Technical Report #99022

An Improved Tabu Search Method for the Weighted Constraint Satisfaction Problem

Koji Nonobe and Toshihide Ibaraki
Department of Applied Mathematics and Physics,
Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan
{nonobe,ibaraki}@i.kyoto-u.ac.jp

August 1999
(revised in January 2000)

Abstract

Aiming at developing a general problem solver for combinatorial optimization problems, we consider in this paper the weighted constraint satisfaction problem (WCSP), which, given a number of constraints and their weights of importance, asks to minimize the total weight of unsatisfied constraints. We propose a tabu search algorithm for WCSP with the features that it uses an evaluation function, defined in terms of the modified weights of constraints, for guiding the search, and that it incorporates an automatic control mechanism of the weights in the evaluation function. Using this code, we solved a number of problems including those from real applications such as generalized assignment, set covering, parallel shop scheduling, timetabling and nurse scheduling. Many problems that arise in cellular manufacturing can also be formulated as WCSP, including the problems of cell formation and tool selection. Our computational results indicate that the control mechanism of weights makes our tabu search more powerful, and our algorithm is practically usable.

Key words: combinatorial optimization problem, weighted constraint satisfaction problem, scheduling, general problem solver, tabu search.

UNE MÉTHODE TABOU AMÉLIORÉE POUR LE PROBLÈME DE SATISFACTION DE CONTRAINTES PONDÉRÉES

Résumé

Dans le but de développer une méthode générale de résolution de problèmes d'optimisation combinatoire, nous considérons dans cet article le problème de satisfaction de contraintes pondérées (WCSP, Weighted Constraint Satisfaction Problem). Étant donné un certain nombre de contraintes et leur poids associé, ce problème demande à minimiser le poids total des contraintes insatisfaites. Nous proposons un algorithme de recherche tabou pour WCSP ayant une double particularité: d'une part, il utilise une fonction d'évaluation définie en terme du poids modifié des contraintes, afin de guider la recherche; d'autre part, il incorpore un mécanisme de contrôle automatique du poids dans la fonction d'évaluation. En utilisant ce code, nous avons résolu un certain nombre de problèmes, issus notamment d'applications réelles, tels que les problèmes d'affectation généralisée, de recouvrement d'ensembles, de gestion d'emploi du temps et de gestion des horaires des infirmières. Beaucoup de problèmes rencontrés en génie génétique se pretênt également à une formulation en terme de WCSP, comme le problème de la formation de cellules ou de la sélection d'outils. Nos résultats de calcul montrent que le mécanisme de contrôle du poids rend notre méthode tabou plus puissante, et notre algorithme est utilisable en pratique.

1 Introduction

The *constraint satisfaction problem* (CSP) seeks an assignment of values to variables such that all the given constraints are satisfied. CSP receives increasing attention as it can formulate many combinatorial problems encountered in real applications. However, the specified constraints are often so tight that problems may have no assignment satisfying all the constraints. In such cases, it is naturally required to satisfy as many important constraints as possible. The *weighted constraint satisfaction problem* (WCSP) is an enhanced framework of CSP that can deal with these over-constrained problems.

Many problems that arise in cellular manufacturing can also be formulated as WCSPs. An example is the cellular formation problem. In this problem, we are asked to decompose the set of all machines into machine cells, each of which processes the assigned part families. Although the types of constraints and objectives may vary depending on the situations, the problem can be written as a WCSP by regarding the machines as variables that take the values of the corresponding machine cells. Furthermore, WCSP can also handle some extensions; e.g., each part has several alternative process plans (routings), and it is required to select the tools to be equiped on the machines [28, 30].

We develop in this paper an efficient WCSP solver that can be used as a *general problem solver* for combinatorial optimization problems. In this approach, the problem at hand is reformulated as a WCSP and then solved by an algorithm for WCSP. The main advantage of this approach is that users do not need to develop algorithms specialized to the problem, but have only to describe them in the form of WCSP. Furthermore, our code contains automatic mechanisms of tuning the program parameters and users do not need to spend time on preconditioning the code (although it may be possible to achieve higher performance in return if users tune program parameters for themselves).

The performance of a general purpose solver may not be as high as that of specialized solvers. However, it is still worth developing as it saves the manpower and time required to develop specialized codes. In this direction, we have already proposed a tabu search algorithm with some encouraging computational results in [27]. Tabu search is a variant of the local search algorithms in which the search history is utilized to make the remaining search process more efficient [15, 16]. In this paper, we describe how the previous tabu search algorithm is modified and improved.

In addition to the classical CSP, its extended frameworks have been studied, mainly in the field of artificial intelligence. As for WCSP, Freuder and Wallace [11] considered its special case called the *maximal constraint satisfaction problem* (Max-CSP), in which all constraints have the same weight; they extended some basic backtracking and local consistency methods for CSP to Max-CSP. Then, a number of papers on WCSP have been published (e.g., *Lecture notes in Computer Science*, Springer-Verlag, *The International Conferences on Principles and Practice of Constraint Programming*, which have been held every year since 1995). Since the sizes of problems we consider in this paper are not so small as being exactly solved, our interest consists in heuristic methods, which are typically based on local search. In [31], Wallace and Freuder investigate some heuristic methods for Max-CSP, which are based on the *min-conflicts* heuristic [23] and combined with some strategies such as "walk", "reset" and "tabu". Comparing the performance of several local search procedures, Wallace [32] says that the min-conflicts heuristic combined with the random walk strategy (MHRW) is a most successful method for Max-CSP. In [12], Galinier and Hao present a tabu search algorithm for Max-CSP, and give computational results showing that their method performs better than MHRW. In [17], Hao and Dorne solve the graph coloring problem and the frequency assignment problem to evaluate different heuristics

adopted within the local search framework. In these reports, the problems solved are Max-CSPs rather than WCSPs (i.e., their constraints are unweighted), and most of the problem instances are not from practical applications, but randomly generated. Lau [20] proposes a heuristic method for WCSP, which is based on the *randomized rounding* of a semidefinite program (SDP) relaxation. In [20], in addition to a worst-case analysis, computational experiments are also conducted, but random instances are used. On the whole, there have not been many attempts to actually conduct comprehensive computational experiments for various WCSP instances arising in real applications. This paper is aimed to improve this situation of WCSP.

The local search type heuristics like tabu search applied to WCSP may not perform well, in comparison with Max-CSP, if the given weights are directly used for the search process. This is because the quality of a solution (assignment) is likely to be dominated by the penalties for the constraints with large weights, and thus constraints with relatively small weights tend to be neglected. To overcome this difficulty, we guide the search by an evaluation function, which is different from the original penalty function but reflects its essence. We also propose a mechanism that dynamically controls the weights in the evaluation function during the search. In our algorithm, the standard tabu search procedure is repeatedly executed, controlling the weights at the end of each round of tabu search. After each round of tabu search, the weights in the evaluation function are increased if they correspond to those constraints violated by the best assignment found in the round; their values are determined according to the amounts of violations, but are always kept within the original values of the weights. The overall control rule resembles the subgradient method applied to Lagrangian relaxations [14].

Using the developed code, we then conduct computational experiments for the generalized assignment problem and the set covering problem, as well as some real application problems such as parallel shop scheduling, timetabling and nurse scheduling. Through these computational experiments, we show that our algorithm performs well for various combinatorial problems with varied weights of constraints. For example, for the generalized assignment problem, our algorithm competes with most of the special purpose algorithms with respect to both the quality of solution and the required computational time. Furthermore, we demonstrate that the control mechanism of weights improves the performance of our algorithm, by comparing it with our previous tabu search [27] and a mixed integer programming (MIP) solver CPLEX.

2 Formulations of CSP and WCSP

CSP is formally defined as follows. We are given n variables X_i $(i = 1, 2, \ldots, n)$, each taking a value in the finite domain D_i, and m constraints C_ℓ $(\ell = 1, 2, \ldots, m)$, each defined by a set of all the legal t_ℓ-tuples on variables $X_{\ell_1}, \ldots, X_{\ell_{t_\ell}}$,

$$C_\ell(X_{\ell_1}, X_{\ell_2}, \ldots, X_{\ell_{t_\ell}}) \subseteq D_{\ell_1} \times D_{\ell_2} \times \cdots \times D_{\ell_{t_\ell}}.$$

The objective of CSP is to find an assignment of exactly one value to each variable (in the rest of the paper, we refer to such an assignment simply as an assignment) such that all constraints are satisfied.

Now, we introduce the *value-variable* x_{ij} for each pair of variable X_i and value $j \in D_i$, 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 $\sum_i |D_i|$ value-variables in total. With these value-variables, an assignment can be

represented by a 0-1 vector

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

satisfying that

$$\sum_{j \in D_i} x_{ij} = 1, \quad \text{for all } i. \tag{2}$$

We denote the set of all assignments by

$$\mathcal{X} = \{\boldsymbol{x} \mid \boldsymbol{x} \text{ satisfies (2)}\}. \tag{3}$$

There may exist no assignment that satisfies all constraints. In such cases, we are often asked to satisfy the maximum number of constraints, if all the constraints are of the same importance. This problem is called the maximal CSP (Max-CSP) [11], and is a special case of the weighted CSP (WCSP), which is defined in the following. WCSP introduces a weight $w_\ell \geq 0$ and a *penalty function* $p_\ell(\boldsymbol{x}) \geq 0$ for each constraint C_ℓ, and is formulated as the following minimization problem:

$$\begin{aligned} \text{minimize} \quad & p(\boldsymbol{x}) = \sum_\ell w_\ell p_\ell(\boldsymbol{x}) \\ \text{subject to} \quad & \boldsymbol{x} \in \mathcal{X}, \end{aligned} \tag{4}$$

where $p_\ell(\boldsymbol{x})$ takes a value depending on the amount of violation of C_ℓ by $\boldsymbol{x}$. Throughout this paper, we assume that $p_\ell(\boldsymbol{x})$ takes value 0 if $\boldsymbol{x}$ satisfies C_ℓ, and a positive integer value otherwise. This assumption on p_ℓ is introduced mainly for the ease of intuitive understanding and description, though it is not mathematically needed. (In [5, 20], each $p_\ell(\boldsymbol{x})$ is regarded to take value 0 (resp., 1) if C_ℓ is satisfied (resp., violated) by $\boldsymbol{x}$.)

In [3], two problems *Semiring-based CSP* and *Valued CSP*, which contain WCSP as a special case, are defined. When WCSP is viewed as a Semiring-based CSP, each penalty term $w_\ell p_\ell(\boldsymbol{x})$ in (4) is referred to as the *value* of C_ℓ, while in a Valued CSP formulation, it corresponds to the *valuation* of C_ℓ.

We note here that the penalty function method has been commonly used in nonlinear and combinatorial optimizations [10] to transform a constrained problem into an unconstrained problem. The original objective function $f(\boldsymbol{x})$ is augmented with $p(\boldsymbol{x})$ of (4) after removing all constraints, and the unconstrained minimization problem with the augmented objective function

$$\tilde{f}(\boldsymbol{x}) = f(\boldsymbol{x}) + \sum_\ell w_\ell p_\ell(\boldsymbol{x}) \tag{5}$$

is then solved.

In many cases, constraints are classified into *hard* constraints and *soft* constraints, meaning that the former must always be satisfied for the obtained assignment to be meaningful, while the latter may remain violated if it is difficult to satisfy them. In the framework of WCSP, hard (resp., soft) constraints can be handled as those with large (resp., small) weights, and therefore both hard and soft constraints are processed on the same mathematical basis. This is different from other approaches such as the *constraint hierarchies* framework [4], which deals with hard and soft constraints in different preference levels.

Given a problem to solve, there may be more than one formulation into a WCSP. In each formulation, we must determine what are variables and domains, and how we describe constraints, their weights and penalty functions. Although this is a very important issue, we do not discuss it

in this paper, partly because it is user's responsibility and partly because the task of a problem solver is to exhibit reasonable performance to whatever formulations provided by the user. For this reason, in our experiments reported later, we assume that problems are formulated in a rather simple way. Each constraint C_ℓ is described as either a linear inequality

$$\sum_{ij} a_{\ell ij} x_{ij} \leq b_\ell, \tag{6}$$

where $a_{\ell ij}$ and b_ℓ are given integers, or an *all_different* constraint

$$\text{all_different}(X_{\ell_1}, X_{\ell_2}, \ldots, X_{\ell_{t_\ell}}), \tag{7}$$

which asks all the specified variables $X_{\ell_1}, X_{\ell_2}, \ldots, X_{\ell_{t_\ell}}$ to take different values. For constraints (6) and (7), we define their penalty functions by

$$p_\ell(\boldsymbol{x}) := \max\left\{\sum_{i,j} a_{\ell ij} x_{ij} - b_\ell, 0\right\} \tag{8}$$

and

$$p_\ell(\boldsymbol{x}) := t_\ell - \left|\left\{j \;\middle|\; \exists i \in \{\ell_1, \ell_2, \ldots, \ell_{t_\ell}\}, x_{ij} = 1\right\}\right|, \tag{9}$$

respectively. Here, however, we emphasize that the subsequent discussion is applicable to any other type of constraints and penalty functions.

3 Algorithm

Our algorithm for WCSP is based on tabu search. Tabu search [15, 16] is a local search with a modification that the replacement of the current solution by its best neighbor continues even if local optimality is attained. To avoid cycling of solutions, a set of solutions including those recently visited are kept in a *tabu list*, and excluded from the candidate list for the next solution. In this section, we first define the basic components of local search such as solution space, neighborhood and evaluation function, and then describe the framework of our tabu search as well as a control mechanism of the weights in the evaluation function.

3.1 Solution space, neighborhood and evaluation function

In our algorithm, the solution space is naturally defined as the set of all assignments $\mathcal{X}$. (We call an assignment a *solution* in the following.)

As for the neighborhood, among from a number of alternatives, we adopt one of the simplest ones, that is, the *shift neighborhood*

$$N(\boldsymbol{x}) = \{\boldsymbol{x}(x_{ij'} \leftarrow 1) \mid i \in \{1, 2, \ldots, n\},\ j' \in D_i,\ x_{ij'} = 0\},$$

where $\boldsymbol{x}(x_{ij'} \leftarrow 1)$ denotes the solution obtained from $\boldsymbol{x}$ by changing the current value j $(\in D_i)$ of a variable X_i to another value j'. Moving from $\boldsymbol{x}$ to $\boldsymbol{x}(x_{ij'} \leftarrow 1) \in N(\boldsymbol{x})$ implies that the value-variable x_{ij} (resp., $x_{ij'}$) is flipped from 1 to 0 (resp., from 0 to 1).

A straightforward method to evaluate solutions for determining the search direction would be to use the original penalty function $p(\boldsymbol{x}) = \sum_\ell w_\ell p_\ell(\boldsymbol{x})$ of (4). As mentioned in Introduction, however, using $p(\boldsymbol{x})$ throughout the search may not be a good idea (particularly for problem instances with varied weights of constraints). This is because the search is likely to be confined

to a restricted region of solutions that satisfy the constraints with large weights, and other constraints of small weights tend to be neglected. However such constraints with small weights may carry important information.

To overcome the above situation, it may be better to use rather small weights for those constraints with large original weights w_ℓ. For this purpose, we introduce new weights $\boldsymbol{v}$ such that $\mathbf{o} \leq \boldsymbol{v} \leq \boldsymbol{w}$, and use the evaluation function

$$q(\boldsymbol{v}, \boldsymbol{x}) = \sum_{\ell} v_\ell p_\ell(\boldsymbol{x}) \tag{10}$$

for the purpose of guiding our tabu search, where $\mathbf{o}$ is the m dimensional zero vector, $\boldsymbol{v} = (v_\ell \mid \ell = 1, 2, \ldots, m)$ and $\boldsymbol{w} = (w_\ell \mid \ell = 1, 2, \ldots, m)$. Of course, since $p(\boldsymbol{x})$ of (4) and $q(\boldsymbol{v}, \boldsymbol{x})$ are generally different, the original $p(\boldsymbol{x})$ is used whenever the qualities of the obtained solutions need to be compared.

In general, it is difficult to find appropriate penalty weights $\boldsymbol{v}$ in advance. As will be explained later, we propose in this paper to control the weights v_ℓ separately and dynamically during the search.

In passing, we briefly survey several method of controlling weights, which are found in the literature. A tabu search for the vehicle routing problem in [13] adopts the evaluation function (5) defined as the sum of the total travel costs and two weighted penalty terms associated with the excess of the vehicle capacity and the excess of the total route duration, respectively. If all the prespecified number of recent solutions are feasible (resp., infeasible) with respect to the vehicle capacity constraint or the route duration constraint, then the corresponding weight is halved (resp., doubled). In [33], which proposes an ejection chain approach to the generalized assignment problem, the objective function is penalized by adding the weighted penalty terms, each corresponding to the infeasibility of each resource capacity constraint. These weights are adaptively controlled during the search, depending on whether all the recent solutions are infeasible or not, and how much are the amounts of constraint violations. As for CSP, the breakout method [26] increases the weights of violated constraints whenever the search is considered to be trapped in a locally optimal solution. A similar idea was independently incorporated into local search procedures for the satisfiability (SAT) problems [29]. In [27], CSP with an objective function is treated, which is a special case of WCSP. The evaluation function is defined by adding the weighted penalty of constraint violations to the original objective function, and the weight of penalty is adaptively controlled so that the rate of the searched solutions being infeasible is kept within a certain range (e.g., between 60 and 80%).

Instead of using the sum of two measures, feasibility and optimality, it is also possible to treat them separately. For example, the tabu search for zero-one mixed integer programming problems in [21] and the genetic algorithm for the set partitioning problem in [8] adopt this approach.

3.2 Tabu search algorithm

Following the standard procedure of tabu search, after replacing the current solution $\boldsymbol{x}^{(h-1)}$ by the next solution $\boldsymbol{x}^{(h)}$ at the h-th iteration, which is the best neighbor $\boldsymbol{x}^{(h-1)}(x_{i^*j^*} \leftarrow 1)$ of $\boldsymbol{x}^{(h-1)}$, we prohibit the variable X_{i^*} from changing its value for the future t iterations. This parameter t is called the *tabu tenure*. In other words, we define our tabu list $T^{(h)}$ at the h-th iteration by

$$T^{(h)} = \left\{ \boldsymbol{x}^{(h-1)}(x_{ij} \leftarrow 1) \;\middle|\; \begin{array}{l} \text{The value of } X_i \text{ has been changed in the recent } t \text{ iterations,} \\ \text{and } j \in D_i \end{array} \right\}.$$

In our algorithm, the tabu tenure t is adaptively controlled to reflect the search history, by the method described in [27].

One round of our standard tabu search aims at solving the minimization problem

$$\begin{array}{ll} \text{minimize} & q(\boldsymbol{v}, \boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} \in \mathcal{X}, \end{array} \tag{11}$$

and is described as follows:

Tabu search: $TS(\boldsymbol{v}, \boldsymbol{x}_{init}, num_iteration)$
Input: A weight vector $\boldsymbol{v}$, an initial solution $\boldsymbol{x}_{init}$ and an iteration upper bound $num_iteration$.
Output: Good solutions $\boldsymbol{x}^{local}$ and $\boldsymbol{x}^{global}$ that, respectively, solve (11) and (4) approximately.

Step 0 (initialization): Set the initial solution by $\boldsymbol{x}^{(0)} := \boldsymbol{x}_{init}$ and the incumbent solutions by $(\boldsymbol{x}^{local}, \boldsymbol{x}^{global}) := (\boldsymbol{x}^{(0)}, \boldsymbol{x}^{(0)})$. Initialize the tabu list by $T^{(1)} := \emptyset$ and let $h := 1$.

Step h (next solution):

(1) Find the best neighbor $\boldsymbol{x}' \in N(\boldsymbol{x}^{(h-1)}) \setminus T^{(h)}$ in the sense of having the smallest evaluation $q(\boldsymbol{v}, \boldsymbol{x}')$, and let the next solution $\boldsymbol{x}^{(h)} := \boldsymbol{x}'$
(2) If $q(\boldsymbol{v}, \boldsymbol{x}^{(h)}) < q(\boldsymbol{v}, \boldsymbol{x}^{local})$ holds, then update $\boldsymbol{x}^{local} := \boldsymbol{x}^{(h)}$.
(3) If $p(\boldsymbol{x}^{(h)}) < p(\boldsymbol{x}^{global})$ holds, then update $\boldsymbol{x}^{global} := \boldsymbol{x}^{(h)}$.
(4) If $h \geq num_iteration$ holds and local optimality has been attained at least once (i.e., at some iteration, $\boldsymbol{x}^{local}$ was not improved), then output $(\boldsymbol{x}^{local}, \boldsymbol{x}^{global})$ and terminate; otherwise update the tabu list to $T^{(h+1)}$. Let $h := h + 1$ and go to Step h.

Our actual implementation contains further elaborations such as aspiration criteria, diversification by long term memory, and neighborhood reduction by restricting the variables checked in $N(\boldsymbol{x}^{(h)})$ to those related to some of the currently violated constraints. In the implementation, tabu list $T^{(h)}$ does not keep tabu solutions explicitly, but is realized by utilizing an array that memorizes the last iteration number H_i at which the value of X_i has been changed to obtain the next solution. The details are described in [27].

3.3 Control mechanism of weights

The above basic round of tabu search TS is repeated in the entire algorithm, while using the best solution $\boldsymbol{x}^{local}$ found in the previous round as the initial solution, and controlling the weight vector $\boldsymbol{v}$ to reflect the search history by then, until some stopping criterion is met; e.g., the prespecified number of rounds or computational time is achieved, or a satisfactory solution is obtained. As explained in Section 3.1, the minimization problem (11), which asks to minimize $q(\boldsymbol{v}, \boldsymbol{x})$ and is approximately solved by TS in each round, can be more tractable for the local search type heuristics to find good solutions than the original problem (4). However, the best solution $\boldsymbol{x}^{(k-1)} := \boldsymbol{x}^{local}$ found in the $(k-1)$-th round may not be good solution in the sense of the original penalty $p(\boldsymbol{x}^{(k-1)})$. Thus we update the weight vector $\boldsymbol{v}^{(k-1)}$ to $\boldsymbol{v}^{(k)}$ in the following manner so that $\boldsymbol{v}^{(k)}$ behaves more closely to $\boldsymbol{w}$.

We assume that at least one of the constraints is violated by $\boldsymbol{x}^{(k-1)}$, since otherwise $\boldsymbol{x}^{(k-1)}$ is concluded to be an optimum solution. We denote by $\boldsymbol{x}^*$ the incumbent solution; i.e., the best solution among those $\boldsymbol{x}^{global}$ obtained in the past $k-1$ rounds of TS.

Case 1: If $q(\boldsymbol{v}^{(k-1)}, \boldsymbol{x}^{(k-1)}) < p(\boldsymbol{x}^*)$ holds, then

$$v_\ell^{(k)} := \min \left\{ w_\ell, v_\ell^{(k-1)} + \frac{p(\boldsymbol{x}^*) - q(\boldsymbol{v}^{(k-1)}, \boldsymbol{x}^{(k-1)})}{\sum_{\ell \in L} p_\ell(\boldsymbol{x}^{(k-1)})^2} p_\ell(\boldsymbol{x}^{(k-1)}) \right\}, \quad \text{for all } \ell, \quad (12)$$

where $L = \{\ell \mid v_\ell^{(k-1)} < w_\ell \text{ and } p_\ell(\boldsymbol{x}^{(k-1)}) > 0\}$.

Case 2: Otherwise, let

$$u_\ell := \alpha \rho v_\ell^{(k-1)}, \quad \text{for all } \ell, \quad (13)$$

where $0 < \alpha < 1$ is a program parameter, and $\rho = \min \left\{ 1, \frac{p(\boldsymbol{x}^*)}{\max_\ell v_\ell^{(k-1)}} \right\}$. Then, apply (12) after substituting u_ℓ for $v_\ell^{(k-1)}$.

This update rule of weights is conceptually based on that of the subgradient method [18]. Each minimization problem (11) with $\boldsymbol{v} = \boldsymbol{v}^{(k)}$ corresponds to a Lagrangian relaxation [14] of the original minimization problem (4), where the weights $\boldsymbol{v}^{(k)}$ are considered as Lagrangian multipliers, and

$$\min_{\boldsymbol{x} \in \mathcal{X}} q(\boldsymbol{v}^{(k)}, \boldsymbol{x}) \leq \min_{\boldsymbol{x} \in \mathcal{X}} p(\boldsymbol{x}) \quad (14)$$

holds by $\mathbf{o} \leq \boldsymbol{v}^{(k)} \leq \boldsymbol{w}$. When the $(k-1)$-th round of tabu search TS is completed, if $q(\boldsymbol{v}^{(k-1)}, \boldsymbol{x}^{(k-1)}) < p(\boldsymbol{x}^*)$ holds (i.e., Case 1 applies), then the weight vector $\boldsymbol{v}^{(k-1)}$ is updated by (12), which is similar to the update formula of Lagrangian multipliers in the subgradient method. In this case, the weights $v_\ell^{(k-1)}$ of constraints C_ℓ, $\ell \in L$ are considered to be too small in comparison with the original weights w_ℓ. Thus, such $v_\ell^{(k-1)}$ are increased by the amount proportional to the penalty $p_\ell(\boldsymbol{x}^{(k-1)})$, unless it exceeds w_ℓ. As a result of increasing $\boldsymbol{v}^{(k-1)}$ to $\boldsymbol{v}^{(k)}$, the new $q(\boldsymbol{v}^{(k)}, \boldsymbol{x}^{(k-1)})$ may become equal to $p(\boldsymbol{x}^*)$; otherwise some weight $v_\ell^{(k-1)}$ reaches w_ℓ.

On the other hand, since each problem (11) is only approximately solved, the best solution $\boldsymbol{x}^{(k-1)}$ in the previous round may not satisfy (14). That is, if $q(\boldsymbol{v}^{(k-1)}, \boldsymbol{x}^{(k-1)}) \geq p(\boldsymbol{x}^*)$ holds (i.e., Case 2 applies), it seems that the tabu search procedure does not perform well for the weight vector $\boldsymbol{v}^{(k-1)}$, and we conclude that it is hopeless to find a solution better than the incumbent solution $\boldsymbol{x}^*$, as long as we use the weights $\boldsymbol{v}^{(k-1)}$ or larger. In this case, we reduce all the weights $\boldsymbol{v}^{(k-1)}$ by multiplying by α (< 1) in (13) before applying (12). After reducing all weights, it becomes possible to increase those weights $v_\ell^{(k-1)}$ which have already reached w_ℓ.

Another multiplier ρ is introduced in (13) so that no weight $v_\ell^{(k)}$ becomes larger than the incumbent penalty value $p(\boldsymbol{x}^*)$. This is explained as follows. To obtain a solution better than $\boldsymbol{x}^*$, the algorithm looks for only those solutions which satisfy all constraints C_ℓ whose weights satisfy $v_\ell \geq p(\boldsymbol{x}^*)$ (recall that $p_\ell(\boldsymbol{x})$ takes an integer value). Therefore, those constraints C_ℓ whose weights satisfy $v_\ell > p(\boldsymbol{x}^*)$ are considered to be over-weighted, and we keep such weights v_ℓ rather small, that is, within the incumbent penalty value $p(\boldsymbol{x}^*)$.

Now, the entire algorithm is described as follows:

CSP algorithm

Step 0 (initialization): Generate an initial solution $\boldsymbol{x}^{(0)} \in \mathcal{X}$ randomly, and initialize the incumbent solution by $\boldsymbol{x}^* := \boldsymbol{x}^{(0)}$. Set initial weights $\boldsymbol{v}^{(1)} := \boldsymbol{w}$ and $k := 1$.

Step k (tabu search and update of weights):

(1) Execute one round of tabu search and compute $(\boldsymbol{x}^{local}, \boldsymbol{x}^{global}) := TS(\boldsymbol{v}^{(k)}, \boldsymbol{x}^{(k-1)}, num_iteration)$, where *num_iteration* is a program parameter. Let the next solution $\boldsymbol{x}^{(k)} := \boldsymbol{x}^{local}$. If $p(\boldsymbol{x}^{global}) < p(\boldsymbol{x}^*)$ holds, then update $\boldsymbol{x}^* := \boldsymbol{x}^{global}$.

(2) If the stopping criterion is met, output $\boldsymbol{x}^*$ and terminate; otherwise let $k := k + 1$, and update weights to $\boldsymbol{v}^{(k)}$ by (12) and (13). Go to Step k.

Our algorithm includes two program parameters, α and *num_iteration*. From our preliminary experiments, it is observed that the computational results are not very sensitive to α unless α is close to 1. Hence we set $\alpha := 0.5$ in our implementation.

The other parameter *num_iteration* is more crucial. If *num_iteration* is larger, TS in Step k (1) is likely to find better solutions, but consumes more computational time. Thus, for small instances such as the university timetable in Section 4, it appears better to set *num_iteration* rather large; e.g., 3000, while for large instances such as the nurse scheduling problem in Section 4, *num_iteration* should be set to be smaller; e.g., 30. Of course this decision highly depends on the available computational time. To improve the applicability of our algorithm, therefore, we control *num_iteration* adaptively as follows. We consider that the algorithm is divided into a number of *phases*, each starting whenever Case 2 is applied in updating the weights in Step k (2). We first set *num_iteration* to 1 and adjust it at the end of each phase: If the incumbent solution was improved during the last phase, we halve *num_iteration*, while we double it if the incumbent solution remains the same for three phases since the last change of *num_iteration*. This rule reduces the task of finding an appropriate value of *num_iteration* and works well for most instances we tested.

4 Computational experiments

To demonstrate the wide applicability and effectiveness of our algorithm, we solved two standard combinatorial optimization problems, the generalized assignment problem and the set covering problem, as well as three practical problems taken from real applications. In this section, we report their computational results. Our algorithm is coded in C and run on a workstation Sun Ultra 2 (300MHz).

4.1 Generalized assignment problem

The *generalized assignment problem* (GAP) asks to find 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. Let $\{1, 2, \cdots, n\}$ be a set of jobs, and $\{1, 2, \cdots, d\}$ be a set of agents. We define the cost c_{ij} (> 0) of assigning job i to agent j, the amount of resource r_{ij} required by assigning job i to agent j, and the resource capacity b_j of agent j. By introducing a 0-1 variable x_{ij} for each pair of job i and agent j such that

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

GAP can be formulated as follows:

$$\text{minimize} \quad \sum_i \sum_j c_{ij} x_{ij} \tag{15}$$

$$\text{subject to} \quad \sum_j x_{ij} = 1, \quad \text{for all } i, \tag{16}$$

$$\sum_i r_{ij} x_{ij} \leq b_j, \quad \text{for all } j. \tag{17}$$

It is not difficult to see that this is in the form of WCSP, in which the variables are corresponding to jobs and the domain of each variable is given as the set of agents. The objective function (15) is realized as a linear inequality constraint

$$\sum_i \sum_j c_{ij} x_{ij} \leq 0, \tag{18}$$

whose penalty function (defined by (8)) gives the total cost $\sum_i \sum_j c_{ij} x_{ij}$ for each assignment $\boldsymbol{x}$. In the WCSP formulation, the weight of (18) is set to be sufficiently smaller than those of constraints (17), so that feasible solutions to GAP have higher priority of being chosen. Notice that constraints (16) are already in the form of (2), and we do not need to consider them explicitly in the WCSP formulation.

In our experiments, we used 18 GAP benchmark instances consisting of three types C, D and E. Each type contains six instances with $n = 100$ or 200, and $d = 5$, 10 or 20. These instances are available via WWW[1], and have been tested by many researchers. We solved them by our WCSP algorithms with and without the automatic control mechanism of weights $\boldsymbol{v}$ (denoted as CONTROLLED and FIXED, respectively), where FIXED corresponds to one of our previous CSP algorithms in [27]. In FIXED, the weight $\boldsymbol{v}$ given to each constraint of (17) is fixed to a sufficiently large value during the search, while that of (15) (i.e., (18)) is fixed to 1. We compared their results with those by other four algorithms specially developed for GAP; a genetic algorithm by Chu and Beasley (CB) [7], a tabu search by Laguna et al. (LKGG) [19], a MAX-MIN ant system by Lourenço and Serra (LS) [22] and an ejection chain approach by Yagiura, Ibaraki and Glover (YIG) [33].

Table 1 shows the best cost and the computational time (in seconds) obtained by these algorithms (in the case of CB, the average values of ten runs are shown). CB, LS and the rest were run on a Silicon Graphics Indigo (R4000, 100MHz), a Sun Ultra 1 (167MHz), and a Sun Ultra 2 (300MHz), respectively. The speeds of Silicon Graphics Indigo (R4000, 100MHz) and Sun Ultra 1 (167MHz), respectively, are estimated about 3–8 and 1.5–2 times slower than Sun Ultra 2 (300MHz) according to the results that we ran our code on some different computers, and [9]. The computational results of LKGG and LS are taken from [33], and the results of CB for type E are not available (denoted N.A.). These results indicate that our algorithm CONTROLLED is inferior to YIG, which is the best among the tested six algorithms, but appears to be at least as good as other GAP algorithms even if the difference of machines is taken into account.

In addition to the fact that the results of FIXED are much worse than those of CONTROLLED, we can also observe the tendency that, for instances of type C (resp., E), the difference between the cost found by FIXED and the best one found by YIG is relatively small (resp., large). This can be explained by the tightness of constraints. The problem instances tested here are generated so that constraints of type C (resp., E) are loosest (resp., tightest) among the three types. Since instances whose constraints are loose may have many good feasible solutions, the objective functions for type C are rather easily minimized even if the search is confined to the set of feasible solutions.

We can improve the performance of FIXED by adjusting the weight vector $\boldsymbol{v}$. For example, by manually tuning $\boldsymbol{v}$, we could obtain solutions of costs 1937, 6386 and 12789, respectively, for

[1] `http://mscmga.ms.ic.ac.uk/jeb/orlib/gapinfo.html` for types C and D,
`http://www.kuamp.kyoto-u.ac.jp/labs/or/members/yagiura/gap/` for type E.

Table 1: Computational results for GAP: Comparison with other special purpose algorithms.

type	size		CB		LKGG		LS		YIG		CONTROLLED		FIXED[5]	
	n	d	cost[1]	time[2]	cost	time[3]	cost	time[4]	cost	time[3]	cost	time[3]	cost	time[3]
C	100	5	1938.3	302.4	1931	150.0	1942	358.5	1931	150.0	1933	150.0	1964	150.0
C	100	10	1406.0	394.2	1403	150.0	1407	384.7	1402	150.0	1402	150.0	1431	150.0
C	100	20	1249.4	669.3	1245	150.0	1247	703.3	1243	150.0	1244	150.0	1297	150.0
C	200	5	3463.9	810.1	3457	300.0	3467	2088.2	3456	300.0	3459	300.0	3501	300.0
C	200	10	2819.5	1046.0	2812	300.0	2818	2331.3	2806	300.0	2810	300.0	2918	300.0
C	200	20	2405.8	1792.3	2396	300.0	2405	3332.9	2392	300.0	2395	300.0	2472	300.0
D	100	5	6394.8	530.3	6386	150.0	6476	2542.9	6357	150.0	6375	150.0	6592	150.0
D	100	10	6427.6	1094.7	6406	150.0	6469	5986.7	6358	150.0	6426	150.0	6684	150.0
D	100	20	6308.5	2126.1	6297	150.0	6358	14125.3	6221	150.0	6330	150.0	6669	150.0
D	200	5	12826.1	1942.8	12788	300.0	12923	21952.1	12746	300.0	12763	300.0	13424	300.0
D	200	10	12627.0	3189.6	12537	300.0	12746	52763.2	12446	300.0	12482	300.0	13374	300.0
D	200	20	12528.1	5565.1	12436	300.0	12617	116969.0	12284	300.0	12403	300.0	13360	300.0
E	100	5	N.A.		12687	20000.0	12836	1358.1	12682	150.0	12733	150.0	15122	150.0
E	100	10	N.A.		11641	20000.0	11780	3332.6	11577	150.0	11709	150.0	14365	150.0
E	100	20	N.A.		8522	20000.0	8717	5516.9	8443	150.0	8646	150.0	10543	150.0
E	200	5	N.A.		25147	20000.0	25317	12342.6	24930	300.0	24984	300.0	33817	300.0
E	200	10	N.A.		23567	20000.0	23620	30865.6	23302	300.0	23468	300.0	39539	300.0
E	200	20	N.A.		22659	20000.0	22779	68442.0	22377	300.0	22790	300.0	38057	300.0

1. the average incumbent cost of 10 runs.
2. the average total CPU time [sec] of 10 runs on Silicon Graphics Indigo (R4000, 100MHz).
3. the total CPU time [sec] on Sun Ultra 2 (300MHz).
4. the total CPU time [sec] on Sun Ultra 1 (167MHz).
5. our previous CSP algorithm in [27].

three instances of types C, D and E with $n = 100$, $d = 5$, within 150 seconds of computational time. But these solutions are still worse than those obtained by CONTROLLED. Furthermore, this approach is against our goal of developing a problem solver that is easy to use.

4.2 Set covering problem

The *set covering problem* (SCP) is a well known combinatorial optimization problem defined as follows:

$$\begin{aligned} \text{minimize} \quad & \sum_i c_i x_i \\ \text{subject to} \quad & \sum_i a_{\ell i} x_i \geq 1, \qquad \ell = 1, 2, \ldots, m, \\ & x_i \in \{0, 1\}, \qquad i = 1, 2, \ldots, n, \end{aligned}$$

where c_i is a positive integer and $a_{\ell i}$ is 0 or 1. Similarly to GAP, we can easily see that SCP is already formulated as a WCSP.

Table 2 shows the computational results for 20 benchmark instances from OR Library[2]. In addition to our two algorithms, CONTROLLED and FIXED, three special purpose SCP algorithms BaCa, BeCh and CFT are tested, where BaCa and CFT are Lagrangian based heuristics by Balas and Carrera [1] and by Caprara, Fischetti and Toth [6], respectively, and BeCh is a genetic algorithm by Beasley and Chu [2]. In Table 2, the sizes of instances, n, m and the density (i.e., the percentage of $a_{\ell i} = 1$) are indicated. For each instance, the best costs and the CPU times (in seconds) to find the best solutions (for CB, the average values of ten runs) are reported, except that the overall CPU times (in second) are reported for BaCa. The results of BaCa, BeCh, CFT and ours are run on VAX 6400, Silicon Graphics Indigo (R4000, 100MHz), DEC station 5000/240 and Sun Ultra 2 (300MHz), respectively. The speeds of VAX 6400, Silicon Graphics Indigo (R4000, 100MHz) and DEC station 5000/240 are perhaps 30–80, 3–8 and 10–20 times slower than Sun Ultra 2 (300MHz), respectively, according to our crude estimation.

First, we can see that our algorithm is drastically improved by incorporating the control mechanism of weights. Since our algorithms do not apply a preprocessing that reduces the sizes of instances by fixing some variables, like other algorithms developed for SCP, our codes are rather slow. However, it should be pointed out that, in most cases, our algorithm CONTROLLED could find good solutions whose costs are close to the best ones.

4.3 Parallel shop scheduling

Let us consider a parallel shop scheduling problem encountered in real applications [25]. The tested problem instance consists of 11 jobs, which are classified into 7 groups, and 4 parallel machines. For each job r, the machines on which job r can be processed are specified, and the total days demanded for processing r is also given. Each job can be processed on more than one machine with preemption and be also processed on several machines in parallel. Jobs are scheduled on a day basis, and the schedule horizon is 31 days. This problem has the following constraints:

- Each machine can process only one job at a time, and runs even on weekends and holidays.
- Once the processing of a job is initiated, it must be continued for at least three days.

[2] `http://mscmga.ms.ic.ac.uk/jeb/orlib/scpinfo.html`

Table 2: Computational results for SCP: Comparison with other special purpose algorithms.

instances	size			BaCa		BeCh		CFT		CONTROLLED		FIXED[6]	
	n	m	density	cost	time[1]	cost[2]	time[3]	cost	time[4]	cost	time[5]	cost	time[5]
E.1	500	5000	10%	29	251.3	29.0	13.5	29	26.0	29	35.9	49	138.4
E.2	500	5000	10%	32	345.6	30.6	213.5	30	408.0	30	579.3	49	56.1
E.3	500	5000	10%	28	367.9	27.7	68.1	27	94.2	27	98.9	60	217.2
E.4	500	5000	10%	29	352.2	28.0	190.8	28	26.3	28	203.9	64	418.7
E.5	500	5000	10%	28	280.2	28.0	12.4	28	36.6	28	36.2	37	300.3
F.1	500	5000	20%	14	306.6	14.0	27.0	14	33.2	14	89.5	18	248.6
F.2	500	5000	20%	15	402.5	15.0	27.6	15	31.2	15	51.1	21	453.0
F.3	500	5000	20%	15	347.9	14.0	94.3	14	248.5	14	351.0	18	93.3
F.4	500	5000	20%	15	340.1	14.0	74.1	14	31.0	14	94.9	22	89.4
F.5	500	5000	20%	14	282.8	13.7	53.7	13	201.1	14	62.1	16	404.4
G.1	1000	10000	2%	183	1480.1	177.7	361.0	176	147.0	177	383.5	723	495.8
G.2	1000	10000	2%	161	1682.1	156.3	127.4	154	783.4	156	179.0	850	594.4
G.3	1000	10000	2%	175	1721.0	167.9	249.7	166	978.0	168	304.9	755	558.2
G.4	1000	10000	2%	176	1509.9	170.3	532.3	168	378.5	172	203.1	849	508.4
G.5	1000	10000	2%	172	1193.4	169.4	194.1	168	237.2	169	570.9	801	529.1
H.1	1000	10000	5%	68	2220.2	64.0	594.4	63	1451.1	64	473.9	157	486.0
H.2	1000	10000	5%	67	1730.5	64.0	187.4	63	887.0	65	535.3	163	357.3
H.3	1000	10000	5%	63	2013.9	59.1	637.3	59	1560.3	61	253.2	144	593.4
H.4	1000	10000	5%	62	1612.2	58.9	770.3	58	237.6	59	434.8	137	488.0
H.5	1000	10000	5%	58	1460.7	55.1	158.9	55	155.4	56	350.3	199	586.4

1. the total CPU time [sec] on VAX 6400.
2. the average incumbent cost of 10 runs.
3. the average CPU time [sec] required to find the incumbent solutions of 10 runs on Silicon Graphics Indigo (R4000, 100MHz).
4. the CPU time [sec] required to find the best solution on DEC station 5000/240.
5. the CPU time [sec] required to find the best solution on Sun Ultra 2 (300MHz).
6. our previous CSP algorithm in [27].

- Changing jobs within the same group on a machine takes no time, but changing a job into another job in a different group requires the setup construction of two days.
- In the starting day of processing of a job and a setup construction, the switching operation is required.
- The setup construction and the switching operation are possible only on weekdays, and their numbers in a day are at most two and three, respectively.

The objective is to minimize the weighted sum of the total amount (days) of unsatisfied demands and the number of switchings, that is,

$$w^a \times (\text{the total amount of unsatisfied demands}) + w^s \times (\text{the number of switchings}) \qquad (19)$$

subject to all constraints.

This problem can be written in the form of WCSP by introducing variable $X_{h,t}$ for each pair of machine h and day t (hence $4 \times 31 = 124$ variables) together with its domain $D_{h,t} = \{\text{job 1, job 2, } \ldots, \text{ job 11, construction}\} \times \{\text{starting day, else}\}$ (24 values in total). Each value stands for the state of machine h on day t, and values with "starting day" correspond to switching operations. For example, the variable $X_{h,t}$ taking value (job 1, starting day) means that on day t, processing of job 1 is initiated on machine h, which requires a switching operation. All the constraints can be described by 10295 linear inequalities. The constraints associated with the objective function (19) have the weight w^a or w^s, while others have sufficiently large weights; i.e., these constraints are hard.

To compare our code with an existing general problem solver, we reformulated the resulting WCSP as the following mixed integer programming (MIP) problem:

$$\begin{aligned}
\text{minimize} \quad & \sum_{\ell} w_\ell z_\ell \\
\text{subject to} \quad & \sum_{i,j} a_{\ell ij} x_{ij} \le b_\ell + z_\ell, \quad \text{for all } \ell, \\
& \sum_{j \in D_i} x_{ij} = 1, \quad \text{for all } i, \\
& x_{ij} \in \{0,1\}, \quad \text{for all } i, \text{ and } j \in D_i, \\
& z_\ell \ge 0, \quad \text{for all } \ell,
\end{aligned}$$

where 0-1 variables x_{ij} are the value-variables of (1), and z_ℓ in the right hand side of constraint indicates whether the original constraint $\sum a_{\ell ij} x_{ij} \le b_\ell$ is satisfied (i.e., $z_\ell = 0$) or violated (i.e., $z_\ell > 0$). This problem was solved by a general MIP solver CPLEX.

We solved four instances of $(w^a, w^s) = (1, 0.5)$, $(1, 1)$, $(1, 2)$ and $(1, 5)$, respectively, by our CSP algorithms without and with the automatic control mechanism of weights $\boldsymbol{v}$ (FIXED and CONTROLLED), and by CPLEX. In FIXED, the weights $\boldsymbol{v}$ in the evaluation function is fixed to the original weights $\boldsymbol{w}$. (It may be possible to improve the performance by using another $\boldsymbol{v}$.) For each instance, we ran our codes ten times, each consuming ten minutes, while CPLEX was run once for ten hours of computational time. In Table 3, the computational results are shown. For our codes, the minimum, average and maximum penalty values (out of ten runs) are indicated if all of the ten runs could find some feasible schedules in the sense of satisfying all hard constraints (we say that such runs are successful); otherwise, the average penalty values for only successful runs and the success rate are described in the columns of "ave." and "max", respectively. These computational results say that the control mechanism of weights is quite effective, and robust against the values of (w^a, w^s), while the solutions obtained by other algorithms get worse rapidly as the difference between w^a and w^s becomes larger.

Table 3: Computational results for the parallel shop scheduling.

(w^a, w^s)	algorithms	penalty			#runs	CPU time per run
		min	ave.	max		
(1, 0.5)	FIXED[1]	24.0	$(33.3)^2$	$(7/10)^3$	10	10 [min]
	CONTROLLED	16.0	17.8	20.0	10	10 [min]
	CPLEX	16.0	—	—	1	10 [hr][4]
(1, 1)	FIXED[1]	38	$(50.3)^2$	$(8/10)^3$	10	10 [min]
	CONTROLLED	17	18.9	20	10	10 [min]
	CPLEX	19	—	—	1	10 [hr][5]
(1, 2)	FIXED[1]	51	$(79.4)^2$	$(7/10)^3$	10	10 [min]
	CONTROLLED	18	21.5	26	10	10 [min]
	CPLEX	24	—	—	1	10 [hr][6]
(1, 5)	FIXED[1]	95	$(126.7)^2$	$(6/10)^3$	10	10 [min]
	CONTROLLED	18	22.2	32	10	10 [min]
	CPLEX	39	—	—	1	10 [hr][7]

1. our previous CSP algorithm in [27].
2. the average values for successful runs.
3. "$n/10$" means that n runs out of 10 are successful.
4. The best solution was found in about 9 hours (the second best one with penalty 16.5 was found in about 50 minutes).
5. The best solution was found in about 2 hours.
6. The best solution was found in about 7.5 hours (the second best one with penalty 25 was found in about 50 minutes).
7. The best solution was found in about 35 minutes.

4.4 University Timetable

Let us consider the problem of constructing a timetable at a university, as described in [24]. This example consists of 30 lectures, 13 professors, 60 students, 10 periods and 3 rooms. Each student selects 8–10 lectures (out of 30) that he/she wants to take. Therefore, all the lectures selected by a student are desired to be assigned to different periods (constraint 1). Each professor gives two or three lectures which must be given in different periods (constraint 2), and each professor has inconvenient periods (constraint 3). Furthermore, some lectures cannot be held in some rooms due to the numbers of available seats (constraint 4). Of course, each room is assigned to at most one lecture in each period (constraint 5). The objective is to find a timetable that minimizes the number of conflicts in constraints 1 and 3 while satisfying the rest. (The complete data of this instance is given in [24] and [27].)

This problem is naturally formulated as a WCSP. We introduce a variable for each lecture, which has a pair of period and room as its value. Thus, there are 30 variables with domains of size 30, respectively. All the constraints can be represented by linear inequalities, but constraint 5 can be more compactly described by an all_different constraint.

We solved this problem by our two codes using fixed and controlled weights, respectively,

Table 4: Computational results for the university timetable.

algorithms	penalty			#runs	CPU time per run
	min	ave.	max		
FIXED[1]	87	88.7	90	10	100 [sec]
CONTROLLED	84	87.0	88	10	100 [sec]
CPLEX	92	—	—	1	10 [hr][2]

1. our previous CSP algorithm in [27].
2. The best solution was found in about 8 minutes.

and CPLEX. Our codes was run ten times, each consuming 100 seconds, and CPLEX was run once for ten hours. The computational results are shown in Table 4 in the same manner as in Table 3. We can see that our codes could find better solutions than CPLEX, and the automatic weight control mechanism improves the performance of our code.

4.5 Nurse scheduling

Finally, we show the computational results for an instance of the nurse scheduling problem, which is based on real data at a hospital in Tokyo. The instance[3] has 25 nurses (numbered from 1 to 25), where each nurse must be assigned every day one of the five alternatives: "day shift", "evening shift", "night shift", "meeting" and "day off". The time span of the schedule is 30 days (one month). The nurses are divided into two teams A and B, where team A has 13 nurses (from 1 to 13) and team B has 12 nurses (from 14 to 25). Among 25 nurses, six in A (from 1 to 6) and five in B (from 14 to 18) are regarded as leaders, respectively. Nurse 25 is also regarded as a leader in day shifts and evening shifts (but not in night shifts). 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 (given by Table 5). Each nurse may have some days which have already been scheduled (indicated by symbols with brackets in Figure 1). All meetings have been scheduled in advance. The numbers of day shifts, evening shifts, night shifts, and days off assigned to each nurse during 30 days must be within their lower and upper bounds, respectively (given by Table 6). Besides, there are the following constraints:

1. Each nurse must get a day shift and a day off at least once a week (more precisely, at least once every consecutive 7 days), respectively.

2. 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.

3. After a series of night shifts is over, the next night shift must be avoided for at least 6 days.

4. When Nurse 25 is in a night shift, at least two of the other nurses must be selected from team A.

[3]This instance was contributed by Prof. A. Ikegami at Seikei University, Japan.

Table 5: The required number of nurses and leaders for each shift of each day; e.g., "8 (4/2)" indicates that at least 8 nurses are required in total, and at least 4 nurses including at least 2 leaders must be selected from each team.

day(s)	day shift	evening shift	night shift
1,7,10,16,20,30	8 (4/2)[1]	4 (2/1)	3 (1/1)
13	7 (3/2)[1]	4 (2/1)	3 (1/1)
21	9 (4/2)[1]	4 (2/1)	3 (1/1)
23	13 (4/2)[1]	4 (2/1)	3 (1/1)
otherwise	10 (4/2)[1]	4 (2/1)	3 (1/1)

1. Since these constraints seem so tight, they are relaxed by decreasing these numbers by 1; e.g., 8 (4/2) are relaxed to 7 (3/1). The original constraints are regarded as soft ones whose weights are 1.

Table 6: The lower and upper bounds on the numbers of day shifts, evening shifts, night shifts and days off, respectively, assigned to the nurses during 30 days.

nurse(s)	day shift	evening shift	night shift	day off
1,2,3,4,5,6,14,15,16,17,18,25	[0, 14]	[4, 6]	[3, 6]	[9, 17]
12	[0, 17]	[2, 2]	[2, 2]	[9, 17]
others	[0, 15]	[4, 6]	[2, 4]	[9, 17]

5. Nurses 1 and 9 must be avoided from working together in evening shifts and night shifts.

6. The pattern of (day off, duty, day off) should be avoided.

7. An isolated night shift should be avoided (as a result of this and constraint 2 (i), night shifts should continue two days), unless it has already been scheduled on the first day (as the continuation from the previous month) or it is scheduled on the last day.

The objective is to find a schedule that satisfies all constraints. Since this objective seems difficult to achieve, however, we shall relax the constraints concerning the number of nurses and leaders for day shifts by decreasing their required numbers by one (as indicated in Table 5). The original tight constraints are regarded as soft ones whose weights are 1. Furthermore, we consider that constraints 6 and 7 are also soft, and their weights are set to 10 and 100, respectively. The adopted objective is to minimize the weighted number of violations of such soft constraints while satisfying all other constraints, whose weights are set to be sufficiently large. To formulate this problem as a WCSP, we introduce a variable for each pair of nurse and day, which takes one of the five alternatives. Thus, there are 750 variables in total, and five values in each domain.

For this instance, we ran our codes ten times, each consuming ten minutes, and ran CPLEX once consuming ten hours. In Table 7, the computational results are given in the same manner

Table 7: Computational results for the nurse scheduling.

algorithms	penalty			#runs	CPU time per run
	min	ave.	max		
FIXED[1]	9	(189.3)[2]	(6/10)[3]	10	10 [min]
CONTROLLED	3	6.6	11	10	10 [min]
CPLEX	439	—	—	1	10 [hr][4]

1. our previous CSP algorithm in [27].
2. the average for the successful runs.
3. the success rate (out of 10).
4. The best solution was found in about 3 hours.

as in Table 3. In all ten runs, CONTROLLED could find quite good solutions that satisfy not only all hard constraints but also all soft constraints whose weights are 10 or 100, while FIXED could find feasible schedules only in six runs, and the schedule found by CPLEX violates a large number of soft constraints. This says that the automatic weight control mechanism makes our code much more powerful.

An example of schedule with penalty value 3 found by our code is illustrated in Figure 1, in which "−", "=", "≡", "+" and "/" stand for day shift, evening shift, night shift, meeting and day off, respectively. In this schedule, the numbers of leaders from team B are not sufficient in day shifts (i.e., only one leader is selected from team B) on the 11th, 12th and 20th, and these are all the violations.

5 Conclusion

In this paper, we developed a tabu search algorithm for WCSP as a general problem solver for combinatorial optimization problems. To improve the performance of our algorithm, we introduced an evaluation function, which is different from the penalty function but reflects its essential part, and incorporated a control mechanism of the weights in the evaluation function. The wide applicability of our code was demonstrated by solving a number of problems including those taken from practical applications.

As mentioned in Section 2, however, it is necessary to argue what are appropriate WCSP formulations of given problems. To this end, we should improve the flexibility of our code so that users can easily incorporate their specified constraints into it. Furthermore, it is necessary to confirm if our control mechanism of weights performs well even when problems are formulated in another way. These are challenges in our future work.

Acknowledgments

We appreciate Prof. S. Morito of Waseda University for the data of parallel shop scheduling in Section 4.3, and Prof. A. Ikegami of Seikei University for the data of nurse scheduling in Section 4.5. We also wish to thank the two anonymous referees for their valuable comments and suggestions. This research was partially supported by the Grant-in-Aid for Scientific Research of

nurses	days																													#each pattern				
	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	/	−	−	/	/	≡	≡	/	−	=	/	−	−	≡	≡	/	=	=	/	−	=	/	−	−	−	≡	≡	/	−	−	11	4	6	0	9
2	[≡]	[/]	−	−	−	−	=	=	[/]	−	−	−	[+]	=	/	−	−	[/]	=	=	/	−	−	≡	≡	/	/	/	/	−	12	5	3	1	9
3	−	[/]	−	=	/	/	/	−	−	−	=	=	=	/	/	−	−	−	/	−	−	≡	≡	/	−	−	=	=	/	≡	12	6	3	0	9
4	=	=	/	−	−	−	−	≡	≡	/	[/]	[/]	/	−	−	≡	≡	/	−	[+]	−	=	=	=	[/]	[/]	−	≡	≡	/	9	5	6	1	9
5	/	−	=	≡	≡	/	−	−	=	/	−	≡	≡	/	=	=	/	−	−	≡	≡	[/]	[/]	[/]	[/]	[/]	−	−	−	−	=	10	5	6	0	9
6	−	≡	≡	/	=	=	/	−	−	≡	≡	/	−	−	−	[/]	[/]	≡	≡	/	/	−	−	−	=	=	/	−	=	/	10	5	6	0	9
7	−	=	/	−	−	/	−	−	=	/	−	−	/	−	=	=	/	−	−	=	/	/	−	−	−	−	−	/	≡	≡	/	14	5	2	0	9
8	≡	≡	/	−	=	/	/	−	−	[+]	−	−	/	−	−	−	−	−	=	/	/	−	−	≡	≡	[/]	[/]	−	=	=	/	12	4	4	1	9
9	[/]	−	=	/	−	−	/	=	≡	≡	/	[/]	−	=	/	−	−	−	=	/	−	−	[/]	−	−	/	−	−	−	≡		14	4	3	0	9
10	[+]	−	≡	≡	/	/	−	−	−	=	=	=	/	/	−	[+]	−	≡	≡	/	/	−	−	=	=	=	/	/	/	−	9	6	4	2	9
11	=	/	−	−	−	=	=	/	[/]	−	−	≡	≡	/	/	[+]	−	−	−	/	/	−	−	−	≡	≡	/	−	=	/		12	4	4	1	9
12	/	−	−	−	≡	≡	[/]	[/]	[/]	−	−	−	−	/	/	[+]	=	/	/	−	−	/	−	−	−	−	=	/	−	−	15	2	2	1	10
13	−	−	−	=	/	−	[+]	/	[/]	[/]	[/]	−	=	≡	≡	/	/	−	−	−	=	=	=	[/]	[/]	[/]	−	−	−	=	11	6	2	1	10
14	−	/	−	=	≡	≡	/	−	−	[+]	[/]	[/]	−	≡	≡	/	=	=	/	−	−	=	/	−	−	=	[/]	/	=	≡	9	6	5	1	9
15	−	−	/	/	/	≡	≡	/	−	−	=	=	=	/	−	−	/	[/]	=	≡	≡	/	−	−	/	−	−	−	−	=	12	5	4	0	9
16	/	−	−	−	=	/	−	=	/	−	−	−	[+]	=	/	/	−	≡	≡	[/]	[+]	−	=	/	−	−	/	≡	≡	/		11	4	4	2	9
17	≡	≡	/	−	−	=	/	−	=	≡	≡	/	[+]	−	−	=	/	−	−	[/]	[/]	≡	≡	/	/	−	=	=	/	−	9	5	6	1	9
18	[/]	=	=	/	−	−	−	≡	≡	[/]	[/]	/	−	−	=	≡	≡	/	/	[+]	−	−	/	=	=	≡	≡	/	−	−	9	5	6	1	9
19	−	/	=	=	/	−	[+]	=	=	/	−	−	−	/	−	[+]	/	−	−	−	≡	≡	[/]	[/]	[/]	[/]	−	−	−	−	13	4	2	2	9
20	/	−	−	−	=	/	−	−	[/]	/	=	=	/	−	−	≡	≡	[/]	[/]	−	−	=	=	/	−	−	−	/	−	[+]	13	5	2	1	9
21	−	=	/	/	−	−	/	[/]	−	=	/	−	=	=	/	−	−	=	=	/	−	−	−	/	−	−	≡	≡	/	[+]	12	6	2	1	9
22	[+]	−	−	−	/	=	≡	≡	[/]	−	−	/	/	−	−	=	/	/	−	=	/	/	−	−	=	=	/	−	−	[+]	12	5	2	2	9
23	/	/	/	/	−	−	−	−	[/]	[/]	≡	≡	/	−	−	−	=	/	−	−	=	/	−	−	−	−	−	=	=	/	[+]	13	4	2	1	10
24	=	[/]	[/]	/	−	−	=	/	−	−	−	−	/	−	=	/	−	−	≡	≡	/	−	−	=	[/]	[/]	−	−	=	=	13	6	2	0	9
25	=	/	≡	≡	/	−	=	/	−	=	/	≡	≡	/	/	−	−	−	−	=	=	/	−	≡	≡	/	−	−	/	−	10	5	6	0	9
−	8	10	10	10	10	10	8	10	10	8	10	10	7	10	10	8	10	10	10	8	9	10	13	10	10	10	10	10	10	8	287				
=	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	4	4	5	4		121			
≡	3	3	3	3	3	4	3	3	3	3	3	4	3	3	3	3	3	3	4	3	3	3	3	3	3	3	3	4	3	3			94		
+	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				20	
/	8	8	8	8	8	7	8	8	8	8	8	7	8	8	8	6	8	8	7	8	8	8	5	8	8	8	8	7	7	6					228

− : day shift
= : evening shift
≡ : night shift
+ : meeting
/ : day off
[ ] : prespecified schedule

Figure 1: A schedule of 25 nurses during 30 days found by our code.

Priority Areas "Algorithm Engineering as a New Paradigm: A Challenge to Hard Computation Problems" of the Ministry of Education, Science, Sports and Culture of Japan. The first author is supported by Research Fellowship of the Japan Society for the Promotion of Science for Young Scientists.

References

[1] E. Balas and M.C. Carrera, "A dynamic subgradient-based branch-and-bound procedure for set covering," *Operations Research* 44 (1996) 875–890.

[2] J.E. Beasley and P.C. Chu, "A genetic algorithm for the set covering problem," *European Journal of Operational Research* 94 (1996) 392–404.

[3] S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex and G. Verfaillie, "Semiring-based CSPs and valued CSPs: Basic properties and comparison," in: M. Jampel, E. Freuder and M. Maher (eds.), *Over-constrained systems*, Springer-Verlag (1996) 111–150.

[4] A. Borning, B. Freeman-Benson and M. Wilson, "Constraint hierarchies," *Lisp and Symbolic Computation* 5 (1992) 223–270.

[5] B. Cabon, S. de Givry and G. Verfaillie, "Anytime lower bounds for constraint violation minimization problems," in: M. Maher and J.-F. Puget (eds.), *Principles and Practice of Constraint Programming — CP98*, Springer-Verlag (1998) 117–131.

[6] A. Caprara, M. Fischetti and P. Toth, "A heuristic method for the set covering problem," in: W.H. Cunningham, T.S. McCormick and M. Queyranne (eds.), *Integer Programming and Combinatorial Optimization: the fifth IPCO Conference*, Springer-Verlag (1996) 72–81.

[7] P.C. Chu and J.E. Beasley, "A genetic algorithm for the generalised assignment problem," *Computers and Operations Research* 24 (1997) 17–23.

[8] P.C. Chu and J.E. Beasley, "Constraint handling in genetic algorithms: The set partitioning problem," *Journal of Heuristics* 11 (1998) 323–357.

[9] J.J. Dongarra, "Performance of various computers using standard linear equations software," Technical Report #CS-89-85, Computer Science Department, University of Tennessee and Mathematical Sciences Section, Oak Ridge National Laboratory (1998).

[10] A.V. Fiacco and G.P. McCormick, *Nonlinear programming: Sequential unconstrained minimization techniques*, Wiley (1968).

[11] E.C. Freuder and R.J. Wallace, "Partial constraint satisfaction," *Artificial Intelligence* 58 (1992) 21–70.

[12] P. Galinier and J.-K. Hao, "Tabu search for maximal constraint satisfaction problems," in: G. Smolka (ed.), *Principles and Practice of Constraint Programming — CP97*, Springer-Verlag (1997) 196–208.

[13] M. Gendreau, A. Hertz and G. Laporte, "A tabu search heuristic for the vehicle routing problem," *Management Science* 40 (1994) 1276–1290.

[14] A.M. Geofferion, "Lagrangian relaxation for integer programming," *Mathematical Programming Study* 2 (1994) 82–114.

[15] F. Glover, "Tabu search — Part I," *ORSA Journal on Computing* 1 (1989) 190–206; Part II, ditto, 2 (1990) 4–32.

[16] F. Glover and M. Laguna, *Tabu Search*, Kluwer Academic Publishers (1997).

[17] J.-K. Hao and R. Dorne, "Empirical studies of heuristic local search for constraint solving," in: E.C. Freuder (ed.), *Principles and Practice of Constraint Programming — CP96*, Springer-Verlag (1996) 194–208.

[18] M. Held, P. Wolfe and H.P. Crowder, "Validation of subgradient optimization," *Mathematical Programming* 6 (1974) 62–88.

[19] M. Laguna, J.P. Kelly, J.L. González-Velarde and F. Glover, "Tabu search for the multilevel generalized assignment problem," *European Journal of Operational Research* 82 (1995) 176–189.

[20] H.C. Lau, "A new approach for weighted constraint satisfaction: Theoretical and computational results," in: E.C. Freuder (ed.), *Principles and Practice of Constraint Programming — CP96*, Springer-Verlag (1996) 323–337.

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

[22] H.R. Lourenço and D. Serra, "Adaptive approach heuristics for the generalized assignment problem," Working Paper #288, Department of Economics and Management, Universitat Pompeu Fabra, Spain (1998).

[23] S. Minton, M.D. Johnston, A.B. Philips and P. Laird, "Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method," *Proceedings of the ninth National Conference on Artificial Intelligence* (AAAI) (1990) 17–24.

[24] S. Miyazaki, K. Iwama and Y. Kambayashi, "Database queries as combinatorial optimization problems," *Proceedings of International Symposium on Cooperative Database Systems for Advanced Applications* (1996) 448–454.

[25] S. Morito, J. Imaizumi and J.W. Park, "A mathematical programming approach to a tightly constrained scheduling problem," *Proceeding of the Production Scheduling Symposium '96* (1996) 85–90 in Japanese.

[26] P. Morris, "The breakout method for escaping from local minima," *Proceedings of the eleventh National Conference on Artificial Intelligence* (AAAI) (1993) 40–45.

[27] K. Nonobe and T. Ibaraki, "A tabu search approach to the constraint satisfaction problem as a general problem solver," *European Journal of Operational Research* 106 (1998) 599–623.

[28] V. Ramabhatta and R. Nagi, "An integrated formulation of manufacturing cell formation with capacity planning and multiple routings," *Annals of Operations Research* 77 (1998) 79–95.

[29] B. Selman and H. Kautz, "Domain-independent extension to GSAT: Solving large structured satisfiability problems," *Proceedings of IJCAI-93* (1993) 290–295.

[30] D. Sinriech and A. Meir, "Process selection and tool assignment in automated cellular manufacturing using Genetic Algorithms," *Annals of Operations Research* 77 (1998) 51–78.

[31] R.J. Wallace and E.C. Freuder, "Heuristic methods for over-constrained constraint satisfaction problems," in: M. Jampel, E. Freuder and M. Maher (eds.), *Over-constrained systems*, Springer-Verlag (1996) 207–216.

[32] R.J. Wallace, "Analysis of heuristic methods for partial constraint satisfaction problems," in: E.C. Freuder (ed.), *Principles and Practice of Constraint Programming — CP96*, Springer-Verlag (1996) 482–496.

[33] M. Yagiura, T. Ibaraki and F. Glover, "An ejection chain approach for the generalized assignment problem," Technical Report #99013, Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, Japan (1999).

Koji NONOBE received the B.E., M.E. and Dr. of Informatics degrees from Kyoto University, in 1995, 1997 and 2000, respectively. Since 2000, he has been with Department of Applied Mathematics and Physics, Graduate school of Informatics, Kyoto University. Currently, he is an assistant professor of Kyoto University. His interest includes combinatorial optimization, metaheuristics, scheduling and their applications.

Toshihide IBARAKI received the B.E., M.E. and Ph.D. degrees in Engineering from Kyoto University, in 1963, 1965 and 1970, respectively. Since 1969, he has been with Department of Applied Mathematics and Physics, Kyoto University, except for two and a half years from 1983 to 1985, during which time he was with Department of Computer and Information Sciences, Toyohashi University of Technology. Currently, he is a professor of Kyoto University. He has held a number of visiting appointments with University of Illinois, University of Waterloo, Simon Fraser University, Rutgers University, University of Canterbury and others. He is the author of "Enumerative Approaches to Combinatorial Optimization," Baltzer AG., a coauthor of "Resource Allocation Problems: Algorithmic Approaches," MIT Press, and the author of several books in Japanese. His interest includes algorithms, optimization, computational complexity and their applications.