A Path Relinking Approach with Ejection Chains for the Generalized Assignment Problem

Mutsunori Yagiura[†], Toshihide Ibaraki[‡] and Fred Glover[††]

[†]Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan; Email: yagiura@i.kyoto-u.ac.jp

[‡]Department of Informatics, School of Science and Technology, Kwansei Gakuen University, 2-1 Gakuen, Sanda 669-1337, Japan; Email: ibaraki@ksc.kwansei.ac.jp

[††]Leeds School of Business, University of Colorado, Boulder, Colorado 80309-0419; Email: fred.glover@colorado.edu

Abstract

The generalized assignment problem is a classical combinatorial optimization problem known to be NP-hard. It can model a variety of real world applications in location, allocation, machine assignment, and supply chains. The problem has been studied since the late 1960s, and computer codes for practical applications emerged in the early 1970s. We propose a new algorithm for this problem that proves to be more effective than previously existing methods. The algorithm 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 the method is not only effective in general, but is especially effective for types D and E instances, which are known to be very difficult.

Key Words: metaheuristics, scatter search, path relinking, ejection chains, generalized assignment problem

1 Introduction

We introduce an effective metaheuristic algorithm for the generalized assignment problem (GAP), which is one of the representative combinatorial optimization problems known to be NP-hard (e.g., [29]). This problem has many important applications, notably including scheduling, supply chain, location and vehicle routing problems. Consequently, the challenge of designing good exact and/or heuristic algorithms for GAP has significant practical as well as theoretical value (e.g., [6, 24, 28, 33]).

Our algorithm features a path relinking approach [9, 10, 12, 19, 23] associated with adaptive memory programming (tabu search), 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 [9, 10], and some of its basic aspects were also introduced in an earlier paper by Ibaraki, Ohashi and Mine [16]. For more about the general principles of the path relinking approach, see e.g., [19, 23]. Preliminary results of our path relinking approach were reported in [31, 32]. To the best of our knowledge, our paper [32] (a preliminary extended abstract of this paper) was the first paper that reported computational results of a path relinking approach for the GAP, in which results for benchmark instances with up to 200 jobs were reported. Alfandari, Plateau and Tolla [1] independently proposed another effective path relinking algorithm slightly earlier, although without reporting any computational results in the original version of their paper. (The full version [2] containing computational results appeared somewhat later, and we compare their method with ours in Section 4.) Our algorithm also features an ejection chain approach, likewise associated with tabu search [11, 33], where Lagrangian relaxation provides adjusted cost information to guide the neighborhood search to promising solutions. Rego and Glover suggested in Section 4.3 of [27] that combining ejection chain methods and path relinking would be fruitful. Moreover, we incorporate an automatic mechanism for adjusting search parameters, to maintain a balance between visits to feasible and infeasible regions.

Computational comparisons are conducted on benchmark GAP instances known as types B, C, D and E. These test problems are taken from the OR-Library,[1] which is the primary repository for such problems, and are supplemented by additional test instances generated by ourselves. The proposed algorithm is compared with many existing heuristic algorithms including the recent path relinking approach by Alfandari, Plateau and Tolla [2, 3], tabu search by Díaz and Fernández [6], a Lagrangian heuristic algorithm by Haddadi and Ouzia [14], tabu search by Yagiura, Ibaraki and Glover [33], variable depth search algorithms by Yagiura, Yamaguchi and Ibaraki [34, 35], another variable depth search algorithm by Racer and Amini [26], tabu search by Laguna et al. [18], MAX-MIN ant system by Lourenço and Serra [20], genetic algorithm by Chu and Beasley [4] and a mixed integer programming solver CPLEX 6.5. The results show that our GAP method is highly effective, especially for instances of types D and E, which have been established as the most difficult problem classes.

Our algorithm is confirmed by extensive computational experiment to be efficient and robust, both in relation to parameter settings and variations in problem structures. The outcomes indicate that useful benefits result by combining path relinking and ejection chain strategies associated with adaptive memory methods, and by making use of classical relaxation methodology. The resulting method yields a powerful and effective tool for practical applications.

[1]URL of OR-Library: http://mscmga.ms.ic.ac.uk/jeb/orlib/gapinfo.html

2 Generalized Assignment Problem

2.1 Definition of the 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 a resource constraint for each agent. Assigning job j to agent i incurs a cost of c_{ij} and consumes an amount a_{ij} of resource, whereas the total amount of the resource available at agent i is b_i. An assignment is a mapping $\sigma\colon J \to I$, where $\sigma(j) = i$ means that job j is assigned to agent i. For convenience, we define a 0–1 variable x_{ij} for each pair of $i \in I$ and $j \in J$ by

$$x_{ij} = 1 \iff \sigma(j) = i.$$

Then the *generalized assignment problem* (GAP) is formulated as follows:

$$\begin{array}{lll} \text{minimize} & cost(\sigma) = \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} & \\ \text{subject to} & \sum_{j \in J} a_{ij} x_{ij} \le b_i, & \forall i \in I \\ & \sum_{i \in I} x_{ij} = 1, & \forall j \in J \\ & x_{ij} \in \{0, 1\}, & \forall i \in I \text{ and } \forall j \in J. \end{array} \tag{1}$$

GAP is known to be NP-hard (e.g., [29]), and the (supposedly) simpler problem of judging the existence of a feasible solution for GAP is NP-complete, since the partition problem [8] can be reduced to this problem with $m = 2$.

2.2 Generation of Instances

As we will discuss some computational results in Section 3, we first explain the instances used for the experiments. There are five types of benchmark instances called types A, B, C, D and E [4, 18, 21]. Out of these, we use four types B, C, D and E, since type A is too easy to see differences among the tested algorithms. Type B instances are also easier than types C, D and E, and hence we report limited results for this type. Instances of these types are generated as follows:

Type B: a_{ij} are random integers from [5, 25], c_{ij} are random integers from [10, 50], and $b_i = 0.7\left\{0.6(n/m)15 + 0.4\max_{i \in I} \sum_{j \in J, i_j^{\min} = i} a_{ij}\right\}$, where $i_j^{\min} = \min\{i \mid c_{ij} \le c_{kj},\ \forall k \in I\}$.

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

Type D: a_{ij} are random integers from [1, 100], $c_{ij} = 111 - a_{ij} + e_1$, where e_1 are random integers from $[-10, 10]$, and $b_i = 0.8 \sum_{j \in J} a_{ij}/m$.

Type E: $a_{ij} = 1 - 10 \ln e_2$, where e_2 are random numbers from (0, 1], $c_{ij} = 1000/a_{ij} - 10e_3$, where e_3 are random numbers from [0, 1], and $b_i = 0.8 \sum_{j \in J} a_{ij}/m$.

Types D and E are somewhat harder than other types, since c_{ij} and a_{ij} are inversely correlated.[2] We tested the following two sets of problem instances.

MEDIUM: Total of 24 instances of types B, C, D and E with n up to 200, where each type consists of six instances. Among them, types B, C and D instances were taken from OR-Library,[3] and type E instances were generated by ourselves, which are available at our site.[4]

LARGE: Total of 27 instances of types C, D and E with n up to 1600, where each type consists of nine instances. All of these instances were generated by ourselves, and are available at our site.

3 Algorithm

3.1 The Path Relinking Algorithm

Our algorithm, called PREC (path relinking & ejection chains), 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, denoted N_{shift} and N_{swap} respectively, are usually used in local search methods for GAP, where

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

The size of these neighborhoods are $O(mn)$ and $O(n^2)$, respectively.

Our algorithm uses an ejection chain neighborhood, which was proposed in our previous work [33]. Though the details of this neighborhood are explained in our previous paper, we will briefly illustrate its basic idea, since the use of ejection chains is one of the keys to the success of our algorithm. An ejection chain move considered in this paper is a sequence of shift moves, in which every two successive moves share a common agent. The ejection chain neighborhood is the set of solutions obtainable by such ejection chain moves. More precisely, the ejection chain neighborhood 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

$$\sigma'(j_r) = \sigma(j_{r-1}), \quad r = 2, 3, \ldots, l,$$

where $\sigma'(j_1)$ is arbitrary. 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}. An ejection chain move is called *cyclic* if $\sigma'(j_1) = \sigma(j_l)$

[2]Types D and E instances should be solved as minimization problems; otherwise they are trivial.
[3]URL of OR-Library: http://mscmga.ms.ic.ac.uk/jeb/orlib/gapinfo.html
[4]URL of our site: http://www-or.amp.i.kyoto-u.ac.jp/~yagiura/gap/

holds, i.e., the first job j_1 is inserted into the agent from which the last job j_l is ejected. The *length* of an ejection chain move is the number of shift moves l in the sequence. Both the shift and swap neighborhoods are subsets of the ejection chain neighborhood, since a shift move is an ejection chain move of length one, and a swap move is a cyclic ejection chain move of length two. However, the size of the ejection chain neighborhood can become exponential unless intelligently controlled. Therefore, in our implementation, we consider only a special subset of alternatives that is divided into three neighborhoods called *shift*, *double shift*, and *long chain*, where a double shift move is an ejection chain move of length two, and a long chain move is an ejection chain of any length.

These neighborhoods are further suppressed to manageable sizes by using heuristic rules based on the Lagrangian relaxation of GAP, in which the assignment constraints (i.e., $\sum_{i\in I} x_{ij} = 1,\ \forall j \in J$) are relaxed. For the double shift neighborhood, the number of candidates for j_2 is restricted to $\max\{m, \log n\}$ when j_1 is fixed, and for the long chain neighborhood, only one candidate for j_l is considered when $j_1, j_2, \ldots, j_{l-1}$ are fixed. See Appendix A for more details of this part. As a result, the sizes of shift, double shift, and long chain neighborhoods become $O(mn)$, $O(n \max\{m, \log n\})$ and $O(n^2)$, respectively. In [33], we also showed that the expected size of the long chain neighborhood was $O(n^{(3/2)+\varepsilon})$ for an arbitrarily small positive ε under a simplified random model, and observed that this expected size accurately represented reality. In our algorithm, these three neighborhoods are used alternately to form an improving phase, which is called *EC probe* in this paper.

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

$$pcost(\sigma) = cost(\sigma) + \sum_{i\in I} \alpha_i p_i(\sigma), \tag{2}$$

where $p_i(\sigma) = \max\left\{0,\ \sum_{j\in J, \sigma(j)=i} a_{ij} - b_i\right\}$ denotes the amount of infeasibility at agent i. The parameters α_i (> 0) are adaptively controlled during the search by using the algorithm in [33]. Here we briefly explain the basic idea of the adaptive control of the penalty weights. The initial values of α_i are decided by solving a quadratic programming problem (abbreviated as QP), whose main aim is to balance the estimated change in the cost and penalty after shift moves. The definition of the QP is slightly complicated and is omitted here [33]. Then, whenever an EC probe stops at a locally optimal solution σ_{lopt}, α_i values are updated by the following rule. If no feasible solution is found during the previous EC probe after the last update of α_i, we increase the penalty weights by $\alpha_i := \alpha_i(1 + \Delta \cdot p_i(\sigma_{\text{lopt}})/b_i)$ for all $i \in I$, where Δ is chosen so that $\max_{i\in I} \Delta \cdot p_i(\sigma_{\text{lopt}})/b_i = \delta_{\text{inc}}$ holds (δ_{inc} is a parameter). Otherwise (i.e., if at least one feasible solution is found during the previous EC probe after the last update of α_i), we decrease the penalty weights by $\alpha_i := \alpha_i(1 - \delta_{\text{dec}})$ for all $i \in I$ that satisfy $p_i(\sigma_{\text{lopt}}) = 0$, where δ_{dec} is a parameter. In the computational experiments in Section 4, δ_{inc} and δ_{dec} are set to 0.01 and 0.1, respectively, as in [33].

EC probes are applied to solutions generated by path relinking, which is a methodology to generate solutions from two or more solutions, and will be described in the next paragraph. As it is preferable to apply path relinking to solutions of high quality, we keep a reference set R ($|R| = \rho$ is a parameter) of good solutions. Initially R is prepared by applying EC probes to randomly generated solutions. (To generate a random solution σ, $\sigma(j)$ is chosen from $[1, m]$ uniformly at random for each $j \in J$.) Then it is updated by reflecting outcomes of the local search. If feasible solutions are found during the search, the incumbent solution (i.e., the best feasible solution) is always stored as a member of R. Other solutions in R are maintained as follows. Whenever an EC probe stops, the locally optimal solution σ_{lopt} is exchanged with the worst (with respect to *pcost*) solution σ_{worst} in R (excluding the incumbent solution), provided that σ_{lopt} is not worse than σ_{worst} and is different from all solutions in R.

Path relinking is applied to two solutions σ_{A} (initiating solution) and σ_{B} (guiding solution) randomly chosen from R, where a random shift is applied to σ_{B} with probability 1/2 (no shift with the remaining probability 1/2) before applying the path relinking (for the purpose of keeping the diversity of the search), and the resulting solution is redefined to be σ_{B}. Let the distance between two solutions σ and σ' be

$$dist(\sigma, \sigma') = \left|\{j \in J \mid \sigma(j) \neq \sigma'(j)\}\right|, \tag{3}$$

i.e., the number of jobs assigned to different agents, and let $d = dist(\sigma_{\text{A}}, \sigma_{\text{B}})$ be the distance between solutions σ_{A} and σ_{B}. We generate a sequence $\sigma_0, \sigma_1, \ldots, \sigma_d$ of solutions from two solutions σ_{A} and σ_{B} as follows. Starting from $\sigma_0 := \sigma_{\text{A}}$, for $k = 1, 2, \ldots, d$, we define σ_k to be the solution in $N_{\text{shift}}(\sigma_{k-1})$ with the best *pcost* among those whose distances to σ_{B} are smaller than that from σ_{k-1}. To be more precise, denoting

$$N'_{\text{shift}}(\sigma, \sigma_{\text{B}}) = \{\sigma' \in N_{\text{shift}}(\sigma) \mid dist(\sigma', \sigma_{\text{B}}) = dist(\sigma, \sigma_{\text{B}}) - 1\},$$

σ_k is defined by

$$\sigma_k = \arg \min_{\sigma \in N'_{\text{shift}}(\sigma_{k-1}, \sigma_{\text{B}})} pcost(\sigma), \quad k = 1, 2, \ldots, d, \tag{4}$$

where $\sigma_d = \sigma_{\text{B}}$ holds by definition. Let S be the best γ (a parameter) solutions with respect to *pcost* from $N'_{\text{shift}}(\sigma_0, \sigma_{\text{B}}) \cup \{\sigma_2, \sigma_3, \ldots, \sigma_{d-1}\}$.[5] (The reason for including $N'_{\text{shift}}(\sigma_0, \sigma_{\text{B}})$ instead of using only σ_1 will be discussed in Section 3.3.) S is the set of solutions generated by the current path relinking, to which EC probes are applied. The next path relinking is initiated whenever all solutions in S are exhausted as the starting solutions for EC probes.

To make the computation efficient, we use a heap (with *pcost* as the key) to choose S from $N'_{\text{shift}}(\sigma_0, \sigma_{\text{B}}) \cup \{\sigma_2, \sigma_3, \ldots, \sigma_{d-1}\}$. Then the computation time of the path relinking to generate set S from σ_{A} and σ_{B} is $O(n + d^2 + d \log \gamma)$.

[5]We set $S := N'_{\text{shift}}(\sigma_0, \sigma_{\text{B}}) \cup \{\sigma_2, \sigma_3, \ldots, \sigma_{d-1}\}$ in case $|N'_{\text{shift}}(\sigma_0, \sigma_{\text{B}}) \cup \{\sigma_2, \sigma_3, \ldots, \sigma_{d-1}\}| \leq \gamma$ holds.

The proposed algorithm PREC is formally described as follows, where the algorithm EC probe starts from a solution σ, and returns a locally optimal solution with respect to the ejection chain neighborhood, which is denoted as EC_PROBE(σ).[6]

Algorithm PREC

Step 1. Let R be ρ randomly generated solutions. Then, for each $\sigma \in R$, replace σ with $\sigma_{\mathrm{lopt}} = \mathrm{EC\_PROBE}(\sigma)$. Let $S := \emptyset$.

Step 2. If $S \neq \emptyset$, proceed to Step 3. Otherwise ($S = \emptyset$), randomly choose two solutions σ_{A} and σ_{B} from R ($\sigma_{\mathrm{A}} \neq \sigma_{\mathrm{B}}$). With probability 1/2, replace σ_{B} with a solution randomly chosen from $N_{\mathrm{shift}}(\sigma_{\mathrm{B}})$. Then let S be the best γ solutions from $N'_{\mathrm{shift}}(\sigma_0, \sigma_{\mathrm{B}}) \cup \{\sigma_2, \sigma_3, \ldots, \sigma_{d-1}\}$, where $d = dist(\sigma_{\mathrm{A}}, \sigma_{\mathrm{B}})$ and σ_k ($k = 1, 2, \ldots, d-1$) is defined by (4) (with $\sigma_0 = \sigma_{\mathrm{A}}$).

Step 3. Choose a solution $\sigma \in S$ randomly. Let $S := S \setminus \{\sigma\}$ and $\sigma_{\mathrm{lopt}} := \mathrm{EC\_PROBE}(\sigma)$.

Step 4. Let σ_{worst} be the solution that maximizes *pcost* among those in R (excluding the incumbent solution). If $pcost(\sigma_{\mathrm{lopt}}) \leq pcost(\sigma_{\mathrm{worst}})$ holds and σ_{lopt} is different from all solutions in R, replace σ_{worst} with σ_{lopt}.

Step 5. If the stopping criterion is satisfied, output the incumbent solution and stop. Otherwise return to Step 2.

Notes. The penalty weights α_i in (2) are updated whenever EC_PROBE(σ) returns a solution. Once a feasible solution is found during the search, the incumbent solution is always stored as a member of R. The incumbent solution in R is updated (i.e., the previous incumbent solution is replaced with the new one) whenever the incumbent solution is updated during the search.

As for Step 5, various stopping criteria are possible. In the computational experiment in Section 4, we stop the search when the CPU time exceeds a prespecified time limit in order to make fair comparisons.

3.2 Algorithms TSEC and PREC

Here we briefly explain our previous algorithm TSEC (tabu search & ejection chains) [33] in order to contrast the algorithms and clarify the contribution of this paper. Both algorithms TSEC and PREC use EC probe and the adaptive control mechanism of the penalty weights. Thus the only difference is the way they generate starting solutions for EC probes. Algorithm TSEC applies an EC probe to a solution generated by applying a shift move to a good solution

[6] When the solution σ is chosen from S, the solution is improved first by the cyclic double shift neighborhood in EC_PROBE as in [33]. This strategy was confirmed to be effective to avoid short cycling. See [33] for more discussion.

σ_{seed}. Solution σ_{seed} is initially generated randomly, and is replaced with the locally optimal solution σ_{lopt} obtained by the previous EC probe whenever $pcost(\sigma_{\text{lopt}}) \leq pcost(\sigma_{\text{seed}})$ holds. Then the next EC probe is applied to the solution generated by applying the shift move that minimizes *pcost* among those not executed yet from the current σ_{seed}. As the path relinking part is simple, algorithmic contribution of this paper may not be large, but the improvement in the performance is drastic as will be shown in Section 4. The worst result of PREC is often better than the best result of TSEC when each of them are run five times for each instance. The performance of TSEC is already very good compared to other existing metaheuristic algorithms, and designing an algorithm that outperforms TSEC is not easy. One of the main messages of this paper is that even a simple implementation of path relinking approach is already quite powerful, and useful benefits result by combining path relinking and ejection chain strategies associated with strategic oscillation realized by adaptive control of penalty weights.

3.3 Rules in Algorithm PREC

We tested many variations of the rules used in algorithm PREC, which will be described below.

Definition of S. It might be more natural to consider only those solutions $\sigma_1, \sigma_2, \ldots, \sigma_{d-1}$ on the path as the candidates to be included in S, a strategy adopted in our previous paper [32]. Including solutions in $N'_{\text{shift}}(\sigma_0, \sigma_{\text{B}})$ is motivated by the success of our TSEC algorithm [33]. The adopted approach was confirmed to be more powerful for large-scale instances of types D and E if longer computation time is allowed, though the difference in the performance was not large.

We also tested another definition of S, in which S is set to the best γ solutions from $N'_{\text{shift}}(\sigma_0, \sigma_{\text{B}}) \cup N'_{\text{shift}}(\sigma_1, \sigma_{\text{B}}) \cup \cdots \cup N'_{\text{shift}}(\sigma_{d-2}, \sigma_{\text{B}})$. This approach was observed to be as powerful as the other two strategies; however, the computation time to generate set S becomes $O(n + d^2 \log \gamma)$. Note that this computation time may not be negligible in comparison with the time for EC probe, since $d = \Omega(n)$ may hold and the practical neighborhood size of EC probe is much smaller than $O(n^2)$.

As we adopted a nonstandard strategy, some readers might be interested in the details of the comparison. We therefore show the details in Appendix B. An important observation is that the difference in the performance is not large, and in this sense, algorithm PREC is robust against the selection strategies of S. Indeed, the performance of PREC is already much better than that of TSEC even with other two strategies.

Perturbation on the guiding solution σ_{B}. In Step 2, a random perturbation is applied to the guiding solution σ_{B} with probability 1/2. We tested to set this probability 0, i.e., no perturbation is added to σ_{B}, and observed that the performance was not much different. Hence this perturbation may not be necessary; however, we included it just in case to keep the diversity of the reference set R. (For example, if the assignment of a job becomes the same for all solutions in R, it cannot be changed during the generation of paths by the path relinking approach. We would like to add a mechanism to avoid such risk.) More strategic way to keep the diversity might be possible, but we adopted a simple one.

More than one guiding solution. Instead of only one guiding solution σ_{B}, we also tried to use more than one guiding solution. In this case, for $k = 1, 2, \ldots, d = dist(\sigma_{\mathrm{A}}, \sigma_{\mathrm{B}})$, σ_k is chosen from those solutions obtainable from σ_{k-1} by a shift move, where the candidates of shift moves are restricted to those that satisfy the following two conditions: (i) the distance from σ_{k-1} to at least one guiding solution becomes smaller, and (ii) the assignment of the job to be shifted has not been changed while generating $\sigma_1, \sigma_2, \ldots, \sigma_{k-1}$. We compared algorithm PREC with 1, 2, 3, 4, 6 and 9 guiding solutions, and observed that the performance was not much different. Hence we set the number of guiding solutions to one for simplicity.

Distance between solutions in R. In Step 4 of algorithm PREC, we accept the new solution σ_{lopt} to be included in R if $pcost(\sigma_{\mathrm{lopt}}) \leq pcost(\sigma_{\mathrm{worst}})$ holds and σ_{lopt} is different from all solutions in R. Actually, our program code is more flexible in that it can keep the distance between any pair of solutions in R at least $d_{\min}$ (a prespecified parameter). The rule is as follows. Let $R' \subseteq R$ be the set of solutions whose distance from σ_{lopt} is less than $d_{\min}$. The reference set R is unchanged if one of the following three conditions is satisfied: (i) R' includes the incumbent solution or σ_{lopt}, (ii) there is a solution $\sigma \in R'$ that satisfies $pcost(\sigma) < pcost(\sigma_{\mathrm{lopt}})$, or (iii) $pcost(\sigma_{\mathrm{worst}}) < pcost(\sigma_{\mathrm{lopt}})$ holds, where σ_{worst} is the solution that maximizes $pcost$ among those in R (excluding the incumbent solution). Then we let $R := R \cup \{\sigma_{\mathrm{lopt}}\} \setminus R'$ if $R' \neq \emptyset$; otherwise σ_{worst} is replaced with σ_{lopt}. With this scheme, $|R| < \rho$ may hold after updating the reference set. In this case, in Step 3, EC probe is applied to a randomly generated solution instead of a solution from S, and the new solution σ_{lopt} is added to R provided that it is different from all solutions in R. This is repeated until $|R| = \rho$ holds. According to preliminary comparison with $d_{\min} = 1, 2$ and 5, not much difference in the performance was observed, and hence we set $d_{\min} = 1$ for simplicity.

Strategy to choose σ_{A} and σ_{B} from R. In Step 2 of algorithm PREC, we choose σ_{A} and σ_{B} from R randomly. We also tried the following deterministic rule. The initiating solution σ_{A} is the one with the minimum $pcost$ among those having at least one solution not combined yet with σ_{A}. Then the guiding solution σ_{B} is chosen among those not combined with σ_{A} yet according to the following preference: (i) Let $freq(\sigma)$ denote how many times σ is used for path relinking. Then a solution with the minimum $freq(\sigma_{\mathrm{B}})$ is chosen. (ii) If more than one solution has the same value of (i), we choose a solution σ_{B} that maximizes the distance between σ_{A} and σ_{B}. (iii) If more than one solution has the same value in both (i) and (ii), a solution with the minimum $pcost$ is chosen. According to our preliminary computational results, not much difference was observed between the two strategies, and hence we adopted the random rule that is simpler.

Summary. The above comparisons are based on limited computational results (see Appendix B for details) on some instances from set MEDIUM (instances with up to $n = 200$ and $m = 20$), and careful comparisons for larger instances (e.g., set LARGE) might lead to different conclusions. More sophisticated strategic rules may also be possible, and can result in better performance. Pursuing such possibilities is of course one of the important directions of further research. An interesting indication of the above comparisons is that the path relinking approach is quite powerful even in its simplest form and is not sensitive to slight changes in the rules and parameters involved in its framework.

3.4 Historical Notes

After submitting our paper on EC probe [33] (but without path relinking) to a journal in 1999, we investigated how locally optimal solutions are distributed for the maximum satisfiability problem (MAXSAT) and GAP [30]. The observation told that locally optimal solutions were mutually quite distant for type D instances (see Appendix C for details), suggesting that intensification only might not be sufficient for achieving high performance. This motivated us to incorporate a population-based approach to EC probe to diversify the search, independently of the work by Alfandari, Plateau and Tolla [1, 2, 3], where path relinking (algorithm APT) was also investigated. Note that preliminary results of our path relinking approach were presented in 2001 [31], and a preliminary version of this paper has appeared as [32] in 2002, slightly after Alfandari, Plateau and Tolla presented their work (without any computational result in the proceedings) in 2001 [1]. The paper [32] already contained computational results for benchmark instances with up to 200 jobs.

4 Computational Results

In this section, algorithm PREC is evaluated on benchmark instances. PREC was 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. SPECint95 of this workstation is 12.3 according to the SPEC site[7] (Standard Performance Evaluation Corporation). The parameters in PREC were set to $\rho = 20$ and $\gamma = 10$.[8]

[7] URL of SPEC site: http://www.specbench.org/

[8] According to our preliminary experiment on a few instances, we observe that the performance of algorithm PREC is not sensitive against these parameter values. We therefore did not tune them carefully. The performance of PREC might become better if these parameters are carefully tuned, but we believe that the difference in the solution quality is small.

4.1 Path Relinking and Uniform Crossover

We first compared the path relinking approach (PREC) with the uniform crossover for combining solutions of the reference set. Uniform crossover is one of the traditional methods to generate a new solution by combining two (or more) solutions, which is often used in genetic algorithms [5, 13]. The uniform crossover in our computation generates a new solution σ_{new} from two solutions σ_{A} and σ_{B} by randomly choosing $\sigma_{\text{new}}(j)$ from $\{\sigma_{\text{A}}(j),\ \sigma_{\text{B}}(j)\}$ with probability $1/2$ for each $j \in J$ independently.

Table 1 compares the two approaches, which are exactly the same except for the mechanisms of combining solutions, i.e., the 'uniform crossover' algorithm is composed of the same five steps of the PREC algorithm, except that Step 2 is modified as explained above. The results of our previous paper [33], in which EC probe was proposed, are also exhibited for comparison purposes (denoted TSEC). Algorithm TSEC is basically the same as PREC except that it applies EC probes to solutions generated by applying a shift move to good solutions as explained in Section 3.2. Each algorithm was executed five times for each instance, and the minimum, average and maximum costs are reported. The time limit for each run is 150 (resp., 300) seconds for instances with $n = 100$ (resp., 200). The mark '$*$' means that the best average cost is attained among the three methods. For the type C instances, which are quite easy for all methods, no significant differences emerge. However, for the type D and E instances, the path relinking clearly dominates the uniform crossover. In fact, for the type D and E instances, the worst result (as shown in the column "max") obtained by path relinking is often better than the best result (as shown in the column "min") obtained by uniform crossover (e.g., type D, $n = 100$, $m = 20$). Similar tendency is observed if the uniform crossover approach is compared with TSEC. The performance of the uniform crossover approach is worse than TSEC especially for types D and E instances with larger m. This indicates that the random perturbation by uniform crossover changes the structure of the solution too drastically and the intensification ability of the search is lost.

4.2 Comparison with Other Heuristics

Algorithm PREC was compared with eight existing heuristic algorithms:

- tabu search based on ejection chains by Yagiura, Ibaraki and Glover [33] (denoted TSEC),
- two algorithms of branching variable depth search by Yagiura, Yamaguchi and Ibaraki [34] (denoted BVDS-l and BVDS-j),
- variable depth search by Yagiura, Yamaguchi and Ibaraki [35] (denoted VDS),
- variable depth search by Racer and Amini [26] (denoted RA),
- tabu search by Laguna et al. [18] (denoted LKGG),
- tabu search for the general purpose constraint satisfaction problem by Nonobe and Ibaraki [25] (denoted NI),

- a MAX-MIN ant system combined with local search and tabu search by Lourenço and Serra [20] (denoted RLS).

TSEC, BVDS-l, BVDS-j, VDS and RA were coded in C language by ourselves, while the codes of LKGG, NI and RLS were provided by the authors. The codes of LKGG and NI are written in C, and that of RLS is written in FORTRAN 77. The parameters for BVDS-l, BVDS-j and VDS are set to the values reported in [34]. RA does not include any parameter. The parameters for LKGG and NI are set to the default values. The RLS codes include various types of algorithms, which can be combined by choosing appropriate options. Here we chose the option ASH+LS+TS as recommended in [20], and other parameters were set to the default values. For comparison purposes, we include in Table 2

- the results of the genetic algorithm by Chu and Beasley [4] (denoted CB), and
- the tabu search by Díaz and Fernández [6] (denoted DF),

though we did not run these algorithms on our computer. We also tested a commercial solver CPLEX 6.5,[9] whose results are shown in column CPLEX.

In order to observe the effectiveness of the path relinking component, we also tested a simplified version of algorithm PREC in which standard local search with the shift and swap neighborhoods is used instead of EC probe. This is called algorithm PRSS (path relinking with shift and swap). The other part of algorithm PRSS is exactly the same as PREC.

The results of some algorithms for types B and E instances are not available (i.e., not reported in their papers), and are denoted 'n.a.' in the table. Note that algorithm PREC was run only once for each instance in the computational results in this section (i.e., in Tables 2–4) in order to make the comparison fair (the same random seed was used for all instances).

Table 2 shows the best costs obtained by these algorithms within 150 seconds for $n = 100$, and 300 seconds for $n = 200$, respectively, unless otherwise stated below the table. The computation time of RLS and CB are longer than this time limit as discussed in [33]. According to the estimate in [6], the total time of each run of algorithm DF is smaller than ours; however, the results in column DF are the best of 30 runs, and if this fact is taken into consideration, the total time of DF is larger than ours. (See the remark at the end of this subsection.) Table 2 also shows the lower bounds (denoted LB) obtained by solving the Lagrangian relaxation of GAP. Each '$*$' mark indicates that the best cost is attained, and '—' means that no feasible solution was found.

Table 3 shows similar results for larger instances from set LARGE, where the time limits are set to 3000 seconds for $n = 400$, 10000 seconds for $n = 900$ and 50000 seconds for $n = 1600$. For these instances, we only tested algorithms TSEC, MLS, BVDS-l, LKGG and NI, since the results of BVDS-j and YYI are similar to BVDS-l, and RA and RLS are not competitive with others in Table 2. The results of DF and CPLEX are also shown. The computation time for

[9] We also tested CPLEX 8.1.0, but the results of CPLEX 6.5 were slightly better on average.

DF is larger than other algorithms as discussed in the remark below.

From the tables, we can observe that PREC is very effective, especially for the type D and E instances. PREC obtained the best solution for most of the tested instances. It is also interesting to note that the performance of our path relinking approach is already competitive with other existing metaheuristic algorithms even without EC probe. The result of PRSS is slightly worse than PREC and TSEC, but is better than BVDS-l, BVDS-j, YYI, RA, LKGG, NI, RLS and CB. The solution quality of PRSS and DF is similar for instances from set MEDIUM, but PRSS obtained better solutions than DF for all instances from set LARGE. The performance of PRSS and CPLEX is similar for the types B, C and D instances, but PRSS obtained better solutions than CPLEX for most of the type E instances.

Since some recent papers [3, 14] reported computational results for the instances from set MEDIUM using smaller computation time than in Table 2, we also tested our algorithm PREC with a shorter time limit. For each instance, the time limit of PREC was set to the minimum of the following two: (i) 15 (resp., 30) seconds for $n = 100$ (resp., 200), and (ii) a tenth of the execution time reported in [14]. The results are shown in Table 4. Column HO is the result by the Lagrangian heuristic algorithm by Haddadi and Ouzia [14], and column APT is the result by the path-relinking approach by Alfandari, Plateau and Tolla [3]. Column 'best cost' shows the cost of the best solution obtained within the time limit, column 'time to best' shows the CPU seconds when the best solution was found first, column 'time limit' is the time limit, and column 'execution time' is the CPU time when the algorithm stopped (algorithms HO and APT did not use CPU time for the stopping criteria). CPU time of HO is on a Dell/Phoenix 486DX (120MHz), and that for APT is on a Sun SPARC station with an UltraSPARC processor (400MHz). According to our rough estimate of the speed of these computers, a 486DX (120MHz) is about ten times slower than our computer, and an UltraSPARC (400MHz) is slightly faster than our computer (see the remark below). Hence the time limit of algorithm PREC is not larger than the execution time of HO and APT. From the table, we can observe that PREC clearly dominates HO and APT.

Although PREC outperformed APT in Table 4, the reader will be interested in the comparison of PRSS and APT, because they are both based on the path relinking approach and use standard shift and swap neighborhoods. These two algorithms are contrasted as follows: (i) PRSS uses more sophisticated method than APT to control the penalty weights, and (ii) the rules of the path relinking component in PRSS (e.g., to update the reference set) are randomized and are simpler, while those in APT are deterministic and are slightly more complicated. The results are shown in Table 5, where both PRSS and APT were run only once for each instance and the time limit for PRSS was set to the execution time of APT (fractional part was rounded down). According to our estimation, the speed of the computer used for APT is slightly faster than ours, and hence the time limit of PRSS is slightly shorter than the total execution time of

APT (see the remark below). The meaning of columns 'best cost,' 'time to best,' 'time limit' and 'execution time' are the same as in Table 4. PRSS obtained better solutions than APT for 13 instances, while APT obtained better solutions than PRSS for five instances. For the other six instances to which PRSS and APT obtained the same cost, PRSS tends to find the best cost faster. In summary, the performance of PRSS is slightly better than APT. Recall that the rules for the path relinking component in PRSS are simpler than those in APT, which may indicate that path relinking is not sensitive to details in the rules.

The results in this section are summarized as follows:

- The proposed PREC algorithm is very effective and its performance is the best among the tested algorithms.
- Though the rules for the path relinking component is simple, our path relinking approach is efficient even without EC probe. Algorithm PRSS dominates most of the existing metaheuristic algorithms except for PREC and TSEC (i.e., those using EC probe).

Remark (comparison of machines): Algorithm DF was run on a workstation Sun SPARC station 10/30 with 4 HyperSPARC processors (100 MHz), whose SPECint95 is 2.35, while algorithm PREC was run on a Sun Ultra 2 Model 2300 (300MHz), whose SPECint95 is 12.3. The average execution time for each run was reported to be at least 83.6, 316.1 and 875.0 seconds for instances with $n = 100$, 200 and 400, respectively, which are approximately equivalent to 16, 60 and 170 seconds on our computer. Since they reported the best of 30 runs, the total computation time of algorithm DF is estimated to be at least 480, 1800 and 5000 seconds on our computer for $n = 100$, 200 and 400, respectively.

Next we estimate the speed of a Dell/Phoenix 486DX (120MHz). Since we could not find the data for SPECint95, we estimated its speed via SPECint92 of a similar computer. The SPECint92 of a 486DX4 (100MHz) is 54.59, and the ratio (SPECint92)/(SPECint95) is about 0.025 for similar CPUs. Hence we estimated that the SPECint95 of a 486DX (120MHz) is at least 1.36 and it is about ten times as slow as our computer.

Finally, we estimate the speed of an UltraSPARC (400MHz). Again, we could not find the data for SPECint95 of the same CPU, but the value for similar CPUs (e.g., UltraSPARC-II, UltraSPARC-IIi, etc.) ranges from 15–18. We therefore estimated that an UltraSPARC (400MHz) is slightly faster than ours.

4.3 Detailed Results of Our Algorithm

To make more rigorous comparison of PREC and TSEC, we show their detailed results in Table 6, where the time limits are set as in Table 9 of [33]. (As the results for type B instances were not reported in [33], we use the same time limit as in Table 2 for type B instances.) The table shows the minimum, average and maximum cost, the number of runs where the best solution is found, and the average computation time to find the best solution (over those runs in which the best cost is found) among five runs of PREC and TSEC. We also show the best known solutions found during our computational experiments, where the values with '†' are known to be optimal. If the values in columns 'best known' and 'min' of PREC are different, the best known solutions were found by PREC by a different parameter setting except for those with a mark '‡,' which were found by Ishibashi [17] (see the remark below). A mark '•' indicates that the maximum value of PREC among the five runs is better than the minimum value of TSEC.

For many of the instances whose optimal solutions are known, algorithm PREC found the optimal solutions in all of the five runs. It is also observed that the performance of PREC is better than TSEC for types D and E instances. In particular, the number of '•' marks is one for type C instances, 11 for type D instances and three for type E instances, which indicates that PREC is effective especially for type D instances. One of the conceivable reasons for this result can be given from the observation on the distribution of locally optimal solutions, whose data are shown in Appendix C. The average distance between locally optimal solutions is small for type C instances but is large for type D instances, and that for type E instances is in the middle of types C and D instances. This indicates that intensification of the search works well for type C instances but more diversification is necessary for types D and E instances, especially for type D instances.

Remark: The algorithm in [17] is a distributed computation version of PREC, where a PC cluster consisting of 16 computers each with a Pentium 4 2.26GHz was used, and the time limit was set to 300, 500, 1000, 3000 and 5000 seconds, respectively, for instances with $n = 100$, 200, 400, 900 and 1600, except that 10000 seconds were used for the instances with $m = 80$ and $n = 1600$. Here we estimate the speed of a Pentium 4 2.26GHz. Since we could not find the data for SPECint95, we estimate its speed via SPECint2000. The SPECint2000 of a Pentium 4 2.26GHz is 830–909, and the ratio (SPECint2000)/(SPECint95) is about 10 for similar CPUs. Hence a Pentium 4 2.26GHz is about 7 times as fast as our computer. Based on these, the execution time of the algorithm in [17] is roughly estimated to be $7 \times 16 = 112$ times of the above time limits if it is run on our computer sequentially; e.g., 1,120,000 seconds for the instances with $m = 80$ and $n = 1600$, which is much larger than the time limits in Table 6.

5 Conclusion

The proposed path relinking approach (PREC) proves to be highly effective for the generalized assignment problem. Isolating the path relinking component of our algorithm and comparing it to the use of a uniform crossover component discloses that the outcomes from path relinking are significantly superior to those of uniform crossover. More extensive comparisons of the PREC algorithm, testing against other leading heuristic approaches for GAP, confirm the high performance of PREC; it found better solutions than other methods for most of the tested types D and E instances.

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] L. Alfandari, A. Plateau and P. Tolla, "A two-phase path relinking algorithm for the generalized assignment problem," *Proceedings of the Fourth Metaheuristics International Conference*, Porto, Portugal, July 16–20, 2001, pp. 175–179.

[2] L. Alfandari, A. Plateau and P. Tolla, "A path relinking algorithm for the generalized assignment problem," in: M.G.C. Resende and J.P. de Sousa, eds., *Metaheuristics: Computer Decision-Making*, Kluwer Academic Publishers, Boston, 2004, 1–17.

[3] L. Alfandari, A. Plateau and P. Tolla, "A two-phase path relinking algorithm for the generalized assignment problem," Technical Report No. 378, CEDRIC, CNAM, 2002.

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

[5] L. Davis, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.

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

[7] M.L. Fisher, "The Lagrangian relaxation method for solving integer programming problems," *Management Science*, 27 (1981) 1–18.

[8] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, New York, 1979.

[9] F. Glover, "Genetic algorithms and scatter search: unsuspected potentials," *Statistics and Computing* 4 (1994) 131–140.

[10] F. Glover, "Tabu search for nonlinear and parametric optimization (with links to genetic algorithms)," *Discrete Applied Mathematics* 49 (1994) 231–255.

[11] 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.)

[12] F. Glover, "A template for scatter search and path relinking," in: *Artificial Evolution, Lecture Notes in Computer Science*, 1363, J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer and D. Snyers, eds., Springer, pp. 13–54, 1997.

[13] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Massachusetts, 1989.

[14] S. Haddadi and H. Ouzia, "An effective Lagrangian heuristic for the generalized assignment problem," *INFOR* 39 (2001) 351–356.

[15] M. Held and R.M. Karp, "The traveling salesman problem and minimum spanning trees: part II," *Mathematical Programming*, 1 (1971) 6–25.

[16] T. Ibaraki, T. Ohashi and H. Mine, "A heuristic algorithm for mixed-integer programming problems," *Mathematical Programming Study* 2 (1974) 115–136.

[17] M. Ishibashi, "Methodological investigations on parallelization of metaheuristic algorithms (in Japanese)," Master Thesis, Department of Computer Science and Communication Engineering, Graduate School of Information Science and Electrical Engineering, Kyushu University, 2003 (The results are available at http://tcslab.csce.kyushu-u.ac.jp/~isibashi/Research/parallel-prec.html, where this web page includes Japanese characters).

[18] 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.

[19] M. Laguna and R. Martí, *Scatter Search: Methodology and Implementations in C*, Kluwer Academic Publishers, Boston, 2003.

[20] H.R. Lourenço and D. Serra, "Adaptive search heuristics for the generalized assignment problem," *Mathware and Soft Computing* 9 (2002) 209–234.

[21] S. Martello and P. Toth, "An algorithm for the generalized assignment problem," *Proceedings of the Ninth IFORS International Conference on Operational Research, Hamburg, Germany, July 1981*, North-Holland, Amsterdam, pp. 589–603.

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

[23] R. Martí, M. Laguna and F. Glover, "Principles of scatter search," *European Journal of Operational Research*, to appear in this volume.

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

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

[26] M. Racer and M.M. Amini, "A robust heuristic for the generalized assignment problem,"

Annals of Operations Research 50 (1994) 487–503.

[27] 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.

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

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

[30] M. Yagiura and T. Ibaraki, "Local Search," in: P.M. Pardalos and M.G.C. Resende, eds., *Handbook of Applied Optimization*, Oxford University Press, 2002, pp. 104–123.

[31] M. Yagiura, T. Ibaraki and F. Glover, "An effective metaheuristic algorithm for the generalized assignment problem," *2001 IEEE International Conference on Systems, Man and Cybernetics*, Tucson, Arizona, October 7–10, 2001, p. 242.

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

[33] M. Yagiura, T. Ibaraki and F. Glover, "An ejection chain approach for the generalized assignment problem," *INFORMS Journal on Computing*, 16 (2004) 133–151.

[34] M. Yagiura, T. Yamaguchi and T. Ibaraki, "A variable depth search algorithm with branching search for the generalized assignment problem," *Optimization Methods and Software* 10 (1998) 419–441.

[35] M. Yagiura, T. Yamaguchi and T. Ibaraki, "A variable depth search algorithm for the generalized assignment problem," in: S. Voß, S. Martello, I.H. Osman and C. Roucairol, eds., *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Kluwer Academic Publishers, Boston, 1999, pp. 459–471.

A Details of the EC Probe

We construct ejection chains by exploiting the information from a *Lagrangian relaxation problem* of (1):

$$\begin{aligned} L(v) \;=\; & \min \sum_{i\in I}\sum_{j\in J}(c_{ij}-v_j)x_{ij} - \sum_{j\in J} v_j \\ & \text{s.t.} \;\sum_{j\in J} a_{ij}x_{ij} \le b_i,\ \forall i \in I \\ & \qquad 0 \le x_{ij} \le 1,\ \forall i\in I \text{ and } \forall j \in J, \end{aligned} \tag{5}$$

where $v = (v_1, v_2, \ldots, v_n) \in R^n$ is a Lagrangian multiplier vector given to the constraint $\sum_{i\in I} x_{ij} = 1,\ j \in J$, and

$$c_{ij}(v) = c_{ij} - v_j$$

is called the *Lagrangian relative cost.* Problem (5) decomposes into m real-valued knapsack problems, and thus can be solved in $O(mn)$ time (a linear-time algorithm can be found in, e.g., [22]). For any v, $L(v)$ gives a lower bound on the objective value of problem (1), and the Lagrangian dual problem is to find a $v \in R^n$ that maximizes the lower bound $L(v)$. Any optimal solution v^* to the dual of the linear programming (LP) relaxation of (1),

$$\begin{aligned} \max \quad & \sum_{j \in J} v_j - \sum_{i \in I} b_i u_i \\ \text{s.t.} \quad & v_j - a_{ij} u_i \leq c_{ij}, \ \forall i \in I \text{ and } \forall j \in J, \\ & u_i \geq 0, \ \forall i \in I, \end{aligned} \tag{6}$$

is an optimal solution to the Lagrangian dual [7]. However, computing such v^* by solving (6) is expensive for large scale instances. Hence we use the *subgradient method* [7, 15], which is often used for finding a near-optimal v. Let $\tilde{v}$ denote the Lagrangian multiplier vector obtained by the subgradient method. Computation time to obtain $\tilde{v}$ is negligible compared to the whole execution of algorithm PREC.

For a good multiplier vector v (such as v^* or $\tilde{v}$), the Lagrangian relative cost $c_{ij}(v)$ tends to represent desirability of assigning job j to agent i (smaller $c_{ij}(v)$ means the assignment is more preferable). The rules to restrict the candidates in the double shift and long chain neighborhoods are based on this observation. Let

$$avail(j) = \begin{cases} a_{\sigma(j),j} - p_{\sigma(j)}(\sigma), & \text{if } a_{\sigma(j),j} > p_{\sigma(j)}(\sigma) \\ a_{\sigma(j),j}, & \text{otherwise,} \end{cases}$$

which defines the amount of resource made available by the ejection of job j. In a double shift move, after ejecting the first job j_1 from $i_1 = \sigma(j_1)$, the candidates of jobs j_2 for the second shift move are restricted to those j whose $c_{ij}(\tilde{v})$ is the ξth (ξ is a parameter) or smaller among jobs that satisfy $a_{\sigma(j),k} \leq avail(j)$ and $\sigma(j) \neq i_1$, where we set $\xi = \max\{m, \log n\}$. In a long chain move, after ejecting the lth job j_l from $i_l = \sigma(j_l)$ $(l = 1, 2, \ldots, n)$, the $(l+1)$st job to be shifted into i_l is the job j that minimizes $c_{ij}(\tilde{v})$ among those that satisfy $a_{\sigma(j),k} \leq avail(j)$ and $\sigma(j) \neq i_l$. If such job j has already been shifted in the current chain (i.e., $j \in \{j_1, j_2, \ldots, j_l\}$), then long chain moves of length $l+1$ or more starting from the current j_1 are not searched. (Actually, the rules in [33] to generate the double shift and long chain moves are slightly more complicated, but we simplified them here to explain the main idea more clearly.) As a result, the sizes of shift, double shift, and long chain neighborhoods become $O(mn)$, $O(n \max\{m, \log n\})$ and $O(n^2)$, respectively. Sophisticated data structures are used to find the candidates for the next shift move at each iteration of generating ejection moves. More details of the EC probe are found in [33].

B Comparison of Rules in Algorithm PREC

In this section, we show comparative data on various rules in algorithm PREC discussed in Section 3.3.

Definition of S. We compared the three strategies to choose S from the path from σ_{A} to σ_{B}: The best γ solutions are chosen as S from

Strategy I $\sigma_1, \sigma_2, \ldots, \sigma_{d-1}$,

Strategy II $N'_{\mathrm{shift}}(\sigma_0, \sigma_{\mathrm{B}}) \cup \{\sigma_2, \sigma_3, \ldots, \sigma_{d-1}\}$,

Strategy III $N'_{\mathrm{shift}}(\sigma_0, \sigma_{\mathrm{B}}) \cup N'_{\mathrm{shift}}(\sigma_1, \sigma_{\mathrm{B}}) \cup \cdots \cup N'_{\mathrm{shift}}(\sigma_{d-2}, \sigma_{\mathrm{B}})$.

The computational results with these strategies are shown in Table 7. The solution quality of Strategy II is slightly better than the other two for type D and E instances, hence we adopted Strategy II. However, the difference in the performance is quite small. More important observation is that the performance of PREC is better than TSEC (e.g., the worst result (i.e., column max) of PREC is better than the best result (i.e., column min) of TSEC for all type D instances) even if Strategy I or III is adopted.

We then observed the ratio $|N'_{\mathrm{shift}}(\sigma_0, \sigma_{\mathrm{B}}) \cap S|/|S|$ in order to see the behavior of Strategy II for type C, D and E instances with $n = 200$ and $m = 10$. The reader might guess that a large portion of S is occupied by solutions from $N'_{\mathrm{shift}}(\sigma_0, \sigma_{\mathrm{B}})$, since the initiating solution σ_0 is usually good and many promising solutions will be clustered around it. However, this is not the case. When γ was set to 10 (the value adopted in the computational experiments in Section 4), the average ratios were about 1/3, 1/2 and 1/3 for type C, D and E instances, respectively, and the ratio did not change much when γ was decreased to 5 except that the ratio became about 1/4 for type E instance. That is, on average, more than a half of S are taken from solutions on the path $\sigma_2, \sigma_3, \ldots, \sigma_{d-1}$.

Other rules. Tables 8–11 show comparisons of other rules discussed in Section 3.3 for types C, D and E instances from set MEDIUM (six instances for each type). We ran the algorithm five times for each instance with time limits 150 and 300 seconds for $n = 100$ and 200, respectively, and report the average error in % from the lower bounds (LB in Table 2), where the average was taken for 6 (instances) $\times$ 5 (runs) = 30 runs for each instance type. The rules and parameters of algorithm PREC are basically the same as in Section 4 except for those investigated.[10]

Table 8 compares the effect of perturbing guiding solutions, Table 9 exhibits the effect of using more than one guiding solution, Table 10 shows the effect of changing the minimum distance between solutions in the reference set R, and Table 11 is the comparison between random and deterministic rules to choose two solutions σ_{A} and σ_{B} from R. The differences in the performance were small and hence we adopted simple rules as discussed in Section 3.3.

[10]For the definition of S, Strategy III was used in Tables 9 and 11 because these data were taken before we decided to use Strategy II in the experiments in Section 4.

C Distribution of Locally Optimal Solutions

We consider the distribution of distances between locally optimal solutions, which motivated us to apply path relinking to GAP, where the distance between two solutions is defined by (3). Similar results were reported in [30]. For simplicity, basic local search with N_{shift}, N_{swap} or $N_{\text{shift}} \cup N_{\text{swap}}$ was used, where (i) initial solutions were generated randomly, (ii) the penalty weights α_i of (2) were set to a constant ($\alpha_i = 2$ for types C and D and $\alpha_i = 20$ for type E, which are appropriate to obtain feasible solutions frequently as reported in [34, 35]), and (iii) the first admissible move strategy was adopted. Figures 1–3 show the distribution of distances between locally optimal solutions for type C, D and E instances, respectively, of size $n = 400$ and $m = 20$, where 1000 solutions were generated and distances between all pairs are considered. For comparison purposes, we also include the results for randomly generated solutions. The expected distance between two random solutions is $n(1 - 1/m) = 380$. From these figures, we can observe that the average distance between locally optimal solutions is much smaller than that of randomly generated solutions, and the average distance becomes closer if we use larger neighborhoods. This tendency is drastic for the type C instance, but not so clear for the type D instance, i.e., the average distance is relatively large even after the local search. The distance for the type E instance is in the middle of type C and D instances. This indicates that, to solve type D instances efficiently, pure intensification is not appropriate and diversification mechanism is needed. As algorithm TSEC is strongly biased to intensification, this observation motivated us to combine population-based approach to diversify the search of EC probe. The above consideration also explains the fact that algorithm PREC was more effective than TSEC for type D and E instances and that the difference was clearer especially for type D instances.

Table 1. Comparison of PREC, uniform crossover and TSEC (with time limits 150 and 300 seconds for $n = 100$ and 200, respectively, five runs per instance)

			PREC			uniform crossover			TSEC		
type	n	m	min	avg.	max	min	avg.	max	min	avg.	max
C	100	5	1931	*1931.0	1931	1931	*1931.0	1931	1931	*1931.0	1931
C	100	10	1402	*1402.0	1402	1402	*1402.0	1402	1402	*1402.0	1402
C	100	20	1243	*1243.0	1243	1243	*1243.0	1243	1243	*1243.0	1243
C	200	5	3456	*3456.0	3456	3456	*3456.0	3456	3456	*3456.0	3456
C	200	10	2807	2807.0	2807	2806	2806.8	2807	2806	*2806.2	2807
C	200	20	2391	2391.6	2392	2391	*2391.0	2391	2391	2391.6	2392
D	100	5	6353	*6353.0	6353	6353	6354.0	6356	6354	6355.6	6357
D	100	10	6348	*6354.2	6359	6359	6363.4	6370	6356	6359.6	6364
D	100	20	6205	*6213.4	6221	6259	6269.0	6275	6215	6220.0	6226
D	200	5	12744	*12744.6	12745	12744	*12744.6	12746	12744	12745.6	12747
D	200	10	12437	*12439.4	12442	12464	12468.0	12473	12445	12445.4	12446
D	200	20	12264	*12267.6	12275	12349	12356.4	12363	12277	12284.4	12289
E	100	5	12681	*12681.0	12681	12681	12681.2	12682	12681	12681.4	12682
E	100	10	11577	*11577.0	11577	11577	*11577.0	11577	11577	*11577.0	11577
E	100	20	8443	8446.0	8450	8526	8547.4	8563	8439	*8440.6	8443
E	200	5	24930	*24930.0	24930	24930	24930.2	24931	24930	*24930.0	24930
E	200	10	23307	23308.4	23310	23312	23313.0	23315	23307	*23308.0	23310
E	200	20	22379	*22380.0	22381	22451	22493.0	22528	22379	22384.0	22391

Table 2. The best costs obtained by the tested algorithms (with time limits 150 and 300 seconds for $n = 100$ and 200, respectively, one run per instance except CB and DF)

type	n	m	LB	PREC	PRSS	TSEC	BVDS-l	BVDS-j	YYI	RA	LKGG	NI	⋆RLS	⋆⋆CB	⋆⋆⋆DF	CPLEX
B	100	5	1839	*1843	*1843	*1843	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	*1843	*1843	*1843
B	100	10	1407	*1407	*1407	*1407	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	*1407	*1407	*1407
B	100	20	1166	*1166	*1166	*1166	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	*1166	*1166	*1166
B	200	5	3550	*3552	*3552	*3552	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	3553	*3552	*3552
B	200	10	2826	*2827	*2827	*2827	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	2831	2828	*2827
B	200	20	2339	*2339	*2339	*2339	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	2340	2340	*2339
C	100	5	1930	*1931	*1931	*1931	*1931	*1931	*1931	1938	*1931	*1931	1942	*1931	*1931	*1931
C	100	10	1400	*1402	*1402	*1402	*1402	1403	*1402	1405	1403	1403	1407	1403	*1402	*1402
C	100	20	1242	*1243	1245	*1243	1244	1244	1246	1250	1245	1245	1247	1244	*1243	*1243
C	200	5	3455	*3456	*3456	*3456	*3456	3457	3457	3469	3457	3465	3467	3458	3457	*3456
C	200	10	2804	2807	2807	*2806	2809	2808	2809	2835	2812	2817	2818	2814	2807	*2806
C	200	20	2391	*2391	2394	2392	2401	2400	2405	2419	2396	2407	2405	2397	*2391	*2391
D	100	5	6350	*6353	6356	6357	6358	6362	6365	—	6386	6415	6476	6373	6357	6358
D	100	10	6342	6356	6373	6358	6367	6370	6380	6532	6406	6487	6469	6379	*6355	6381
D	100	20	6177	*6211	6235	6221	6275	6245	6284	6428	6297	6368	6358	6269	6220	6280
D	200	5	12741	*12744	12745	12746	12755	12755	12778	—	12788	12973	12923	12796	12747	12750
D	200	10	12426	*12438	12468	12446	12480	12473	†12496	12799	12537	12889	12746	12601	12457	12457
D	200	20	12230	*12269	12345	12284	12440	12318	†12335	12665	12436	12793	12617	12452	12351	12393
E	100	5	12673	*12681	*12681	12682	*12681	12682	12685	12917	††12687	‡12686	12836	n.a.	*12681	*12681
E	100	10	11568	*11577	*11577	*11577	11585	11599	11585	12047	††11641	‡11590	11780	n.a.	11581	11593
E	100	20	8431	8444	8474	*8443	8499	8484	8490	9004	††8522	‡8509	8717	n.a.	8460	8565
E	200	5	24927	*24930	24933	*24930	24942	24933	24948	25649	††25147	‡24958	25317	n.a.	24931	*24930
E	200	10	23302	23310	23311	*23307	23346	23348	23340	24717	††23567	‡23396	23620	n.a.	23318	23321
E	200	20	22377	*22379	22398	22391	22475	22437	†22452	24117	††22659	‡22551	22779	n.a.	22422	22457

†Results after 1,000 seconds on Sun Ultra 2 Model 2300

††Results after 20,000 seconds on Sun Ultra 2 Model 2300

‡Results after 5,000 seconds on Sun Ultra 2 Model 2300

⋆Computation time is reported in [33]

⋆⋆Results in [4]

⋆⋆⋆Results in [6]

Table 3. Best costs of the tested algorithms for larger instances (with time limits 3000, 10000, and 50000 seconds for $n = 400, 900$ and 1600, respectively, one run per instance except DF)

type	n	m	LB	PREC	PRSS	TSEC	MLS	BVDS-l	LKGG	NI	†DF	CPLEX
C	400	10	5596	*5597	*5597	*5597	5645	5605	5608	5613	5598	*5597
C	400	20	4781	*4782	4783	*4782	4868	4795	4792	4793	4786	*4782
C	400	40	4244	4245	4246	*4244	4327	4259	4251	4247	4248	*4244
C	900	15	11339	*11341	11342	*11341	11459	11368	11362	11365	n.a.	11343
C	900	30	9982	*9984	9988	9985	10187	10022	10007	10000	n.a.	9991
C	900	60	9325	*9328	9335	*9328	9533	9386	9341	9340	n.a.	9365
C	1600	20	18802	*18803	18804	*18803	19011	18892	18831	18822	n.a.	18805
C	1600	40	17144	*17145	17148	17147	17457	17262	17170	17165	n.a.	17157
C	1600	80	16284	*16289	16294	16291	16594	16380	16303	16301	n.a.	16324
D	400	10	24959	*24969	25002	24974	25330	25032	25145	26004	25039	25003
D	400	20	24561	*24587	24654	24614	25165	24780	24872	25496	24747	24682
D	400	40	24350	*24417	24563	24463	—	24724	24726	25405	24707	24675
D	900	15	55403	*55414	55539	55435	56277	55614	56423	57504	n.a.	55474
D	900	30	54833	*54868	55131	54910	56101	55210	55918	57630	n.a.	55002
D	900	60	54551	*54606	54810	54666	—	55123	55379	56699	n.a.	54937
D	1600	20	97823	*97837	97961	97870	99358	98248	100171	101122	n.a.	97921
D	1600	40	97105	*97113	97460	97177	99114	97721	99290	100574	n.a.	97323
D	1600	80	97034	*97052	97341	97109	—	98146	98439	100471	n.a.	97512
E	400	10	45745	*45746	45751	*45746	46163	45878	172185	46243	45781	45757
E	400	20	44876	*44879	44894	44882	45628	45079	137153	45492	45007	45138
E	400	40	44557	*44574	44608	44589	45673	44898	63669	45574	44921	44829
E	900	15	102420	*102422	102426	102423	103512	102755	463142	104932	n.a.	102497
E	900	30	100426	*100434	100449	100442	102200	100956	527451	104173	n.a.	100762
E	900	60	100144	*100169	100244	100185	102395	100917	479650	105494	n.a.	100769
E	1600	20	180642	*180646	180649	180647	182590	181143	936609	187207	n.a.	180880
E	1600	40	178293	*178302	178321	178311	181029	179036	1026259	187451	n.a.	178599
E	1600	80	176816	*176857	176969	176866	180744	178205	1026417	189774	n.a.	177926

†Results in [6]

Table 4. The best costs obtained by the tested algorithms with a small time limit (with one run per instance)

			PREC			HO		APT		
			best	time to	time	best	execution	best	time to	execution
type	n	m	cost	best	limit	cost	time	cost	best	time
B	100	5	*1843	1.45	2	*1843	29.33	*1843	10.0	26.3
B	100	10	*1407	0.42	9	*1407	96.61	*1407	7.3	48.1
B	100	20	*1166	2.90	7	*1166	74.97	*1166	11.4	32.8
B	200	5	*3552	1.78	26	*3552	262.38	3553	121.9	194.6
B	200	10	*2827	25.94	30	2832	481.86	2829	37.6	180.2
B	200	20	*2339	10.17	30	2341	475.44	2340	132.7	196.0
C	100	5	*1931	0.78	4	1932	41.64	*1931	6.6	44.8
C	100	10	*1402	2.43	6	1405	64.92	*1402	31.5	51.2
C	100	20	*1244	5.25	12	1248	125.12	*1244	47.5	114.7
C	200	5	*3456	16.53	24	3457	244.31	3458	146.1	231.8
C	200	10	*2807	13.99	30	2810	531.34	2810	104.3	174.9
C	200	20	*2394	4.41	30	2397	749.73	2396	146.4	247.0
D	100	5	*6357	1.41	3	6358	37.13	6365	71.5	79.8
D	100	10	6373	4.02	9	*6369	91.01	6372	77.3	92.3
D	100	20	*6228	10.62	15	6261	183.62	6267	108.8	131.6
D	200	5	*12746	23.30	30	12747	304.18	12747	318.1	339.3
D	200	10	*12445	15.66	30	12460	531.35	12457	105.2	258.0
D	200	20	*12298	25.77	30	12320	1324.58	12333	308.7	528.4
E	100	5	12685	3.12	15	n.a.	n.a.	*12681	96.1	133.4
E	100	10	*11577	11.29	15	n.a.	n.a.	11597	71.9	82.2
E	100	20	*8447	12.46	15	n.a.	n.a.	8541	79.4	104.3
E	200	5	24933	17.66	30	n.a.	n.a.	*24931	184.0	268.2
E	200	10	*23313	24.37	30	n.a.	n.a.	23316	372.9	493.7
E	200	20	*22389	29.46	30	n.a.	n.a.	22453	435.8	688.9

Note: CUP time for PREC is on a Sun Ultra 2 Model 2300 (300MHz), that for HO is on a Dell/Phoenix 486DX (120MHz), and that for APT is on a Sun SPARC station with an UltraSPARC processor (400MHz).

Table 5. Comparison of algorithms PRSS and APT (with one run per instance)

			PRSS			APT		
			best	time to	time	best	time to	execution
type	n	m	cost	best	limit	cost	best	time
B	100	5	*1843	1.41	26	*1843	10.0	26.3
B	100	10	*1407	0.18	48	*1407	7.3	48.1
B	100	20	*1166	7.24	32	*1166	11.4	32.8
B	200	5	*3552	2.59	194	3553	121.9	194.6
B	200	10	*2827	91.64	180	2829	37.6	180.2
B	200	20	*2339	18.21	196	2340	132.7	196.0
C	100	5	*1931	1.34	44	*1931	6.6	44.8
C	100	10	*1402	1.86	51	*1402	31.5	51.2
C	100	20	1245	17.99	114	*1244	47.5	114.7
C	200	5	*3456	2.12	231	3458	146.1	231.8
C	200	10	*2807	48.47	174	2810	104.3	174.9
C	200	20	*2394	191.32	247	2396	146.4	247.0
D	100	5	*6356	14.24	79	6365	71.5	79.8
D	100	10	6373	74.12	92	*6372	77.3	92.3
D	100	20	*6235	129.33	131	6267	108.8	131.6
D	200	5	*12745	249.89	339	12747	318.1	339.3
D	200	10	12468	246.32	258	*12457	105.2	258.0
D	200	20	12334	518.30	528	*12333	308.7	528.4
E	100	5	*12681	3.82	133	*12681	96.1	133.4
E	100	10	*11577	14.64	82	11597	71.9	82.2
E	100	20	*8479	102.63	104	8541	79.4	104.3
E	200	5	24933	201.00	268	*24931	184.0	268.2
E	200	10	*23307	368.17	493	23316	372.9	493.7
E	200	20	*22393	524.50	688	22453	435.8	688.9

Note: CUP time for PRSS is on a Sun Ultra 2 Model 2300 (300MHz), and that for APT is on a Sun SPARC station with an UltraSPARC processor (400MHz).

Table 6. Detailed results of algorithms PREC and TSEC

					PREC					TSEC					
				best	cost of 5 runs			#best	avg. time	cost of 5 runs			#best	avg. time	time
type	n	m	LB	known	min	avg.	max	found	to best	min	avg.	max	found	to best	limit
B	100	5	1839	†1843	1843	*1843.0	1843	5/5	0.95	1843	*1843.0	1843	5/5	0.81	150
B	100	10	1407	†1407	1407	*1407.0	1407	5/5	0.33	1407	*1407.0	1407	5/5	0.22	150
B	100	20	1166	†1166	1166	*1166.0	1166	5/5	3.64	1166	*1166.0	1166	5/5	9.00	150
B	200	5	3550	†3552	3552	*3552.0	3552	5/5	2.35	3552	*3552.0	3552	5/5	3.13	300
B	200	10	2826	†2827	2827	*2827.0	2827	5/5	18.97	2827	*2827.0	2827	5/5	16.00	300
B	200	20	2339	†2339	2339	*2339.0	2339	5/5	11.53	2339	*2339.0	2339	5/5	5.19	300
C	100	5	1930	†1931	1931	*1931.0	1931	5/5	0.52	1931	*1931.0	1931	5/5	0.6	3000
C	100	10	1400	†1402	1402	*1402.0	1402	5/5	3.20	1402	*1402.0	1402	5/5	3.0	3000
C	100	20	1242	†1243	1243	*1243.0	1243	5/5	35.83	1243	*1243.0	1243	5/5	22.5	3000
C	200	5	3455	†3456	3456	*3456.0	3456	5/5	12.09	3456	*3456.0	3456	5/5	3.7	6000
C	200	10	2804	†2806	2806	2806.4	2807	3/5	2710.87	2806	*2806.0	2806	5/5	403.8	6000
C	200	20	2391	†2391	2391	*2391.0	2391	5/5	1268.83	2391	*2391.0	2391	5/5	301.8	6000
C	400	10	5596	†5597	5597	*5597.0	5597	5/5	52.5	5597	*5597.0	5597	5/5	103.4	3000
C	400	20	4781	4782	4782	*4782.0	4782	5/5	306.6	4782	4782.4	4783	3/5	1956.3	3000
C	400	40	4244	†4244	4244	*4244.6	4245	2/5	904.0	4244	*4244.6	4245	2/5	1602.0	3000
C	900	15	11339	11340	11340	11340.8	11341	1/5	4526.8	11340	*11340.4	11341	3/5	5837.2	10000
C	900	30	9982	†9982	9982	*9982.8	9984	3/5	4841.5	9984	9984.6	9985	2/5	5186.7	10000
C	900	60	9325	9327	9328	*9328.0	9328	5/5	4006.7	9328	9329.0	9330	1/5	9259.0	10000
C	1600	20	18802	‡†18802	18803	*18803.0	18803	5/5	3924.2	18803	18803.2	18804	4/5	19484.3	50000
C	1600	40	17144	17145	17145	*17146.2	17147	1/5	33571.0	17147	17147.4	17148	3/5	19102.4	50000
C	1600	80	16284	‡16285	16287	*16288.2	•16289	2/5	46018.6	16291	16292.4	16294	2/5	25493.9	50000
D	100	5	6350	†6353	6353	*6353.0	6353	5/5	83.62	6353	*6353.0	6353	5/5	649.2	3000
D	100	10	6342	6348	6348	*6349.2	6351	3/5	988.07	6349	6351.8	6354	1/5	2440.7	3000
D	100	20	6177	‡6190	6192	*6196.0	•6201	1/5	2254.88	6206	6210.6	6214	1/5	1591.9	3000
D	200	5	12741	12742	12742	*12743.0	12744	1/5	5712.49	12743	12743.2	12744	4/5	3564.8	6000
D	200	10	12426	‡12432	12433	*12436.6	12441	1/5	5520.51	12440	12441.6	12443	1/5	5829.9	6000
D	200	20	12230	‡12241	12245	*12252.6	•12259	1/5	5777.10	12277	12278.6	12281	3/5	1757.7	6000
D	400	10	24959	‡24963	24967	*24969.2	•24971	1/5	1497.1	24974	24976.4	24979	1/5	2611.8	3000
D	400	20	24561	‡24574	24578	*24586.6	•24592	1/5	2238.6	24604	24609.0	24616	1/5	81.6	3000
D	400	40	24350	‡24392	24409	*24419.6	•24433	1/5	2017.9	24456	24461.2	24464	1/5	1385.9	3000
D	900	15	55403	‡55409	55413	*55414.6	•55420	3/5	3543.4	55425	55433.4	55436	1/5	114.2	10000
D	900	30	54833	‡54852	54868	*54874.0	•54878	1/5	7082.8	54903	54908.8	54912	1/5	241.0	10000
D	900	60	54551	‡54568	54595	*54603.4	•54610	1/5	9733.3	54656	54666.6	54680	1/5	1844.0	10000
D	1600	20	97823	‡97832	97837	*97838.8	•97840	1/5	42608.0	97867	97872.4	97877	1/5	153.8	50000
D	1600	40	97105	‡†97105	97106	*97111.0	•97115	1/5	48721.3	97160	97166.0	97177	1/5	1300.7	50000
D	1600	80	97034	‡97035	97047	*97051.0	•97054	1/5	44257.0	97097	97103.0	97110	2/5	6002.2	50000
E	100	5	12673	†12681	12681	*12681.0	12681	5/5	40.36	12681	*12681.0	12681	5/5	97.3	3000
E	100	10	11568	†11577	11577	*11577.0	11577	5/5	10.09	11577	*11577.0	11577	5/5	31.5	3000
E	100	20	8431	†8436	8436	*8437.6	8440	3/5	1800.95	8436	8438.4	8439	1/5	1144.2	3000
E	200	5	24927	†24930	24930	*24930.0	24930	5/5	29.67	24930	*24930.0	24930	5/5	20.0	6000
E	200	10	23302	†23307	23307	23307.4	23308	3/5	554.15	23307	*23307.0	23307	5/5	866.8	6000
E	200	20	22377	†22379	22379	*22379.0	22379	5/5	741.88	22379	*22379.0	22379	5/5	627.8	6000
E	400	10	45745	45746	45746	*45746.0	45746	5/5	1072.8	45746	*45746.0	45746	5/5	863.6	3000
E	400	20	44876	44877	44877	*44878.2	•44879	1/5	2330.3	44882	44883.4	44885	1/5	2665.3	3000
E	400	40	44557	‡44562	44565	*44572.4	44586	1/5	1215.6	44579	44584.6	44589	1/5	2602.0	3000
E	900	15	102420	102421	102421	*102421.2	102422	4/5	3395.0	102422	102423.0	102424	1/5	4701.0	10000
E	900	30	100426	‡100429	100431	*100433.8	•100435	1/5	6445.1	100438	100440.6	100443	2/5	5255.6	10000
E	900	60	100144	‡100153	100169	*100178.0	100194	1/5	9999.9	100177	100181.2	100185	1/5	8139.7	10000
E	1600	20	180642	180646	180646	*180646.2	180647	4/5	19358.8	180647	180647.4	180648	3/5	19142.6	50000
E	1600	40	178293	‡178294	178298	*178300.8	•178303	1/5	46494.2	178311	178313.2	178316	2/5	35026.0	50000
E	1600	80	176816	‡176828	176848	*176856.4	176872	1/5	25161.3	176856	176862.8	176869	1/5	49790.3	50000

†: An exact optimal solution
‡: A solution found by Ishibashi [17]
•: The maximum value of PREC is better than the minimum value of TSEC.

Table 7. Comparison of strategies to choose S in PREC (with time limits 3000, 10000 and 50000 seconds for $n = 400$, 900 and 1600, respectively, five runs per instance)

			PREC									TSEC	
			Strategy I			Strategy II			Strategy III				
			min	avg.	max	min	avg.	max	min	avg.	max	min	avg.	max
C	400	10	5597	*5597.0	5597	5597	*5597.0	5597	5597	*5597.0	5597	5597	*5597.0	5597
C	400	20	4782	*4782.0	4782	4782	*4782.0	4782	4782	*4782.0	4782	4782	4782.4	4783
C	400	40	4244	4244.6	4245	4244	4244.6	4245	4244	*4244.4	4245	4244	4244.6	4245
C	900	15	11340	*11340.4	11341	11340	11340.8	11341	11340	11340.8	11341	11340	*11340.4	11341
C	900	30	9982	*9982.4	9984	9982	9982.8	9984	9983	9983.8	9984	9984	9984.6	9985
C	900	60	9327	*9327.4	9328	9328	9328.0	9328	9327	*9327.4	9328	9328	9329.0	9330
C	1600	20	18803	*18803.0	18803	18803	*18803.0	18803	18803	*18803.0	18803	18803	18803.2	18804
C	1600	40	17145	*17145.8	17146	17145	17146.2	17147	17145	*17145.8	17146	17147	17147.4	17148
C	1600	80	16287	16288.2	16289	16287	16288.2	16289	16288	*16288.0	16288	16291	16292.4	16294
D	400	10	24967	24968.4	24969	24967	24969.2	24971	24964	*24966.4	24969	24974	24976.4	24979
D	400	20	24586	24587.6	24590	24578	*24586.6	24592	24583	24587.6	24592	24604	24609.0	24616
D	400	40	24422	24432.4	24447	24409	*24419.6	24433	24416	24425.4	24432	24456	24461.2	24464
D	900	15	55413	55417.4	55422	55413	*55414.6	55420	55414	55415.8	55418	55425	55433.4	55436
D	900	30	54868	*54871.2	54877	54868	54874.0	54878	54866	54871.4	54877	54903	54908.8	54912
D	900	60	54599	54609.8	54621	54595	*54603.4	54610	54598	54607.6	54618	54656	54666.6	54680
D	1600	20	97841	97843.2	97844	97837	*97838.8	97840	97836	97840.0	97845	97867	97872.4	97877
D	1600	40	97114	97115.2	97117	97106	*97111.0	97115	97110	97114.8	97119	97160	97166.0	97177
D	1600	80	97047	97054.2	97064	97047	97051.0	97054	97045	*97050.4	97059	97097	97103.0	97110
E	400	10	45746	*45746.0	45746	45746	*45746.0	45746	45746	45746.4	45747	45746	*45746.0	45746
E	400	20	44877	44880.4	44884	44877	*44878.2	44879	44877	44879.2	44882	44882	44883.4	44885
E	400	40	44570	44576.0	44582	44565	*44572.4	44586	44569	44574.4	44584	44579	44584.6	44589
E	900	15	102421	*102421.0	102421	102421	102421.2	102422	102421	102421.2	102422	102422	102423.0	102424
E	900	30	100432	100433.4	100436	100431	100433.8	100435	100430	*100431.4	100433	100438	100440.6	100443
E	900	60	100171	100178.0	100188	100169	100178.0	100194	100163	*100171.2	100182	100177	100181.2	100185
E	1600	20	180646	180646.4	180647	180646	*180646.2	180647	180646	180646.4	180647	180647	180647.4	180648
E	1600	40	178300	178302.2	178306	178298	*178300.8	178303	178300	178301.2	178303	178311	178313.2	178316
E	1600	80	176847	176860.8	176895	176848	176856.4	176872	176849	*176856.0	176865	176856	176862.8	176869

Table 8. Effect of perturbing guiding solutions (average error in % from LB; with time limits 150 and 300 seconds for $n = 100$ and 200, respectively, five runs per instance)

type	with perturbation	without perturbation
C	0.073	0.073
D	0.212	0.209
E	0.062	0.050

Table 9. Effect of using more than one guiding solution (average error in % from LB; with time limits 150 and 300 seconds for $n = 100$ and 200, respectively, five runs per instance)

	the number of guiding solutions					
type	1	2	3	4	6	9
C	0.074	0.069	0.072	0.070	0.071	0.076
D	0.199	0.212	0.195	0.213	0.212	0.231
E	0.058	0.053	0.057	0.050	0.054	0.054

Table 10. Effect of restricting the distance between solutions in the reference set (average error in % from LB; with time limits 150 and 300 seconds for $n = 100$ and 200, respectively, five runs per instance)

	minimum distance		
type	1	2	5
C	0.073	0.073	0.068
D	0.212	0.187	0.203
E	0.062	0.061	0.056

Table 11. Comparison of two strategies to choose initiating and guiding solutions from R (average error in % from LB; with time limits 150 and 300 seconds for $n = 100$ and 200, respectively, five runs per instance)

type	random	deterministic
C	0.074	0.071
D	0.199	0.205
E	0.058	0.051

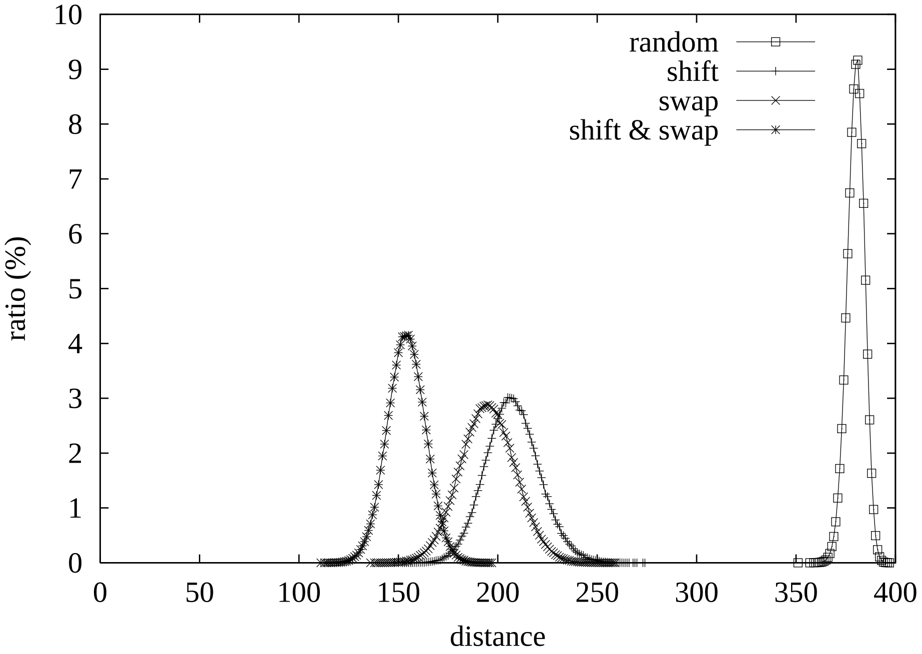

Figure 1. The distribution of distances between locally optimal solutions for the type C instance with $n = 400$ and $m = 20$

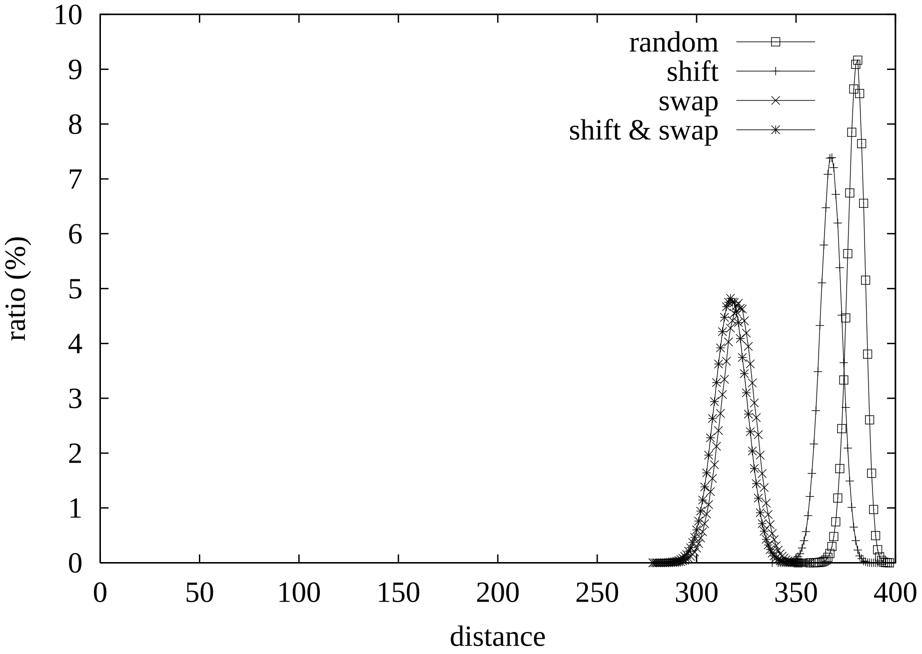

Figure 2. The distribution of distances between locally optimal solutions for the type D instance with $n = 400$ and $m = 20$

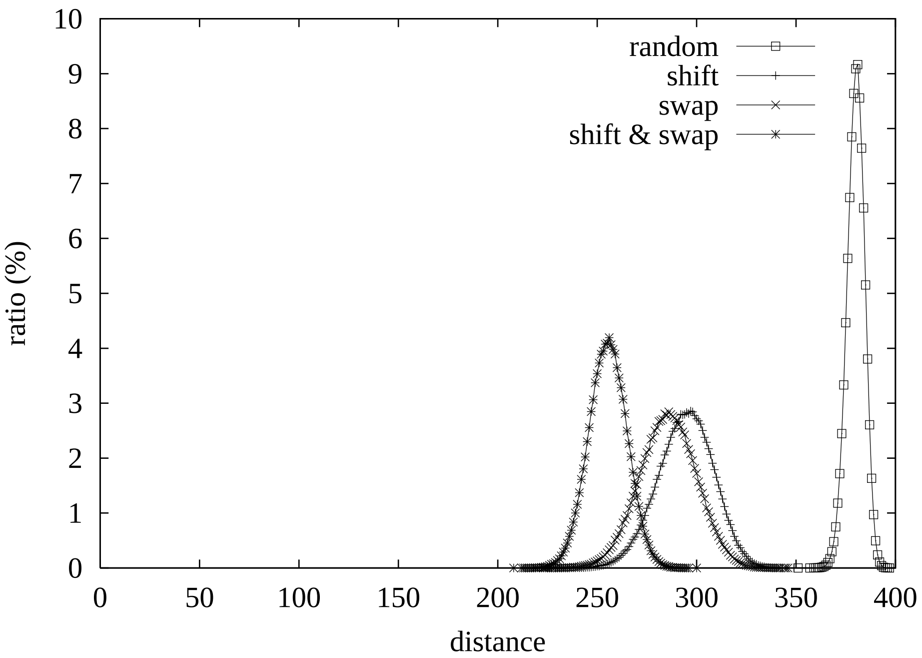

Figure 3. The distribution of distances between locally optimal solutions for the type E instance with $n = 400$ and $m = 20$