A Path Relinking Approach for the Multi-Resource Generalized Quadratic Assignment Problem

Mutsunori Yagiura[1], Akira Komiya[2], Kenya Kojima[2], Koji Nonobe[3], Hiroshi Nagamochi[2], Toshihide Ibaraki[4], and Fred Glover[5]

[1] Graduate School of Information Science, Nagoya University, Nagoya, Japan,
yagiura@nagoya-u.jp
[2] Graduate School of Informatics, Kyoto University, Kyoto, Japan,
nag@i.kyoto-u.ac.jp
[3] Faculty of Engineering and Design, Hosei University, Tokyo, Japan,
nonobe@hosei.ac.jp
[4] School of Science and Technology, Kwansei Gakuin University, Sanda, Japan,
ibaraki@ksc.kwansei.ac.jp
[5] Leeds School of Business, University of Colorado, Boulder, USA,
fred.glover@colorado.edu

Abstract. We consider the multi-resource generalized quadratic assignment problem (MR-GQAP), which has many applications in various fields such as production scheduling, and constitutes a natural generalization of the generalized quadratic assignment problem (GQAP) and the multi-resource generalized assignment problem (MRGAP). We propose a new algorithm PR-CS for this problem that proves highly effective. PR-CS features a path relinking approach, which is a mechanism for generating new solutions by combining two or more reference solutions. It also features an ejection chain approach, which is embedded in a neighborhood construction to create more complex and powerful moves. Computational comparisons on benchmark instances show that PR-CS is more effective than existing algorithms for GQAP, and is competitive with existing methods for MRGAP, demonstrating the power of PR-CS for handling these special instances of MR-GQAP without incorporating special tailoring to exploit these instances.

1 Introduction

We consider the *multi-resource generalized quadratic assignment problem* (MR-GQAP), which is a natural generalization of the *generalized quadratic assignment problem* (GQAP) [1, 8] and the *multi-resource generalized assignment problem* (MRGAP) [2, 13]. For this problem, we are given n jobs, m agents, assignment costs of jobs, a cost matrix between jobs, a cost matrix between agents, and coefficients for resource constraints. The objective of MR-GQAP is to find a minimum cost assignment of jobs to agents subject to cardinality constraints and multi-resource constraints for each agent, where the following two types of

costs are considered: One is individual cost associated with each assignment, and the other is mutual cost associated with a pair of assignments. MR-GQAP is NP-hard because both MRGAP and GQAP are NP-hard. MR-GQAP is very general and includes such problems as the graph coloring problem, a special case of the channel assignment problem, and so forth. MR-GQAP is also motivated by some problems emerging from real-world applications such as production scheduling problems in steal industry. It also includes the *quadratic assignment problem* (QAP) as a special case of GQAP, and the *generalized assignment problem* (GAP) as a special case of MRGAP.

For GQAP, Lee and Ma [8] proposed linearization approaches and a branch-and-bound algorithm, and Cordeau et al. [1] have recently proposed a sophisticated memetic algorithm. For MRGAP, Gavish and Pirkul [2] proposed a branch-and-bound algorithm and two simple Lagrangian heuristics, and Yagiura et al. [13] devised a very large-scale neighborhood search algorithm. Nonlinear variants are also discussed, e.g., in [11]. For more restricted special cases such as GAP and QAP, much effort has been devoted to develop efficient exact and heuristic algorithms. To the best of our knowledge, however, not much has been done for GQAP and MRGAP in spite of their practical importance.

In this paper, we propose a heuristic algorithm *PR-CS* (path relinking with chained shift neighborhood) for MR-GQAP. PR-CS features a *path relinking* approach, which provides an *evolutionary* mechanism for generating new solutions by combining two or more reference solutions. The idea of path relinking was proposed by Glover [3, 4], and some of its basic aspects were also introduced in an earlier paper by Ibaraki et al. [6]. For more about the general principles of the path relinking approach, see e.g., [7, 9]. PR-CS also features the idea of *ejection chains* [5], which is embedded in a neighborhood construction to create more complex and powerful moves. We call the resulting neighborhood the *chained shift neighborhood*, which generalizes standard shift and swap neighborhoods. The problem of judging the existence of a feasible solution for MR-GQAP is NP-complete. We therefore allow our search to visit infeasible solutions that may violate resource constraints as well, and evaluate the amount of the violation as penalty. The performance of the algorithm crucially depends on penalty weights, and hence we incorporate an adaptive mechanism for controlling them to maintain a balance between visiting feasible and infeasible regions.

We conduct computational experiments to observe the effectiveness of each component of the above mentioned methodologies, and to confirm that their combination provides a successful framework for algorithm design. We test PR-CS on MR-GQAP instances generated by us, and on benchmark instances of GQAP and MRGAP. We first compare PR-CS with a basic algorithm (without path relinking mechanism) using only the shift and swap neighborhoods, and observe the effectiveness of the path relinking approach and the chained shift neighborhood. We then compare PR-CS with existing algorithms for GQAP and MRGAP, which are specially tailored to these specific problems, and a general solver for the constraint satisfaction problem. The computational results of PR-CS are quite promising considering its generality.

2 Formulation

Given n jobs $J = \{1, 2, \ldots, n\}$ and m agents $I = \{1, 2, \ldots, m\}$, we undertake to determine a minimum cost assignment of each job to exactly one agent under cardinality constraints and multi-resource constraints for each agent, where s resources $K = \{1, 2, \ldots, s\}$ are considered. In this problem, for $i, i' \in I$, $j, j' \in J$, and $k \in K$, the following data are given as the input:

c_{ij}: the cost of processing job j at agent i,
$u_{jj'}$: the cost coefficient between jobs j and j',
$w_{ii'}$: the cost coefficient between agents i and i',
a_{ijk}: the amount of resource k consumed by job j if it is assigned to agent i,
b_{ik}: the upper bound of resource k available at agent i,
t_i^{UB}: the upper bound on the number of jobs assigned to agent i,
t_i^{LB}: the lower bound on the number of jobs assigned to agent i.

Assigning a job j to an 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}. Moreover, for any pair of jobs j, j', assigning jobs j and j' to agents i and i' respectively incurs a cost of $f(u_{jj'}, w_{ii'})$, where $f : R^2 \to R$ is a given function. 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 σ: $J \to I$, 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 problem we consider in this paper is formally described as follows:

$$\text{minimize} \quad cost(\sigma) = \sum_{j \in J} c_{\sigma(j),\, j} + \sum_{j, j' \in J} f\left(u_{jj'}, w_{\sigma(j)\sigma(j')}\right) \tag{1}$$

$$\text{subject to} \quad t_i^{\mathrm{LB}} \leq |J_i^\sigma| \leq t_i^{\mathrm{UB}}, \qquad \forall i \in I \tag{2}$$

$$\sum_{j \in J_i^\sigma} a_{ijk} \leq b_{ik}, \qquad \forall i \in I \text{ and } \forall k \in K. \tag{3}$$

We mainly consider the case with $f(u, w) = uw$, and call the problem the *multi-resource generalized quadratic assignment problem* (MR-GQAP). We call (2) the *cardinality constraints* and (3) the *resource constraints*. This problem includes GQAP as its special case with $f(u, w) = uw$, $s = 1$, $t_i^{\mathrm{LB}} = 0, t_i^{\mathrm{UB}} = n$ for all $i \in I$ and $a_{ij1} = a_{i'j1}$ for all $i, i' \in I$ and $j \in J$. MRGAP is a special case of MR-GQAP with $f(u, w) \equiv 0$ and $t_i^{\mathrm{LB}} = 0, t_i^{\mathrm{UB}} = n$ for all $i \in I$. QAP is a special case of GQAP with $a_{ij1} = 1$ for all i and j (hence resource constraints can also be described as cardinality constraints), and GAP is a special case of MRGAP with $s = 1$. Note that GQAP does not include GAP because the resource constraint of GQAP must satisfy $a_{ij1} = a_{i'j1}$ for all $i, i' \in I$ and $j \in J$. MR-GQAP is NP-hard in the strong sense, and the (supposedly) simpler problem of judging the existence of a feasible solution for GAP is NP-complete, since the partition problem can be reduced to GAP with $m = 2$.

3 Algorithm

Our algorithm PR-CS is based on local search, where the initial solutions of local search are generated by path relinking. We describe its basic components in the following subsections.

3.1 Local Search, Search Space and Neighborhood

Local search starts from an initial solution σ, and repeatedly replaces the current solution σ with a better solution σ' in its *neighborhood* $N(\sigma)$ until no better solution is found in the neighborhood. The resulting solution is called *locally optimal*. Shift and swap neighborhoods, denoted N_{shift} and N_{swap} respectively, are often used in local search methods for assignment type problems, where $N_{\mathrm{shift}}(\sigma)$ is the set of solutions obtainable from σ by changing the assignment of one job, and $N_{\mathrm{swap}}(\sigma)$ is the set of solutions obtainable from σ by exchanging the assignments of two jobs. The sizes of these neighborhoods are $O(mn)$ and $O(n^2)$, respectively. 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 $N_{\mathrm{chain}}(\sigma)$ is the set of solutions σ' obtainable from σ by changing the assignments of l ($l = 2, 3, \ldots, n$) arbitrary jobs $j_1, j_2, \ldots, j_l$ simultaneously so that

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

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 l of jobs shifted in the move. This is based on the idea of ejection chains by Glover [5]. Since the size of such a neighborhood can become exponential in l, we carefully limit its size by utilizing ejection trees to be explained in Section 3.3. Since $|N_{\mathrm{shift}}| \leq |N_{\mathrm{swap}}| \leq |N_{\mathrm{chain}}|$ usually holds, N_{swap} is searched only if N_{shift} does not contain an improving solution, and N_{chain} is searched only if $N_{\mathrm{shift}} \cup N_{\mathrm{swap}}$ does not contain an improving solution.

The search space of our local search is the set of assignments σ that satisfy the cardinality constraints (2), but may violate the resource constraints (3). Note that it is easy to judge the existence of an assignment σ that satisfies (2): There exists a σ satisfying (2) if and only if

$$\sum_{i \in I} t_i^{\mathrm{LB}} \leq n \leq \sum_{i \in I} t_i^{\mathrm{UB}}. \tag{4}$$

In the rest of this paper, we assume (4) without loss of generality. In the shift neighborhood, we only evaluate solutions satisfying (2). Note that all solutions in the swap and chained shift neighborhoods satisfy (2) if the current solution satisfies (2). The search space is connected if both shift and swap

neighborhoods are used; i.e., for any two solutions σ and σ' that satisfy (2), there exists a sequence $\sigma = \sigma_0, \sigma_1, \ldots, \sigma_{l'} = \sigma'$ such that σ_r satisfies (2) and $\sigma_r \in N_{\mathrm{shift}}(\sigma_{r-1}) \cup N_{\mathrm{swap}}(\sigma_{r-1})$ holds for all $r = 1, 2, \ldots, l'$. As the search may visit the infeasible region, we evaluate solutions by an objective function penalized by infeasibility:

$$pcost(\sigma) = cost(\sigma) + \sum_{\substack{i \in I \\ k \in K}} \alpha_{ik} p_{ik}(J_i^\sigma), \tag{5}$$

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 the rules similar to those in [12]. The basic idea is simple and intuitively explained as follows: The weights are updated whenever a locally optimal solution is found, and are increased slightly if no feasible solution is found during the last call to local search, and are decreased otherwise (i.e., at least one feasible solution is found during the search). We omit the details due to space limitation.

For convenience, we denote by LS-SS(σ, σ_{incum}) the local search with the shift and swap neighborhoods that starts from a solution σ, where it improves the solution σ to a locally optimal solution and also updates the incumbent solution σ_{incum} (i.e., the best feasible solution found by then) if it finds a better feasible solution during the search.

3.2 An Efficient Implementation of Neighborhood Search

As it takes $O(n^2 + ns + ms)$ time to calculate *pcost* (5) of one solution from scratch, it takes $O(mn^3 + mn^2 s + m^2 ns)$ time to calculate all the solutions in the shift neighborhood, if we adopt a naive implementation. In this section, we propose an efficient implementation of the shift neighborhood in which it memorizes the changes of *pcost* induced by all shift operations in a table of size $O(mn)$. Below, we consider the cost changes and the resulting penalty incurred by shifting job j from agent $\sigma(j)$ to agent i. Let δ_j^{c-}, δ_{ij}^{c+}, δ_j^{p-}, δ_{ij}^{p+} be defined as follows:

$$\delta_j^{c-} = - \sum_{j' \in J \setminus \{j\}} \left\{ f(u_{jj'}, w_{\sigma(j),\sigma(j')}) + f(u_{j'j}, w_{\sigma(j'),\sigma(j)}) \right\} - c_{\sigma(j),j} - f(u_{jj}, w_{\sigma(j),\sigma(j)}), \tag{6}$$

$$\delta_{ij}^{c+} = \sum_{j' \in J \setminus \{j\}} \left\{ f(u_{jj'}, w_{i,\sigma(j')}) + f(u_{j'j}, w_{\sigma(j'),i}) \right\} + c_{ij} + f(u_{jj}, w_{ii}), \tag{7}$$

$$\delta_j^{p-} = - \sum_{k \in K} \alpha_{\sigma(j),k} \left\{ p_{\sigma(j),k}(J_{\sigma(j)}^\sigma) - p_{\sigma(j),k}(J_{\sigma(j)}^\sigma \setminus \{j\}) \right\}, \tag{8}$$

$$\delta_{ij}^{p+} = \sum_{k \in K} \alpha_{ik} \{ p_{ik}(J_i^\sigma \cup \{j\}) - p_{ik}(J_i^\sigma) \}. \tag{9}$$

We can decompose the operation of shifting a job j from agent $\sigma(j)$ to agent i into two steps; we first remove job j from agent $\sigma(j)$ and insert it into agent i. In this process, δ_j^{c-} and δ_j^{p-} represent the increases (actually, the decreases times -1) of cost and penalty by the removal of job j from agent $\sigma(j)$, and δ_{ij}^{c+} and δ_{ij}^{p+} represent the increases in the cost and penalty, respectively, by the insertion of job j to agent i. We can calculate the values of δ_j^{c-} and δ_{ij}^{c+} in $O(n)$ time. If the amount of resource $k \in K$ used by agent $i \in I$ (i.e., $\sum_{j \in J_i^\sigma} a_{ijk}$) at the current solution σ is memorized in a table (this table can be prepared in $O((m+n)s)$ time), then δ_j^{p-} and δ_{ij}^{p+} can be calculated in $O(s)$ time. It therefore takes $O(mn(n+s))$ time to compute the table of $\delta_j^{c-}, \delta_{ij}^{c+}, \delta_j^{p-}$, and δ_{ij}^{p+} for all i and j.

If the above table is given, the increase in *pcost* by shifting job j from agent $\sigma(j)$ to agent i is given by

$$\delta_{ij} = \delta_j^{c-} + \delta_j^{p-} + \delta_{ij}^{c+} + \delta_{ij}^{p+} \tag{10}$$

($\delta_{ij} < 0$ means that we can get an improved solution by this shift operation), which can be calculated in $O(1)$ time. We can therefore calculate *pcost* of all the solutions in the shift neighborhood in $O(mn)$ time excluding the time to prepare the table.

We then consider the computation time to renew the table when the current solution is changed by a shift operation. Assume that job j' is shifted from agent i' to agent i'', and let $\hat{\delta}_j^{c-}$, $\hat{\delta}_{ij}^{c+}$, $\hat{\delta}_j^{p-}$, $\hat{\delta}_{ij}^{p+}$ denote the values of δ_j^{c-}, δ_{ij}^{c+}, δ_j^{p-}, δ_{ij}^{p+} before the shift operation, respectively. For $j \neq j'$, δ_j^{c-} and δ_{ij}^{c+} after the shift move are given by

$$\begin{aligned}
\delta_j^{c-} &= \hat{\delta}_j^{c-} - f(u_{jj'}, w_{\sigma(j),i''}) - f(u_{j'j}, w_{i'',\sigma(j)}) \\
&\quad + f(u_{jj'}, w_{\sigma(j),i'}) + f(u_{j'j}, w_{i',\sigma(j)}) \\
\delta_{ij}^{c+} &= \hat{\delta}_{ij}^{c+} - f(u_{jj'}, w_{ii'}) - f(u_{j'j}, w_{i'i}) + f(u_{jj'}, w_{ii''}) + f(u_{j'j}, w_{i''i}),
\end{aligned}$$

and the computation time of this update for each pair of i and j is $O(1)$. For the remaining case (i.e., $j = j'$), we need to calculate $\delta_{j'}^{c-}$ by (6), which takes $O(n)$ time, and $\delta_{ij'}^{c+} = \hat{\delta}_{ij'}^{c+}$ holds for all $i \in I$. Therefore it takes $O(mn)$ time to renew the table of δ_j^{c-} and δ_{ij}^{c+} for all pairs of i and j.

For the table of δ_j^{p-} and δ_{ij}^{p+}, we calculate δ_j^{p-} according to (8) for all jobs $j \in J$ such that $\sigma(j) = i'$ or $\sigma(j) = i''$, and calculate δ_{ij}^{p+} according to (9) for all $j \in J$ and $i \in \{i', i''\}$. In this case, we do not need to renew the table for other i and j (i.e., $\delta_j^{p-} = \hat{\delta}_j^{p-}$ holds if $\sigma(j) \neq i', i''$, and $\delta_{ij}^{p+} = \hat{\delta}_{ij}^{p+}$ holds for all $j \in J$ if $i \neq i', i''$). Since the number of δ_j^{p-} and δ_{ij}^{p+} requiring updates is $O(n)$, and it takes $O(s)$ time for each update, it takes $O(ns)$ time to renew δ_j^{p-} and δ_{ij}^{p+} for all i and j.

The time to renew the table of $\sum_{j \in J_i^\sigma} a_{ijk}$ for all i and k is $O(s)$, because we only need to calculate the changes at agents i' and i''. In total, we can renew

the table of δ_j^{c-}, δ_{ij}^{c+}, δ_j^{p-} and δ_{ij}^{p+} for all i and j and that of $\sum_{j \in J_i^\sigma} a_{ijk}$ for all i in $O(n(m+s))$ time.

In conclusion, a shift move can be executed in $O(n(m+s))$ time once the tables are initialized. Note that it takes $O(nm(n+s))$ time to initialize the tables when an initial solution for local search is given or the penalty weights α_{ik} are changed. Although time for initializing the tables is larger than the computation time needed for each move, we can usually ignore it because the number of moves in a local search is much larger than the number of initialization of tables. For many instances, $s \ll m$ holds, and the computation time for a shift move becomes $O(mn)$, which is the same as the size of N_{shift}. In such cases, we can evaluate one solution in the shift neighborhood in $O(1)$ amortized time.

Based on a similar idea, we can evaluate a solution in the swap neighborhood in $O(s)$ time using $\delta_j^{c-}, \delta_{ij}^{c+}, \delta_j^{p-}$ and δ_{ij}^{p+}, and renew the tables in $O(n(m+s))$ time. (The details are omitted due to space limitation.) For many instances, s can be considered as a fixed constant and $m \leq n$ hold, and hence the computation time for a move becomes $O(n^2)$, which is the same as the size of N_{swap}.

3.3 Search in the Chained Shift Neighborhood

In this section, we briefly explain the idea of our algorithm for finding an improved solution in the chained shift neighborhood using ejection trees. The ejection tree is a rooted tree, in which each vertex corresponds to a job, and the path from the root to a vertex corresponds to a chained shift move.

We consider a set of n ejection trees $T(\sigma,1), T(\sigma,2), \ldots, T(\sigma,n)$ corresponding to the current solution σ. The root vertex of $T(\sigma,j)$ corresponds to job j, and other vertices correspond to other jobs. Let $j(v)$ denote the job assigned to a vertex v, and $\rho(v)$ denote the parent of v with depth $d_v \geq 1$. Let $j_0^v (= j), j_1^v, \ldots, j_{d_v}^v$ denote the sequence of jobs in the path from the root to a vertex v in depth d_v. Then the chained shift move corresponding to this path is as follows:

$$\begin{aligned} &\sigma'(j_d^v) = \sigma(j_{d-1}^v), \quad d = 1, 2, \ldots, d_v \\ &\sigma'(j_0^v) = \sigma(j_{d_v}^v), \end{aligned}$$

where σ' is the new solution generated by the move. Let σ_v be the solution obtained by the chained shift operation that corresponds to the path to v from its root.

It is clear that we can generate all possible solutions in the chained shift neighborhood by considering appropriate ejection trees; however, generating all solutions in this neighborhood is not realistic. We therefore limit the search by the following heuristic rules.

- The search is restricted to the vertices of depth $\leq d_{\max}$ (a parameter).
- In each depth d, we choose the vertices with the smallest $\lceil \gamma/d \rceil$ (γ is a parameter) values of $\Delta^-(v)$ among the set of vertices generated in depth d (≥ 1), and generate only the descendants of the chosen vertices, where $\Delta^-(v)$ is the difference in *pcost* between the current solution σ and the

incomplete solution obtained by ejecting the assignment of the job j_0^v from the solution σ_v.

In the experiment in Section 4, we set $d_{\max} = \min\{m, 5\}$ and $\gamma = 4$.

We implement our algorithm so that it evaluates each solution σ_v in $O(d_v + s)$ time, by using an idea similar to those in Section 3.2; however, its details are quite complicated and are omitted. The whole computation time to search the chained shift neighborhood is $O(n^2 s + nm)$.

Even with such an elaborate implementation, the search in the chained shift neighborhood is still expensive compared to the search in the shift and swap neighborhoods. We therefore invoke the search in the chained shift neighborhood only if the current solution σ is locally optimal with respect to N_{shift} and N_{swap}, and $pcost(\sigma) < 1.01 cost(\sigma_{\text{incum}})$ holds, where σ_{incum} is the incumbent solution. We denote by LS-CS(σ, σ_{incum}) the local search with the shift, swap and chained shift neighborhoods that starts from a solution σ (it receives the current solution σ and the incumbent solution σ_{incum} and modifies them if possible).

3.4 Path Relinking and Reference Set

Path Relinking. We generate initial solutions for LS-SS and/or LS-CS by a path relinking approach, which is a method to construct solutions from two solutions. We define a path to be the set of solutions obtained by repeatedly applying the shift operations from a solution to the other. If two solutions σ_1 and σ_2 are given, the path relinking gives a set of initial solutions S along the path between σ_1 and σ_2. Let J' be the set of jobs assigned to different agents between σ_1 and σ_2. To construct a path from σ_1 to σ_2, in each step, we shift a job $j \in J'$ such that $\delta_{\sigma_2(j),j}$ (i.e., the increase in *pcost* calculated by (10)) is minimum. In our algorithm, we apply local search to at most ω solutions in the path having small *pcost*, where ω is a parameter. For a given pair of σ_1 and σ_2, our path relinking procedure, denoted PR(σ_1, σ_2), is formally described as follows.

Procedure PR(σ_1, σ_2)
Step 1. Let $\sigma := \sigma_1$, $S := \emptyset$ and $J' := \{j \in J \mid \sigma_1(j) \neq \sigma_2(j)\}$.
Step 2. Choose a job $j \in J'$ with minimum $\delta_{\sigma_2(j),j}$, and let $\sigma(j) := \sigma_2(j)$.
Step 3. Let $S := S \cup \{\sigma\}$, and remove j from J'. If $|J'| \geq 2$, return to Step 2; otherwise proceed to Step 4.
Step 4. If $|S| \leq \omega$, output S, otherwise let S' be the set of solutions in S with ω smallest values of *pcost*, and output S'.

Reference Set. We keep a set R of good solutions, and choose the two solutions σ_1 and σ_2 for path relinking from R. It is preferable to keep good solutions in the reference set R to make path relinking more effective, while similar solutions in R are not desirable from the view point of diversification. As candidates for R, we test only locally optimal solutions obtained in the previous call to local search.

We define the distance D between two solutions σ_1 and σ_2 to be the number of jobs that are assigned to different agents; i.e.,

$$D(\sigma_1, \sigma_2) = |\{j \in J \mid \sigma_1(j) \neq \sigma_2(j)\}|.$$

We keep R in such a way that the distance between any two solutions is at least κ (a parameter) for attaining diversification.

We now explain the rule for renewing the reference set R. Let σ be the locally optimal solution obtained in the previous local search, and $\sigma_{\rm worst}$ be a solution with the maximum *pcost* in R. We consider the following two cases: (1) All solutions in R have distances from σ larger than or equal to κ, and (2) otherwise. In case (1), if $|R| < \zeta$ (a parameter) holds, then we add σ into R; otherwise, if $pcost(\sigma) < pcost(\sigma_{\rm worst})$ holds, then we exchange σ and $\sigma_{\rm worst}$, i.e., we let $R := R \setminus \{\sigma_{\rm worst}\} \cup \{\sigma\}$. In case (2), if $pcost(\sigma) < pcost(\sigma')$ holds for all $\sigma' \in R$ such that distance between σ and σ' is smaller than κ, then we add σ into R, and remove all the solutions whose distance from σ is smaller than κ. The procedure to renew the reference set for a given locally optimal solution σ, denoted RNR(σ, R, κ), is summarized as follows.

Procedure RNR(σ, R, κ)

Step 1. If $D(\sigma, \sigma') \geq \kappa$ holds for all $\sigma' \in R$, go to Step 2; otherwise go to Step 3.

Step 2. If $|R| < \zeta$, let $\tau := +\infty$ and $A := \emptyset$; otherwise let $\sigma_{\rm worst}$ be a solution in R such that $pcost(\sigma') \leq pcost(\sigma_{\rm worst})$ for all $\sigma' \in R$, and then let $\tau := pcost(\sigma_{\rm worst})$ and $A := \{\sigma_{\rm worst}\}$. Go to Step 4.

Step 3. Let $A := \{\sigma' \in R \mid D(\sigma, \sigma') < \kappa\}$, and let $\sigma_{\rm best}$ be a solution in A such that $pcost(\sigma') \geq pcost(\sigma_{\rm best})$ for all $\sigma' \in A$. Then let $\tau := pcost(\sigma_{\rm best})$ and go to Step 4.

Step 4. If $pcost(\sigma) < \tau$, then let $R := R \setminus A \cup \{\sigma\}$.

3.5 The Whole Framework of the Algorithm

Our algorithm PR-CS basically applies LS-SS or LS-CS to solutions generated by the path relinking method. Its details are summarized in this section.

At the beginning of the search, the reference set R is empty, and the size of R may increase or decrease when procedure RNR is called. If $|R| < \zeta$ holds and the set of initial solutions generated by the previous call to the path relinking is exhausted, then we apply the local search to randomly generated solutions until $|R| = \zeta$ holds, where the generated locally optimal solutions are added to R according to the rule in Section 3.4.

We also adopt the following rules to realize intensification and diversification. Let $R_{\rm best}$ be the set of solutions in R with ξ smallest values of *pcost*, where ξ is a parameter. Then we choose σ_1 from $R_{\rm best}$ (to intensify the search) and σ_2 from R both randomly. After choosing two solutions, we add random shifts to σ_2 in 1% of jobs for diversification. Moreover, we increase the minimum distance for renewing the reference set to 2κ if the number of calls to local search (LS-SS or LS-CS) after the last update of the incumbent solution is more than or equal to 2θ (θ is a parameter).

As the search in the chained shift neighborhood takes much time compared to shift and swap neighborhoods, we invoke LS-CS only if the current penalty weights are judged as appropriate,[6] and the number r of calls to local search from the last update of the incumbent solution satisfies $\theta \leq r < 2\theta$ or $r \geq 4\theta$. (Recall that we double the parameter κ for procedure RNR for diversification when $r \geq 2\theta$ holds. When this rule applies, we first use LS-SS in its early stage, i.e., when $2\theta \leq r < 4\theta$ holds.)

The whole framework of our algorithm is described as follows, where ω, ζ, ξ and κ are parameters. In the computational experiments in Section 4, we set $\omega = 5$, $\zeta = 10$, $\xi = 3$, $\theta = 2\xi\zeta\omega$, and the initial value of κ to 3. We stop the search when a prespecified amount of time is spent.

Algorithm PR-CS

Phase 1 (Initialization)

Step 1. Let $R := \emptyset$, $S := \emptyset$, $r := 0$ and $\kappa' := \kappa$.

Step 2. Randomly generate a solution that satisfies (2) (recall that this is always possible by the assumption (4)), and apply a local search with the shift and swap neighborhoods, where each solution σ is evaluated by the total penalty excess $\sum_{i \in I,\, k \in K} p_{ik}(J_i^\sigma)$ breaking ties by $cost(\sigma)$. Let σ be the locally optimal solution obtained by the local search. If σ is feasible, let $\sigma_{\rm incum} := \sigma$ ($\sigma_{\rm incum}$ keeps the incumbent solution).

Step 3. Initialize the penalty weights.

Phase 2 (Construction of the reference set)

Step 4. Let σ be a randomly generated solution that satisfies (2). If the current penalty weights are appropriate, $pcost(\sigma) < 1.01 cost(\sigma_{\rm incum})$ holds, and $\theta \leq r < 2\theta$ or $r \geq 4\theta$ holds, invoke LS-CS$(\sigma, \sigma_{\rm incum})$; otherwise invoke LS-SS$(\sigma, \sigma_{\rm incum})$. Let $r := r+1$. If $r = 2\theta$, then let $\kappa' := 2\kappa$. If $\sigma_{\rm incum}$ is updated, then let $r := 0$ and $\kappa' := \kappa$. Update the penalty weights.

Step 5. Invoke RNR(σ, R, κ'). If the stopping criterion is satisfied, output the best feasible solution $\sigma_{\rm incum}$ found during the search and halt.

Phase 3 (Construction of the set of initial solutions)

Step 6. If $|R| < \zeta$, go to Step 4; otherwise go to Step 7.

Step 7. Let $R_{\rm best}$ be the subset of R containing the solutions with ξ smallest values of $pcost$. Choose two solutions σ_1 and σ_2 ($\sigma_1 \neq \sigma_2$) randomly, σ_1 from $R_{\rm best}$ and σ_2 from R.

Step 8. Apply random shifts to σ_2, and let the new solution be σ_2'. Invoke PR(σ_1, σ_2') and let S be its output.

Phase 4 (Improvement of solutions)

Step 9. Randomly choose a solution σ in S, and remove it from S. Then, if the current penalty weights are appropriate, $pcost(\sigma) < 1.01 cost(\sigma_{\rm incum})$ holds, and $\theta \leq r < 2\theta$ or $r \geq 4\theta$ holds, invoke LS-CS$(\sigma, \sigma_{\rm incum})$; otherwise invoke LS-SS$(\sigma, \sigma_{\rm incum})$. Let $r := r + 1$. If $r = 2\theta$, then let $\kappa' := 2\kappa$. If $\sigma_{\rm incum}$ is updated, then let $r := 0$ and $\kappa' := \kappa$. Update the penalty weights.

[6] We judge the current penalty weights to be appropriate if the rule for incrementing the penalty weights and that for decrementing them are both invoked ten times.

Step 10. Invoke RNR(σ, R, κ'). If the stopping criterion is satisfied, output the best feasible solution $\sigma_{\rm incum}$ found during the search and halt.
Step 11. If $S \neq \emptyset$, go to Step 9, otherwise go to Step 6.

4 Computational Experiments

We conducted computational experiments of our algorithm PR-CS for MRGQAP, and also compared the results with those of existing algorithms for GQAP and MRGAP. PR-CS was coded in C++ language and run on an IBM IntelliStation Z Pro (two Intel Xeon 3.2 GHz processors with 2 GB memory, where the computation was done on a single processor). The instances used in our experiments are available at our site.[7]

4.1 Multi-Resource Generalized Quadratic Assignment Problem

We generated test instances by adding quadratic costs to the benchmark instances of GAP and MRGAP called types C, D and E (see [12, 13] for the details of these types). For each instance, we added three different types of quadratic costs, types 1, 2 and 3. For all the three types, we used $f(u, v) = uv$. In type 1, we set $u_{jj} = 0$ for all j and $w_{ii} = 0$ for all i, and we generated $u_{jj'}$ for all $j \neq j'$ and $w_{ii'}$ for all $i \neq i'$ randomly. In type 2, the quadratic cost takes a positive value only if two jobs are assigned to the same agent; i.e., $u_{jj'} = 0$ (1) if $j = j'$ ($j \neq j'$) and $w_{ii'} = C$ (0) if $i = i'$ ($i \neq i'$) (C is a positive constant chosen from [1, 20]). Type 3 is the cost that takes a positive value only if two jobs are assigned to different agents; i.e., $u_{jj'} = 0$ (1) if $j = j'$ ($j \neq j'$) and $w_{ii'} = 0$ (C) if $i = i'$ ($i \neq i'$) (C is a positive constant chosen from [1, 10]).

To see the effectiveness of path relinking and cyclic neighborhood, we compare our algorithm PR-CS with the random multi-start local search (denoted MLS) that repeatedly calls LS-SS from randomly generated solutions, and the PR-CS algorithm without the chained shift neighborhood (denoted PR-SS). In MLS, we incorporate the adaptive control mechanism of penalty weights, and PR-SS is exactly the same as PR-CS except that it does not invoke LS-CS. We also compare PR-CS with a general solver for the constraint satisfaction problem by Nonobe and Ibaraki (denoted NI)[10]. We also tested CPLEX 9.0.0 (a general mixed integer programming solver); however, it took too much time even to find a feasible solution; e.g., CPLEX could not find a feasible solution for an instance of $n = 100$ and $m = 10$ in one hour on a PC with Xeon 3.01 GHz.

PR-SS and MLS were also coded in C++ language and run on the same PC as PR-CS. NI was run on a PC with Intel Pentium III 1 GHz and 1GB memory. The time limits for MLS, PR-SS and PR-CS are 300 seconds, and that for NI is 1200 seconds. The number of runs of each algorithm for each instance is one.

Table 1 shows the costs obtained by the tested algorithms, where the column "type" shows the type of the original GAP or MRGAP instance, the column

[7] URL of our site: http://www.al.cm.is.nagoya-u.ac.jp/~yagiura/mrgqap/

Table 1. Comparison of four algorithms for instances of MR-GQAP

instance	n	m	s	type	quadratic cost	NI	MLS	PR-SS	PR-CS
qc05501	50	5	1	C	1	15897	*15822	*15822	*15822
qc05502	50	5	1	C	2	*1315	*1315	*1315	*1315
qc05503	50	5	1	C	3	2849	*2846	*2846	*2846
qc101001	100	10	1	C	1	41798	40430	*40320	*40320
qc101002	100	10	1	C	2	23290	23150	*23110	*23110
qc101003	100	10	1	C	3	104950	103840	*103710	*103710
mqc1010041	100	10	4	C	1	24263	20182	*20127	20137
mqc1010042	100	10	4	C	2	10472	10458	*10452	*10452
mqc1010043	100	10	4	C	3	89442	86066	*86060	*86060
qc102001	200	10	1	C	1	220452	214912	214094	*214028
qc102002	200	10	1	C	2	14379	14246	14224	*14222
qc102003	200	10	1	C	3	75658	72690	*72215	*72215
mqc1020041	200	10	4	C	1	62465	50833	50387	*50193
mqc1020042	200	10	4	C	2	14375	14263	14237	*14234
mqc1020043	200	10	4	C	3	76275	72631	72519	*72486
qd05501	50	5	1	D	1	38636	*38543	*38543	*38543
qd05502	50	5	1	D	2	24411	24420	*24309	*24309
qd05503	50	5	1	D	3	52770	52620	*52460	*52460
qd101001	100	10	1	D	1	43019	36686	36571	*36540
qd101002	100	10	1	D	2	8559	8328	8178	*8171
qd101003	100	10	1	D	3	15960	15390	*15048	15159
mqd1010041	100	10	4	D	1	23486	18653	*18593	*18593
mqd1010042	100	10	4	D	2	8617	8414	8231	*8228
mqd1010043	100	10	4	D	3	16077	15526	*15174	*15174
qd102001	200	10	1	D	1	269574	250141	243529	*243234
qd102002	200	10	1	D	2	25012	24036	23885	*23884
qd102003	200	10	1	D	3	84833	82215	79172	*79160
mqd1020041	200	10	4	D	1	60342	44220	*43766	*43766
mqd1020042	200	10	4	D	2	25058	24212	*23896	23897
mqd1020043	200	10	4	D	3	84707	79630	78968	*78962
qe05501	50	5	1	E	1	64434	*64148	*64148	*64148
qe05502	50	5	1	E	2	8691	*8635	*8635	*8635
qe05503	50	5	1	E	3	10320	*10309	*10309	*10309
qe101001	100	10	1	E	1	53054	49992	*49210	*49210
qe101002	100	10	1	E	2	30005	29859	29723	*29720
qe101003	100	10	1	E	3	30955	29866	*29541	29548
mqe1010041	100	10	4	E	1	39774	36354	*35607	*35607
mqe1010042	100	10	4	E	2	30359	30133	29799	*29787
mqe1010043	100	10	4	E	3	31106	30139	29639	*29628
qe102001	200	10	1	E	1	324138	306782	303556	*303468
qe102002	200	10	1	E	2	64797	61426	61342	*61338
qe102003	200	10	1	E	3	146321	133018	*131261	131287
mqe1020041	200	10	4	E	1	89056	79750	*78338	78357
mqe1020042	200	10	4	E	2	64698	61809	61398	*61397
mqe1020043	200	10	4	E	3	148245	132445	*131299	131304

Table 2. Comparison of MA and PR-CS for instances of GQAP

			MA		PR-CS		
instance	n	m	value	time (s)	value	TTB (s)	TL (s)
20-15-35	20	15	1471896	96	1471896	0.300	9
20-15-55	20	15	1723638	102	1723638	0.204	10
20-15-75	20	15	1953188	102	1953188	4.856	10
30-06-95	30	6	5160920	114	5160920	0.132	11
30-07-75	30	7	4383923	156	4383923	0.056	15
30-08-55	30	8	3501695	96	3501695	0.496	9
30-10-65	30	10	3620959	210	3620959	1.440	21
30-20-35	30	20	3379359	564	3379359	0.528	50
30-20-55	30	20	3593105	462	3593105	0.756	46
30-20-75	30	20	4050938	522	4050938	0.084	50
30-20-95	30	20	5710645	5232	5710645	511.024	520
35-15-35	35	15	4456670	456	4456670	0.348	45
35-15-55	35	15	4639128	384	4639128	0.492	38
35-15-75	35	15	6301723	396	6301723	26.150	39
35-15-95	35	15	6670264	864	6670264	31.326	50
40-07-75	40	7	7405793	180	7405793	0.308	18
40-09-95	40	9	7667719	1140	7667719	9.697	50
40-10-65	40	10	7265559	240	7265559	0.368	24
50-10-65	50	10	10513029	504	10513029	0.324	50
50-10-75	50	10	11217503	606	11217503	0.544	50
50-10-95	50	10	12845598	1254	12845598	0.276	50
CPU			Sun 1.2 GHz		Xeon 3.2 GHz		

"quadratic cost" shows the type of the quadratic cost, and each '$*$' mark indicates the best objective value among the four algorithms in the table. For MR-GQAP, the performance of PR-CS and PR-SS is much better than MLS and NI, and that of PR-CS is slightly better than PR-SS.

4.2 Generalized Quadratic Assignment Problem

We tested benchmark instances of GQAP [1, 8], and compared PR-CS with the memetic algorithm by Cordeau et al. (denoted MA)[1], which is specially tailored for GQAP. We refer the results of MA reported in [1], in which MA was run on a SUN workstation (1.2 GHz).[8]

For benchmark instances of Lee and Ma [8], both PR-CS and MA succeeded in obtaining exact optimal solutions for all instances. The computation time of PR-CS to obtain an optimal solution for each instance is less than 0.2 seconds, while the computation time of MA ranges from 1 to 8 seconds. These instances

[8] According to the SPEC site (http://www.spec.org/), the values of SPECint2000 are around 700–722 for Sun workstations (1.2 GHz) and around 1289–1579 for Xeon (3.2 GHz). Hence the speed of the Xeon seems to be 2–3 times faster than the Sun.

Table 3. Comparison with NI, TS and MLS for MRGAP instances

type	NI	TS	PR-CS
C	0.140	0.060	0.052
D	2.118	0.885	0.992
E	1.682	0.358	0.464

are somewhat easy and CPLEX was able to solve all of them exactly in less than 10 seconds for two-thirds of the instances, in 10–60 seconds for the rest except one instance, and with more than 200 seconds for one instance.

Table 2 shows the results for benchmark instances of Cordeau et al. [1]. The column "time" shows the computation time of MA, the column "TL" shows the time limit of PR-CS, the column "TTB" shows the time when the best solutions were found and the columns "value" show the objective values of the best solutions obtained by the algorithms. We set the time limit of PR-CS to the smaller value of one tenth of the computation time of MA and 50 seconds, except for instance 30-20-95 for which we set the time limit to one tenth of the computation time of MA. These time limits are not longer than the time spent by MA if the speed of computers are taken into consideration. From Table 2, we can observe that the solution values obtained by the two algorithms are exactly the same for all instances.

These results indicate that PR-CS is at least as good as MA. The computation times reported for MA are the time when it stopped, and hence it is not easy to draw a decisive conclusion; however, PR-CS seems to spend less computation time than MA.

4.3 Multi-Resource Generalized Assignment Problem

We test algorithm PR-CS on benchmark instances of MRGAP with up to 200 jobs, 20 agents and 8 resources, and compare its performance with NI and the tabu search by Yagiura et al. (denoted TS)[13], which is specially tailored for MRGAP. TS and NI for MRGAP were run on a workstation Sun Ultra 2 Model 2300 (300 MHz, 1 GB memory).[9] The time limits of NI and TS are 300 and 600 seconds for $n = 100$ and 200, respectively, and the time limits of PR-CS are 30 and 60 seconds for $n = 100$ and 200, respectively. The number of runs of each algorithm for each instance is one.

Table 3 shows the average gap in % of the costs obtained by the algorithms within the time limit from the lower bound reported in [13], where the average was taken over 24 instances for each of types C, D and E. From the table, we can observe that the performance of PR-CS is much better than NI, and is competitive with TS. It is worth noting that the average gap of PR-CS is slightly better than TS for type C instances. Considering its generality, these competitive results are quite encouraging.

[9] We estimate that Xeon (3.2 GHz) is about 10 times faster than the Sun (300 MHz).

5 Conclusion

In this paper, we proposed a heuristic algorithm PR-CS for MR-GQAP, which incorporated the path relinking and ejection chain components. Through computational experiments on randomly generated instances of MR-GQAP, we confirmed that such algorithmic components are effective for improving the performance of local search. We also observed that PR-CS is more efficient than general purpose solvers developed for constraint satisfaction and mixed integer programming problems. Computational results on benchmark instances of GQAP and MRGAP, special cases of MR-GQAP, disclosed that PR-CS was highly efficient in that its performance was competitive with (or sometimes even better than) existing algorithms specially tailored for GQAP and MRGAP. Considering the generality of our algorithm PR-CS, these results are quite satisfactory.

References

1. Cordeau, J., Gaudioso, M., Laporte, G., Moccia, L., A memetic heuristic for the generalized quadratic assignment problem, *INFORMS Journal on Computing* **18** (2006) 433–443
2. Gavish, B., Pirkul, H., Algorithms for the multi-resource generalized assignment problem, *Management Science* **37** (1991) 695-713
3. Glover, F., Genetic algorithms and scatter search: unsuspected potentials, *Statistics and Computing* **4** (1994) 131–140
4. Glover, F., Tabu search for nonlinear and parametric optimization (with links to genetic algorithms), *Discrete Applied Mathematics* **49** (1994) 231–255
5. Glover, F., 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)
6. Ibaraki, T., Ohashi, T., Mine, H., A heuristic algorithm for mixed-integer programming problems, *Mathematical Programming Study* **2** (1974) 115–136.
7. Laguna, M., Martí, R., *Scatter Search: Methodology and Implementations in C*, Kluwer Academic Publishers, Boston, 2003
8. Lee, C., Ma Z., The generalized quadratic assignment problem, Technical Report. Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, Ontario, Canada, 2003
9. Martí, R., Laguna, M., Glover, F., Principles of scatter search, *European Journal of Operational Research*, **169** (2006) 359–372
10. Nonobe, K., Ibaraki, T., A tabu search approach to the CSP (constraint satisfaction problem) as a general problem solver, *European Journal of Operational Research* **106** (1998) 599–623
11. Voss, S., Heuristics for nonlinear assignment problems, In: P.M. Pardalos, L.S. Pitsoulis, eds., *Nonlinear Assignment Problems*, Kluwer Academic Publishers, Dordrecht, 2000, 175–215.
12. Yagiura, M., Ibaraki, T., Glover, F., An ejection chain approach for the generalized assignment problem, *INFORMS Journal on Computing* **16** (2004) 133–151
13. Yagiura, M., Iwasaki, S., Ibaraki, T., Glover, F., A very large-scale neighborhood search algorithm for the multi-resource generalized assignment problem, *Discrete Optimization* **1** (2004) 87–98