2j_1} \leftarrow 1$ is prohibited by $H^{(k)}$, we interpret that the swap $(x_{i_1j_2} \leftarrow 1, x_{i_2j_1} \leftarrow 1)$ is also prohibited.
- In marking attributes, as described in Section 4.1, if the move is a swap operation, then the two attributes of these successive moves are marked as a pair.
- The aspiration criterion of Section 4.2 is modified as follows for a swap operation $(x_{i_1j_2} \leftarrow 1, x_{i_2j_1} \leftarrow 1)$. Assume that $a^{(k')}$ and $a^{(k'')}$ denote the attributes of two moves $x_{i_1j_2} \leftarrow 1$ and $x_{i_2j_1} \leftarrow 1$, respectively, where $k', k'' < k$, and at least one of them is in $H^{(k)}$. Let $\boldsymbol{x}' = \boldsymbol{x}^{(k)}(x_{i_1j_2} \leftarrow 1,\ x_{i_2j_1} \leftarrow 1)$. Then the aspiration criterion is satisfied if all of $a^{(k')}$ and $a^{(k'')}$, which are in $H^{(k)}$, satisfy the condition in Eq. (10).

6.2. Computational experiments with the swap neighborhood

6.2.1. Generalized assignment problem

Let us consider the GAP again. Table 8 shows the results with two types of neighborhoods for two types of problem instances ($n = 48$, $d = 8$) and ($n = 60$, $d = 10$), as described in Section 5. (The first one is chosen as a rather difficult instance, while the latter one is a representative of large scale instances.) The table gives success rates (out of 10 runs) after 5, 10, 15, 30, 45 and 60 s of computation, respectively, where program parameters are set to $\theta = 0.5$, $LB = 0.6$, $UB = 0.8$, $\sigma = 3.0$, and $w_o^{(0)} = 1$. Roughly speaking, one iteration with $N(\boldsymbol{x}) \cup N_{\text{swap}}(\boldsymbol{x})$ consumes 3–7 times more computational time than that with $N(\boldsymbol{x})$. Therefore, the search with $N(\boldsymbol{x})$ gives better performance in the early stage (e.g., 5–10 s). As the search proceeds, however, the performance of $N(\boldsymbol{x}) \cup N_{\text{swap}}(\boldsymbol{x})$ becomes better than that of using $N(\boldsymbol{x})$ only. Therefore, the use of the enlarged neighborhood is recommended if enough computational time is allowed.

To conclude the computational experiment on the GAP, we compare in Table 9 the performance of our algorithm (CSP) with two other heuristic algorithms, TS1 due to Osman [26] and GA$_b$ due to Chu–Beasley [7], respectively. As described in Section 5.2, there are five problem instances for each problem size defined by n and d; hence 60 instances in total. Our tabu search CSP used here is the

Table 8
Effect of the swap neighborhood on GAP instances

		$N(x)$						$N(x) \cup N_{\text{swap}}(x)$r>					
		5[s]	10[s]	15[s]	30[s]	45[s]	60[s]	5[s]	10[s]	15[s]	30[s]	45[s]	60[s]
$n = 48$	Instance 1	0/10	0/10	0/10	0/10	2/10	2/10	2/10	3/10	3/10	4/10	6/10	9/10
$d = 8$	Instance 2	4/10	7/10	10/10	10/10	10/10	10/10	7/10	10/10	10/10	10/10	10/10	10/10
	Instance 3	0/10	1/10	1/10	2/10	2/10	2/10	0/10	2/10	3/10	6/10	8/10	8/10
	Instance 4	0/10	0/10	1/10	2/10	2/10	4/10	2/10	3/10	5/10	6/10	9/10	10/10
	Instance 5	1/10	1/10	1/10	6/10	9/10	10/10	6/10	7/10	8/10	10/10	10/10	10/10
$n = 60$	Instance 1	10/10	10/10	10/10	10/10	10/10	10/10	8/10	8/10	8/10	10/10	10/10	10/10
$d = 10$	Instance 2	9/10	10/10	10/10	10/10	10/10	10/10	7/10	10/10	10/10	10/10	10/10	10/10
	Instance 3	10/10	10/10	10/10	10/10	10/10	10/10	6/10	8/10	10/10	10/10	10/10	10/10
	Instance 4	6/10	8/10	10/10	10/10	10/10	10/10	2/10	5/10	6/10	10/10	10/10	10/10
	Instance 5	9/10	9/10	9/10	10/10	10/10	10/10	1/10	6/10	6/10	9/10	10/10	10/10

Table 9
Comparison of our tabu search with other heuristics for GAP

Problem size	TS1 [a] (%)	GA$_b$ [b] (%)	CSP [c]		
				Success rate	CPU time
$n = 15$, $d = 5$	0.00	0.00	0.000	50/50	0.07
$n = 20$, $d = 5$	0.10	0.00	0.000	50/50	0.07
$n = 25$, $d = 5$	0.00	0.00	0.000	50/50	0.13
$n = 30$, $d = 5$	0.03	0.00	0.000	50/50	0.69
$n = 24$, $d = 8$	0.00	0.00	0.000	50/50	0.57
$n = 32$, $d = 8$	0.03	0.01	0.000	50/50	0.93
$n = 40$, $d = 8$	0.00	0.00	0.000	50/50	1.79
$n = 48$, $d = 8$	0.09	0.05	0.005	47/50	19.23
$n = 30$, $d = 10$	0.06	0.00	0.000	50/50	4.42
$n = 40$, $d = 10$	0.08	0.04	0.004	48/50	9.13
$n = 50$, $d = 10$	0.02	0.00	0.000	50/50	5.03
$n = 60$, $d = 10$	0.04	0.01	0.000	50/50	8.58
The number of instances whose optimal solutions are found	45/60	60/60	60/60		

[a] Tabu search using long term memory with best-admissible selection [26].

[b] GA with heuristic operator involving two local improvement steps [7].

[c] Our value-based adaptive method with swap neighborhood and the control mechanism of w_0.

value-based adaptive method equipped with the control mechanism of w_0 (in Section 5) and the swap neighborhood (in this section). For each problem instance, 10 runs are repeated, where each run is cut off either if it finds an optimal solution or exceeds 60 s. Table 9 shows the average percentage deviation from the optimal values. It also shows in bottom row the number of problem instances (out of 60) whose optimal solutions are found. In the columns of our algorithms, the number of success runs (out of 50) and the average CPU time (in seconds) in one run are also given, for each problem size. From these results, our CSP algorithm is considered slower than TS1, and faster than GA$_b$, but gives better solutions than either of them. We may conclude that our algorithm is at least competitive with these existing ones specially designed for the GAP.

6.2.2. *University timetable*

Consider the problem of constructing a timetable at a university, as described in [25]. This example consists of 30 lectures, 13 professors, 60 students, 10 periods $T_1, T_2, \ldots, T_{10}$, and three rooms R_1, R_2 and R_3. As shown in Table 10, each student selects 8–10 lectures (those marked by circles) which he/she wants to take, out of 30 lectures. Therefore, all the lectures selected by a student must be given in different periods (constraint 1). Each professor gives two or three lectures specified in Table 11, which must be given in different periods (constraint 2) and has inconvenient periods, which are also indicated in Table 11 (constraint 3). Furthermore, since the numbers of seats in rooms R_1, R_2, R_3 are 20, 40, and 60, respectively, some lectures cannot be held in small rooms, as described in Table 12 (constraint 4). Of course, each room is assigned to at most one lecture in each period (constraint 5).

This problem can be easily formulated as a CSP instance by introducing $n = 30$ variables (corresponding to lectures), with domain size $d = 10$ (the number of periods) × 3 (the number of rooms). All the constraints are represented by 774 linear inequalities.

By a preliminary analysis, it becomes obvious that there is no solution satisfying all constraints. Therefore, we classify the constraints into those which must be satisfied so that the obtained

Table 10
Selection list of lectures by students

Students	Lectures																													
	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	O	O	O	—	O	—	O	—	O	—	—	—	—	—	—	O	—	—	O	—	—	—	—	—	—	—	O	—	—	—
2	O	O	O	O	—	—	O	O	O	—	—	O	O	—	—	—	—	—	—	—	—	—	—	—	—	—	—	O	—	—
3	O	—	O	O	O	—	—	O	—	—	O	—	—	O	—	O	—	—	O	—	—	—	—	—	—	—	—	O	—	—
4	—	O	—	O	O	—	—	O	O	—	O	—	O	O	—	O	—	—	O	—	—	—	—	—	—	—	—	—	—	—
5	—	O	O	O	O	—	O	—	O	—	—	—	O	—	—	—	—	—	—	—	—	—	—	—	—	—	O	O	—	—
6	—	—	—	O	O	—	O	O	O	—	O	O	O	—	—	—	—	—	—	—	—	—	—	—	—	—	O	O	—	—
7	—	—	O	—	O	—	—	O	O	—	O	—	O	O	—	O	—	—	—	—	—	—	—	—	—	—	O	O	—	—
8	O	—	—	O	O	—	O	—	—	—	—	O	—	O	—	O	—	—	O	—	—	—	—	—	—	—	—	O	—	—
9	—	—	O	O	—	—	O	—	O	—	O	O	O	O	—	O	—	—	O	—	—	—	—	—	—	—	—	—	—	—
10	—	O	—	O	O	—	—	O	—	—	—	—	O	O	—	O	—	—	O	—	—	—	—	—	—	—	—	O	—	—
11	O	—	—	O	—	O	O	—	O	O	O	—	O	—	—	O	—	—	—	—	—	—	—	—	—	—	—	O	—	—
12	—	—	O	O	—	—	O	—	O	O	O	—	O	—	O	O	—	—	—	—	—	—	—	—	—	—	—	O	—	—
13	—	—	O	—	—	O	O	—	O	—	—	—	—	—	O	O	O	—	O	—	—	—	—	—	—	—	O	O	—	—
14	—	—	O	—	—	O	—	—	—	O	O	—	O	—	—	O	O	—	O	—	—	—	—	—	—	—	O	O	—	—
15	—	—	—	O	O	—	O	—	—	—	O	—	O	—	O	—	—	—	O	—	—	—	—	—	—	—	O	O	—	—
16	—	—	O	—	—	—	O	—	—	O	O	—	O	O	—	O	—	—	O	—	—	—	—	—	—	—	O	O	—	—
17	O	—	—	O	O	O	O	—	O	—	—	—	O	—	O	O	—	—	—	—	—	—	—	—	—	—	—	O	—	—
18	—	—	—	O	O	O	—	—	O	O	—	—	O	—	O	O	—	—	—	—	—	—	—	—	—	—	O	O	—	—
19	—	—	O	—	O	O	O	—	—	O	O	—	O	—	O	O	—	—	—	—	—	—	—	—	—	—	—	O	—	—
20	O	—	O	—	—	O	—	—	—	O	O	—	O	—	O	—	O	—	O	—	—	—	—	—	—	—	—	O	—	—
21	—	—	—	O	—	—	—	O	—	O	—	O	—	O	O	—	—	—	—	—	—	—	—	—	—	—	—	O	O	—
22	O	—	—	O	O	—	—	—	—	—	O	—	—	—	O	—	O	—	O	—	—	—	—	—	—	—	—	O	O	—
23	—	—	—	O	—	O	—	O	—	O	O	O	—	O	—	—	—	—	—	—	—	—	—	—	—	—	O	O	O	—
24	—	—	O	O	—	—	O	—	—	O	O	—	—	O	O	—	O	—	—	—	—	—	—	—	—	—	O	—	O	—
25	—	—	—	—	O	—	O	O	—	O	—	—	—	O	O	—	O	—	—	—	—	—	—	—	—	—	—	O	—	—
26	—	—	O	—	—	O	—	O	O	—	O	—	—	—	O	—	O	—	O	—	—	—	—	—	—	—	O	—	O	—
27	—	—	O	—	—	—	—	O	O	—	—	O	—	O	—	—	O	—	O	—	—	—	—	—	—	—	O	O	O	—
28	—	—	O	—	O	O	—	—	—	—	—	O	—	O	O	O	—	—	O	—	—	—	—	—	—	—	—	O	—	—
29	O	—	—	—	O	O	—	O	—	O	O	O	—	—	O	—	O	—	O	—	—	—	—	—	—	—	—	—	—	—
30	—	—	—	—	O	—	O	—	O	—	—	O	—	O	—	—	O	—	—	—	—	—	—	—	—	—	O	—	O	—
31	—	—	O	—	O	—	O	—	—	O	O	—	—	—	—	—	O	O	—	O	—	—	—	—	—	—	—	O	O	—
32	—	—	O	—	O	—	—	—	—	O	—	—	—	—	—	—	O	—	O	O	O	O	—	—	—	—	—	—	O	—
33	—	—	—	O	—	—	O	—	—	—	O	—	—	—	—	O	O	—	O	—	O	—	O	—	—	—	—	O	O	—
34	—	—	O	—	—	—	O	—	—	O	—	—	—	—	—	—	O	—	O	O	—	O	O	—	—	—	—	O	O	—
35	—	—	O	O	—	—	—	—	—	O	O	—	—	—	—	—	O	O	O	—	—	O	—	—	—	—	—	O	O	—
36	—	—	—	O	O	—	—	—	—	—	O	—	—	—	—	O	O	—	O	—	O	O	—	—	—	—	—	O	O	—
37	—	—	—	—	O	—	O	—	—	O	O	—	—	—	—	—	O	—	—	O	—	O	O	—	—	—	—	—	O	—
38	—	—	—	O	—	—	O	—	—	—	O	—	—	—	—	O	O	—	O	O	O	—	—	—	—	—	—	O	O	—
39	—	—	—	—	O	—	O	—	—	—	O	—	—	—	—	—	O	—	O	O	—	O	O	—	—	—	—	O	O	—
40	—	—	—	O	—	—	O	—	—	O	—	—	—	—	—	—	—	—	O	—	O	O	O	—	—	—	—	O	O	—
41	—	—	—	O	O	O	—	—	—	O	—	—	—	—	O	—	O	—	—	—	—	O	—	—	O	—	—	—	—	O
42	—	—	—	—	O	—	O	—	—	O	—	—	—	—	O	—	—	O	—	—	—	O	O	—	O	—	—	O	—	O
43	—	—	—	O	—	—	O	—	—	—	—	—	—	—	O	—	—	O	—	O	—	O	—	O	O	—	—	O	—	O
44	—	—	—	O	—	—	O	—	—	—	—	—	—	—	—	—	O	O	—	O	—	—	O	—	O	—	—	—	—	O
45	—	—	—	O	—	—	—	—	—	—	—	—	—	—	—	—	O	O	—	—	O	O	O	—	O	O	—	O	—	O
46	—	—	—	—	O	O	O	—	—	O	—	—	—	—	—	—	—	—	—	—	—	O	—	O	O	O	—	O	—	O
47	—	—	—	—	O	—	—	—	—	—	—	—	—	—	O	—	O	O	—	O	—	O	—	—	O	O	—	O	—	O
48	—	—	—	—	—	—	O	—	—	O	—	—	—	—	—	—	O	—	—	O	O	—	—	O	O	O	—	O	—	O
49	—	—	—	—	O	O	—	—	—	—	—	—	—	—	—	—	O	O	—	O	—	O	—	—	O	—	—	O	—	O
50	—	—	—	—	—	O	O	—	—	O	—	—	—	—	O	—	—	O	—	—	—	O	—	O	O	O	—	O	—	—
51	—	—	—	—	O	—	O	—	—	O	—	—	—	—	—	—	O	O	—	O	—	O	—	O	O	—	—	—	—	O
52	—	—	—	—	O	O	—	—	—	—	—	—	—	—	—	—	—	O	O	O	—	O	O	O	O	—	—	O	—	—
53	—	—	—	—	O	O	—	—	—	O	—	—	—	—	—	—	—	—	O	O	O	—	O	—	—	O	—	O	—	O
54	—	—	—	—	O	—	O	—	—	—	—	—	—	—	—	—	O	—	—	—	O	—	O	—	O	—	—	O	—	O
55	—	—	—	—	O	O	—	—	—	O	—	—	—	—	—	—	—	O	—	O	—	—	O	—	—	O	—	O	—	O
56	—	—	—	—	—	—	O	—	—	O	—	—	—	—	—	—	O	O	—	O	—	O	O	O	O	O	—	—	—	—
57	—	—	—	—	—	O	—	—	—	—	—	—	—	—	—	—	O	O	—	—	O	—	O	—	—	O	—	O	—	O
58	—	—	—	—	—	—	O	—	—	O	—	—	—	—	—	—	—	O	—	—	O	—	—	O	O	O	—	O	—	O
59	—	—	—	—	—	O	—	—	—	O	—	—	—	—	—	—	O	—	O	—	O	—	—	O	—	—	—	O	—	O
60	—	—	—	—	—	O	O	—	—	O	—	—	—	—	—	—	O	—	—	O	—	O	—	O	O	—	—	O	—	O

Table 11
Lectures and inconvenient periods of professors

Professors	Lectures			Inconvenient periods		
1	1	7		1	3	8
2	2	8		2	3	5
3	3	13	15	1	4	9
4	4	16	27	2	5	10
5	5	28		7	9	
6	6	14		3	6	10
7	9	12		1	2	5
8	10	24	29	6	7	
9	11	17	19	4	8	10
10	18	20		1	6	8
11	21	26		3	7	
12	22	30		4	5	6
13	23	25		2	8	9

Table 12
The rooms in which lectures cannot be held

Lecture	4	5	6	7	10	11	17	19	28
Room	1	1	1	1	1	1	1	1	1, 2

solution is meaningful as a timetable (i.e., constraints 2, 4 and 5; *hard constraints*) and the rest (constraints 1 and 3; *soft constraints*). We then try to find a timetable that satisfies all hard constraints and minimizes the number of conflicts in soft constraints. The distinction between hard and soft constraints is realized by setting the weights w_h in Eq. (6) corresponding to hard constraints (resp., soft constraints) as 100 (resp., 1).

For two neighborhoods, $N(\boldsymbol{x})$ and $N(\boldsymbol{x}) \cup N_{\mathrm{swap}}(\boldsymbol{x})$, this instance was tested 10 times respectively, consuming 60 s in each run. Table 13 shows the results by value-based adaptive method for two types of neighborhoods, that is, the average penalty values of 10 incumbent solutions, and the content of the best solutions in 10 runs; the number of students having the stated number of conflicted lectures, the total number of inconvenient periods assigned to professors, and the total penalty. Table 14 is an example of the best timetable found by our algorithms. The performance of the search is clearly improved by the incorporation of the swap neighborhood.

In [25] from which this instance is taken, a local search algorithm is applied after formulating this as a SAT problem. The best timetable found therein has penalty value 94, compared to our value of 84, which may clearly indicate a superiority of our approach as a tool to find solutions with a small number of conflicts.

7. 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 collect-

Table 13
Computational results for the university timetable

Neighborhood	Average penalty	Best solution out of 10 trials						
		Number of students with conflicts					Number of conflicts for Professors	Penalty
		0	1	2	3	More than 4		
$N(x)$	88.5	6	27	23	4	0	2	87
$N(x) \cup N_{\mathrm{swap}}(x)$	84.8	7	29	19	5	0	2	84

Table 14
An example of computed timetable (penalty = 84)

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
R_1	26	30	9	15	23	3	8	27	16	12
R_2	4	13	19	20	11	21	6	1	18	2
R_3	17	29	25	14	24	5	7	22	10	28

ing 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.

7.1. High school timetable

The data [31] are taken from a real high school located in the suburb of Tokyo. This high school has 30 classes, 60 teachers, 30 classrooms and additional three 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 (six periods a day from Monday through Friday, and four periods on Saturday morning) under various constraints to be described shortly. To formulate this problem as a CSP, we introduce $n = 780 + 13 = 793$ variables X_i and domain $D = \{1, 2, \ldots, 34\}$ corresponding to the periods in a week; i.e., there are $793 \times 34 = 26962$ value–variables x_{ij}. There is no objective function.

There are a large number of constraints to be satisfied. Some of them are common to all timetable 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 12 747 linear inequalities.

We ran our tabu search algorithm 30 times, each with 300 s as an upper bound on computational time, and could find solutions that satisfy all constraints except one, on most runs. By analyzing the violated constraint and related ones, we could show that it is not possible to satisfy all constraints. Hence the solutions found by our tabu search turned out to be the best possible.

Table 15 shows the success rate (out of 30), the average and maximum incumbent penalty, and the average length of t, for variable-based and value-based adaptive methods, respectively. As mentioned in Section 4, the variable-based algorithm appears to bring better results for large scale instances like this one.

There are a number of papers on timetabling such as [8,18,28,31]. In addition to [31], from which our instance is taken, [8] also contains computational results on large scale instances with

Table 15
Results for timetabling

	Success rate	Penalty (1 is optimal)		Average of t
		Average	Max	
Variable-based	26/30	1.3	4	9.3
Value-based	18/30	1.9	8	12.2

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 at least as good as these special-purpose algorithms in [8,31].

7.2. Nurse scheduling

The nurse scheduling is also a very difficult problem discussed in various papers (e.g., [19,27]). Our example, which is based on real data from a hospital in Tokyo, has 25 nurses, where each nurse must be assigned every day one of the five alternatives: day shift, evening shift, night shift, scheduled meeting and day off. It is reported in [19] that the nurse scheduling in hospitals in Japan has some special features, such as every nurse can be assigned to any shift, the schedule changes every day, and the frequent pairing of the same nurses is considered undesirable. These may make the scheduling more difficult.

For a schedule of 25 nurses on 30 days (one month), the 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, six in A and five 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 four nurses are leaders, an evening shift requires four nurses including two leaders, and a night shift requires three nurses including two 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) three successive night shifts, (ii) four successive evening shifts, (iii) five 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 six days.

The resulting formulation has 9731 inequalities to describe the foregoing constraints and others. It turns out, however, that there are some conflicting constraints which make a solution satisfying all constraints impossible. Therefore, we are asked to achieve a certain compromise among constraints, by taking into account the fact that some constraints are not very rigid (soft constraints). 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 other hard constraints are satisfied. To achieve this, we introduced weights w_h of Eq. (6) for all inequalities and manually controlled them so that the target can be achieved smoothly.

We ran our tabu search algorithms 30 times, each consuming 300 s. The results for the variable-based and value-based methods are presented in Table 16, which shows the minimum, average, and maximum incumbent penalty values and the average of tabu tenure t. As in the instance of timetabling described above, the variable-based ap-

Table 16
Results for nurse scheduling

	Penalty			Average of t
	Min	Average	Max	
Variable-based	0.8	15.9	106.7	13.6
Value-based	2.1	43.7	125.1	7.9

proach is more effective. Although the preliminary experiment is necessary to obtain appropriate weights w_h given to constraints, some solutions which have only conflicts in soft constraints (meaning that these solutions can be used as practical schedulings) are found. An example is illustrated in Fig. 3. In this schedule, there are some conflicts in soft constraints; the numbers of leaders on day shifts are not sufficient for seven days, the numbers of nurses on day shifts are not sufficient for one day.

8. Conclusion and discussion

We developed tabu search algorithms by employing the CSP as a general purpose vehicle to formulate problems, in which further elaborations are added such as an automatic control mechanism for the tabu tenure t, incorporating an objective function with an automatic control mechanism based on the weight w_0, and enlarging the neighborhood by allowing swap operations. Although our code is aimed at being general pur-

		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
A	1*	/	–	–	/	≡	≡	/	–	–	/	–	=	/	/	–	=	=	≡	≡	/	/	–	–	=	=	=	/	–	≡	≡
	2*	≡	/	/	–	–	–	–	=	/	≡	≡	/	+	–	–	/	/	/	–	=	=	≡	≡	/	–	–	=	=	=	/
	3*	/	/	–	–	=	=	/	≡	≡	/	–	–	/	–	=	≡	≡	/	–	–	/	/	=	≡	≡	/	–	–	–	–
	4*	–	–	=	=	/	≡	≡	/	–	–	/	/	=	=	/	–	–	=	/	+	–	–	–	–	/	/	≡	≡	/	–
	5*	–	≡	≡	/	–	–	=	=	/	–	=	≡	≡	/	/	–	–	–	–	≡	≡	/	/	/	/	–	–	/	–	=
	6*	=	=	/	≡	≡	/	–	–	=	=	/	–	–	≡	≡	/	/	–	=	/	–	=	/	–	–	≡	≡	/	/	–
	7	/	≡	≡	/	–	–	=	/	=	=	=	/	–	–	/	–	–	/	–	–	–	/	–	=	≡	≡	/	–	–	/
	8	=	=	/	/	–	–	/	–	–	+	–	=	/	–	–	–	=	≡	≡	/	–	–	–	–	/	/	–	=	/	/
	9	/	–	–	–	–	=	/	–	–	–	/	/	/	≡	≡	/	–	–	/	–	=	=	/	–	–	–	=	/	–	–
	10	+	/	–	–	/	/	–	≡	≡	/	–	–	–	/	/	+	–	=	=	=	/	–	–	–	=	=	/	≡	≡	/
	11	–	–	=	=	=	/	–	–	/	/	–	–	/	–	–	+	/	–	–	/	/	–	–	/	–	–	=	≡	≡	/
	12	≡	/	/	–	–	–	/	/	/	–	–	–	=	=	/	+	–	/	–	–	–	/	–	–	–	–	/	–	–	≡
	13	–	–	≡	≡	/	–	+	–	/	/	/	–	–	–	=	=	/	–	≡	≡	/	–	=	/	/	/	–	–	=	=
B	14*	=	/	–	=	≡	≡	/	/	–	+	/	/	≡	≡	/	–	–	–	=	=	=	/	–	=	≡	≡	/	–	–	/
	15*	–	=	=	/	–	–	=	/	–	=	≡	≡	/	–	–	–	/	/	/	–	–	=	≡	≡	/	–	–	/	/	–
	16*	≡	≡	/	–	=	/	/	–	≡	≡	/	–	+	/	–	=	≡	≡	/	/	+	–	–	/	=	=	=	/	–	–
	17*	/	/	–	–	–	=	≡	≡	/	–	=	/	+	–	≡	≡	/	–	–	/	/	–	=	/	–	=	≡	≡	/	–
	18*	/	–	≡	≡	/	/	–	–	–	/	/	–	–	/	=	=	=	/	–	+	≡	≡	/	–	–	/	–	=	=	=
	19	–	–	–	–	/	/	+	–	–	–	–	=	/	/	–	+	–	=	/	–	≡	≡	/	/	/	/	–	=	=	=
	20	/	/	–	–	=	/	–	=	/	–	–	≡	≡	/	–	–	=	/	/	–	=	/	–	–	–	–	/	–	–	+
	21	/	–	–	/	/	–	=	/	=	=	=	/	–	–	–	≡	≡	/	–	–	–	/	–	–	–	–	=	/	/	+
	22	+	=	=	/	–	–	–	–	/	≡	≡	/	–	–	/	/	–	–	/	/	–	–	=	=	=	/	–	–	/	+
	23	–	–	–	=	/	–	≡	≡	/	/	–	–	/	=	=	/	/	–	=	=	/	–	–	/	–	–	–	/	–	+
	24	=	/	/	–	–	=	/	/	–	–	–	–	=	/	–	–	/	–	–	/	–	=	≡	≡	/	/	–	–	–	–
	25	–	–	/	/	/	–	–	=	=	/	–	=	=	=	/	/	–	=	≡	≡	/	/	–	–	–	–	/	–	≡	≡
total	–	8	10	10	10	10	11	8	10	9	8	11	10	7	10	10	8	10	10	10	8	9	10	13	10	11	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	4	4	4	4	4	4	5	4	4	4
	≡	3	3	4	3	3	3	3	4	3	3	3	3	3	3	3	3	3	3	4	3	3	3	3	3	3	3	3	4	4	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	7	8	8	7	8	7	9	8	7	8	8	8	8	6	8	8	7	8	8	8	5	8	7	8	7	7	7	6

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

*: leader

Fig. 3. A schedule of 25 nurses on 30 days.

pose, the computational results for the problems chosen from a wide variety of applications disclose that it is competitive with special-purpose algorithms developed in the respective problem domains.

Our code contains a number of parameters in order to attain enough versatility for a wide range of applications. We summarize below how these parameters are being handled.

- t (tabu tenure): Automatically controlled.
- w_0 (weight of the objective function): Automatically controlled by other parameters [LB,UB] and σ. σ may always be fixed to 3.0, as the performance of our code is not sensitive to it. However, the range [LB,UB] may be determined by taking into account the difficulty of obtaining feasible solutions during search; e.g., $[0.6, 0.8]$ if it is rather hard to obtain feasible solutions, and $[0.4, 0.6]$ if it is easy.
- w_h (weights of constraints): Usually all weights are set to $w_h = 1$. However, higher performance can be expected if these are adjusted in accordance with importance of the constraints. In case "hard" constraints and "soft" constraints are specified by the user (as in the problems of university timetable and nurse scheduling), hard constraints should get much larger w_h values (e.g., 100 times larger) than those for soft constraints.
- Neighborhood: The default is the shift neighborhood. If the problem being solved is rather hard, incorporation of a swap neighborhood appears to be effective.
- value-based or variable-based (representation of tabu list): Default setting is variable-based. The value-based approach may be recommended if the size of a given problem instance is rather small, and hence t is large relative to problem size (e.g., if t is reset to 1 for the reason that T includes all the neighbors of the current solution $\boldsymbol{x}^{(k)}$).

As our project is still on-going, however, further elaborations are necessary to increase its power and applicability. Toward this goal, the following points may deserve consideration:

1. Although we attempted to automatically adjust program parameters t and w_0, further efforts may be necessary to control the remaining weights w_h automatically.
2. At the present stage, the diversification is enforced only by repeating the search from different initial solutions. For this purpose, more sophisticated elements of tabu search, such as strategic oscillation, path relinking and target analysis, may be worthwhile to include. However, their effectiveness sometimes appears to be sensitive to the types of problems being solved. It is an interesting challenge to investigate their roles in general purpose solvers like our code.
3. Incorporation of neighborhoods other than $N(\boldsymbol{x})$ and $N_{\text{swap}}(\boldsymbol{x})$ may also be useful. Flexible choices of neighborhood structures will enlarge the applicability of our algorithm, but at the same time will necessitate establishment of a rule of choosing an appropriate neighborhood.
4. Most of the problems tested in this paper are combinatorial in the sense that all the coefficients a_{hij} in constraints Eq. (4) are 0 or ± 1. There is an indication (e.g., see the discussion at the end of Section 5.2) that our algorithm may not be very effective for those problems with general a_{hij}. To overcome this, it appears necessary to exploit mathematical programming techniques (e.g., linear programming) to capture global structural information of the entire feasible region and to use it in the search process.
5. To deal with objective functions, there are methods other than those described in Section 5, such as introducing the ratio-based penalty function [21]. These methods should be tested and evaluated.
6. As observed in timetabling and nurse scheduling in Sections 6 and 7, two types of constraints, hard and soft, are common in real applications. In this paper, we dealt with this issue by tuning the weights w_h given to such constraint. However, as the performance of the search highly depends on these weights, it is important to investigate how to determine them without spending too much time on preliminary experiment.
7. Currently, it is the users' responsibility to determine how to formulate the given problem (including how to describe the constraints, i.e.,

either by inequalities or by other means). It might be necessary to establish a guideline that helps users to choose appropriate formulations.

8. To make the system easily accessible for many users, it should be equipped with a friendly interface so that the user can easily input his/her problem instances in the CSP form and execute the CSP algorithm effectively. Development of such an interface is an important future project.

Acknowledgements

We appreciate C. Mannino of University of Rome La Sapienza for the data of the set covering problems in Table 7, K. Iwama of Kyoto University for the data of university timetable in Section 6, M. Watanabe and K. Kaneko of C and C Research Laboratories, NEC Corporation for the data of high school timetable in Section 7, and A. Ikegami of Seikei University for the data of nurse scheduling also in Section 7. We also wish to thank the two anonymous referees for their valuable comments and suggestions. This research was partially supported by the Scientific Grant-in-Aid by the Ministry of Education, Science, Sports and Culture of Japan. The first author is supported by Research Fellowships of the Japan Society for the Promotion of Science for Young Scientists.

References

[1] D. Avis, A note on some computationally difficult set covering problems, Mathematical Programming 18 (1980) 138–145.

[2] E. Balas, A. Ho, Set covering algorithms using cutting planes, heuristics, and subgradient optimization: A computational study, Mathematical Programming 12 (1980) 37–60.

[3] R. Battiti, G. Tecchiolli, The reactive tabu search, ORSA Journal on Computing 6 (1994) 126–140.

[4] J.E. Beasley, An algorithm for set covering problem, European Journal of Operational Research 31 (1987) 85–93.

[5] D.G. Cattrysse, L.N. Van Wassenhove, A survey of algorithms for the generalized assignment problem, European Journal of Operational Research 60 (1992) 260–272.

[6] D.G. Cattrysse, M. Salomon, L.N. Van Wassenhove, A set partitioning heuristic for the generalized assignment problem, European Journal of Operational Research 72 (1994) 167–174.

[7] P.C. Chu, J.E. Beasley, A genetic algorithm for the generalised assignment problem, Working paper, The Management School, Imperial College, 1995.

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

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

[10] T.A. Feo, K. Venkatraman, J.F. Bard, A GRASP for a difficult single machine scheduling problem, Computers and Operations Research 18 (1991) 635–643.

[11] C. Fleurent, J.A. Ferland, Genetic and hybrid algorithms for graph coloring, in: G. Laporte, I.H. Osman, P.L. Hammer (Eds.), Metaheuristics in Combinatorial Optimization, Annals of Operations Research (1996).

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

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

[14] D.R. Fulkerson, G.L. Nemhauser, L.E. Trotter, Two computationally difficult set covering problems that arise in computing the 1-width of incidence matrices of Steiner triple systems, Mathematical Programming Study 2 (1974) 72–81.

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

[16] F. Glover, Tabu search fundamentals and uses, Technical Report, University of Colorado, 1995.

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

[18] A. Hertz, Tabu search for large scale timetabling problems, European Journal of Operational Research 54 (1991) 39–47.

[19] A. Ikegami, A. Niwa, M. Ohkura, Nurse scheduling problem in Japan, Communications of the Operations Research Society of Japan 41 (1996) 436–442 in Japanese.

[20] M. Laguna, T.A. Feo, H.C. Elrod, A greedy randomized adaptive search procedure for the two-partition problem, Operations Research 42 (1994) 677–687.

[21] A. Løkketangen, F. Glover, Probabilistic move selection in tabu search for zero-one mixed integer programming problems, in: I.H. Osman, J.P. Kelly (Eds.), Meta-Heuristics: Theory and Applications, Kluwer Academic Publishers, Boston, Ch. 28, 1996, pp. 467–487.

[22] L.A.N. Lorena, F. Belo Lopes, A surrogate heuristic for set covering problems, European Journal of Operational Research 79 (1994) 138–150.

[23] C. Mannino, A. Sassano, Solving hard set covering problems, Operations Research Letters 18 (1995) 1–5.

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

[25] S. Miyazaki, K. Iwama, Y. Kambayashi, Database queries as combinatorial optimization problems, Proceedings of International Symposium on Cooperative Database Systems for Advanced Applications, 1996, pp. 448–454.

[26] I.H. Osman, Heuristics for the generalised assignment problem: simulated annealing and tabu search approaches, OR Spektrum 17 (1995) 211–225.

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

[28] A. Schaerf, Tabu search techniques for large high-school timetabling problems, Proceedings of the thirteenth National Conference on Artificial Intelligence (AAAI) and the eighth Innovate Applications of Artificial Intelligence (IAAI), 1996, pp. 363–368.

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

[30] E. Tsang, Foundations of Constraint Satisfaction, Academic Press, London, 1993.

[31] M. Yoshikawa, K. Kaneko, T. Yamanouchi, M. Watanabe, A constraint-based high school scheduling system, IEEE Expert 11 (1996) 63–72.