A Very Large-Scale Neighborhood Search Algorithm for the Multi-Resource Generalized Assignment Problem

Mutsunori Yagiura,[1] Shinji Iwasaki,[2] Toshihide Ibaraki,[1] Fred Glover[3]

[1]Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan, {yagiura,ibaraki}@i.kyoto-u.ac.jp

[2]Web Service Group, NTT Data Corporation, Kayabacho Tower Bldg., 21-2, Shinkawa 1-Chome, Chuo-Ku, Tokyo 104-0033, Japan, iwasakis@nttdata.co.jp

[3]Leeds School of Business, University of Colorado, Boulder, Colorado 80309-0419, fred.glover@colorado.edu

Abstract: We propose a metaheuristic algorithm for the multi-resource generalized assignment problem (MRGAP). MRGAP is a generalization of the generalized assignment problem, which is one of the representative combinatorial optimization problems known to be NP-hard. The algorithm features a very large-scale neighborhood search, which is a mechanism of conducting the search with complex and powerful moves, where the resulting neighborhood is efficiently searched via the improvement graph. We also incorporate an adaptive mechanism for adjusting search parameters, to maintain a balance between visits to feasible and infeasible regions. Computational comparisons on benchmark instances show that the method is effective, especially for type D and E instances, which are known to be quite difficult.

Keywords: multi-resource generalized assignment problem, ejection chain, very large-scale neighborhood search, adaptive parameter adjustment, strategic oscillation

1 Introduction

We propose a metaheuristic algorithm for the *multi-resource generalized assignment problem* (MRGAP) [13]. MRGAP seeks a minimum cost assignment of n jobs to m agents subject to multi-resource constraints for each agent. MRGAP is a generalization of the generalized assignment problem (GAP) [23, 42, 43], which is one of the representative combinatorial optimization problems known to be NP-hard [37].

Many exact and heuristic algorithms have been proposed for GAP. Among recent exact algorithms for GAP are a branch-and-price algorithm by Savelsbergh [38] and a branch-and-cut algorithm by Nauss [23], where exact optimal solutions to many benchmark instances with up to 200 jobs and 20 agents were obtained by Nauss [23]. Among various heuristic and metaheuristic algorithms developed for GAP are a combination of the greedy method and local search by Martello and Toth [20, 21]; a tabu search and simulated annealing approach by Osman [26]; a genetic algorithm by Chu and Beasley [7]; variable depth search methods by Amini and Racer [4, 31]; a tabu search approach by Laguna et al. [16]; a set partitioning heuristic by Cattrysse, Salomon and Van Wassenhove [6]; a relaxation heuristic by Lorena and Narciso [18]; a GRASP and MAX-MIN ant system combined with local search and tabu search by Lourenço and Serra [19]; a linear relaxation heuristic by Trick [41]; a tabu search algorithm by Díaz and Fernández [8]; an ejection chain approach and a path relinking approach by Yagiura, Ibaraki and Glover [42, 43]; and so on. Motivated by practical applications, various generalizations of GAP have been proposed, e.g., the multi-level generalized assignment problem by Laguna et al. [16]; the dynamic multi-resource generalized assignment problem by Shtub and Kogan [39]; the generalized multi-assignment problem by Park, Lim and Lee [27]; the multi-resource generalized assignment problem with additional constraints by Privault and Herault [30]; and so forth. MRGAP is a natural generalization of GAP, and has many practical applications, e.g., in distributed computer systems and in the trucking industry [11, 12, 22, 29]. For MRGAP, Gavish and Pirkul proposed a branch-and-bound algorithm and two simple Lagrangian heuristics [13]. To the best of our knowledge, however, not much has been done for MRGAP after the work of Gavish and Pirkul in spite of its practical importance.

Our algorithm is based on tabu search, and features a *very large-scale neighborhood search*, which is a mechanism of conducting the search with complex and powerful moves, where the resulting neighborhood is efficiently searched via the *improvement graph* [1, 3, 14]. The idea of the very large-scale neighborhood search is based on the concept of the *ejection chains* [14]. Ejection chains generalize the alternating path constructions of graph theory [5, 9] and also generalize the well-known Lin and Kernighan algorithms [15, 17], which were successfully applied to graph partitioning and traveling salesman problems. Applications of the ejection chain approach include [16, 28, 32, 33, 36]. Recent developments have provided especially effective ejection chain approaches both for the traveling salesman problem [10, 35] and for the vehicle routing problem [34]. We also incorporate the strategic oscillation component of tabu search, which is realized by an automatic mechanism for adjusting search parameters, to maintain a balance between visits to feasible and infeasible regions.

We conducted computational experiments on benchmark instances called types C, D and E, and compared the proposed method with other existing algorithms. The results show that our algorithm is effective, especially for types D and E instances, which are known to be quite difficult.

2 Multi-Resource Generalized Assignment Problem

Given n jobs $J = \{1, 2, \ldots, n\}$ and m agents $I = \{1, 2, \ldots, m\}$, we undertake to determine a minimum cost assignment subject to assigning each job to exactly one agent and satisfying multi-resource constraints for each agent, where s resources $K = \{1, 2, \ldots, s\}$ are considered. Assigning job j to agent i incurs a cost of c_{ij} and consumes an amount a_{ijk} of each resource $k \in K$, whereas the total amount of the resource k available at agent i is b_{ik}. Throughout the paper, we assume $a_{ijk} \geq 0$ and $b_{ik} > 0$ for all $i \in I$, $j \in J$ and $k \in K$. An assignment is a mapping $\sigma\colon J \to I$, where $\sigma(j) = i$ means that job j is assigned to agent i. Let

$$J_i^\sigma = \{j \in J \mid \sigma(j) = i\}, \quad \forall i \in I,$$

which is the set of jobs assigned to agent i in assignment σ. Then the *multi-resource generalized assignment problem* (MRGAP) is formulated as follows:

$$\begin{array}{ll} \text{minimize} & cost(\sigma) = \displaystyle\sum_{j \in J} c_{\sigma(j),j} \\ \text{subject to} & \displaystyle\sum_{j \in J_i^\sigma} a_{ijk} \leq b_{ik}, \quad \forall i \in I \text{ and } \forall k \in K. \end{array} \tag{1}$$

MRGAP is known to be NP-hard (e.g., [37]), and the (supposedly) simpler problem of judging the existence of a feasible solution for GAP (i.e., MRGAP with $s = 1$) is NP-complete, since the partition problem can be reduced to MRGAP with $m = 2$ and $s = 1$.

3 Algorithm

3.1 Outline of the algorithm

Our algorithm, called TS-CS (tabu search with chained shift neighborhood), is an extension of local search. Local search starts from an initial solution σ and repeatedly replaces σ with a better solution in its *neighborhood* $N(\sigma)$ until no better solution is found in $N(\sigma)$. The resulting solution σ is *locally optimal* in the sense that no better solution exists in its neighborhood. *Shift* and *swap* neighborhoods N_{shift} and N_{swap} are usually used in local search methods for GAP, where

$$\begin{aligned} N_{\text{shift}}(\sigma) &= \{\sigma' \mid \sigma' \text{ is obtained from } \sigma \text{ by changing the assignment of one job}\}, \text{ and} \\ N_{\text{swap}}(\sigma) &= \{\sigma' \mid \sigma' \text{ is obtained from } \sigma \text{ by exchanging the assignments of two jobs}\}. \end{aligned}$$

The size of the shift neighborhood is $O(mn)$ and that of the swap neighborhood is $O(n^2)$. In addition to these standard neighborhoods, our algorithm uses a *chained shift* neighborhood, which consists of solutions obtainable by certain sequences of shift moves. The chained shift neighborhood $N_{\rm chain}(\sigma)$ is the set of solutions σ' obtainable from σ by shifting l ($l = 2, 3, \ldots, n$) jobs $j_1, j_2, \ldots, j_l$ simultaneously, in such a way that satisfies

$$\begin{aligned} \sigma'(j_r) &= \sigma(j_{r-1}), \quad r = 2, 3, \ldots, l \\ \sigma'(j_1) &= \sigma(j_l). \end{aligned}$$

In other words, for $r = 2, 3, \ldots, l$, job j_r is shifted from agent $\sigma(j_r)$ to agent $\sigma(j_{r-1})$ after ejecting job j_{r-1}, and then the cycle is closed by assigning job j_1 to agent $\sigma(j_l)$. The *length* of a chained shift move is the number of jobs l shifted in the move. This is based on the idea of ejection chains by Glover [14]. Since the size of such a neighborhood can become exponential in l, we carefully limit its size by utilizing improvement graphs [1, 3]. Since $|N_{\rm shift}| \le |N_{\rm swap}| \le |N_{\rm chain}|$ holds, $N_{\rm swap}$ is searched only if $N_{\rm shift}$ does not contain an improving solution, and $N_{\rm chain}$ is searched only if $N_{\rm shift} \cup N_{\rm swap}$ does not contain an improving solution, unless otherwise stated.

When the search visits the infeasible region, we evaluate the solutions by an objective function penalized by infeasibility:

$$pcost(\sigma) = cost(\sigma) + \sum_{\substack{i \in I \\ k \in K}} \alpha_{ik} p_{ik}(J_i^\sigma), \tag{2}$$

where

$$p_{ik}(S) = \max\left\{0, \sum_{j \in S} a_{ijk} - b_{ik}\right\}$$

for $i \in I$, $k \in K$ and a subset $S \subseteq J$ of the jobs. The parameters α_{ik} (> 0) are adaptively controlled during the search by using an algorithm similar to the method in [42].

Whenever the local search stops at a locally optimal solution $\sigma_{\rm lopt}$, it resumes from an initial solution generated by the following rule. We keep a solution $\sigma_{\rm seed}$, which is initially generated randomly, and is replaced with $\sigma_{\rm lopt}$ if $pcost(\sigma_{\rm lopt}) \le pcost(\sigma_{\rm seed})$ holds (the most recent values for α_{ik} are used in *pcost*) during the search. Then we choose as the initial solution the solution in $N_{\rm shift}(\sigma_{\rm seed}) \setminus T$ with the smallest *pcost*, where T is the set of solutions already generated with shift moves from the current $\sigma_{\rm seed}$. Then the local search starts from the initial solution with neighborhood $N_{\rm swap}$ and $N_{\rm chain}$, i.e., the search in $N_{\rm shift}$ is forbidden until an improved solution is found. This strategy is confirmed to be effective to avoid cycling of short period [42].

We will describe the details of the proposed algorithm in the following sections. Section 3.2 explains the improvement graph, Section 3.3 describes how we search an improved solution in the chained shift neighborhood using the improvement graph, Section 3.4 is about the adaptive control mechanism of the penalty weights, and Section 3.5 gives the whole framework of the proposed algorithm.

3.2 Improvement graph

The improvement graph $G(\sigma) = (V, E)$ is a directed graph with a weight $w_{j_1 j_2}$ on each arc $(j_1, j_2) \in E$, where each vertex j in V corresponds to a job j in J (i.e., $V = J$), and

$$E = \{(j_1, j_2) \mid j_1, j_2 \in V, \sigma(j_1) \neq \sigma(j_2)\}. \tag{3}$$

An arc $(j_1, j_2) \in E$ corresponds to two partial shift moves, ejection of job j_1 from agent $\sigma(j_1)$ and insertion of job j_2 into $\sigma(j_1)$ (note that the arc orientation is opposite to the actual move of job j_2; this convention is more appropriate to indicate the order of participating actions). A chained shift move of jobs $j_1, j_2, \ldots, j_l$ corresponds to a cycle $j_1 \to j_2 \to \cdots \to j_l \to j_1$ in graph $G(\sigma)$. The length of a cycle is the number of arcs in it, which is the length of the corresponding chained shift move. We denote

$$i_r = \sigma(j_r), \quad r = 1, 2, \ldots, l$$

and assume $j_{l+1} = j_1$ and $i_{l+1} = i_1$ for convenience. Figure 1 illustrates two partial shift moves of jobs j_1 and j_2 and the corresponding arc in the improvement graph, while Figure 2 illustrates a chained shift move of length three and the corresponding cycle in the improvement graph, where a round rectangle represents an agent and a circle represents a job in the left part of the figures.

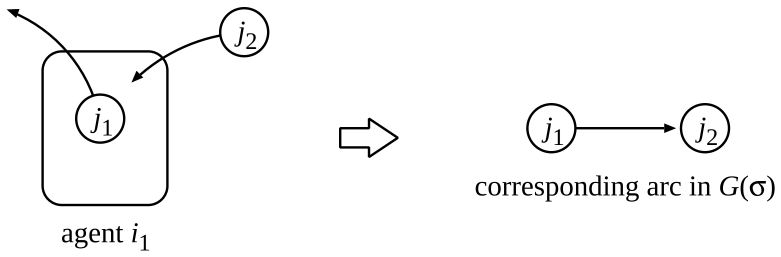

Figure 1. Two partial shift moves and the corresponding arc in $G(\sigma)$

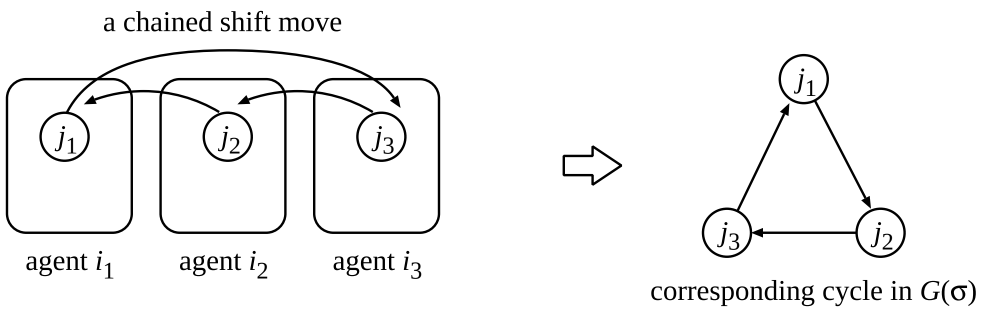

Figure 2. A chained shift move of length three and the corresponding cycle in $G(\sigma)$

For convenience, we denote by

$$pcost_i(S) = \sum_{j \in S} c_{ij} + \sum_{k \in K} \alpha_{ik} p_{ik}(S)$$

the penalized cost for agent $i \in I$ when the jobs in $S \subseteq J$ are assigned to i. Then the penalized cost of (2) can be computed as

$$pcost(\sigma) = \sum_{i \in I} pcost_i(J_i^\sigma).$$

The weight of an arc (j_1, j_2) is then defined as

$$w_{j_1 j_2} = pcost_{i_1}(J_{i_1}^\sigma \cup \{j_2\} \setminus \{j_1\}) - pcost_{i_1}(J_{i_1}^\sigma),$$

which is the change in $pcost_{i_1}$ when job j_1 is ejected from agent i_1 and job j_2 is inserted into i_1.

Let us consider a chained shift move of jobs $j_1, j_2, \ldots, j_l$ and the corresponding cycle $j_1 \to j_2 \to \cdots \to j_l \to j_1$ in graph $G(\sigma)$. We call a cycle *subset-disjoint* if $i_r \neq i_{r'}$ holds for any r and r' ($r \neq r' \in \{1, 2, \ldots, l\}$). Then, if the cycle is subset-disjoint, the change in *pcost* by the chained shift move is computed as

$$\begin{aligned} & \sum_{r=1}^{l} pcost_{i_r}(J_{i_r}^\sigma \cup \{j_{r+1}\} \setminus \{j_r\}) - \sum_{r=1}^{l} pcost_{i_r}(J_{i_r}^\sigma) \\ = & \sum_{r=1}^{l} \{pcost_{i_r}(J_{i_r}^\sigma \cup \{j_{r+1}\} \setminus \{j_r\}) - pcost_{i_r}(J_{i_r}^\sigma)\} \\ = & \sum_{r=1}^{l} w_{j_r j_{r+1}}. \end{aligned}$$

That is, the change in *pcost* by a chained shift move is the same as the total weight of arcs in the corresponding cycle. Note that this property may not hold if the cycle is not subset-disjoint. With this property, we can find an improved solution in the chained shift neighborhood by searching for a subset-disjoint cycle with a negative total weight. Unfortunately, the problem of finding such a cycle in an improvement graph is known to be NP-hard in general [40], while the problem of finding a negative cycle, not necessarily subset-disjoint, is known to be polynomially solvable (e.g., [2]). In the case of the traveling salesman problem a special strategy has been proposed that identifies subgraphs of the improvement graph, whose cycles encompass exponential numbers of solutions, over which the problem of finding a negative cycle can be solved in polynomial time [14]. By contrast, in the present context, we allow consideration of the entire graph, but use a tailored heuristic method to find an improved solution in the chained shift neighborhood.

Since the efficiency of the search in the improvement graph crucially depends on the number of arcs, we reduce the number of arcs by the following rule. Let

$$\delta_{j_1 j_2} = \left| \{k \in K \mid p_{i_1 k}(J_{i_1}^\sigma \setminus \{j_1\} \cup \{j_2\}) > 0\} \right|,$$

which is the number of overloaded resources at agent $i_1 = \sigma(j_1)$ when job j_1 is ejected from i_1 and j_2 is inserted into i_1. We then restrict the arcs to

$$\tilde{E} = \left\{ (j_1, j_2) \in E \;\middle|\; \delta_{j_1 j_2} = \min_{j' \in J} \delta_{j_1 j'} \right\}. \tag{4}$$

That is, outgoing arcs from a vertex j_1 is restricted to those whose number of overloaded resources is the minimum among the outgoing arcs from j_1. With this restriction, we may lose some improved solution in the chained shift neighborhood, but the search will become more efficient. Moreover, the search is biased toward the feasible region. In particular, if $\delta_{j_r j_{r+1}} = 0$ holds for all $r = 1, 2, \ldots, l$ in a subset-disjoint cycle $j_1 \to j_2 \to \cdots \to j_l \to j_1$, then the solution obtained by applying the corresponding chained shift move is feasible.

3.3 Search in the chained shift neighborhood

In this section, we explain the algorithm to find an improved solution in the chained shift neighborhood via the improvement graph $\tilde{G}(\sigma) = (V, \tilde{E})$. We do not restrict our search to subset-disjoint cycles, since we expect that the total arc weight of a cycle is often a good estimate on the change in *pcost* of the corresponding chained shift move even if the cycle is not subset-disjoint. Note that a cycle of length at most three is always subset-disjoint, which is immediate from the definition (3). Moreover, as shown in Appendix A, a short cycle of length at most $\sqrt{m}$ tends to be subset-disjoint if the assignment σ is random. On the other hand, a longer cycle tends not to be subset-disjoint (Appendix A), and a cycle of length more than m cannot be subset-disjoint. We therefore restrict the search to cycles of length at most m.

Our algorithm is based on dynamic programming. Let $f^*(j_1, j, l)$ be the minimum total weight of a path among those of length l from vertex j_1 to j for $j_1, j \in J,\ l = 1, 2, \ldots, m-1$. We assume $w_{j_1 j_2} = +\infty$ for $(j_1, j_2) \notin \tilde{E}$ throughout this section. Then $f^*(j_1, j, l)$ can be computed by

$$f^*(j_1, j, l) = \begin{cases} w_{j_1 j}, & l = 1 \\ \min\limits_{j' \in J} \left\{ f^*(j_1, j', l-1) + w_{j' j} \right\}, & l = 2, 3, \ldots, m-1 \end{cases} \tag{5}$$

for $j_1, j \in J$. During the recursion, we keep a j' that attains the minimum in (5), for each (j_1, j, l). Then a path of weight $f^*(j_1, j, l)$ can be reconstructed by tracing such j' from l to 1.

By definition, $f^*(j_1, j, l) + w_{j j_1}$ represents the minimum weight among those cycles of length l that include edge (j, j_1). We therefore can find a cycle of negative weight in the improvement graph by searching (j_1, j, l) with $f^*(j_1, j, l) + w_{j j_1} < 0$. While searching such (j_1, j, l), we recompute *pcost* of the corresponding solution whenever we find (j_1, j, l) that satisfies $f^*(j_1, j, l) + w_{j j_1} < 0$, since $f^*(j_1, j, l) + w_{j j_1}$ may not be the same as the change in *pcost* if the cycle is not subset-disjoint. This procedure is summarized as follows, which is called *Search-Chained-Shift* (SCS).

Procedure SCS

Input: the current solution σ.

Output: 'no' if it failed in finding an improved solution in $N_{\text{chain}}(\sigma)$; otherwise an improved solution $\sigma' \in N_{\text{chain}}(\sigma)$.

Step 1. Let $S := J$.

Step 2. If $S = \emptyset$ holds, output 'no' and stop. Otherwise, choose a $j_1 \in S$ randomly, and let $S := S \setminus \{j_1\}$ and $l := 1$.

Step 3. Compute $f^*(j_1, j, l)$ for all j by (5).

Step 4. Sort $j \in J$ in nondecreasing order of $f^*(j_1, j, l) + w_{jj_1}$. Then, for each $j \in J$ in this order, compute the change in *pcost* of the corresponding chained shift move, and if an improved solution σ is found, output σ immediately and stop.

Step 5. If $l \le m - 2$, let $l := l + 1$ and return to Step 3; otherwise return to Step 2.

Procedure SCS can find a cycle of negative weight (not necessarily subset-disjoint) if it exists; however, it does not necessarily mean that an improved solution is found in the chained shift neighborhood, since the weight of a cycle may not be the same as the change in *pcost* by the corresponding chained shift move.

The recursion of (5) is computed as in Figure 3. The time complexity for computing $f^*(j_1, j, l)$ for all $j_1, j \in J$ and $l = 1, 2, \ldots, m-1$ is $O(nm(n + |\tilde{E}|)) = O(nm|\tilde{E}|)$, which is the dominant part of the time complexity of procedure SCS. More precisely, $O(n \log n)$ time is necessary for sorting for each call to Step 4, which requires $O(n^2 m \log n)$ time in total. Moreover, $O(ls)$ time is necessary to recompute *pcost* for each (j_1, j, l) with $f^*(j_1, j, l) + w_{jj_1} < 0$, where s is the number of resources. This may require $O(n^2 m^2 s)$ time in total, since the number of such (j_1, j, l) is $O(n^2 m)$ and $l \le m$. Hence the worst case time complexity is $O(nm(|\tilde{E}| + n \log n + nms))$. However, the latter two parts in this complexity usually do not dominate $O(nm|\tilde{E}|)$, because (1) the improvement graph is usually dense even with the restriction of (4), and (2) the number of combinations (j_1, j, l) with $f^*(j_1, j, l) + w_{jj_1} < 0$ is usually very small.

In order to make the search more efficient, we employ the following restricted dynamic programming. Let

$$\lambda(x) = \begin{cases} +\infty, & x \ge 0 \\ x, & x < 0, \end{cases}$$

for $j_1 = 1, 2, \ldots, n$ **do**

  $f^*(j_1, j, 1) := w_{j_1 j}$ for all $j \in J$.

  for $l = 2, 3, \ldots, m-1$ **do**

    $f^*(j_1, j, l) := +\infty$ for all $j \in J$.

    for each $(j', j) \in \tilde{E}$ **do**

      $f^*(j_1, j, l) := f^*(j_1, j', l-1) + w_{j'j}$ if $f^*(j_1, j', l-1) + w_{j'j} < f^*(j_1, j, l)$.

    end for

  end for

end for.

Figure 3. Algorithm to compute the recursion of (5)

then the recursion of the restricted dynamic programming is

$$f_-^*(j_1, j, l) = \begin{cases} \lambda(w_{j_1 j}), & l = 1 \\ \min\limits_{j' \in J} \{\lambda(f_-^*(j_1, j', l-1) + w_{j'j})\}, & l = 2, 3, \ldots, m-1 \end{cases} \tag{6}$$

for $j_1, j \in J$. Compared to (5), the recursion of (6) restricts the search to only those paths $j_1 \to j_2 \to \cdots \to j_l$ in which $f_-^*(j_1, j_{l'}, l'-1)$ is negative for all $l' = 2, 3, \ldots, l$. The recursion of (6) is usually more efficient than (5), because we can omit the computation if $f_-^*(j_1, j', l-1) + w_{j'j} \geq 0$ holds. To realize this, we sort the outgoing edges of each vertex in nondecreasing order of the weight, and scan the outgoing edges (j', j) from a vertex j' in this order until $f_-^*(j_1, j', l-1) + w_{j'j} \geq 0$ holds. The sorting is necessary only once for each vertex for a given $\tilde{G}(\sigma)$. This takes $O(n^2 \log n)$ time in total, which is negligible compared to the time needed for the DP recursion. Procedure SCS with this restricted dynamic programming can also find a cycle of negative weight (not necessarily subset-disjoint) if it exists. This fact is immediate from the following well-known lemma (see, e.g., [17]).

Lemma 1 If $\sum_{r=1}^{l} w_{j_r j_{r+1}} < 0$ holds, then there exists a u such that $\sum_{r=u}^{t} w_{j_r j_{r+1}} < 0$ holds for all $t = u, u+1, \ldots, l$, and $\sum_{r=1}^{t} w_{j_r j_{r+1}} + \sum_{i=u}^{l} w_{j_r j_{r+1}} < 0$ holds for all $t = 1, 2, \ldots, u-1$.

Proof: Let t^* be the largest index among those t that maximizes $\sum_{r=1}^{t} w_{j_r j_{r+1}}$. Then $\sum_{r=1}^{t} w_{j_r j_{r+1}} + \sum_{r=t^*+1}^{l} w_{j_r j_{r+1}} \leq \sum_{r=1}^{t^*} w_{j_r j_{r+1}} + \sum_{r=t^*+1}^{l} w_{j_r j_{r+1}} < 0$ holds if $1 \leq t \leq t^*$, and $\sum_{r=t^*+1}^{t} w_{j_r j_{r+1}} = \sum_{r=1}^{t} w_{j_r j_{r+1}} - \sum_{r=1}^{t^*} w_{j_r j_{r+1}} < 0$ holds if $t^* + 1 \leq t \leq l$. Hence $u = t^* + 1$ satisfies the conditions in the lemma. ∎

3.4 Adaptive control of the penalty weights

In this section, we explain the adaptive mechanism for controlling the penalty weights α_{ik} in algorithm TS-CS. The performance of TS-CS highly depends on the values of α_{ik}.

Initially we set $\alpha_{ik} := \beta$ for all i and k, where we use $\beta = 1$ in the computational experiment in Section 4. We then update α_{ik} by the following rule whenever a locally optimal solution σ_{lopt} with respect to the current α_{ik} is reached. Here we use functions

$$\begin{aligned} q_{ik}^{\text{inc}}(\sigma_{\text{lopt}}) &= p_{ik}(\sigma_{\text{lopt}})/b_{ik}, \\ q_{ik}^{\text{dec}}(\sigma_{\text{lopt}}) &= \begin{cases} -1, & \text{if } p_{ik}(\sigma_{\text{lopt}}) = 0 \\ 0, & \text{otherwise.} \end{cases} \end{aligned}$$

The rule depends on whether a feasible solution was found or not during the search after the last update of α_{ik}.

<u>Case 1</u> (If no feasible solution was found after the last update of α_{ik}): the α_{ik} values are increased for all $i \in I$ and $k \in K$ by

$$\alpha_{ik} := \alpha_{ik}(1 + \Delta_{ik}),$$

where

$$\Delta_{ik} = \begin{cases} \mathit{step\_size\_inc} \cdot \dfrac{q_{ik}^{\text{inc}}(\sigma_{\text{lopt}})}{\max\limits_{i' \in I,\, k' \in K} |q_{i'k'}^{\text{inc}}(\sigma_{\text{lopt}})|}, & \text{if } \max\limits_{i' \in I,\, k' \in K} |q_{i'k'}^{\text{inc}}(\sigma_{\text{lopt}})| > 0 \\ 0, & \text{otherwise} \end{cases}$$

($\mathit{step\_size\_inc} > 0$ is a prespecified parameter).

<u>Case 2</u> (Otherwise): All α_{ik} are decreased. The rule to update α_{ik} is the same as Case 1 except that $q_{ik}^{\text{dec}}(\sigma_{\text{lopt}})$ and *step_size_dec* (a prespecified parameter satisfying $0 < \mathit{step\_size\_dec} < 1$) are used instead of $q_{ik}^{\text{inc}}(\sigma_{\text{lopt}})$ and *step_size_inc*.

The meaning of these rules is explained as follows. In the functions $q_{ik}^{\text{inc}}(\sigma_{\text{lopt}})$, the penalties $p_{ik}(\sigma_{\text{lopt}})$ are divided by b_{ik} to normalize among agents and resources. If no feasible solution was found after the last update of α_{ik} (Case 1), we increase α_{ik} by $\alpha_{ik}\Delta_{ik}$ for all $i \in I$ and $k \in K$, where the $q_{ik}^{\text{inc}}(\sigma_{\text{lopt}})$ terms are the weights to emphasize overloaded agents, and Δ_{ik} are proportional to $q_{ik}^{\text{inc}}(\sigma_{\text{lopt}})$ normalized by $\max_{i' \in I,\, k' \in K} |q_{i'k'}^{\text{inc}}(\sigma_{\text{lopt}})|$ so that the maximum amount of changes among all agents becomes *step_size_inc*. The opposite case (Case 2) is similarly explained. In our preliminary experiments, we found that the performance of TS-CS is robust with parameters *step_size_inc* and *step_size_dec*. In Section 4, we use $\mathit{step\_size\_inc} = 0.01$ and $\mathit{step\_size\_dec} = 0.1$.

3.5 The whole framework of the algorithm

The whole framework of the proposed algorithm TS-CS is described in this section. The search starts from a random solution. The current solution is improved by local search with the shift, swap and chained shift neighborhoods, where the swap (resp., chained shift) neighborhood is searched only if the current solution is locally optimal with respect to the shift (resp., swap) neighborhood. Whenever the local search stops at a locally optimal solution σ_{lopt}, the penalty weights α_{ik} are updated, and then the search resumes from an initial solution generated by applying a shift move to a solution σ_{seed}, which is initially generated randomly, and is replaced with σ_{lopt} if $\mathit{pcost}(\sigma_{\mathrm{lopt}}) \leq \mathit{pcost}(\sigma_{\mathrm{seed}})$ holds. Tabu list T is used to avoid short cycling when we choose a shift move from σ_{seed}, where T consists of those solutions to which the moves are forbidden. The search stops when the execution time exceeds *timelim* (a prespecified parameter). We check the feasibility for all the solutions generated during the search, and update the incumbent solution whenever a feasible solution is found.

Algorithm TS-CS

Step 1. Generate a solution σ randomly; i.e., for each $j \in J$, choose an i uniformly at random from $\{1, 2, \ldots, m\}$ and let $\sigma(j) := i$. Let $T := \emptyset$ and $\sigma_{\mathrm{seed}} := \sigma$.

Step 2. Search the shift neighborhood with the first admissible move strategy. If a solution $\sigma' \in N_{\mathrm{shift}}(\sigma)$ that satisfies $\mathit{pcost}(\sigma') < \mathit{pcost}(\sigma)$ is found, let $\sigma := \sigma'$ and return to Step 2.

Step 3. Search the swap neighborhood with the first admissible move strategy. If a solution $\sigma' \in N_{\mathrm{swap}}(\sigma)$ that satisfies $\mathit{pcost}(\sigma') < \mathit{pcost}(\sigma)$ is found, let $\sigma := \sigma'$ and return to Step 2.

Step 4. Search the chained shift neighborhood by procedure SCS. If a solution $\sigma' \in N_{\mathrm{chain}}(\sigma)$ that satisfies $\mathit{pcost}(\sigma') < \mathit{pcost}(\sigma)$ is found, let $\sigma := \sigma'$ and return to Step 2.

Step 5. If the computation time exceeds *timelim*, output the incumbent solution and stop.

Step 6. Update the penalty weights α_{ik} by using the procedure in Section 3.4.

Step 7. If $\mathit{pcost}(\sigma) \leq \mathit{pcost}(\sigma_{\mathrm{seed}})$ and $\sigma \neq \sigma_{\mathrm{seed}}$, let $T := \emptyset$ and $\sigma_{\mathrm{seed}} := \sigma$; otherwise let $\sigma := \sigma_{\mathrm{seed}}$.

Step 8. Let σ' be the solution that minimizes *pcost* in $N_{\mathrm{shift}}(\sigma_{\mathrm{seed}}) \setminus T$. Let $\sigma := \sigma'$ and $T := T \cup \{\sigma'\}$. Return to Step 3.

4 Computational Results

We compared the proposed algorithm TS-CS with the following three algorithms: (1) tabu search without chained shift neighborhood (denoted TS-noCS), (2) a general problem solver for the weighted constraint satisfaction problem proposed in [24] (denoted TS-WCSP), and (3) a commercial exact solver CPLEX 6.5 (denoted CPLEX).[1] Note that TS-noCS is the same as TS-CS except that it does not use the chained shift neighborhood. All the algorithms were coded in C and run on a workstation Sun Ultra 2 Model 2300 (two UltraSPARC II 300MHz processors with 1 GB memory), where the computation was executed on a single processor.

Test instances were generated randomly by using benchmark instances for GAP. (We use GAP instances for $s = 1$.) There are five types of benchmark instances of GAP called types A, B, C, D and E [7, 16]. Out of these, we use three types C, D and E, since the other two are too easy to see differences among the tested algorithms. Instances of these types are generated as follows:

Type C: a_{ij1} are random integers from [5, 25], c_{ij} are random integers from [10, 50], and $b_{i1} = 0.8 \sum_{j \in J} a_{ij1}/m$.

Type D: a_{ij1} are random integers from [1, 100], $c_{ij} = 111 - a_{ij1} + e_{ij}$, where e_{ij} are random integers from $[-10, 10]$, and $b_{i1} = 0.8 \sum_{j \in J} a_{ij1}/m$.

Type E: $a_{ij1} = 1 - 10 \ln e_{ij}$, where e_{ij} are random numbers from (0, 1], $c_{ij} = 1000/a_{ij1} - 10\hat{e}_{ij}$, where $\hat{e}_{ij}$ are random numbers from [0, 1], and $b_{i1} = 0.8 \sum_{j \in J} a_{ij1}/m$.

Types D and E are somewhat harder than type C, since c_{ij} and a_{ij1} are inversely correlated. We tested 18 instances of types C, D and E with n up to 200. Among them, types C and D instances were taken from OR-Library,[2] and type E instances were generated by ourselves, and are available at our web site.[3] For each of these GAP instances, we generated MRGAP instances by setting $a_{ijk} = 3a_{ij1}/4 + \gamma_{ijk}a_{ij1}/2$ for each $k = 2, 3, \ldots, s$ as in [13], where γ_{ijk} are random numbers from [0, 1], and then setting b_{ik} for $k = 2, 3, \ldots, s$ in the same manner as b_{i1} according to the problem type. As a result, there are 72 instances, i.e., one instance for each combination of the type (C, D or E), n (100 or 200), m (5, 10 or 20) and s (1, 2, 4 or 8). The generated MRGAP instances are available at our web site.[4]

Tables 1, 2 and 3 show the results of TS-CS, TS-noCS, TS-WCSP and CPLEX, where the time limit is set to 300 (resp., 600) seconds for each instance with $n = 100$ (resp., 200). Columns "best" show the objective values of the best solutions obtained by the algorithms within the time limit, and columns "TTB" (time to best) show the CPU seconds when the best solutions were found for the first time. (Precise TTB for CPLEX is not available,

[1] We also tested a later version, CPLEX 8.1.0, for some instances, but the results are not much different.
[2] URL of OR-Library: http://mscmga.ms.ic.ac.uk/jeb/orlib/gapinfo.html
[3] URL of our web site for GAP instances: http://www-or.amp.i.kyoto-u.ac.jp/~yagiura/gap/
[4] URL of our web site for MRGAP instances: http://www-or.amp.i.kyoto-u.ac.jp/~yagiura/mrgap/

and hence an approximate value is shown.) Columns "LB" show the lower bounds on the optimal values, where the mark "†" means that the value is optimal. (Most of these lower bounds were obtained by CPLEX, where the time limit was set to 3600 seconds. Some optimal values for GAP (i.e., $s = 1$) were found by Nauss [23], and some LBs for GAP were reported in [42], which were found by solving a Lagrangian relaxation problem. If $s \geq 2$, then optimal values and LBs were found by CPLEX.) In the tables, each "$*$" mark represents that the best cost is attained, and "—" means that no feasible solution was found. The average of "LB," "best"and "TTB" for $n = 100$ and 200, respectively, are also shown.

From the tables, we can observe the following.

- The performance of algorithm TS-CS is better than TS-noCS especially for types D and E instances. This indicates that incorporating the chained shift neighborhood is effective for hard instances.
- The performance of TS-CS is better than TS-WCSP and CPLEX. CPLEX is very effective for type C instances; however, TS-CS is better for types D and E instances.

5 Conclusion

In this paper, we considered the multi-resource generalized assignment problem and proposed a tabu search algorithm in which a sophisticated neighborhood called the chained shift neighborhood is incorporated. It was confirmed through computational comparisons on benchmark instances that the method is effective, especially for type D and E instances, which are known to be very difficult.

Acknowledgments

The authors are grateful to anonymous referees for valuable comments to improve this paper. This research was partially supported by Scientific Grant-in-Aid by the Ministry of Education, Culture, Sports, Science and Technology of Japan, and by Informatics Research Center for Development of Knowledge Society Infrastructure (COE program of the Ministry of Education, Culture, Sports, Science and Technology, Japan).

References

[1] R.K. Ahuja, O. Ergun, J.B. Orlin, and A.P. Punnen, "A survey of very large-scale neighborhood search techniques," *Discrete Applied Mathematics*, 123 (2002) 75–102.

[2] R.K. Ahuja, T.L. Magnanti and J.B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, 1993.

[3] R.K. Ahuja, J.B. Orlin, and D. Sharma, "Very large-scale neighborhood search," *International Transactions in Operational Research*, 7 (2000) 301–317.

[4] M.M. Amini and M. Racer, "A hybrid heuristic for the generalized assignment problem," *European Journal of Operational Research*, 87 (1995) 343–348.

[5] C. Berge (translated by A. Doig), *The Theory of Graphs and Its Applications*, Methuen, London, 1962; (re-issued by Wiley, New York, 1964).

[6] D.G. Cattrysse, M. Salomon and 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 and J.E. Beasley, "A genetic algorithm for the generalized assignment problem," *Computers and Operations Research*, 24 (1997) 17–23.

[8] J.A. Díaz and E. Fernández, "A tabu search heuristic for the generalized assignment problem," *European Journal of Operational Research*, 132 (2001) 22–38.

[9] J. Edmonds, "Maximum matching and a polyhedron with 0, 1-vertices," *J. Research of the National Bureau of Standards*, 69B (1965) 125–130.

[10] D. Gamboa, C. Rego and F. Glover, "Data structures and ejection chains for solving large-scale traveling salesman problems," Research Report HCES-05-02, Hearin Center for Enterprise Science, The University of Mississippi, 2002.

[11] B. Gavish and H. Pirkul, "Allocation of databases and processors in a distributed computing system," in: J. Akoka, ed., *Management of Distributed Data Processing*, North Holland Publishing Company, Amsterdam, 1982.

[12] B. Gavish and H. Pirkul, "Computer and database location in distributed computer systems," *IEEE Transactions on Computers*, 35 (1986) 583–590.

[13] B. Gavish and H. Pirkul, "Algorithms for the multi-resource generalized assignment problem," *Management Science*, 37 (1991) 695-713.

[14] F. Glover, "Ejection chains, reference structures and alternating path methods for traveling salesman problems," Research Report, University of Colorado, Boulder, CO. (abbreviated version published in *Discrete Applied Mathematics*, 65 (1996) 223–253.)

[15] B.W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell System Technical Journal*, 49 (1970) 291–307.

[16] 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.

[17] S. Lin and B.W. Kernighan, "An effective heuristic algorithm for the traveling salesman problem," *Operations Research*, 21 (1973) 498–516.

[18] L.A.N. Lorena and M.G. Narciso, "Relaxation heuristics for a generalized assignment problem," *European Journal of Operational Research*, 91 (1996) 600–610.

[19] H.R. Lourenço and D. Serra, "Adaptive search heuristics for the generalized assign-

ment problem," *Mathware and Soft Computing*, 9 (2002) 209–234.

[20] S. Martello and P. Toth, "An algorithm for the generalized assignment problem," *Proceedings of the Ninth IFORS International Conference on Operational Research*, pp. 589–603, 1981.

[21] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Chichester, 1990.

[22] R.A. Murphy, "A private fleet model with multi-stop backhaul," Working Paper 103, Optimal Decision Systems, Green Bay, WI54306, 1986.

[23] R.M. Nauss, "Solving the generalized assignment problem: An optimizing and heuristic approach," *INFORMS Journal on Computing*, 15 (2003) 249–266.

[24] K. Nonobe and T. Ibaraki, "An improved tabu search method for the weighted constraint satisfaction problem," *INFOR*, 39 (2001) 131–151.

[25] H. Ono, M. Yagiura and T. Ibaraki, "A decomposability index in logical analysis of data," submitted for publication, 2003.

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

[27] J.S. Park, B.H. Lim and Y. Lee, "A Lagrangian dual-based branch-and-bound algorithm for the generalized multi-assignment problem," *Management Science*, 44 (1998) S271–282.

[28] E. Pesch and F. Glover, "TSP ejection chains," *Discrete Applied Mathematics*, 76 (1997) 165–181.

[29] H. Pirkul, "An integer programming model for the allocation of databases in a distributed computer system," *European Journal of Operational Research*, 26 (1986) 401–411.

[30] C. Privault and L. Herault, "Solving a real world assignment problem with a metaheuristic," *Journal of Heuristics*, 4 (1998) 383–398.

[31] M. Racer and M.M. Amini, "A robust heuristic for the generalized assignment problem," *Annals of Operations Research*, 50 (1994) 487–503.

[32] C. Rego, "Relaxed tours and path ejections for the traveling salesman problem," *European Journal of Operational Research*, 106 (1998) 522–538.

[33] C. Rego, "A subpath ejection chain method for the vehicle routing problem," *Management Science*, 44 (1998) 1447–1459.

[34] C. Rego, "Node ejection chains for the vehicle routing problem: sequential and parallel algorithms," HCES-02-01, Hearin Center for Enterprise Science, The University of Mississippi, University, MS, 2001.

[35] C. Rego and F. Glover, "Local Search and Metaheuristics for the Traveling Salesman Problem," in: G. Gutin and A. Punnen, eds., *The Traveling Salesman Problem and Its Variations*, Kluwer Academic Publishers, Netherlands, pp. 309–368, 2002.

[36] C. Rego and C. Roucairol, "A parallel tabu search algorithm using ejection chains for the vehicle routing problem," in: I.H. Osman and J.P. Kelly, eds., *Meta-Heuristics: Theory & Applications*, Kluwer Academic Publishers, Boston, pp. 661–675, 1996.

[37] S. Sahni and T. Gonzalez, "P-complete approximation problems," *J. ACM*, 23 (1976) 555–565.

[38] M. Savelsbergh, "A branch-and-price algorithm for the generalized assignment problem," *Operations Research*, 45 (1997) 831–841.

[39] A. Shtub and K. Kogan, "Capacity planning by the dynamic multi-resource generalized assignment problem (DMRGAP)," *European Journal of Operational Research*, 105 (1998) 91–99.

[40] P.M. Thompson and J.B. Orlin, "The theory of cyclic transfers," Working Paper No. OR 200-89, Operations Research Center, MIT, Cambridge, 1989.

[41] M.A. Trick, "A linear relaxation heuristic for the generalized assignment problem," *Naval Research Logistics*, 39 (1992) 137–151.

[42] M. Yagiura, T. Ibaraki and F. Glover, "An ejection chain approach for the generalized assignment problem," *INFORMS Journal on Computing*, 16 (2004) to appear.

[43] M. Yagiura, T. Ibaraki and F. Glover, "A path relinking approach for the generalized assignment problem," *Proc. International Symposium on Scheduling*, Japan, June 4–6 (2002) 105–108.

A Probabilistic analysis

We analyze the probability that a cycle in an improvement graph is subset-disjoint when the assignment $\sigma(j)$ is chosen from $[1, m]$ uniformly at random for each job $j \in J$. For a cycle of length l ($l = 2, 3, \ldots, m$), this probability is

$$1 \cdot \frac{m-1}{m} \cdot \frac{m-2}{m} \cdots \frac{m-l+1}{m} = \frac{m^{\underline{l}}}{m^l},$$

where $m^{\underline{l}} = m(m-1)(m-2)\cdots(m-l+1)$ is the *falling factorial power*. As shown in [25, 42],

$$\frac{m^{\underline{l}}}{m^l} = \begin{cases} 1 + o(1), & \text{if } l \le m^{\frac{1}{2}-\varepsilon} \\ e^{-\frac{1}{2}} + o(1), & \text{if } l = m^{\frac{1}{2}} \\ o(1), & \text{if } l \ge m^{\frac{1}{2}+\varepsilon} \end{cases}$$

holds for any small constant $\varepsilon > 0$, where $o(1) \to 0$ as $m \to +\infty$.[5] That is, a cycle of length less (resp., more) than $\sqrt{m}$ tends to be subset-disjoint (resp., not subset-disjoint) if m is sufficiently large.

[5] Actually, the middle case is not explicitly shown in [25, 42], but the proof is immediate by a similar argument.

Table 1. Results for type C instances

				TS-CS		TS-noCS		TS-WCSP		CPLEX	
n	m	s	LB	best	TTB	best	TTB	best	TTB	best	TTB
100	5	1	†1931	*1931	1.25	*1931	0.61	1933	30.00	*1931	2
100	5	2	†1933	*1933	2.39	*1933	58.54	*1933	15.68	*1933	0
100	5	4	†1943	*1943	172.35	*1943	197.17	1944	91.65	*1943	19
100	5	8	†1950	*1950	61.19	*1950	185.08	1956	179.87	*1950	26
100	10	1	†1402	*1402	5.49	*1402	28.92	*1402	15.87	*1402	9
100	10	2	†1409	*1409	133.34	1410	74.06	1411	261.89	*1409	35
100	10	4	†1419	*1419	37.40	*1419	53.13	*1419	93.59	*1419	38
100	10	8	†1435	1436	147.34	1440	208.26	*1435	227.31	*1435	271
100	20	1	†1243	1245	55.48	1245	43.30	1245	140.22	*1243	8
100	20	2	†1250	1251	45.39	1252	46.16	1253	202.30	*1250	5
100	20	4	†1254	1257	186.09	1256	294.35	1258	135.77	*1254	30
100	20	8	†1267	1269	205.60	1275	261.05	*1267	11.86	1272	199
average			1536.3	1537.1	87.78	1538.0	120.89	1538.0	117.17	1536.8	53.5
200	5	1	†3456	*3456	170.10	3458	5.91	3460	107.54	*3456	35
200	5	2	†3461	*3461	47.64	*3461	219.14	3462	180.82	*3461	17
200	5	4	†3466	*3466	167.53	*3466	196.33	3469	389.58	*3466	173
200	5	8	†3473	*3473	530.30	*3473	447.35	3478	294.82	3474	56
200	10	1	†2806	2807	46.23	2808	156.07	2811	53.63	*2806	249
200	10	2	†2811	*2812	316.68	2813	167.54	*2812	352.65	*2812	37
200	10	4	†2819	2821	399.99	2821	444.46	2823	151.04	*2819	347
200	10	8	2833	*2837	344.24	2842	381.29	2842	480.93	2842	134
200	20	1	†2391	2393	369.54	2399	136.93	2394	31.13	*2391	296
200	20	2	†2397	*2398	313.10	2400	155.53	2403	348.75	*2398	175
200	20	4	2408	*2409	430.56	2416	165.83	2415	15.17	2415	115
200	20	8	2415	2422	47.29	2424	419.02	2423	176.03	*2419	451
average			2894.7	2896.3	265.27	2898.4	241.28	2899.3	215.17	2896.6	173.8

Notes. The mark "†" means that the value is optimal, and the marc "$*$" means that the best cost among the tested algorithms is attained.

Table 2. Results for type D instances

n	m	s	LB	TS-CS best	TS-CS TTB	TS-noCS best	TS-noCS TTB	TS-WCSP best	TS-WCSP TTB	CPLEX best	CPLEX TTB
100	5	1	†6353	*6357	109.87	6359	209.92	6370	115.87	6358	43
100	5	2	6352	*6359	136.10	6371	53.42	6380	106.03	6360	27
100	5	4	6362	*6379	207.25	6381	179.95	6404	297.04	6386	172
100	5	8	6388	*6425	67.89	6428	239.62	6500	264.45	6428	244
100	10	1	6342	*6361	246.00	6377	79.09	6418	192.77	6381	132
100	10	2	6340	*6378	174.39	6405	168.45	6411	241.94	6419	88
100	10	4	6361	*6430	274.92	6438	184.59	6516	126.63	6468	166
100	10	8	6388	*6478	241.80	6520	232.30	6679	255.64	6528	83
100	20	1	6177	*6231	194.94	6270	217.92	6305	204.90	6280	60
100	20	2	6165	*6261	253.83	6305	56.77	6389	223.84	6316	19
100	20	4	6182	*6321	277.59	6331	178.93	6529	58.70	6406	148
100	20	8	6206	6482	234.49	*6481	270.78	6736	34.08	6588	68
average			6301.3	6371.8	201.59	6388.8	172.65	6469.8	176.82	6409.8	104.2
200	5	1	12741	12751	191.63	12756	81.33	12760	87.70	*12750	62
200	5	2	12751	*12766	441.53	12772	110.91	12778	171.48	*12766	534
200	5	4	12745	12775	178.92	12778	151.80	12799	78.63	*12762	286
200	5	8	12755	12805	527.78	12809	292.53	12844	348.73	*12787	432
200	10	1	12426	12463	330.52	12482	555.38	12478	279.46	*12457	27
200	10	2	12431	*12477	476.07	12518	512.70	12533	590.21	12482	578
200	10	4	12432	*12496	471.01	12552	540.32	12586	548.07	12532	112
200	10	8	12448	*12571	481.10	12592	513.42	12812	346.33	12577	209
200	20	1	12230	*12312	230.72	12365	346.77	12409	445.56	12393	297
200	20	2	12227	*12332	597.11	12384	568.76	12442	587.38	12425	21
200	20	4	12237	*12396	576.45	12488	386.71	12605	573.62	12472	425
200	20	8	12254	*12485	337.27	12650	481.50	12918	250.78	12548	132
average			12473.1	12552.4	403.34	12595.5	378.51	12663.7	359.00	12579.3	259.6

Notes. The mark "†" means that the value is optimal, and the marc "$*$" means that the best cost among the tested algorithms is attained.

Table 3. Results for type E instances

				TS-CS		TS-noCS		TS-WCSP		CPLEX	
n	m	s	LB	best	TTB	best	TTB	best	TTB	best	TTB
100	5	1	†12681	*12681	54.08	12682	124.34	12753	2.72	*12681	44
100	5	2	†12692	*12692	120.93	*12692	210.25	12727	28.90	*12692	212
100	5	4	†12810	12812	104.03	12812	83.04	12893	79.48	*12810	24
100	5	8	†12738	*12738	53.98	12739	210.64	12876	152.79	12749	61
100	10	1	†11577	*11577	90.03	11584	91.42	11712	209.98	11584	200
100	10	2	†11582	*11587	179.65	11604	290.33	11665	41.33	11612	80
100	10	4	11636	*11676	289.32	11689	106.93	11864	102.15	11753	257
100	10	8	11619	*11701	260.90	11756	230.60	11836	86.49	11739	258
100	20	1	†8436	*8447	142.39	8488	110.94	8655	144.76	8565	72
100	20	2	10123	*10150	207.09	10219	247.15	10471	79.65	10251	234
100	20	4	10794	*11029	160.57	11075	82.28	11271	117.97	11443	78
100	20	8	11224	*11610	265.33	11817	285.93	11957	143.07	12458	291
average			11492.7	11558.3	160.69	11596.4	172.82	11723.3	99.11	11694.8	150.9
200	5	1	†24930	24933	43.21	24933	399.77	25002	101.77	*24930	18
200	5	2	†24933	24936	430.29	*24933	520.42	25024	519.47	*24933	131
200	5	4	24990	*24999	537.27	25017	555.53	25091	280.81	25003	228
200	5	8	†24943	24950	192.28	24970	449.00	25090	271.98	*24943	68
200	10	1	†23307	*23312	411.12	23326	30.46	23414	494.72	23321	140
200	10	2	23310	*23317	436.01	23333	51.82	23538	303.94	23325	386
200	10	4	23344	*23363	376.65	23412	56.24	23628	83.62	23543	309
200	10	8	23339	23412	198.96	*23410	357.69	23714	550.29	23744	56
200	20	1	†22379	*22386	178.16	22455	519.64	22815	207.84	22457	185
200	20	2	22387	*22408	333.97	22459	121.13	22834	377.76	22558	302
200	20	4	22395	*22439	462.53	22517	566.93	22990	413.64	22782	238
200	20	8	22476	*22614	317.52	—	—	23057	394.42	23482	85
average			23561.1	23589.1	326.50	—	—	23849.8	333.36	23751.8	178.8

Notes. The mark "†" means that the value is optimal, and the marc "$*$" means that the best cost among the tested algorithms is attained.