J Heuristics (2011) 17:527–565
DOI 10.1007/s10732-010-9144-0

GRASP with path-relinking for the generalized quadratic assignment problem

Geraldo R. Mateus · Mauricio G.C. Resende · Ricardo M.A. Silva

Received: 5 January 2009 / Revised: 26 April 2010 / Accepted: 15 August 2010 /
Published online: 1 September 2010

Abstract The generalized quadratic assignment problem (GQAP) is a generalization of the NP-hard quadratic assignment problem (QAP) that allows multiple facilities to be assigned to a single location as long as the capacity of the location allows. The GQAP has numerous applications, including facility design, scheduling, and network design. In this paper, we propose several GRASP with path-relinking heuristics for the GQAP using different construction, local search, and path-relinking procedures. We introduce a novel approximate local search scheme, as well as a new variant of path-relinking that deals with infeasibilities. Extensive experiments on a large set of test instances show that the best of the proposed variants is both effective and efficient.

Keywords Generalized quadratic assignment problem · Heuristic · GRASP · Path-relinking · Experimental algorithm

G.R. Mateus
Dept. of Computer Science, Federal University of Minas Gerais, CEP 31270-010, Belo Horizonte, MG, Brazil
e-mail: mateus@dcc.ufmg.br

M.G.C. Resende (✉)
Algorithms and Optimization Research Department, AT&T Labs Research, 180 Park Avenue, Room C241, Florham Park, NJ 07932, USA
e-mail: mgcr@research.att.com

R.M.A. Silva
Computational Intelligence and Optimization Group, Dept. of Computer Science, Federal University of Lavras, C.P. 3037, CEP 37200-000, Lavras, MG, Brazil
e-mail: rmas@dcc.ufla.br

R.M.A. Silva
Centro de Informática (CIn), Federal University of Pernambuco, Av. Professor Luís Freire s/n, Cidade Universitária, Recife, PE, Brazil

1 Introduction

Given n facilities and m locations, the generalized quadratic assignment problem (GQAP) consists in feasibly assigning each facility to a location. Each facility demands some amount of location capacity and each location has a fixed amount of capacity to distribute among facilities. An assignment is feasible if each location has sufficient capacity to accommodate the demands of all facilities assigned to it. Given nonnegative flows between all pairs of facilities and nonnegative distances between all pairs of locations, the GQAP seeks a feasible assignment that minimizes the sum of products of flows and distances in addition to a linear assignment component. Ideally, two facilities with high inter-facility flow are either assigned to the same location or to two nearby locations. The GQAP is a generalization of the quadratic assignment problem (QAP) that allows multiple facilities to be assigned to a single location as long as the capacity of the location permits. The QAP (see, e.g., Li et al. 1994; Oliveira et al. 2004, and Pardalos et al. 1994) is a special case of the GQAP where each facility demands one unit of capacity and each location has one unit of capacity to distribute.

Let $N = \{1, \ldots, n\}$ denote the set of facilities and $M = \{1, \ldots, m\}$ the set of locations. Furthermore, denote by $A_{n\times n} = (a_{ii'})$ the flow between facilities $i \in N$ and $i' \in N$, such that $a_{ii'} \in \Re^+$ if $i \neq i'$ and otherwise $a_{ii'} = 0$, by $B_{m\times m} = (b_{jj'})$ the distance between locations $j \in M$ and $j' \in M$, such that $b_{jj'} \in \Re^+$ if $j \neq j'$ and otherwise $b_{jj'} = 0$, and by $C_{n\times m} = (c_{ij})$, the cost of assigning facility $i \in N$ to location $j \in M$, such that $c_{ij} \in \Re^+$. Let $z \in \Re^+$ be a scaling factor called the unit traffic cost, $q_i \in \Re^+$ be the capacity demanded by facility $i \in N$, and $Q_j \in \Re^+$, the capacity of location $j \in M$. The GQAP consists in finding $X_{n\times m} = (x_{ij})$, with $x_{ij} = \{0, 1\}$, where facility $i \in N$ is assigned to location $j \in M$ if and only if $x_{ij} = 1$, such that the constraints

$$\sum_{j\in M} x_{ij} = 1, \quad \forall i \in N, \tag{1}$$

$$\sum_{i\in N} q_i x_{ij} \leq Q_j, \quad \forall j \in M, \tag{2}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i \in N, \forall j \in M \tag{3}$$

are satisfied and the objective function

$$\sum_{i\in N}\sum_{j\in M} c_{ij}x_{ij} + z\sum_{i\in N}\sum_{j\in M}\sum_{i'\in N, i'\neq i}\sum_{j'\in M} a_{ii'}b_{jj'}x_{ij}x_{i'j'} \tag{4}$$

is minimized. Constraints (1) guarantee that each facility is assigned to exactly one location, while constraints (2) ensure that location capacities are not violated.

The generalized quadratic assignment problem, introduced by Lee and Ma (2005), is a relatively new combinatorial optimization problem. Lee and Ma proposed three linear programming relaxations and a branch and bound algorithm for the GQAP, as well as suite of test problems on which they tested their algorithms. Elloumi et al. (2003) proposed three families of lower bounding techniques for the constrained

module allocation problem, a special case of the GQAP. They also propose a suite of test problems. Cordeau et al. (2006) proposed a new suite of test problems and an integer linear programming formulation with which they solved small instances of their test problems (having 5 to 16 facilities and 3 to 30 locations). A heuristic based on truncating the branch and bound tree was also proposed. The authors finally describe a memetic algorithm which was shown to find optimal solutions for all small instances tested and, on larger instances (having 20 to 50 facilities and 6 to 20 locations) was able to find better solutions than those found with the truncated branch and bound. Hahn et al. (2008) use the *reformulation linearization technique* (RLT) dual ascent procedure to find optimal solutions to some problems from Cordeau et al. (2006), Lee and Ma (2005), and Elloumi et al. (2003), having up to 20 facilities and 15 locations. For problems of these dimensions, the runtimes of their algorithm were no worse than those of previous exact methods. Pessoa et al. (2008) proposed a new lower bound for the GQAP based on the Lagrangian relaxation of the RLT formulation proposed in Hahn et al. (2008). To efficiently solve the relaxed problem, they combined the Hahn et al. (2008) dual ascent procedure with the general-purpose volume algorithm of Barahona and Anbil (2000). They tested the new bounding procedure using the branch-and-bound method of Hahn et al. (2008) on a subset of instances from Cordeau et al. (2006), Lee and Ma (2005), and Elloumi et al. (2003).

GRASP, or greedy randomized adaptive search procedure, is a metaheuristic for finding approximate solutions to combinatorial optimization problems formulated as

$$\min f(x) \quad \text{subject to} \quad x \in \mathcal{X},$$

where $f(\cdot)$ is an objective function to be minimized and $\mathcal{X}$ is a discrete set of feasible solutions. It was first introduced by Feo and Resende (1989) in a paper describing a probabilistic heuristic for set covering. Since then, GRASP has experienced continued development (Feo and Resende 1995; Resende and Ribeiro 2002, 2010) and has been applied in a wide range of problem areas (Festa and Resende 2002, 2009a, 2009b).

In this paper, we present a GRASP with path-relinking heuristic (Laguna and Martí 1999; Resende and Ribeiro 2005) for the generalized quadratic assignment problem. Extensive computational experiments on benchmark test problems show the effectiveness of these heuristics.

The paper is organized as follows. In Sect. 2, we describe the GRASP with path-relinking procedure. We consider variants of the greedy randomized construction procedure, the local search, and the path-relinking intensification procedure. Computational results are described in Sect. 3 and concluding remarks are made in Sect. 4.

2 GRASP with path-relinking for GQAP

A GRASP is a multi-start heuristic where at each iteration a greedy randomized solution is constructed to be used as a starting solution for local search. Local search repeatedly substitutes the current solution by a better solution in the neighborhood of the current solution. Each such replacement is called a *move*. If there is no better solution in the neighborhood, the current solution is declared a local minimum and

the search stops. The best local minimum found over all GRASP iterations is output as the solution.

GRASP iterations are independent, i.e. solutions found in previous GRASP iterations do not influence the algorithm in the current iteration. The use of previously found solutions to influence the procedure in the current iteration can be thought of as a memory mechanism. One way to incorporate memory into GRASP is with path-relinking (Glover 1996; Glover et al. 2000). In GRASP with path-relinking (Laguna and Martí 1999; Resende and Ribeiro 2005), an elite set of diverse good-quality solutions is maintained to be used during each GRASP iteration. After a solution is produced with greedy randomized construction and local search, that solution is combined with a randomly selected solution from the elite set using the path-relinking operator. The best of the combined solutions is a candidate for inclusion in the elite set and is added to the elite set if it meets quality and diversity criteria.

Algorithm 1 shows pseudo-code for a GRASP with path-relinking heuristic for the GQAP. The algorithm takes as input the set N of facilities, the set M of locations, the flow matrix A, the distance matrix B, the assignment cost matrix C, the scaling factor z, the facility demands q_i, $i \in N$, and the location capacities Q_j, $j \in M$, and outputs an assignment vector π^* specifying the location of each facility in the best solution found.

After initializing the elite set P as empty in line 1, the GRASP with path-relinking iterations are computed in lines 2 to 24 until a stopping criterion is satisfied. This criterion could be, for example, a maximum number of iterations, a target solution quality, or a maximum number of iterations without improvement. During each iteration, a greedy randomized solution π' is generated in line 3. If the elite set P does not have at least ρ elements, then if π' is feasible and sufficiently different from all other elite set solutions, π' is added to the elite set in line 22. To define the term *sufficiently different* more precisely, let $\Delta(\pi', \pi)$ be defined as the minimum number of moves needed to transform π' into π or vice-verse. For a given level of difference δ, we say π' is sufficiently different from all elite solutions in P if $\Delta(\pi', \pi) > \delta$ for all $\pi \in P$, which we indicate with the notation $\pi' \not\approx P$. If the elite set P does have at least ρ elements, then the steps in lines 5 to 19 are computed. The greedy randomized construction procedure is not guaranteed to generate a feasible solution. It is described in detail in Sect. 2.1. If the greedy randomized procedure returns an infeasible solution, a feasible solution π' is selected uniformly at random from the elite set in line 6 to be used as a surrogate for the greedy randomized solution. An approximate local search is applied using π' as a starting point in line 8, resulting in an approximate local minimum, which we denote by π'. Since elite solutions are made up of approximate local minima, then applying an approximate local search to an elite solution will, with high probability, result in a different approximate local minimum. The approximate local search is not guaranteed to find an exact local minimum. It is described in detail in Sect. 2.2. Since π' is an approximate local minimum, the application of an approximate local search to it will, with high probability, result in a different approximate local minimum. Next, path-relinking is applied in line 10 between π' and an elite solution π^+, randomly chosen in line 9. Solution π^+ is selected with probability proportional to $\Delta(\pi', \pi^+)$. In line 11, the approximate local search is applied to π'. If the elite set is full, then if π' is of better quality than

Data : $N, M, A, B, C, z, q_i, Q_j$.
Result: Solution $\pi^* \in \chi$.
1 $P \leftarrow \emptyset$;
2 **while** *stopping criterion not satisfied* **do**
3 $\quad \pi' \leftarrow$ `GreedyRandomized`$(\cdot)$;
4 $\quad$ **if** *elite set P has at least* ρ *elements* **then**
5 $\quad\quad$ **if** π' *is not feasible* **then**
6 $\quad\quad\quad$ Randomly select a new solution $\pi' \in P$;
7 $\quad\quad$ **end**
8 $\quad\quad \pi' \leftarrow$ `ApproxLocalSearch`(π');
9 $\quad\quad$ Randomly select a solution $\pi^+ \in P$;
10 $\quad\quad \pi' \leftarrow$ `PathRelinking`(π', π^+);
11 $\quad\quad \pi' \leftarrow$ `ApproxLocalSearch`(π');
12 $\quad\quad$ **if** *elite set P is full* **then**
13 $\quad\quad\quad$ **if** $c(\pi') \leq \max\{c(\pi) \mid \pi \in P\}$ **and** $\pi' \not\approx P$ **then**
14 $\quad\quad\quad\quad$ Replace the element most similar to π' among all elements with cost worst than π';
15 $\quad\quad\quad$ **end**
16
17 $\quad\quad$ **else if** $\pi' \not\approx P$ **then**
18 $\quad\quad\quad P \leftarrow P \cup \{\pi'\}$;
19 $\quad\quad$ **end**
20
21 $\quad$ **else if** π' *is feasible* **and** $\pi' \not\approx P$ **then**
22 $\quad\quad P \leftarrow P \cup \{\pi'\}$;
23 $\quad$ **end**
24 **end**
25 **return** $\pi^* = \min\{c(\pi) \mid \pi \in P\}$;

Algorithm 1: Pseudo-code for `GRASP+PR`: GRASP with path-relinking heuristic

the worst elite solution and $\pi' \not\approx P$, then it will be added to the elite set in line 14 in place of some elite solution. Among all elite solutions having cost no better than that of π', a solution π most similar to π', i.e. with the smallest $\Delta(\pi', \pi)$ value, is selected to be removed from the elite set. Ties are broken at random. Otherwise, if the elite set is not full, π' is simply added to the elite set in line 18 if $\pi' \not\approx P$.

2.1 Greedy randomized construction

Recall that a solution to the GQAP consists of n assignments of facilities to locations. The construction procedure builds a solution one assignment at time.

Suppose a number of assignments have already been made. To make the next assignment, the procedure needs to select a new facility and a location. Locations are made available, one at time. The procedure randomly determines whether to use a new location or a previously chosen location, favoring a new location when the pre-

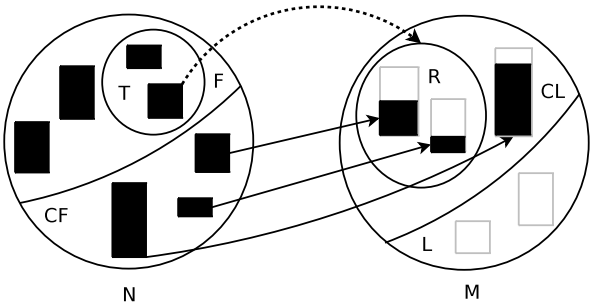

Fig. 1 Greedy randomized construction process. Facilities *on the left* are assigned to locations *on the right*. Facilities in set F have not yet been assigned. Locations in set L have not yet been assigned to. Facilities in set CF have already been assigned. Locations in set CL have been or can potentially be assigned to. Facilities in T have demands less than or equal to the maximum slack of facilities in set CL. After randomly selecting a facility to assign from set T, set R is defined as the set of locations that can accommodate that facility. One of these locations is selected at random to accommodate the facility

viously chosen locations have insufficient or barely sufficient available capacity. Later in this section, we define this choice more precisely. If the procedure determines that a previously chosen location is to be selected, it then determines which facilities can be assigned to a previously chosen location having the maximum available capacity and randomly selects one of these facilities to be assigned. Of the locations that can accommodate this facility, one is selected at random and the assignment is made. Otherwise, if there is no previously chosen location with sufficient capacity or if the available capacity is barely sufficient, a new location is selected at random from the set of yet unchosen locations. Figure 1 illustrates this procedure. In the figure, the set N of facilities is partitioned into the set CF of assigned facilities and the set F of unassigned ones. Likewise, the set M of locations is partitioned into the set CL of previously chosen locations and the set L of unselected locations. Facilities in CF are assigned to locations in CL. The facility set T consists of all unassigned facilities with demands less than or equal to the maximum available capacity of locations in CL. After a facility from T is randomly selected, set R consists of previously selected locations that can accommodate it. A location is randomly selected from set R and the facility is assigned to it.

The above procedure is not guaranteed to produce a feasible solution. The greedy randomized construction procedure, shown in Algorithm 2, addresses this difficulty by repeatedly applying the steps described above. The main loop in lines 1 to 21 is repeated a maximum number of times or until all facilities are assigned, i.e. when $F = \emptyset$. In each iteration of the procedure, the working sets are initialized in line 2 and the threshold probability is set to 1 in line 3. The purpose of the threshold is to control whether a new location should be selected. Since it is initially set to 1, then in the first iteration of the until loop in lines 4 to 19, the procedure always selects a new (first) location. At each iteration of the until loop, the threshold is updated in line 17 such that it will be more likely that a new location is selected when there are few facilities that can be assigned to locations in the current set R. The until loop consists of two stages. With probability equal to the threshold, the first stage (lines 5 to 9) selects a new location in line 6, updates the sets L and CL in line 7, and in line 8

Data : $\bar{t} =$ *maximum number of tries*
Result: Solution $x \in \mathcal{X}$
1 **while** $k < \bar{t}$ **and** $F \neq \emptyset$ **do**
2 $F \leftarrow N$; $CF \leftarrow \emptyset$; $L \leftarrow M$; $CL \leftarrow \emptyset$; $T \leftarrow \emptyset$;
3 Set `threshold` $\leftarrow 1$;
4 **repeat**
5 **if** $L \neq \emptyset$ **and** `random`$([0, 1]) \leq$ `threshold` **then**
6 Randomly select a location $l \in L$;
7 Update sets $L \leftarrow L \setminus \{l\}$ and $CL \leftarrow CL \cup \{l\}$;
8 Set $T \subseteq F$ to be all facilities with demands less than or equal to the maximum slack in CL;
9 **end**
10 **if** $T \neq \emptyset$ **then**
11 Randomly select a facility $f \in T$;
12 Update sets $T \leftarrow T \setminus \{f\}$; $F \leftarrow F \setminus \{f\}$; and $CF \leftarrow CF \cup \{f\}$;
13 Create set $R \subseteq CL$ to be all locations having slack greater than or equal to demand of facility f;
14 Randomly select a location $l \in R$;
15 Assign facility f to location l;
16 Set $T \subseteq F$ to be all facilities with demands less than or equal to the maximum slack in CL;
17 Set `threshold` $\leftarrow 1 - |T|/|F|$;
18 **end**
19 **until** $T = \emptyset$ **and** $L = \emptyset$;
20 $k \leftarrow k + 1$;
21 **end**
22 **if** $F \neq \emptyset$ **then**
23 Solution not found;
24 **else**
25 **return** assignment $x \in \mathcal{X}$;
26 **end**

Algorithm 2: Pseudo-code for `GreedyRandomized`: Greedy randomized construction procedure

determines the set T of facilities that can be assigned to some selected location. In the second stage (lines 10 to 18), the procedure randomly selects a facility from set T in line 11, updates the sets T, F, and CF in line 12, creates the location set R in line 13, randomly selects a location from that set in line 14, makes the assignment of the facility to the location in line 15, determines the set T of facilities that can be assigned to some selected location in line 16, and updates the threshold probability in line 17. The until loop is repeated until both sets T and L are empty in line 19. The while loop ends either with a valid assignment in line 25 (indicated by $F = \emptyset$) or with no solution found determined in line 23.

The construction procedure makes use of randomization to select a location in the first stage of the until loop and to select a facility and location in the second stage. To select a location in the first stage (line 6), the procedure favors locations in L with high capacity and that are close to high-capacity locations in CL. For all $k \in L$, let

$$H_k = \sum_{l \in CL} \frac{Q_k \cdot Q_l}{b_{kl}}.$$

The probability that location k is chosen is

$$\frac{H_k}{\sum_{r \in L} H_r}.$$

To select a facility in the second stage (line 11), the procedure favors facilities in T having high demand and high flows to other facilities. For all facilities $k \in T$, let

$$W_k = q_k \sum_{l \in N \setminus \{k\}} a_{kl}.$$

The probability that facility k is selected is

$$\frac{W_k}{\sum_{r \in T} W_r}.$$

To select a location in the second stage (line 14), the procedure favors locations in R that have high capacity slack and that are close to high-capacity locations in CL. Furthermore, the procedure favors locations in R for which there is a small increase in the objective function resulting from the assignment to it of the chosen facility in T. For all locations $k \in R$, let

$$Z_k = \sum_{l \in CL \setminus \{k\}} \frac{\sigma_k \cdot Q_l}{d \cdot b_{kl}},$$

where σ_k is the available capacity of location k and

$$d = c_{tk} + z \sum_{i' \in CF} \sum_{j' \in CL} a_{ti'} b_{kj'} x_{i'j'}$$

is the increase in the objective function resulting from the assignment to it of the chosen facility in $t \in T$. The probability that location $k \in R$ is selected is

$$\frac{Z_k}{\sum_{r \in R} Z_r}.$$

2.2 Approximate local search

When the construction procedure of Sect. 2.1 produces a feasible solution π, it is not guaranteed to be locally optimal. In GRASP heuristics, a local search procedure is

usually applied starting at π to find a local minimum. The heuristic described in this paper makes use of an *approximate local search* procedure. This procedure uses two neighborhood structures which we call *1-move* and *2-move*. A solution in the 1-move neighborhood of π is obtained by changing one facility-to-location assignment in π. Likewise, a solution in the 2-move neighborhood of π is obtained by simultaneously changing two facility-to-location assignments in π.

One way to carry out a local search in these neighborhoods is to evaluate moves in the 1-move neighborhood and move to the first improving solution. This is called the *first fit* local search. If no 1-move improving solution exists, 2-move neighborhood solutions are evaluated and a move is made to the first improving solution. Another way to carry out the local search is to evaluate all 1-move and 2-move neighborhood solutions and move to the best improving solution. This is called the *best fit* local search. In both variants, the search is repeated until no improving solution in the neighborhoods exists. A tradeoff approach is used here. Instead of evaluating all of the 1-move and 2-move neighborhood solutions, we sample these neighborhoods and populate a candidate list with improving solutions. One of the solutions from the candidate list is randomly selected and a move is made to that solution. The search is repeated until no improving solution is sampled. Because solutions are sampled, not all neighbors may be evaluated. Consequently, the best solution found may not be a local minimum. We call this solution an *approximate local minimum*.

Pseudo-code for the approximate local search is shown in Algorithm 3. The procedure takes as input the starting solution π and two parameters, *MaxCLS* and *MaxItr*, which control the sampling. The repeat until loop in lines 1 to 13 is repeated until an approximate local minimum is produced. In line 2, the sampling counter *count* and the candidate list *CLS* are initialized. At each iteration of the inner loop in lines 3 to 9, the 1-move and 2-move neighborhoods of π are sampled without replacement by procedure `Move`(π) in line 4. If this neighbor is an improving feasible solution, it is inserted into *CLS* in line 6. This is repeated until either the candidate list is full or a maximum number of neighbors have been sampled. In lines 10 to 12, if the candidate list is not empty, an assignment $\pi \in CLS$ is randomly chosen. If the set *CLS* is empty after the sampling process, the procedure terminates returning π as an approximate local minimum in line 14. Otherwise, the procedure moves to a solution in *CLS*, repeating the outer loop.

We propose two possible schemes for carrying out a move. In the first, the *greedy* scheme, the move is made to the solution in *CLS* having the smallest cost. In the second, the *random* scheme, the move is made at random to some solution in *CLS* with a bias favoring lower-cost solutions. One reason for using this approximate local search is the size of the neighborhood, which in the standard local search needs to be completely evaluated at least once.

Since there are many more 2-move neighbors than 1-move neighbors, we sample each neighbor as follows. With probability p_1 we sample from the 1-move neighborhood while with probability $p_2 = 1 - p_1$ we sample from the 2-move neighborhood. Note that by doing this, the set *CLS* can have both 1-move and 2-move neighbors.

The approximate local search procedure makes use of randomization to select a solution π from set *CLS* in line 11. The procedure favors solutions in *CLS* with smaller cost, according to the objective function (4). The probability that solution

Data : π, *MaxCLS*, *MaxItr*
Result: Approximate local minimum π
repeat
  $count \leftarrow 0$; $CLS \leftarrow \emptyset$;
  repeat
    $\pi' \leftarrow \texttt{Move}(\pi)$;
    if π' *is feasible* **and** $cost(\pi') < cost(\pi)$ **then**
      $CLS \leftarrow CLS \cup \{\pi'\}$;
    end
    $count \leftarrow count + 1$;
  until $|CLS| \geq MaxCLS$ **or** $count \geq MaxItr$;
  if $CLS \neq \emptyset$ **then**
    Randomly select a solution $\pi \in CLS$;
  end
until $CLS = \emptyset$;
return π;

Algorithm 3: Pseudo-code for `ApproxLocalSearch`: Approximate local search procedure

π is chosen is

$$\frac{G_\pi}{\sum_{\pi' \in CLS} G_{\pi'}},$$

where, for all $\pi' \in CLS$,

$$G_{\pi'} = \frac{1}{f(\pi')},$$

where $f(\cdot)$ is the objective function (4).

2.3 Path-relinking

Path-relinking (PR) is a search strategy that explores trajectories connecting two solutions (Glover 1996; Glover et al. 2000). Given two solutions (one which we call the *initial* solution and the other the *target* solution), their common elements are kept fixed and the space of solutions spanned by the remaining elements is explored with a greedy algorithm. Starting from an initial solution, the scheme moves from one solution to another until the target solution is reached. The objective consists in finding a solution that is better than both the initial and target solutions. Faria et al. (2005) introduced greedy randomized adaptive path-relinking, where instead of moving between two solutions in a greedy way, the moves are done in a greedy randomized fashion: one from among a candidate list with the most promising moves in the path being investigated is randomly selected.

One way of carrying out PR is through *forward* PR, where the best of the two given solutions is used as the target solution and the other as the initial solution.

Another way is through *backward* PR (Aiex et al. 2005; Resende and Ribeiro 2003), where the best of the two given solutions is used as the initial solution and the other as the target solution. In back and forward PR, forward PR is applied and is followed by backward PR. Another way to proceed is with *mixed* PR (Glover 1996; Ribeiro and Rosseti 2002), where two paths are simultaneously explored, the first emanating from the initial solution and the other from the target solution, until they meet at an intermediary solution equidistant from both extremes.

According to Ribeiro et al. (2002) and Aiex et al. (2003), better solutions are found in the backward approach than in the forward scheme, where the worst of the two solutions to be relinked is used as the initial solution. Back and forward PR usually finds better solutions than either forward or backward PR but at the expense of longer running times. Ribeiro and Rosseti (2002) showed that mixed PR outperformed forward, backward, and back and forward PR. As we will see in Sect. 3, for the GRASP with path-relinking heuristic proposed in this paper, the forward and backward schemes are better than the mixed scheme.

In this paper, a new variant of path-relinking is proposed, motived by the fact that a single move guided by the target solution does not guarantee the feasibility of the newly constructed solution. For instance, suppose that among the differences between a solution π' and the target solution is the location assigned to some specific facility. Performing a move in π' that assigns the facility to the same location assigned to it in the target solution can result in a solution that may or not be feasible. If the capacity constraint of this location is not violated, the new solution is feasible. Otherwise, the repair procedure `makeFeasible` is applied in an attempt to make it feasible. The repair procedure is described in detail in Sect. 2.4. In a path-relinking *step*, the move described above, with or without a repair procedure, is repeated for each facility of solution π' assigned to a location that is different from the one in the target solution. Let $\mathcal{G}$ denote the set of assignments resulting from these moves and let $\mathcal{B} \subseteq \mathcal{G}$ denote a subset of the best solutions in set $\mathcal{G}$.

The path-relinking process consists of a sequence of path-relinking steps. In each step, a feasible solution is either greedily or randomly selected from $\mathcal{B}$ to become the new solution π of the next step. This process continues until the target solution is reached or when no single feasible solution is obtained from π', i.e. when $\mathcal{B} = \emptyset$. The best solution in this path, including the start and target solutions, is returned as the final result.

Algorithm 4 shows pseudo-code for the path-relinking procedure. The algorithm takes as input π_s and π_t, the starting and target solutions, respectively, and outputs the best solution π^* in path from π_s to π_t. Initially, the best solution in the path is set as π^* in line 1 and its corresponding objective function is assigned to f^* in line 2. In line 3, the current solution π' is initialized with π_s, and the working sets *Fix* and *nonFix* are respectively initialized empty and with N. The while loop in lines 5 to 35 is repeated until all facilities in π' are assigned to the same locations assigned to them in π_t, i.e. the set $\varphi(\pi', \pi_t) = \{i \in N \mid \pi'(i) \neq \pi_t(i)\}$ is empty, where $\pi'(i)$ and $\pi_t(i)$ are the locations assigned to facility i in solutions π' and π_t, respectively.

After the set $\mathcal{B}$ of best solutions is set to empty in line 6, each while loop iteration consists of two stages. The first stage in lines 7 to 20 implements the path-relinking step. It creates set $\mathcal{B}$ with the best feasible solutions constructed from the current solution π'. In line 6, $\mathcal{B}$ is initialized as empty. Each facility $v \in \varphi(\pi', \pi_t)$ is analyzed

Data : Starting solution π_s, target solution π_t, and candidate size factor η
Result: Best solution π^* in path from π_s to π_t

1 $\pi^* \leftarrow \text{argmin}\{f(\pi_s), f(\pi_t)\}$;
2 $f^* \leftarrow f(\pi^*)$;
3 $\pi' \leftarrow \pi_s$; $Fix \leftarrow \emptyset$; $nonFix \leftarrow N$;
4 Compute difference $\varphi(\pi', \pi_t)$ between solution π' and π_t;
5 **while** $\varphi(\pi', \pi_t) \neq \emptyset$ **do**
6 $\quad \mathcal{B} \leftarrow \emptyset$;
7 $\quad$ **for** $\forall v \in \varphi(\pi', \pi_t)$ **do**
8 $\quad\quad$ Move the facility v in π' to the same location l assigned to v in π_t;
9 $\quad\quad \bar{\pi} \leftarrow$ `makeFeasible`(π', v);
10 $\quad\quad$ **if** $\bar{\pi}$ *is feasible* **then**
11 $\quad\quad\quad$ **if** $|\mathcal{B}| \geq \eta \cdot |\varphi(\pi', \pi_t)|$ **then**
12 $\quad\quad\quad\quad$ **if** $c(\bar{\pi}) \leq \max\{c(\pi) \mid \pi \in \mathcal{B}\}$ **and** $\bar{\pi} \notin \mathcal{B}$ **then**
13 $\quad\quad\quad\quad\quad$ replace the element most similar to $\bar{\pi}$ among all elements with cost worst than $\bar{\pi}$;
14 $\quad\quad\quad\quad$ **end**
15
16 $\quad\quad\quad$ **else if** $\bar{\pi} \notin \mathcal{B}$ **then**
17 $\quad\quad\quad\quad \mathcal{B} \leftarrow \mathcal{B} \cup \{\bar{\pi}\}$;
18 $\quad\quad\quad$ **end**
19 $\quad\quad$ **end**
20 $\quad$ **end**
21 $\quad$ **if** $\mathcal{B} \neq \emptyset$ **then**
22 $\quad\quad$ Randomly select a solution $\pi \in \mathcal{B}$;
23 $\quad\quad$ Compute difference $\varphi(\pi, \pi_t)$ between solution π and π_t;
24 $\quad\quad$ Set $I = \varphi(\pi', \pi_t) \setminus (\varphi(\pi', \pi_t) \cap \varphi(\pi, \pi_t))$;
25 $\quad\quad$ Randomly select a facility $i \in I$;
26 $\quad\quad$ $Fix \leftarrow Fix \cup \{i\}$; $nonFix \leftarrow nonFix \setminus \{i\}$;
27 $\quad\quad$ $\pi' \leftarrow \pi$;
28 $\quad\quad$ **if** $f(\pi') < f^*$ **then**
29 $\quad\quad\quad$ $f^* \leftarrow f(\pi')$;
30 $\quad\quad\quad$ $\pi^* \leftarrow \pi'$;
31 $\quad\quad$ **end**
32 $\quad$ **else**
33 $\quad\quad$ **return** assignment π^*;
34 $\quad$ **end**
35 **end**
36 **return** assignment π^*;

Algorithm 4: Pseudo-code for `PathRelinking`: Path-relinking procedure

in lines 7 to 20 to create the set $\mathcal{B}$ with the best feasible solutions constructed from the current solution π'. Procedure `makeFeasible` (described in detail in Sect. 2.4) is applied in line 8 to facility v to attempt to create a new solution $\bar{\pi}$ from π'. The

application of `makeFeasible` to facility v can either result in a feasible or infeasible solution. In case `makeFeasible` returns a feasible solution $\bar{\pi} \notin \mathcal{B}$, $\bar{\pi}$ is added to $\mathcal{B}$ if $\mathcal{B}$ is not yet full. Otherwise, if $\mathcal{B}$ is full and solution $\bar{\pi} \notin \mathcal{B}$ is not worse than any elite solution, then $\bar{\pi}$ is added to $\mathcal{B}$ replacing some other elite solution.

In the second stage (lines 21 to 34), the procedure first randomly selects a solution π from set $\mathcal{B}$ in line 22. Then, in line 25, it selects at random a facility $i \in I = \varphi(\pi', \pi_t) \setminus (\varphi(\pi', \pi_t) \cap \varphi(\pi, \pi_t))$, where I is defined in line 24 as the set containing all unfixed facilities whose locations were corrected in the previous path-relinking step. A facility is corrected when its location becomes the one assigned to it in the target solution. After fixing facility $i \in I$, sets *Fix* and *nonFix* are updated in line 26. Finally, the next path-relinking step solution π' is set as π in line 27 and, if $f(\pi') < f^*$, then the best cost f^* and best solution π^* are updated in lines 29 and 30, respectively. However, if in some path-relinking step no feasible solution is obtained from π', the while loop is interrupted, returning the current solution π^* as the result in line 33. If the target solution is reached, then π^* is returned in line 36.

This path-relinking is different from the standard variant because given solutions π_s and π_t, their common elements are not kept fixed a priori, such that a small portion of the solution space spanned by the remaining elements is explored. The new variant fixes one facility at a time at each step.

The path-relinking procedure makes use of randomization to select a solution π from set $\mathcal{B}$ is line 22 and to select a facility i from set I in line 25. To select a solution in $\mathcal{B}$, the procedure favors those solutions having smaller cost, according to the objective function $f(\cdot)$. For all $s \in \mathcal{B}$, let

$$U_s = \frac{1}{f(s)}.$$

The probability that solution π is chosen is

$$\frac{U_\pi}{\sum_{r \in \mathcal{B}} U_r}.$$

To select a facility i from set I, a uniform probability distribution is used.

2.4 Repair procedure

Suppose π is an intermediary solution obtained while applying path-relinking from π_s to π_t, as illustrated in Fig. 2. The set N of facilities is partitioned into two sets, the set *Fix* of fixed facilities and the set *nonFix* of the remaining facilities. Let $f \in$ *nonFix* be a facility whose location in π_t is l and in π is $u \neq l$, where $l, u \in M$.

According to Fig. 2, performing a move in π that assigns facility f to location l is infeasible since the capacity Q_l of location l is insufficient to accommodate that facility. In this case, the facility set *FTL* $\subseteq$ *nonFix* is created with all unfixed facilities assigned to l in π. Next, set $T \subseteq$ *FTL* is constructed. It is made up of all facilities in *FTL* having demands at most equal to the maximum available capacity of locations in M. Let i be a facility that is randomly selected from T. Set R consists of locations in M that can accommodate facility i. A location j is selected at random from set R

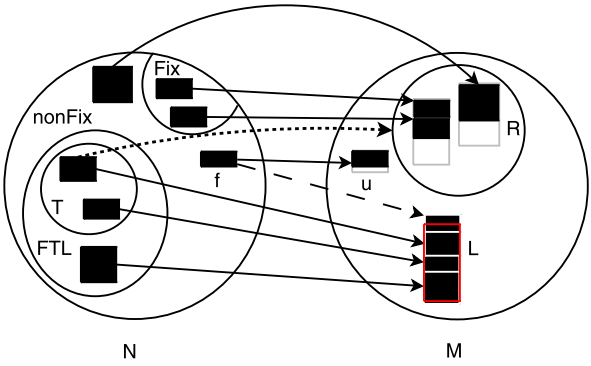

Fig. 2 Single path-relinking process with repair. Facilities in set N *on the left* are assigned to locations in set M *on the right*. Facilities in set $Fix \subseteq N$ have already been fixed. Facilities in set $nonFix \subseteq N$ have not yet been fixed. Facilities in $FTL \subseteq nonFix$ are assigned to location l in solution π. Facilities in $T \subseteq FTL$ have demands less than or equal to the maximum slack of facilities in set M. After randomly selecting a facility from set T to assign, set R is defined as the set of locations that can accommodate that facility

and facility i is assigned to it. This process is repeated until the capacity of location l has a nonnegative slack.

Algorithm 5 shows pseudo-code for the procedure `makeFeasible`$(\cdot)$. The algorithm takes as input a solution π, a facility $f \in \varphi(\pi, \pi_t)$, and the maximum number of tries $\bar{\tau}$, to output a feasible solution if possible. As input, facility f in π was moved to the same location l assigned to it in π_t. If the slack of location l is nonnegative ($\sigma_l \geq 0$), i.e. the capacity of l is not violated, the algorithm returns the updated feasible solution π in line 2. Otherwise, in lines 3 to 24, the repair process is applied as follows. While the capacity of location l remains violated and the number k of tries is less than a maximum limit established a priori, a copy π' of solution π is made in line 6 before the repeat loop in lines 7 to 16. In this loop, a facility set $FTL \subseteq nonFix$ is created in line 8 with all yet unfixed facilities in N assigned to l in π'. Next, in line 9, set $T \subseteq FTL$ is constructed with all facilities in FTL having demands less than or equal to the maximum available capacity of locations in M. Then, if set T is not empty, a facility i is randomly selected from it in line 12 to create a set $R \subseteq M$ with all locations in M that can accommodate the facility. Finally, in line 13, a location j is randomly selected from set R and facility i is assigned to it in line 14. This loop is repeated until either the capacity of location l has a nonnegative slack or set T is empty. At the end, if the slack of l is nonnegative, the returned solution π' is feasible. Otherwise, it is infeasible.

The repair procedure makes use of randomization to select a facility i from set T in line 11 and to select a location j from set R in line 13. To select a facility in T, the procedure favors those with higher demand. Then, the probability that facility i is chosen is

$$\frac{q_i}{\sum_{r \in T} q_r},$$

where we recall that q_i is the capacity demanded by facility $i \in N$. To select a location j from set R, a uniform probability distribution is used.

```
Data  : Solution π, facility f, and τ̄ = maximum number of tries
Result: If possible, a feasible solution
if σ_l ≥ 0 then
    return feasible solution π;
else
    k ← 0;
    while k < τ̄ and σ_l < 0 do
        π′ ← π;
        repeat
            Set FTL ⊆ nonFix to be all facilities in π′ assigned to l;
            Set T ⊆ FTL to be all facilities with demands less than or equal to
            the maximum slack in M;
            if T ≠ ∅ then
                Randomly select a facility i ∈ T;
                Set R ⊆ M to be all locations having slack greater than or equal
                to demand of facility i;
                Randomly select a location j ∈ R;
                Assign facility i to location j: π′(i) ← j;
            end
        until T = ∅ or σ_l ≥ 0;
        k ← k + 1;
    end
    if σ_l < 0 then
        return infeasible solution π′;
    else
        return feasible solution π′;
    end
end
```

Algorithm 5: Pseudo-code for `makeFeasible`: Repair procedure in path-relinking

3 Experimental results

In this section, we present results on computational experiments with the GRASP with path-relinking (GRASP-PR) heuristic introduced in this paper. First, we describe our test environment. Next, we determine an appropriated combination of values for the three most important parameters of the heuristic. Finally, we compare our implementation with other heuristics from the literature on a suite of test problems.

3.1 Test environment

All experiments with GRASP-PR were done on a Dell PE1950 computer with dual quad core 2.66 GHz Intel Xeon processors and 16 Gb of memory, running Red Hat Linux nesh version 5.1.19.6 (CentOS release 5.2, kernel 2.6.18-53.1.21.el5). The GRASP-PR heuristic was implemented in Java and compiled into bytecode with

`javac` version 1.6.0_05. The random-number generator is an implementation of the Mersenne Twister algorithm (Matsumoto and Nishimura 1998) from the COLT[1] library.

3.2 Parameter tuning for the GRASP-PR heuristic

Several parameters are used to describe the GRASP-PR heuristics. Some of these parameters were set to fixed values. These parameters and their fixed values are:

- size $|P|$ of elite set: 10
- minimum number ρ of elements in elite set to perform path-relinking: 2
- maximum number $\bar{t}$ of trials in the greedy randomized construction: 10
- 1-move sample probability p_1 in the local search: 0.5
- maximum number *MaxItr* of neighbors sampled in local search: 100
- maximum size *MaxCLS* of the candidate list in the local search: 10
- candidate size factor η in the non-mixed path-relinking: 0.5
- maximum number $\bar{\tau}$ of tries in `makeFeasible`: 10
- minimum difference δ with respect to elite solutions needed for trial solution to be inserted into elite set during path-relinking: 4

To select the strategies that define the GRASP-PR variant that we use in the computational testing, we conducted the following experiment. For each combination of algorithm strategy and instance, we make 200 independent runs of the heuristic, recording the time taken to find the best known solution for the instance, and plot its runtime distribution. We consider four large instances of Cordeau et al. (2006): 20-15-75, 30-20-75, 40-10-65, and 50-10-95, and tested the GRASP-PR variants defined by the following strategies:

- Path-relinking direction: Forward (`f`), backward (`b`), mixed (`m`)
- Criteria to select a facility from set T in the `makeFeasible` procedure: Randomly (`r`), such as in Algorithm 5, or greedily (`g`), where the facility with the highest demand in T is selected
- Criteria to select a solution from candidate list *CLS* in the approximate local search: Randomly (`r`), such as in Algorithm 3, or greedily (`g`), where the best cost solution in set *CLS* is selected
- Criteria to select solution to move to in the path-relinking phase: Randomly (`r`), as in Algorithm 4 or greedily (`g`), where the best-valued solution in $\mathcal{B}$ is selected

Among the $3 \times 2^3 = 24$ possible combinations of these strategies, Figs. 3 and 4 illustrate the combinations for which the GRASP-PR heuristics found the best known solution for the four test instances in less than 100,000 seconds. These figures show time-to-target plots (Aiex et al. 2002, 2007) (or runtime distributions) for each algorithm variant. For each of the four instances, 200 independent runs of each variant were conducted and the time to find the target solution was recorded. The sorted running times make up the plots. Note that some of the combinations are not shown

[1] COLT is a open source library for high performance scientific and technical computing in Java. See http://acs.lbl.gov/~hoschek/colt/.

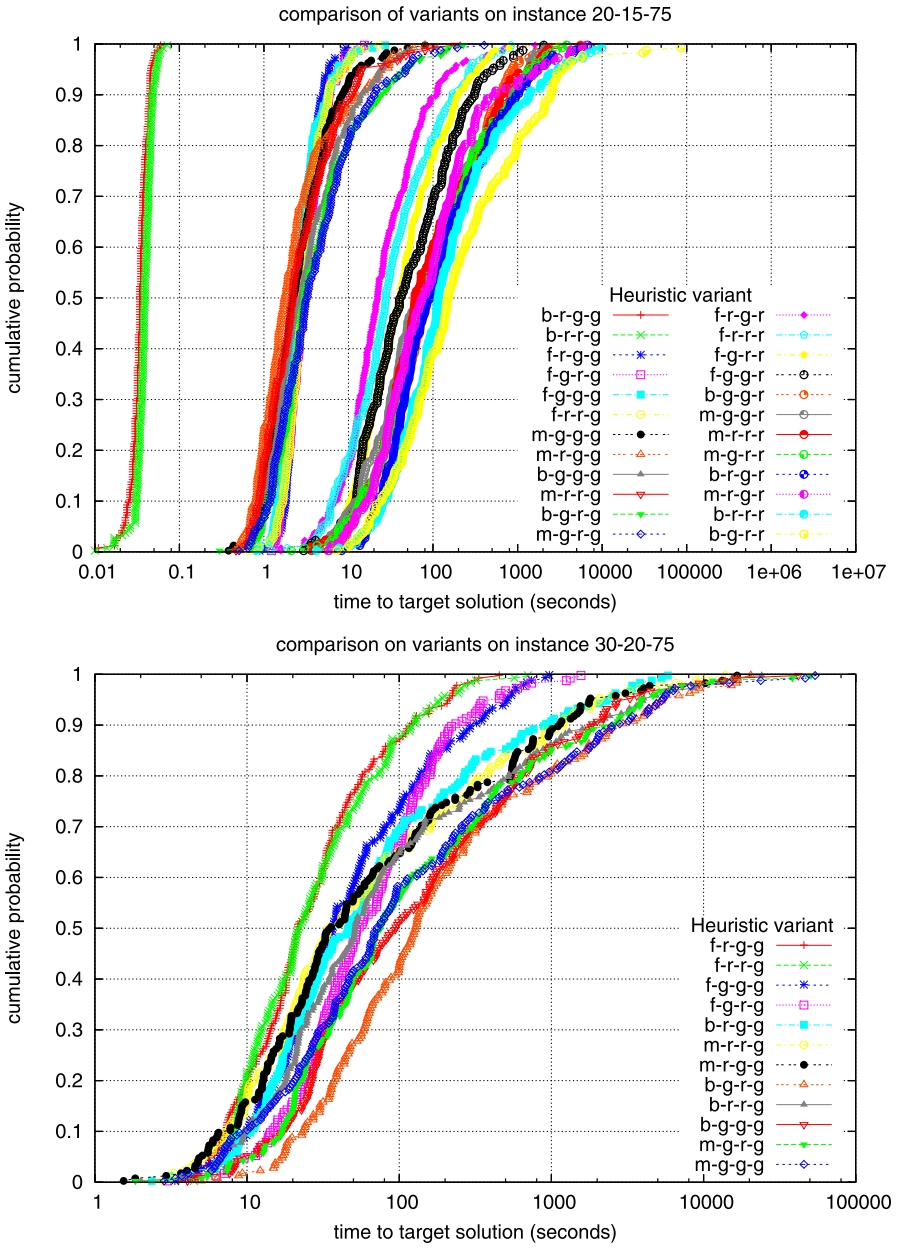

Fig. 3 Plots of cumulative probability distributions of GRASP-PR running times for instances 20-15-75 and 30-20-75 using several variants of the GRASP with path-relinking heuristic

in the figures because they failed to find their target solutions within the time limit. In the figures, the strategies are represented in such a way that the first parameter corresponds to the path-relinking direction, the second parameter corresponds to the

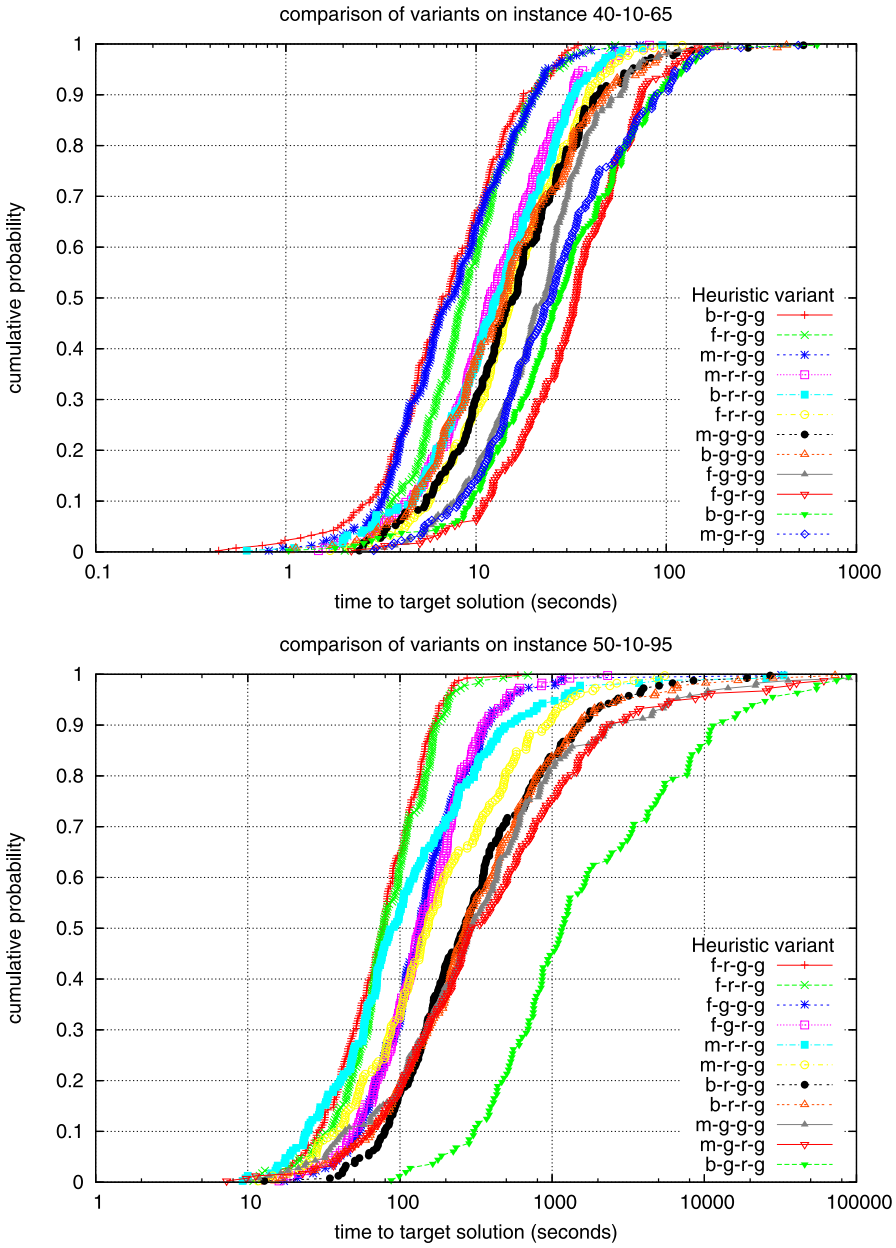

Fig. 4 Plots of cumulative probability distributions of GRASP-PR running times for instances 40-10-65 and 50-10-95 using several variants of the GRASP with path-relinking heuristic

facility selection criterion, the third parameter defines the solution selection criterion, and the fourth parameter defines which path-relinking move strategy is used. For example, `f-g-r-g` represents the strategy that uses forward path-relinking, greedy fa-

Table 1 Time in seconds for 25%, 50%, 75%, and 100% of GRASP-PR runs to find the best-valued solutions for instance 20-15-75

GRASP-PR variants	25%	50%	75%	100%
b-r-g-g	0.030	0.034	0.039	0.060
b-r-r-g	0.034	0.039	0.044	0.072
f-r-g-g	2.109	2.746	3.540	17.069
f-g-r-g	2.105	2.963	4.164	15.426
f-g-g-g	1.932	2.508	3.521	27.175
f-r-r-g	2.070	2.799	3.893	22.986
m-g-g-g	1.341	2.242	4.471	80.463
m-r-g-g	1.014	1.731	3.541	80.291
b-g-g-g	1.467	2.545	6.244	209.327
m-r-r-g	1.199	2.187	4.643	181.163
b-g-r-g	1.859	2.993	6.798	227.481
m-g-r-g	1.651	3.172	7.435	402.552
f-r-g-r	12.293	22.018	48.717	1610.136
f-r-r-r	12.741	27.553	67.312	837.054
f-g-r-r	17.977	39.560	90.693	3029.421
f-g-g-r	17.456	40.174	128.335	2043.053
b-g-g-r	33.275	78.100	267.390	5529.753
m-g-g-r	28.210	63.343	224.118	3820.697
m-r-r-r	30.408	61.083	285.124	5595.585
m-g-r-r	33.145	87.868	258.000	3778.661
b-r-g-r	40.362	90.625	303.850	6801.102
m-r-g-r	29.987	72.820	186.898	6484.493
b-r-r-r	51.821	114.362	324.944	9920.335
b-g-r-r	57.024	148.409	691.220	85,854.522

cility selection, random solution selection, and greedy move in path-relinking. Along with Figs. 3 and 4, Tables 1, 2, 3, and 4 show the running times for 25%, 50%, 75%, and 100% of the runs to find the best-valued solutions for instances 20-15-75, 30-20-75, 40-10-65, and 50-10-95, respectively.

Figures 3 and 4 show the dominance of strategies `f-r-g-g`, `f-r-r-g`, `b-r-g-g`, and `b-r-r-g` over the other combinations. Considering the top two combinations for each instance, the frequencies of occurrence of each strategy among the winners were

$$\begin{bmatrix} \texttt{f}:5 \\ \texttt{b}:3 \\ \texttt{m}:0 \end{bmatrix} - \begin{bmatrix} \texttt{r}:8 \\ \texttt{g}:0 \end{bmatrix} - \begin{bmatrix} \texttt{g}:5 \\ \texttt{r}:3 \end{bmatrix} - \begin{bmatrix} \texttt{g}:8 \\ \texttt{r}:0 \end{bmatrix}.$$

We choose the most popular strategy (`f-r-g-g`) as the preferred GRASP-PR heuristic in the remaining experiments. This heuristic uses forward path-relinking, randomly selects a facility from set T in procedure `makeFeasible`, greedily selects a solution from the candidate list in the approximate local search, and greedily selects a solution to move to in the path-relinking phase.

Table 2 Time in seconds for 25%, 50%, 75%, and 100% of GRASP-PR runs to find the best-valued solutions for instance 30-20-75

GRASP-PR variants	25%	50%	75%	100%
f-r-g-g	11.581	21.537	45.499	454.887
f-r-r-g	11.085	21.738	52.666	701.121
f-g-g-g	18.380	36.205	104.795	969.521
f-g-r-g	26.769	52.559	122.278	1563.631
b-r-g-g	17.170	45.885	178.478	5830.817
m-r-r-g	14.439	33.797	238.672	13,933.570
m-r-g-g	14.476	34.509	204.042	16,598.494
b-g-r-g	44.274	127.487	472.226	20,432.680
b-r-r-g	20.736	51.221	268.023	24,120.400
b-g-g-g	29.507	93.994	465.674	41,981.810
m-g-r-g	25.481	75.039	378.109	50,797.920
m-g-g-g	25.219	71.297	383.568	54,138.195

Table 3 Time in seconds for 25%, 50%, 75%, and 100% of GRASP-PR runs to find the best-valued solutions for instance 40-10-65

GRASP-PR variants	25%	50%	75%	100%
b-r-g-g	4.252	6.596	11.775	34.417
f-r-g-g	5.554	8.537	13.345	53.852
m-r-g-g	4.27	7.385	13.022	72.927
m-r-r-g	7.406	11.748	20.498	81.569
b-r-r-g	7.301	12.68	23.889	95.407
f-r-r-g	9.018	15.105	25.87	121.209
m-g-g-g	9.152	15.575	26.77	526.456
b-g-g-g	6.939	13.97	29.169	428.596
f-g-g-g	12.827	22.772	33.514	211.246
f-g-r-g	19.056	33.469	52.842	190.087
b-g-r-g	14.539	27.271	52.388	621.276
m-g-r-g	13.93	24.733	44.394	494.348

To conclude the tuning, we ran an experiment to determine the best local search strategy. We consider GRASP with path-relinking using four local search strategies:

1. use the approximate local search scheme during the GRASP and after path-relinking
2. use the approximate local search scheme during the GRASP but do not do local search after path-relinking
3. use first fit local search scheme during the GRASP and after path-relinking
4. use best fit local search scheme during the GRASP and after path-relinking

The best fit scheme searches the entire neighborhood of the current solution looking for the best improving solution. If no improving solution is found, then the current solution is declared a local minimum.

For each variant, we made 200 independent runs on instances 20-15-35 and 40-10-65 of Cordeau et al. (2006). Each run was stopped when the best known solution

Table 4 Time in seconds for 25%, 50%, 75%, and 100% of GRASP-PR runs to find the best-valued solutions for instance 50-10-95

GRASP-PR variants	25%	50%	75%	100%
f-r-g-g	44.319	77.375	118.371	595.610
f-r-r-g	53.916	77.523	136.206	693.350
f-g-g-g	78.560	135.611	215.584	32,120.332
f-g-r-g	82.669	134.995	228.472	2317.059
m-r-r-g	48.898	86.436	236.827	33,131.920
m-r-g-g	79.790	150.354	404.184	5500.276
b-r-g-g	132.240	262.648	671.102	27,117.906
b-r-r-g	124.392	270.583	631.574	72,390.710
m-g-g-g	115.116	292.063	673.854	89,439.910
m-g-r-g	127.618	291.607	971.098	61,027.715
b-g-r-g	543.194	1150.462	4784.563	88,714.290

for the instance was found and its running time was recorded. Runtime distribution plots for these runs are shown in Fig. 5. The time to target plots clearly show that the approximate local search scheme is preferable to either first fit or best fit local search. As expected, first fit is shown to be better than best fit. Furthermore, the plots show that it is advisable to perform local search after each call to the path-relinking procedure.

Finally, to demonstrate the need for the repair procedure `makeFeasible` in the path-relinking procedure, we conducted the following experiment using instances 40-10-65 and 50-10-95 of Cordeau et al. (2006). For each instance, we repeated 200 independent runs of GRASP-PR using path-relinking with repair, path-relinking without repair (simply removing the `makeFeasible` from Algorithm 4 and stopping path-relinking when no further feasible movement could be made by the path-relinking procedure). For each of these two heuristics, we tested variants with and without the approximate local search after path-relinking (line 11 of Algorithm 1), resulting in a total of four variants: PR with repair and local search, PR with repair and no local search, PR without repair and with local search, and PR with neither repair nor local search. The results are summarized in the two plots in Fig. 6.

The top plot in the figure shows runtime distributions for instance 40-10-65. For that experiment we used as target solution value the best known solution for this instance. The plot clearly shows that GRASP-PR with PR having both repair and posterior approximate local search dominates the other variants. In particular, using repair dominates not using repair.

The dominance of PR with repair can be better seen in the bottom plot, corresponding to the more difficult instance 50-10-95. In those runs, we first ran the four variants with the best known solution value of 12,845,598 as the target solution value in the stopping criterion. Both variants using repair (with and without posterior local search) found the solution on all 200 runs in less than 600 seconds (about 70% of the runs terminated in less than 100 seconds). On the other hand, neither variant without repair was able to find a single solution in less than 15,000 seconds. To show the relative performance of those variants we used the weak target value of 16,004,409 for the variant without repair but with local search and the even weaker value 17,7461,596

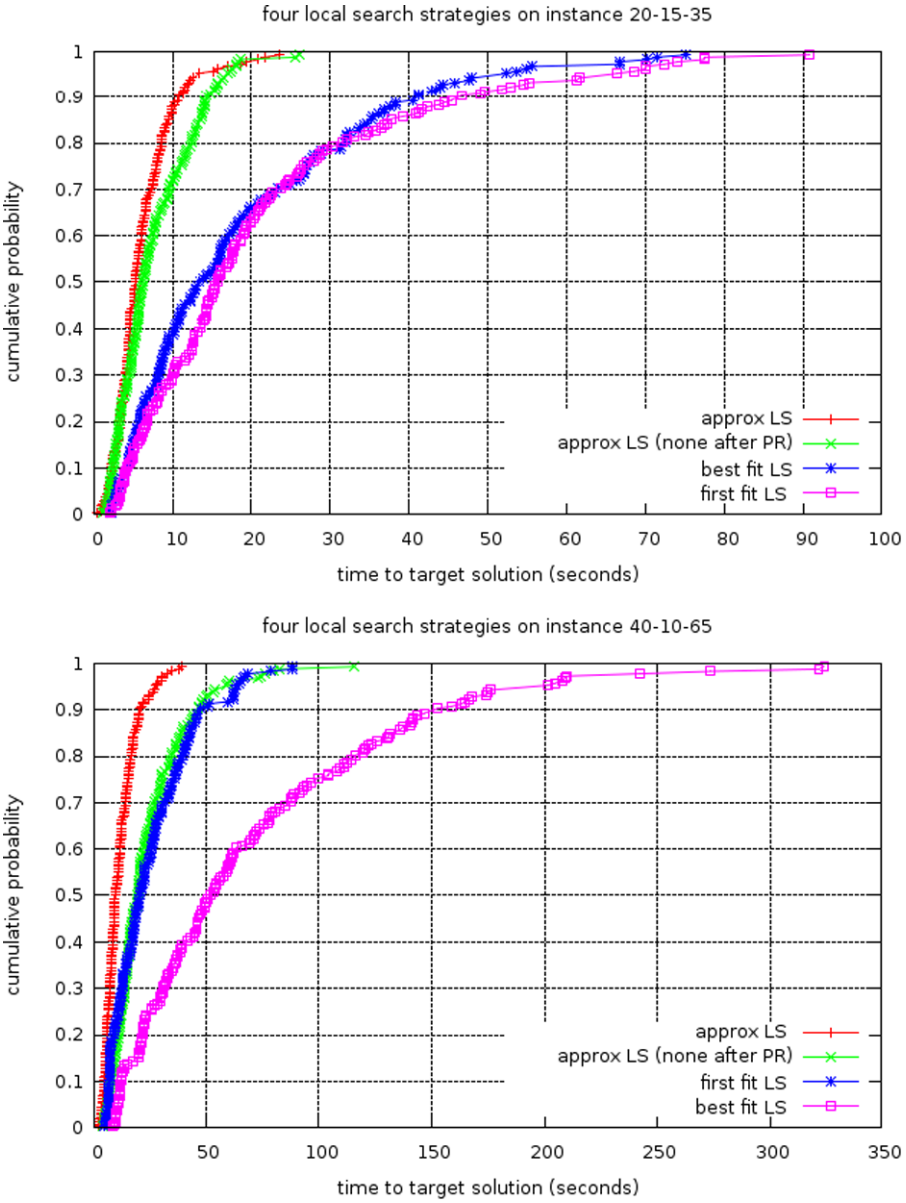

Fig. 5 Plots of cumulative probability distributions of GRASP-PR running times for instances 20-15-35 and 40-10-65 using four variants for the local search strategy. The figures show that applying approximate local search is better than using best fit or first fit local search and that applying local search after path-relinking is better than not applying any local search after path-relinking

for the variant without either repair or local search. Even with these weaker target values, the two variants only found the target solution in less than 15,000 seconds in fewer than 30% of the runs, as can be seen in the lower plot in Fig. 6. Using those

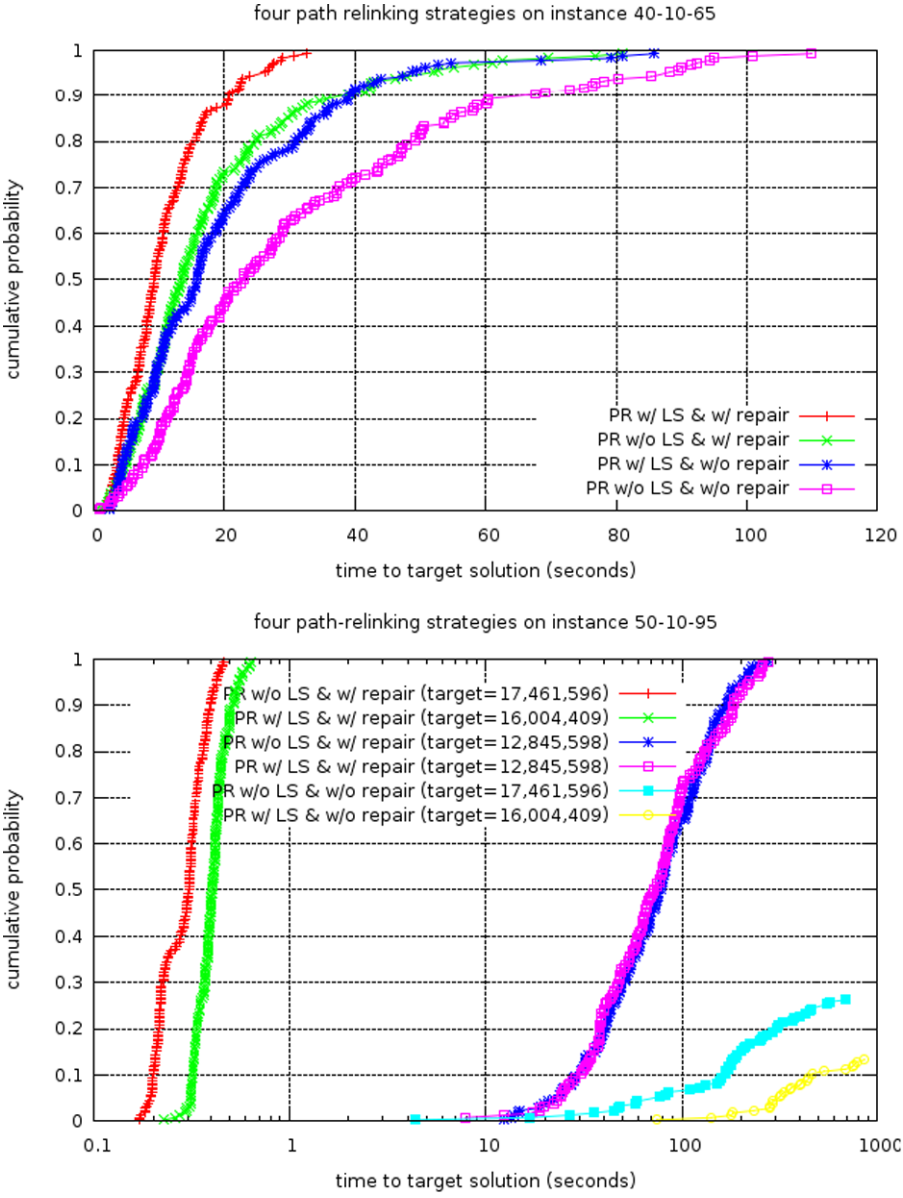

Fig. 6 Plots of cumulative probability distributions of GRASP-PR running times for instances 40-10-65 and 50-10-95 using four variants for the path-relinking strategy. 200 independent runs of the algorithms were made and each run was stopped after 15,000 seconds. The bottom plot for instance 50-10-95 shows all measured times for six combinations of path-relinking strategy and target value. The figures show that, independent of using approximate local search after path-relinking, it is always better to use the repair procedure in path-relinking

weaker targets, the variants using repair (with and without local search) found the target solutions in less than one second on all 200 independent runs each.

3.3 Comparison of the GRASP-PR heuristic with other algorithms

In the experiments to follow, we make use of the test problems used by Lee and Ma (2005), Cordeau et al. (2006), and Hahn et al. (2008). In the comparisons, we show nominal and normalized running times. Times are normalized by the processor clock speed. For example, 96.0 seconds reported on a 1.2 GHz processor and 0.16 seconds on a 2.66 GHz processor have corresponding normalized times of $96.0 \times 1.2 = 115.2$ and $0.16 \times 2.66 = 0.43$ seconds, respectively. All nominal running times reported will be followed by their normalized value (between parentheses).

Among the largest instances are those presented by Cordeau et al. (2006) with 20 to 50 facilities and 6 to 20 locations. Cordeau et al. developed a memetic algorithm which found promising results for these larger instances. Table 5 compares these results with ones obtained with the GRASP-PR heuristic. The first column of the table shows the names of the instances. These instances are labeled with three parameters: $n = |N|$, $m = |M|$, and $s \in \{1, \ldots, 100\}$. Parameter s controls the tightness of the capacity constraints. The higher the value of s, the higher is the tightness of the capacity constraints.

The third column lists the time (in seconds) reported by Cordeau et al. (2006) for their memetic algorithm to find the solutions whose values are given in the second column. It should be noted that Cordeau et al. implemented their memetic algorithm in C and ran their experiment on a 1.2 GHz Sun workstation. The solution values in the second column are the best values found by the memetic algorithm.

For each instance, we made 200 independent runs of GRASP-PR, with the exception of instance 30-20-95 for which we made only 18 runs. Each run stopped when a solution value as good as the one in column 2 was found. Columns 4 to 7 give the minimum, maximum, and average times, as well as the standard deviation of these runs to find solutions with values equal to those listed in column 2. Finally, column 8 lists the time for 95% of the GRASP-PR runs to find solutions having those same target values.

On all runs for each instance in Table 5, the GRASP-PR heuristic found solutions with the target values shown in column 2. The average performance improvement with respect to the memetic algorithm varied between a factor of 1.482 (0.667) and 59.186 (26.634), except for instances 30-20-95, 35-15-95, and 50-10-75, for which the times reported by Cordeau et al. were smaller than the average times for the GRASP-PR heuristic. Note, however, that on all instances, including 30-20-95, 35-15-95, and 50-10-75, the smallest GRASP-PR running times were far less than those reported for the memetic algorithm. Figure 7 shows time-to-target-plots (Aiex et al. 2007) for GRASP-PR on four instances from Table 5. For each group with identical number of facilities, we chose to plot the running-time distribution for the instance on which the memetic algorithm took longest to solve.

To optimally solve the GQAP, Lee and Ma (2005) proposed a branch and bound (B&B) algorithm having single assignment as the branching rule, as done by Burkard (1991) for the quadratic assignment problem (QAP). Furthermore, they proposed

Table 5 Summary of results for memetic and GRASP-PR heuristics on the Cordeau et al. (2006) instances. Times are given in 1.2 GHz Sun workstation seconds running C for the memetic algorithm and 2.66 GHz Intel Xeon processor running Java for GRASP-PR. Normalized times are given between parentheses

Memetic algorithm			GRASP-PR times to target solution				
Instance	Solution	Time	Min	Max	Avg	S.d.	0.95
20-15-35	**1471896**	96.0 (115.2)	0.16 (0.43)	38.87 (103.40)	7.05 (18.76)	6.47 (17.21)	21.04 (55.98)
20-15-55	**1723638**	102.0 (122.4)	0.24 (0.64)	14.42 (38.36)	2.87 (7.62)	2.18 (5.79)	7.69 (20.46)
20-15-75	**1953188**	102.0 (122.4)	0.26 (0.69)	12.82 (34.10)	2.01 (5.36)	1.72 (4.57)	5.25 (13.97)
30-06-95	**5160920**	114.0 (136.8)	0.55 (1.45)	23.81 (63.32)	2.59 (6.88)	2.22 (5.90)	6.44 (17.13)
30-07-75	**4383923**	156.0 (187.2)	0.50 (1.32)	38.47 (102.33)	7.80 (20.74)	5.47 (14.56)	18.18 (48.348)
30-08-55	**3501695**	96.0 (115.2)	0.18 (0.48)	4.89 (13.01)	1.61 (4.29)	0.95 (2.53)	3.60 (9.58)
30-10-65	**3620959**	210.0 (252.0)	2.75 (7.31)	1032.80 (2747.25)	121.94 (324.37)	146.059 (388.52)	514.82 (1369.41)
30-20-35	**3379359**	564.0 (676.8)	1.08 (2.87)	4441.40 (11814.11)	79.03 (210.23)	312.62 (831.57)	166.21 (442.11)
30-20-55	**3593105**	462.0 (554.4)	1.28 (3.40)	150.11 (399.30)	25.16 (66.92)	21.19 (56.37)	66.82 (177.74)
30-20-75	**4050938**	522.0 (626.4)	2.11 (5.61)	759.81 (2021.11)	41.43 (110.22)	68.39 (181.89)	148.43 (394.82)
30-20-95	**5710645**	5232.0 (6278.4)	833.99 (2218.41)	2533608.00 (6739397.28)	543019.01 (1444430.57)	747962.39 (1989579.96)	2186440.80 (5815932.53)
35-15-35	**4456670**	456.0 (547.2)	8.41 (22.37)	1717.94 (4569.72)	306.11 (814.26)	242.49 (645.04)	775.25 (2062.17)
35-15-55	**4639128**	384.0 (460.8)	4.33 (11.52)	75.69 (201.34)	21.13 (56.21)	11.95 (31.79)	42.47 (112.96)
35-15-75	**6301723**	396.0 (475.2)	5.18 (13.77)	621.83 (1654.07)	68.23 (181.50)	74.17 (197.30)	183.19 (487.30)
35-15-95	**6670264**	864.0 (1036.8)	6.61 (17.58)	19171.48 (50996.14)	1454.00 (3867.64)	3057.43 (8132.76)	6949.08 (18484.55)
40-07-75	**7405793**	180.0 (216.0)	4.53 (12.037)	377.06 (1002.97)	59.37 (157.91)	51.21 (136.22)	159.00 (422.94)
40-09-95	**7667719**	1140.0 (1368.0)	6.18 (16.44)	5017.56 (13346.70)	417.00 (1109.22)	610.28 (1623.34)	1490.31 (3964.24)
40-10-65	**7265559**	240.0 (288.0)	0.84 (2.23)	115.06 (306.05)	17.87 (47.53)	15.88 (42.23)	52.73 (140.27)
50-10-65	**10513029**	504.0 (604.8)	2.52 (6.70)	84.64 (225.15)	24.56 (65.32)	16.34 (43.48)	64.04 (170.34)
50-10-75	**11217503**	606.0 (727.2)	22.79 (60.62)	24507.34 (65189.52)	1352.41 (3597.41)	3085.42 (8207.22)	4404.50 (11715.97)
50-10-95	**12845598**	1254.0 (1504.8)	9.97 (26.52)	1059.59 (2818.51)	89.36 (237.70)	91.95 (244.59)	200.20 (532.53)

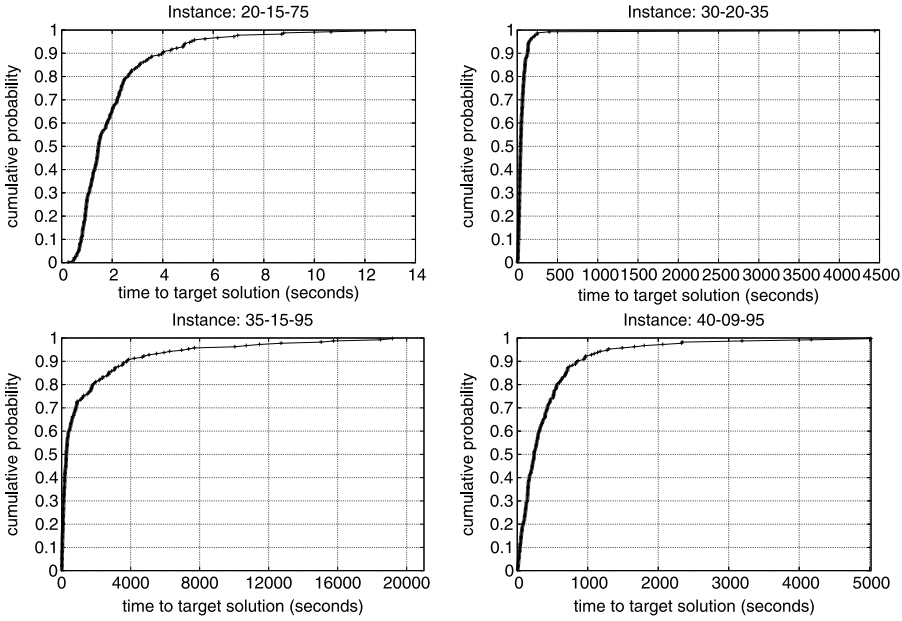

Fig. 7 Time-to-target plots. Plots of cumulative probability distributions of GRASP-PR running times for four instances: 20-15-75, 30-20-35, 35-15-95, and 40-09-95

three linearization methods, F-Y, K-B, and L3, based on Frieze and Yadegar (1983), Kaufman and Broeckx (1978), and Padberg and Rijal (1996) linearizations for the QAP, respectively. Lee and Ma created a suite of test problems with 10 to 16 facilities and 3 to 8 locations on which they tested their algorithms. Table 6 compares these results with ones obtained with the GRASP-PR heuristic.

The first and second columns of Table 6 show the instance names and their corresponding optimum values. These instances are labeled in three parts: the number before the $\times$ is $n = |N|$, the number after the $\times$ is $m = |M|$, and the optional last two letters indicate the following additional characteristics: facilities having a wide range of demands (E), facilities having roughly the same demand (F), locations having a wide range of capacities (G), locations having roughly the same capacity (H). Columns 3 to 6 list the times (in seconds) reported by Lee and Ma (2005) for their four algorithms (F-Y, K-B, L3, and B&B) to find solutions whose values are given in the second column. Times in boldface are the best among the Lee and Ma algorithms.

For each instance, we made 200 independent runs of GRASP-PR. Each run stopped when a solution value as good as the one in column 2 was found. Columns 7 to 10 denote, respectively, the minimum, maximum, and average times, and standard deviation of these runs to find solutions with values equal to those of column 2. Finally, column 11 lists the time for 95% of the runs to find solutions having those same target values. GRASP-PR found the target value on all 200 runs for each of the instances in Table 6, with an average performance improvement varying between a factor of 11.2 (2.8) and 1004.6 (251.15) in relation to the best time of the algorithms of Lee and Ma.

Table 6 Summary of results for the four algorithms of Lee and Ma (2005) and the GRASP-PR heuristic on the Lee and Ma (2005) instances. Times in seconds are measured on a 677 MHz Intel Pentium III running AMPL/CPLEX 8.1 for the Lee and Ma (2005) algorithms and on a 2.66 GHz Intel Xeon running Java for GRASP-PR. Normalized times are given *between parentheses*

		Lee and Ma (2005) times				GRASP-PR time to target solution				
Instance	Solution	F-Y	K-B	L3	B&B	Min	Max	Avg	S.d.	0.95
$10 \times 5EG$	**1021050**	39.2 (26.5)	12.2 (8.3)	**11.5 (7.8)**	161.5 (109.3)	0.031 (0.082)	0.663 (1.764)	0.207 (0.551)	0.118 (0.314)	0.466 (1.240)
$10 \times 5EH$	**1188500**	49.8 (33.7)	21.4 (14.5)	**20.4 (13.8)**	361.1 (244.5)	0.038 (0.101)	0.459 (1.221)	0.185 (0.491)	0.097 (0.257)	0.386 (1.027)
$10 \times 5FG$	**1433500**	51.6 (34.9)	**5.2 (3.5)**	21.3 (14.4)	196.2 (132.8)	0.040 (0.106)	0.785 (2.088)	0.282 (0.747)	0.149 (0.395)	0.543 (1.444)
$10 \times 5FH$	**1471950**	96.2 (65.1)	26.5 (17.9)	**17.1 (11.6)**	723.9 (490.1)	0.039 (0.104)	0.559 (1.487)	0.243 (0.646)	0.124 (0.331)	0.467 (1.242)
$12 \times 3EG$	**1323900**	5.4 (3.7)	**2.0 (1.4)**	2.1 (1.4)	72.1 (48.8)	0.039 (0.104)	0.417 (1.109)	0.147 (0.390)	0.071 (0.188)	0.263 (0.700)
$12 \times 3EH$	**1771450**	17.6 (11.9)	8.4 (5.7)	**5.7 (3.9)**	418.3 (283.2)	0.038 (0.101)	0.622 (1.655)	0.217 (0.578)	0.113 (0.300)	0.453 (1.205)
$12 \times 3FG$	**1788750**	19.6 (13.3)	6.3 (4.3)	**6.0 (4.1)**	251.4 (170.2)	0.032 (0.085)	0.474 (1.261)	0.136 (0.363)	0.078 (0.209)	0.304 (0.809)
$12 \times 3FH$	**1970500**	17.3 (11.7)	14.8 (10.0)	**6.0 (4.1)**	359.2 (243.2)	0.039 (0.104)	0.433 (1.152)	0.148 (0.394)	0.090 (0.240)	0.345 (0.918)
$12 \times 4EG$	**1500150**	31.8 (21.5)	**11.2 (7.6)**	11.3 (7.7)	202.6 (137.2)	0.038 (0.101)	0.995 (2.647)	0.351 (0.934)	0.182 (0.485)	0.689 (1.833)
$12 \times 4EH$	**2081370**	102.9 (69.7)	**29.6 (20.0)**	30.0 (20.3)	626.5 (424.1)	0.045 (0.120)	0.717 (1.907)	0.327 (0.870)	0.135 (0.360)	0.564 (1.500)
$12 \times 4FG$	**1770500**	88.5 (59.9)	55.3 (37.4)	**20.5 (13.9)**	925.3 (626.4)	0.046 (0.122)	0.79 (2.101)	0.326 (0.867)	0.165 (0.438)	0.614 (1.633)
$12 \times 4FH$	**2213500**	82.0 (55.5)	**5.9 (4.0)**	17.8 (12.1)	251.4 (170.2)	0.072 (0.192)	1.117 (2.971)	0.524 (1.394)	0.208 (0.553)	0.858 (2.282)
$12 \times 5EG$	**1519100**	119.1 (80.6)	65.4 (44.3)	**32.3 (21.9)**	565.3 (382.7)	0.054 (0.144)	1.264 (3.362)	0.284 (0.756)	0.153 (0.408)	0.517 (1.375)
$12 \times 5EH$	**2086100**	567.9 (384.5)	**20.6 (13.9)**	59.3 (40.1)	510.6 (345.7)	0.023 (0.061)	1.399 (3.721)	0.605 (1.608)	0.220 (0.586)	0.951 (2.530)
12×6	**1960750**	651.5 (441.1)	**18.2 (12.3)**	229.7 (155.5)	641.9 (434.6)	0.086 (0.229)	1.110 (2.953)	0.526 (1.399)	0.207 (0.552)	0.845 (2.248)
12×8	**1244300**	1091.0 (738.6)	1333.9 (903.1)	**278.2 (188.3)**	1445.8 (978.8)	0.079 (0.210)	1.060 (2.820)	0.403 (1.072)	0.218 (0.581)	0.804 (2.139)

Table 6 (*Continued*)

Instance	Solution	Lee and Ma (2005) times				GRASP-PR time to target solution				
		F-Y	K-B	L3	B&B	Min	Max	Avg	S.d.	0.95
14 × 6	**2351000**	408.6 (276.6)	**52.5 (35.5)**	78.3 (53)	1030.0 (697.3)	0.068 (0.181)	1.295 (3.445)	0.632 (1.682)	0.252 (0.670)	1.083 (2.881)
14 × 7	**2365650**	1670.2 (1130.7)	**172.1 (116.5)**	315.0 (213.3)	2459.1 (1664.8)	0.092 (0.245)	1.563 (4.158)	0.558 (1.483)	0.194 (0.517)	0.811 (2.157)
14 × 8	**2622470**	2717.6 (1839.8)	**256.4 (173.6)**	275.3 (186.4)	2034.5 (1377.4)	0.073 (0.194)	0.936 (2.490)	0.396 (1.053)	0.183 (0.488)	0.746 (1.984)
14 × 9	**2326370**	3759.1 (2544.9)	7881.7 (5335.9)	**483.2 (327.1)**	3278.9 (2219.8)	0.057 (0.152)	0.919 (2.445)	0.479 (1.274)	0.160 (0.425)	0.748 (1.990)
15 × 6	**2707850**	1614.1 (1092.7)	**109.4 (74.1)**	284.8 (192.8)	2160.1 (1462.4)	0.109 (0.290)	7.769 (20.666)	1.009 (2.685)	0.836 (2.225)	2.084 (5.543)
15 × 7	**2720070**	1971.3 (1334.6)	658.7 (445.9)	**296.2 (200.5)**	3119.2 (2111.7)	0.201 (0.535)	2.090 (5.559)	0.798 (2.123)	0.375 (0.997)	1.653 (4.397)
15 × 8	**2856850**	4056.1 (2746.0)	**397.5 (269.1)**	1242.1 (840.9)	5007.9 (3390.3)	0.414 (1.101)	21.727 (57.794)	2.336 (6.214)	2.575 (6.849)	8.032 (21.365)
16 × 6	**2709300**	1815.7 (1229.2)	630.3 (426.7)	**338.4 (229.1)**	4884.4 (3306.7)	0.121 (0.322)	0.934 (2.484)	0.458 (1.218)	0.149 (0.396)	0.730 (1.942)
16 × 7	**2809870**	22423.7 (15180.8)	**988.7 (669.3)**	7208.4 (4880.1)	5921.9 (4009.1)	0.472 (1.256)	13.962 (37.139)	1.752 (4.660)	1.577 (4.195)	4.431 (11.786)

Table 7 Summary of results for RLT1_B&B algorithm and GRASP-PR heuristic. Times in seconds are measured on a 733 MHz Intel Itanium processor (Intel Fortran 8.0 compiler) for the RLT1_B&B algorithm and on a 2.66 GHz Intel Xeon running Java for GRASP-PR. Normalized times are given *between parentheses*

		RLT1_B&B algorithm		GRASP-PR times to target solution				
Instance	Solution	Tot. time	Time to bks	Min	Max	Avg	S.d.	0.95
2005De	**2302**	17640.0 (12930.12)	14390.0 (10547.87)	0.096 (0.255)	6.069 (16.144)	1.531 (4.073)	0.891 (2.371)	3.302 (8.783)
2005Aa	**3059**	136.0 (99.69)	128.0 (93.82)	0.180 (0.479)	14.152 (37.644)	3.420 (9.097)	2.371 (6.306)	19.268 (51.253)
2408Ca	**1028**	6.3 (4.618)	3.3 (2.419)	0.149 (0.396)	2.307 (6.137)	0.714 (1.899)	0.447 (1.189)	1.594 (4.240)
2408Aa	**5643**	719862.0 (527658.85)	185726.0 (136137.16)	1.604 (4.267)	50.445 (134.184)	10.288 (27.366)	8.369 (22.262)	29.735 (79.095)
20-15-35	**1471896**	735.0 (538.76)	528.8 (387.61)	1.604 (4.267)	50.445 (134.184)	10.288 (27.366)	8.369 (22.262)	29.735 (79.095)
14 × 9*EG*	**2326370**	169.0 (123.88)	90.7 (66.48)	1.604 (4.267)	50.445 (134.184)	10.288 (27.366)	8.369 (22.262)	29.735 (79.095)
15 × 8	**2856850**	130.0 (95.29)	39.5 (28.95)	1.604 (4.267)	50.445 (134.184)	10.288 (27.366)	8.369 (22.262)	29.735 (79.095)
16 × 7-1115	**2809870**	548.0 (401.68)	512.6 (375.74)	1.604 (4.267)	50.445 (134.184)	10.288 (27.366)	8.369 (22.262)	29.735 (79.095)

Hahn et al. (2008) proposed a level-1 reformulation-linearization technique (RLT) dual ascent procedure in a branch-and-bound scheme (RLT1_B&B) for the GQAP. They solved four instances (c2005De, 2005Aa, 2408Ca, and 2408Aa) from the website[2] of S. Elloumi, one instance (20-15-35) from Cordeau et al. (2006), and three instances ($14 \times 9EG$, 15×8, and 16×17-1115) from Lee and Ma (2005), as shown in the first column of Table 7. That table compares the results of the RLT1_B&B algorithm with ones obtained with the GRASP-PR heuristic. Column 3 lists the time (in seconds on a 733 MHz Intel Itanium processor) reported by Hahn et al. (2008) for RLT1_B&B to find the optimum solutions with the values given in column 2. For each instance, we made 200 independent runs of the GRASP-PR heuristic. Each of these runs stopped when the optimal solution value was found. Columns 5 to 8 denote, respectively, the minimum, maximum, and average times, and standard deviation of the time to find optimal solutions. Finally, column 9 lists the times needed for 95% of GRASP-PR runs to find optimally-valued solutions.

On all runs, GRASP-PR achieved the target values (optimal solutions) on all instances in Table 7. The average performance improvement, with respect to the RLT1_B&B algorithm, varied between a factor of 8.8 (2.46) and over 69,000 (1930). With respect to the time taken by RLT1_B&B to first find an optimally-valued solution, the improvement of GRASP-PR varied from 4.6 (1.27) and over 18 thousand (4974.68).

Pessoa et al. (2008) proposed a new lower bound for the GQAP based on the Lagrangian relaxation of the RLT formulation proposed in Hahn et al. (2008). As shown in the first column of Table 8, Pessoa et al. tested their approach on four instances (c2005De, 2005Aa, 2408Ca, and 2408Aa) from the website of S. Elloumi, 12 instances (20-15-35, 20-15-55, 20-15-75, 30-08-55, 30-20-35, 30-20-55, 30-20-75, 30-20-95, 30-15-35, 35-15-55, 35-15-75, and 35-15-95) from Cordeau et al. (2006), and three instances ($14 \times 9EG$, 15×8, and 16×17-1115) from Lee and Ma (2005). Table 8 compares these results with ones obtained with the GRASP-PR heuristic. The column 3 lists the time (in seconds on a 1.6 GHz Ultra 45 Sun workstation) reported by Pessoa et al. (2008) for their hybrid B&B algorithm to find the optimal solutions and prove their optimality, using the optimum values given in column 2 as upper bounds.

For each instance, we made 200 independent runs of the GRASP-PR heuristic. Each of these runs stopped when an optimally-valued solution (column 2) was found. Columns 4 to 7 denote the minimum, maximum, and average times, and standard deviation of the time needed to find the optimally-valued solutions. Finally, column 8 gives the time needed for 95% of these runs to find solutions having those same target values. The GRASP-PR heuristic achieved the target values on all instances in Table 8, with an average performance improvement, with respect to the hybrid B&B algorithm, varying between a factor of 132.7 (79.62) and over 100,000 (60,000), with the exception of instance 30-20-95. The smallest GRASP-PR running time for instance 30-20-95, however, was less than the running time reported by Pessoa et al. (2008) for the hybrid B&B algorithm.

Elloumi et al. (2003) proposed three linearization methods (L1, L2, and L3), three semidefinite programming formulations (S0, S1, and S2), and a Lagrangian decom-

[2] http://cedric.cnam.fr/oc/TAP/all_CTAP_instances/.

Table 8 Summary of results for Hybrid_B&B algorithm and GRASP-PR heuristic. Times are given in seconds on a 1.6 GHz Ultra 45 Sun workstation (C++ compiler) for the Hybrid_B&B algorithm and on a 2.66 GHz Intel Xeon running Java for GRASP-PR. Normalized times are given *between parentheses*

	Hybrid_B&B algorithm		GRASP-PR times				
Instance	Opt soln	Time	Min	Max	Avg	S.d.	0.95
c2005De	**5435**	10085.0 (16136.0)	0.083 (0.221)	16.764 (44.592)	2.129 (5.664)	1.745 (4.642)	4.965 (13.207)
2005Aa	**3059**	3018.0 (4828.8)	0.180 (0.479)	14.152 (37.644)	3.420 (9.097)	2.371 (6.306)	19.268 (51.253)
2408Ca	**1028**	117.0 (187.2)	0.149 (0.396)	2.307 (6.137)	0.714 (1.899)	0.447 (1.189)	1.594 (4.240)
2408Aa	**5643**	1031637.0 (1650619.2)	1.604 (4.267)	50.445 (134.184)	10.288 (27.366)	8.369 (22.262)	29.735 (79.095)
20-15-35	**1471896**	2729.0 (4366.4)	0.16 (0.43)	38.87 (103.40)	7.05 (18.76)	6.47 (17.21)	21.04 (55.98)
20-15-55	**1723638**	4370.0 (6992.0)	0.24 (0.64)	14.42 (38.36)	2.87 (7.62)	2.18 (5.79)	7.69 (20.46)
20-15-75	**1953188**	1340.0 (2144.0)	0.26 (0.69)	12.82 (34.10)	2.01 (5.36)	1.72 (4.57)	5.25 (13.97)
30-08-55	**3501695**	2826.0 (4521.6)	0.18 (0.48)	4.89 (13.01)	1.61 (4.29)	0.95 (2.53)	3.60 (9.58)
30-20-35	**3379359**	228177.0 (365083.2)	1.08 (2.87)	4441.40 (11814.11)	79.03 (210.23)	312.62 (831.57)	166.21 (442.11)
30-20-55	**3593105**	707027.0 (1131243.2)	1.28 (3.40)	150.11 (399.30)	25.16 (66.92)	21.19 (56.37)	66.82 (177.74)
30-20-75	**4050938**	18803.0 (30084.8)	2.11 (5.61)	759.81 (2021.11)	41.43 (110.22)	68.39 (181.89)	148.43 (394.82)
30-20-95	**5710645**	2612.0 (4179.2)	833.99 (2218.41)	2533608.00 (6739397.28)	543019.01 (1444430.57)	747962.39 (1989579.96)	2186440.80 (5815932.53)
35-15-35	**4456670**	978664.0 (1565862.4)	8.41 (22.37)	1717.94 (4569.72)	306.11 (814.26)	242.49 (645.04)	775.25 (2062.17)
35-15-55	**4639128**	413241.0 (661185.6)	4.33 (11.52)	75.69 (201.34)	21.13 (56.21)	11.95 (31.79)	42.47 (112.96)
35-15-95	**6670264**	205931.0 (329489.6)	6.61 (17.58)	19171.48 (50996.14)	1454.00 (3867.64)	3057.43 (8132.76)	6949.08 (18484.55)
14×9	**2326370**	139.0 (222.4)	0.057 (0.152)	0.919 (2.445)	0.479 (1.274)	0.160 (0.425)	0.748 (1.990)
15×8	**2856850**	359.0 (574.4)	0.414 (1.101)	21.727 (57.794)	2.336 (6.214)	2.575 (6.849)	8.032 (21.365)
16×7	**2809870**	311.0 (497.6)	0.472 (1.256)	13.962 (37.139)	1.752 (4.660)	1.577 (4.195)	4.431 (11.786)

position (D0) algorithm for the *constrained module allocation problem (CMAP)*, or *constrained task assignment problem (CTAP)*. The CTAP is a special case of the GQAP, where a slightly different cost function is used. For these experiments, we adopted this cost function to be able to compare the GRASP-PR heuristic with their algorithms. Elloumi et al. used the instances introduced in Elloumi (1991) and also used in Roupin (2004). In these instances, four configurations (A, B, C, and D) are considered. For each configuration, two classes of instances are generated: a class with a complete communication graph and a second class where the density of the communication graph is 50%, resulting in a total of eight types of instances. For each type, five instances (a–e), with 10 facilities and three locations, and five instances (a–e), with 20 facilities and five locations, are generated. These instances were downloaded from a website of S. Elloumi. Only a subset of these instances have their optimal values available in the website. Therefore, we decided to divide the instances into two tables. Table 9 compares the running times of the algorithms of Elloumi et al. with those of GRASP-PR on the CTAP instances with known optimal values listed in the website of S. Elloumi. Tables 10–12 list the other instances, jointly with the best solution obtained with the GRASP-PR heuristic. We computed these values because they could not be found in the website. The tables list the average CPU times, and in parentheses, their corresponding average errors with the best known solutions for the seven bounds of Elloumi et al. (2003).

In Tables 9–12, the first and second columns show the instance names and their corresponding optimum or best known values. These instances are labeled in five parts: a letter "c", if the communication graph is complete, $n = |N|$, $m = |M|$, the configuration (A, B, C, or D), and the instance identifier (a, b, c, d, or e).

Columns 3 to 9 list the average time (in seconds) reported by Elloumi et al. (2003) per instance category for their seven methods (L1, L2, L3, S0, S1, S2, and D0) to find solutions whose values are given in the second column. Least mean error values are indicated by boldface with ties broken by running time.

For each instance, we made 200 independent runs of the GRASP-PR heuristic. Each run stopped when a solution value equal to the one in column 2 was found. Columns 10 to 13 denote, respectively, the minimum, maximum, and average times, and standard deviation of these times to find solutions with values equal to those of column 2. Finally, column 14 lists the times taken by the faster 95% of these 200 runs.

The GRASP-PR heuristic found the target values on all 200 runs for each instance in Tables 9–12, with an improvement varying between a factor of 7.3 (1.1) and over 5000 (750) with respect to the best method of Elloumi et al. (2003). For each problem group, the improvement was computed by taking the mean value of the average GRASP-PR running times for that group with respect to the running times in boldface.

4 Concluding remarks

In this paper, we propose a new GRASP with path-relinking heuristic for the generalized quadratic assignment problem. To better explore the solution space, this heuristic incorporates randomization in three novel components: a construction procedure, a local search procedure, and a path-relinking procedure with repair.

Table 9 Summary (Part I) of results for the seven algorithms of Elloumi et al. (2003) and the GRASP-PR heuristic on the instances used in Elloumi et al. (2003). Times in seconds are measured on a 400 MHz Pentium II for the algorithms of Elloumi et al. (2003) and on a 2.66 GHz Intel Xeon running Java for GRASP-PR. Normalized times are given *between parentheses*

		Algs. of Elloumi et al.: avg time in secs (avg error = (bks − lower)/bks)							GRASP-PR times to bks				
Instance	Bks	L1	L2	L3	S0	S1	S2	D0	Min	Max	Avg	S.d.	0.95
c1003Aa	**1616**								0.038 (0.101)	0.927 (2.466)	0.191 (0.507)	0.139 (0.370)	0.518 (1.378)
c1003Ab	**1390**								0.035 (0.093)	0.991 (2.636)	0.210 (0.559)	0.154 (0.409)	0.505 (1.343)
c1003Ac	**1730**	1 (0.4)	1 (0.4)	1 (0.4)	6 (2.4)	948 (379.2)	**884** (**353.6**)	30 (12)	0.037 (0.098)	0.609 (1.620)	0.204 (0.544)	0.133 (0.354)	0.482 (1.282)
c1003Ad	**1289**	(68%)	(12%)	(23%)	(69%)	(9%)	(**3**%)	(15%)	0.031 (0.082)	0.535 (1.423)	0.168 (0.447)	0.114 (0.304)	0.419 (1.115)
c1003Ae	**1048**								0.040 (0.106)	0.996 (2.649)	0.322 (0.856)	0.182 (0.483)	0.670 (1.782)
c1003Ba	**1299**								0.023 (0.061)	0.815 (2.168)	0.209 (0.555)	0.127 (0.338)	0.505 (1.343)
c1003Bb	**865**								0.041 (0.109)	0.838 (2.229)	0.331 (0.882)	0.194 (0.516)	0.684 (1.819)
c1003Bc	**1154**	1 (0.4)	1 (0.4)	1 (0.4)	144 (57.6)	998 (399.2)	**890** (**356**)	25 (10)	0.032 (0.085)	1.142 (3.038)	0.335 (0.891)	0.222 (0.591)	0.880 (2.341)
c1003Bd	**834**	(94%)	(29%)	(37%)	(94%)	(26%)	(**12**%)	(33%)	0.034 (0.090)	0.768 (2.043)	0.249 (0.661)	0.151 (0.401)	0.546 (1.452)
c1003Be	**812**								0.031 (0.082)	0.668 (1.777)	0.189 (0.503)	0.126 (0.334)	0.462 (1.229)
c1003Ca	**455**								0.033 (0.088)	0.516 (1.373)	0.185 (0.491)	0.117 (0.312)	0.426 (1.133)
c1003Cb	**467**								0.033 (0.088)	1.089 (2.897)	0.114 (0.303)	0.099 (0.263)	0.301 (0.801)
c1003Cc	**475**	1 (0.4)	**1** (**0.4**)	1 (0.4)	3 (1.2)	96 (38.4)	274 (109.6)	19 (7.6)	0.034 (0.090)	0.558 (1.484)	0.164 (0.437)	0.110 (0.292)	0.408 (1.085)
c1003Cd	**472**	(4%)	(**1**%)	(4%)	(11%)	(1%)	(1%)	(1%)	0.038 (0.101)	0.340 (0.904)	0.105 (0.279)	0.075 (0.199)	0.285 (0.758)
c1003Ce	**350**								0.033 (0.088)	0.349 (0.928)	0.112 (0.299)	0.091 (0.241)	0.298 (0.793)
c1003Da	**843**								0.022 (0.059)	0.756 (2.011)	0.185 (0.493)	0.134 (0.356)	0.470 (1.250)
c1003Db	**879**								0.032 (0.085)	0.662 (1.761)	0.150 (0.399)	0.108 (0.288)	0.417 (1.109)
c1003Dc	**1230**	1 (0.4)	1 (0.4)	1 (0.4)	2 (0.8)	798 (319.2)	**568** (**227.2**)	27 (10.8)	0.032 (0.085)	0.606 (1.612)	0.196 (0.522)	0.122 (0.326)	0.477 (1.269)
c1003Dd	**956**	(100%)	(25%)	(39%)	(100%)	(22%)	(**8**%)	(32%)	0.029 (0.077)	0.494 (1.314)	0.185 (0.493)	0.114 (0.304)	0.435 (1.157)
c1003De	**848**								0.036 (0.096)	0.618 (1.644)	0.199 (0.529)	0.125 (0.333)	0.478 (1.271)

Table 10 Summary (Part II) of results for the seven algorithms of Elloumi et al. (2003) and the GRASP-PR heuristic on the instances used in Elloumi et al. (2003). Times in seconds are measured on a 400 MHz Pentium II for the algorithms of Elloumi et al. (2003) and on a 2.66 GHz Intel Xeon running Java for GRASP-PR. Normalized times are given *between parentheses*

		Algs. of Elloumi et al.: avg time in secs (avg error = (bks − lower)/bks)							GRASP-PR times to bks				
Instance	Bks	L1	L2	L3	S0	S1	S2	D0	Min	Max	Avg	S.d.	0.95
1003Aa	**731**								0.032 (0.085)	0.377 (1.003)	0.081 (0.215)	0.040 (0.106)	0.441 (1.173)
1003Ab	**713**								0.019 (0.051)	0.238 (0.633)	0.070 (0.187)	0.032 (0.085)	0.318 (0.846)
1003Ac	**645**	1 (0.4)	1 (0.4)	1 (0.4)	4 (1.6)	575 (230)	**458** (**183.2**)	26 (10.4)	0.032 (0.085)	0.343 (0.912)	0.085 (0.226)	0.042 (0.110)	0.393 (1.045)
1003Ad	**688**	(37%)	(21%)	(27%)	(41%)	(12%)	(**5**%)	(25%)	0.029 (0.077)	0.127 (0.338)	0.061 (0.162)	0.020 (0.053)	0.128 (0.340)
1003Ae	**715**								0.02 (0.053)	0.142 (0.378)	0.063 (0.169)	0.021 (0.057)	0.109 (0.290)
1003Ba	**306**								0.02 (0.053)	0.24 (0.638)	0.097 (0.259)	0.043 (0.115)	0.513 (1.365)
1003Bb	**528**								0.032 (0.085)	0.383 (1.018)	0.123 (0.328)	0.063 (0.169)	0.487 (1.295)
1003Bc	**326**	1 (0.4)	1 (0.4)	1 (0.4)	12 (4.8)	860 (344)	**898** (**359.2**)	25 (10)	0.029 (0.077)	0.260 (0.692)	0.110 (0.291)	0.046 (0.122)	0.517 (1.375)
1003Bd	**364**	(85%)	(48%)	(57%)	(85%)	(29%)	(**11**%)	(57%)	0.021 (0.056)	0.559 (1.487)	0.139 (0.371)	0.077 (0.206)	0.295 (0.785)
1003Be	**324**								0.02 (0.053)	0.248 (0.660)	0.090 (0.239)	0.040 (0.107)	0.165 (0.439)
1003Ca	**346**								0.033 (0.088)	0.126 (0.335)	0.061 (0.164)	0.017 (0.044)	0.33 (0.878)
1003Cb	**424**								0.025 (0.067)	0.674 (1.793)	0.098 (0.262)	0.059 (0.157)	0.465 (1.237)
1003Cc	**347**	1 (0.4)	**1** (**0.4**)	1 (0.4)	4 (1.6)	189 (75.6)	156 (62.4)	19 (7.6)	0.033 (0.088)	0.082 (0.218)	0.047 (0.124)	0.009 (0.023)	0.230 (0.612)
1003Cd	**434**	(2%)	(**1**%)	(3%)	(4%)	(2%)	(1%)	(1%)	0.031 (0.082)	0.326 (0.867)	0.066 (0.175)	0.026 (0.070)	0.097 (0.258)
1003Ce	**285**								0.028 (0.074)	0.074 (0.197)	0.043 (0.116)	0.006 (0.017)	0.048 (0.128)
1003Da	**219**								0.02 (0.053)	0.386 (1.027)	0.103 (0.275)	0.057 (0.153)	0.629 (1.673)
1003Db	**402**								0.030 (0.080)	0.425 (1.131)	0.119 (0.316)	0.058 (0.155)	0.619 (1.647)
1003Dc	**297**	1 (0.4)	1 (0.4)	1 (0.4)	3 (1.2)	1631 (652.4)	**855** (**342**)	25 (10)	0.028 (0.074)	0.196 (0.521)	0.079 (0.210)	0.031 (0.083)	0.431 (1.146)
1003Dd	**445**	(100%)	(43%)	(52%)	(100%)	(27%)	(**13**%)	(55%)	0.025 (0.067)	0.318 (0.846)	0.112 (0.297)	0.060 (0.159)	0.398 (1.059)
1003De	**358**								0.029 (0.077)	0.940 (2.500)	0.073 (0.195)	0.067 (0.179)	0.147 (0.391)

Table 11 Summary (Part III) of results for the seven algorithms of Elloumi et al. (2003) and the GRASP-PR heuristic on the instances used in Elloumi et al. (2003). Times in seconds are measured on a 400 MHz Pentium II for the algorithms of Elloumi et al. (2003) and on a 2.66 GHz Intel Xeon running Java for GRASP-PR. Normalized times are given *between parentheses*

	Algs. of Elloumi et al.: avg time in secs (avg error = (bks − lower)/bks)							GRASP-PR times to bks				
Instance Bks	L1	L2	L3	S0	S1	S2	D0	Min	Max	Avg	S.d.	0.95
2005Aa **3059**								0.180 (0.479)	14.152 (37.644)	3.420 (9.097)	2.371 (6.306)	19.268 (51.253)
2005Ab **2954**								0.143 (0.380)	3.089 (8.217)	1.041 (2.768)	0.550 (1.464)	4.585 (12.196)
2005Ac **3012**	1 (0.4)	82 (32.8)	4 (1.6)	59 (23.6)	1800 (720)	**1800** (**720**)	238 (95.2)	0.199 (0.529)	4.173 (11.100)	1.119 (2.975)	0.592 (1.574)	3.961 (10.536)
2005Ad **3174**	(85%)	(12%)	(27%)	(85%)	(10%)	(**2**%)	(19%)	0.108 (0.287)	3.561 (9.472)	1.056 (2.809)	0.508 (1.350)	2.51 (6.677)
2005Ae **3054**								0.163 (0.434)	9.363 (24.906)	1.685 (4.483)	1.338 (3.559)	4.570 (12.156)
2005Ba **2442**								0.663 (1.764)	54.714 (145.539)	14.731 (39.184)	10.785 (28.687)	49.609 (131.960)
2005Bb **2088**								0.210 (0.559)	42.142 (112.098)	4.070 (10.827)	5.092 (13.544)	17.237 (45.850)
2005Bc **1986**	1 (0.4)	75 (30)	4 (1.6)	600 (240)	1800 (720)	**1800** (**720**)	223 (89.2)	0.281 (0.747)	11.458 (30.478)	2.643 (7.031)	1.885 (5.015)	9.428 (25.078)
2005Bd **2449**	(98%)	(17%)	(32%)	(98%)	(16%)	(**3**%)	(26%)	0.413 (1.099)	8.503 (22.618)	2.452 (6.522)	1.597 (4.248)	6.331 (16.840)
2005Be **2453**								0.363 (0.966)	7.904 (21.025)	1.938 (5.154)	1.099 (2.923)	4.177 (11.111)
2005Ca **783**								0.062 (0.165)	0.936 (2.490)	0.141 (0.375)	0.069 (0.183)	0.686 (1.825)
2005Cb **636**								0.060 (0.160)	0.319 (0.849)	0.162 (0.432)	0.058 (0.153)	0.843 (2.242)
2005Cc **772**	1 (0.4)	78 (31.2)	4 (1.6)	71 (28.4)	1800 (720)	**1800** (**720**)	239.8 (95.9)	0.057 (0.152)	0.702 (1.867)	0.244 (0.649)	0.114 (0.302)	1.387 (3.689)
2005Cd **682**	(19%)	(1%)	(10%)	(31%)	(1%)	(**0**%)	(1%)	0.057 (0.152)	0.268 (0.713)	0.139 (0.369)	0.046 (0.123)	0.239 (0.636)
2005Ce **732**								0.060 (0.160)	0.406 (1.080)	0.165 (0.440)	0.065 (0.173)	0.300 (0.798)
2005Da **2413**								0.578 (1.537)	41.654 (110.800)	5.591 (14.872)	6.004 (15.970)	25.535 (67.923)
2005Db **2316**								0.291 (0.774)	12.017 (31.965)	2.271 (6.040)	1.625 (4.323)	8.646 (22.998)
2005Dc **1965**	1 (0.4)	66 (26.4)	4 (1.6)	34 (13.6)	1800 (720)	**1800** (**720**)	206 (82.4)	0.072 (0.192)	3.126 (8.315)	1.188 (3.161)	0.636 (1.692)	3.343 (8.892)
2005Dd **2211**	(100%)	(18%)	(30%)	(100%)	(14%)	(**3**%)	(24%)	0.085 (0.226)	2.940 (7.820)	0.863 (2.295)	0.378 (1.005)	1.402 (3.729)
2005De **2302**								0.096 (0.255)	6.069 (16.144)	1.531 (4.073)	0.891 (2.371)	3.302 (8.783)

Table 12 Summary (Part IV) of results for the seven algorithms of Elloumi et al. (2003) and the GRASP-PR heuristic on the instances used in Elloumi et al. (2003). Times in seconds are measured on a 400 MHz Pentium II for the algorithms of Elloumi et al. (2003) and on a 2.66 GHz Intel Xeon running Java for GRASP-PR. Normalized times are given *between parentheses*

		Algs. of Elloumi et al.: avg time in secs (avg error = (bks − lower)/bks)							GRASP-PR times to bks				
Instance	Bks	L1	L2	L3	S0	S1	S2	D0	Min	Max	Avg	S.d.	0.95
2408Aa	**5643**	Instances in http://cedric.cnam.fr/oc/TAP/all_CTAP_instances/							1.604 (4.27)	50.445 (134.18)	10.288 (27.37)	8.369 (22.26)	29.735 (79.10)
2408Ca	**1028**	but not in Elloumi et al. (2003)							0.149 (0.40)	2.307 (6.14)	0.714 (1.90)	0.447 (1.19)	1.594 (4.24)
c2005Aa	**6412**								1.209 (3.22)	18.117 (48.19)	5.280 (14.05)	3.281 (8.73)	13.211 (35.14)
c2005Ab	**6260**								0.253 (0.67)	15.489 (41.20)	3.125 (8.31)	2.047 (5.45)	7.442 (19.80)
c2005Ac	**6491**	1 (0.4)	72 (28.8)	4 (1.6)	85 (34)	1800 (720)	**1800** (**720**)	214 (85.6)	0.346 (0.92)	20.114 (53.50)	4.466 (11.88)	3.146 (8.37)	11.287 (30.02)
c2005Ad	**6267**	(69%)	(29%)	(40%)	(70%)	(12%)	(**3**%)	(35%)	0.237 (0.63)	8.763 (23.31)	1.935 (5.15)	1.088 (2.90)	3.982 (10.59)
c2005Ae	**6194**								0.185 (0.49)	11.353 (30.20)	2.923 (7.78)	2.132 (5.67)	7.264 (19.32)
c2005Ba	**5420**								1.457 (3.88)	42.049 (111.85)	10.596 (28.19)	6.976 (18.56)	26.247 (69.82)
c2005Bb	**5370**								0.665 (1.77)	24.880 (66.18)	4.433 (11.79)	3.292 (8.76)	11.138 (29.63)
c2005Bc	**5645**	1 (0.4)	81 (32.4)	4 (1.6)	199 (79.6)	1800 (720)	**1800** (**720**)	156 (62.4)	1.163 (3.09)	40.923 (108.86)	9.447 (25.13)	6.924 (18.42)	25.171 (66.96)
c2005Bd	**5420**	(98%)	(17%)	(32%)	(98%)	(16%)	(**3**%)	(26%)	0.260 (0.69)	22.465 (59.76)	3.345 (8.90)	3.027 (8.05)	9.235 (24.57)
c2005Be	**5836**								0.725 (1.93)	25.527 (67.90)	3.901 (10.38)	3.161 (8.41)	10.125 (26.93)
c2005Ca	**1181**								0.134 (0.36)	3.356 (8.93)	1.059 (2.82)	0.619 (1.65)	2.145 (5.71)
c2005Cb	**1017**								0.071 (0.19)	3.385 (9.00)	0.377 (1.00)	0.287 (0.76)	0.746 (1.98)
c2005Cc	**1197**	1 (0.4)	**69** (**27.6**)	4 (1.6)	58 (23.2)	1800 (720)	1800 (720)	228 (91.2)	0.11 (0.29)	3.558 (9.46)	0.996 (2.65)	0.810 (2.16)	3.04 (8.09)
c2005Cd	**1038**	(2%)	(**0**%)	(7%)	(12%)	(1%)	(0%)	(1%)	0.061 (0.16)	0.402 (1.07)	0.171 (0.46)	0.061 (0.16)	0.295 (0.79)
c2005Ce	**1166**								0.057 (0.15)	0.564 (1.50)	0.195 (0.52)	0.084 (0.22)	0.387 (1.03)
c2005Da	**5139**								0.421 (1.12)	7.787 (20.71)	2.584 (6.87)	1.441 (3.83)	5.872 (15.62)
c2005Db	**5519**								0.548 (1.46)	29.663 (78.90)	4.824 (12.83)	3.935 (10.47)	12.613 (33.55)
c2005Dc	**5907**	1 (0.4)	69 (27.6)	4 (1.6)	34 (13.6)	1800 (720)	**1800** (**720**)	127 (50.8)	0.834 (2.22)	55.999 (148.96)	9.945 (26.45)	9.786 (26.03)	36.641 (97.47)
c2005Dd	**5494**	(100%)	(56%)	(63%)	(100%)	(20%)	(**9**%)	(66%)	0.505 (1.34)	3.977 (10.58)	1.494 (3.98)	0.699 (1.86)	2.878 (7.66)
c2005De	**5435**								0.083 (0.22)	16.764 (44.59)	2.129 (5.67)	1.745 (4.64)	4.965 (13.21)

With the exception of the construction procedure, randomization is not common in the components of a GRASP heuristic. Randomization in construction is usually confined to the selection of a solution element from the restricted candidate list. Similarly, path-relinking procedures do not usually incorporate randomization. A well-known exception is the greedy randomized adaptive path-relinking variant of Faria et al. (2005).

In addition to enabling better exploration of the solution space, randomization is useful to help deal with infeasibilities. The randomized construction steps are repeated until either a feasible solution is constructed or a maximum number of repetitions is reached. The main role of randomization in the path-relinking phase is to enable the repair of infeasible solutions that can be produced when an attribute of the guiding solution is incorporated in the current solution.

Randomization in the local search is used to sample moves from those leading to an improving solution. Since the entire neighborhood is not evaluated, this sampling speeds up the search without compromising its effectiveness. Since not all neighborhood solutions are evaluated and the procedure terminates when no improving solution is sampled, the resulting solution may not be a true local minimum. We call such solutions approximate local minima.

The algorithm was implemented in Java and tested extensively on a set of 128 instances available in the literature (Cordeau et al. 2006; Lee and Ma 2005; Elloumi et al. 2003). Previous papers on this subject only considered small subsets of these instances. For each instance (except 30-20-95 of Cordeau et al. 2006), 200 independent runs of the heuristic were done.[3] Each run terminated when the best-known solution for that instance was found. For all instances, the heuristic found the best-known solution on all runs, demonstrating its robustness.

For all instances, except 30-20-95, 35-15-95, and 50-10-75 of Cordeau et al. (2006), the average running time of the GRASP with path-relinking heuristic was never greater than the running time reported for the other heuristics in the literature. In addition to the aforementioned three instances of Cordeau et al. (2006), average normalized times of the GRASP with path-relinking were longer than the reported times for the memetic algorithm only on instances 30-10-65 and 35-15-35. However, normalized minimum running times of the GRASP with path-relinking heuristic were smaller than those of the memetic algorithm on all instances tested. Even on instance 30-20-95, where we measured the longest running times, the fastest of the 18 solution times of the GRASP with path-relinking heuristic implemented in Java was 834 seconds on a 2.66 GHz processor (normalized to 2218 seconds) while the time reported for the C implementation by Cordeau et al. (2006) was 5232 seconds on a 1.2 GHz processor (normalized to 6278 seconds).

With respect to exact methods, our average running times were often orders of magnitude smaller than those reported in the literature. This is not surprising since the same is observed on the quadratic assignment problem where heuristics can quickly find a solution having an optimal objective function value whereas exact methods on large computational grids sometimes take months of computation to prove optimality of these solutions (Anstreicher et al. 2002).

[3] Because of long running times, on instance 30-20-95, only 18 independent runs were done.

Taking into account the effects of processor speed and programming languages, we believe that the heuristic proposed in this paper is competitive with the algorithms proposed in the literature for solving the generalized quadratic assignment problem.

Acknowledgements The research of R.M.A. Silva was partially supported by the Brazilian National Council for Scientific and Technological Development (CNPq) and the Foundation for Support of Research of the State of Minas Gerais, Brazil (FAPEMIG).

References

Aiex, R.M., Resende, M.G.C., Ribeiro, C.C.: Probability distribution of solution time in GRASP: An experimental investigation. J. Heuristics **8**, 343–373 (2002)

Aiex, R.M., Binato, S., Resende, M.G.C.: Parallel GRASP with path-relinking for job shop scheduling. Parallel Comput. **29**, 393–430 (2003)

Aiex, R.M., Pardalos, P.M., Resende, M.G.C., Toraldo, G.: GRASP with path-relinking for three-index assignment. INFORMS J. Comput. **17**, 224–247 (2005)

Aiex, R.M., Resende, M.G.C., Ribeiro, C.C.: TTTPLOTS: A Perl program to create time-to-target plots. Opt. Lett. **1**, 201–212 (2007)

Anstreicher, K., Brixius, N., Goux, J.P., Linderoth, J.: Solving large quadratic assignment problems on computational grids. Math. Program. **91**, 563–588 (2002)

Barahona, F., Anbil, R.: The volume algorithm: producing primal solutions with a subgradient method. Math. Program. **87**, 385–399 (2000)

Burkard, R.E.: Locations with spatial interactions: the quadratic assignment problem. In: Discrete Location Theory, pp. 89–98. Wiley, New York (1991)

Cordeau, J.-F., Gaudioso, M., Laporte, G., Moccia, L.: A memetic heuristic for the generalized quadratic assignment problem. INFORMS J. Comput. **18**, 433–443 (2006)

Elloumi, S.: Contribution à la résolution des programmes non linéaires en variables 0-1, application aux problèmes de placement de tâches dans les systèmes distribués (thèse de doctorat in informatique). Technical report, Conservatoire National des Arts et Métiers, Paris (1991)

Elloumi, S., Roupin, F., Soutif, E.: Comparison of different lower bounds for the constrained module allocation problem. Technical Report 473, CNAM-Laboratoire Cédric, 292 Rue St Martin, 75141 Paris Cedex 03, France (2003)

Faria, H. Jr., Binato, S., Resende, M.G.C., Falcão, D.J.: Transmission network design by a greedy randomized adaptive path relinking approach. IEEE Trans. Power Syst. **20**(1), 43–49 (2005)

Feo, T.A., Resende, M.G.C.: A probabilistic heuristic for a computationally difficult set covering problem. Oper. Res. Lett. **8**, 67–71 (1989)

Feo, T.A., Resende, M.G.C.: Greedy randomized adaptive search procedures. J. Global Optim. **6**, 109–133 (1995)

Festa, P., Resende, M.G.C.: GRASP: An annotated bibliography. In: Ribeiro, C.C., Hansen, P. (eds.) Essays and Surveys on Metaheuristics, pp. 325–367. Kluwer Academic, Norwell (2002)

Festa, P., Resende, M.G.C.: An annotated bibliography of GRASP—Part I: Algorithms. Int. Trans. Oper. Res. **16**, 1–24 (2009a)

Festa, P., Resende, M.G.C.: An annotated bibliography of GRASP—Part II: Applications. Int. Trans. Oper. Res. **16**, 131–172 (2009b)

Frieze, A., Yadegar, J.: On the quadratic assignment problem. Discrete Appl. Math. **5**, 89–98 (1983)

Glover, F.: Tabu search and adaptive memory programing—Advances, applications and challenges. In: Barr, R.S., Helgason, R.V., Kennington, J.L. (eds.) Interfaces in Computer Science and Operations Research, pp. 1–75. Kluwer Academic, Norwell (1996)

Glover, F., Laguna, M., Martí, R.: Fundamentals of scatter search and path relinking. Control Cybern. **39**, 653–684 (2000)

Hahn, P.M., Kim, B.-J., Guignard, M., MacGregor Smith, J., Zhu, Y.-R.: An algorithm for the generalized quadratic assignment problem. Comput. Optim. Appl. **401**, 351–372 (2008)

Kaufman, L., Broeckx, F.: An algorithm for the quadratic assignment problem using benders decomposition. Eur. J. Oper. Res. **2**, 204–211 (1978)

Laguna, M., Martí, R.: GRASP and path relinking for 2-layer straight line crossing minimization. INFORMS J. Comput. **11**, 44–52 (1999)

Lee, C.-G., Ma, Z.: The generalized quadratic assignment problem. Technical Report MIEOR TR2005-01, Department of Mechanical and Industrial Engineering at the University of Toronto (2005)

Li, Y., Pardalos, P.M., Resende, M.G.C.: A greedy randomized adaptive search procedure for the quadratic assignment problem. In: Pardalos, P.M., Wolkowicz, H. (eds.) Quadratic Assignment and Related Problems. DIMACS Series in Discrete Matehematics and Theoretical Computer Science, vol. 16, pp. 237–261. American Mathematical Society, Providence (1994)

Matsumoto, M., Nishimura, T.: Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Trans. Model. Comput. Simul. **8**, 3–30 (1998)

Oliveira, C.A.S., Pardalos, P.M., Resende, M.G.C.: GRASP with path-relinking for the quadratic assignment problem. In: Ribeiro, C.C., Martins, S.L. (eds.) Efficient and Experimental Algorithms. Lecture Notes in Computer Science, vol. 3059, pp. 356–368. Springer-Verlag, Berlin (2004)

Padberg, M., Rijal, M.: Location, Scheduling, Design and Integer Programming. Kluwer Academic, Norwell (1996)

Pardalos, P.M., Rendl, F., Wolkowicz, H.: The quadratic assignment problem: A survey and recent development. In: Pardalos, P.M., Wolkowicz, H. (eds.) The Quadratic Assignment and Related Problems. DIMACS Series in Discrete Matehematics and Theoretical Computer Science, vol. 16, pp. 1–42. American Mathematical Society, Providence (1994)

Pessoa, A.A., Hahn, P.M., Guignard, M., Zhu, Y.-R.: An improved algorithm for the generalized quadratic assignment problem. Technical Report MIEOR TR2005-01, Electrical and Systems Engineering at the University of Pennsylvania (2008)

Resende, M.G.C., Ribeiro, C.C.: Greedy randomized adaptive search procedures. In: Glover, F., Kochenberger, G. (eds.) Handbook of Metaheuristics, pp. 219–249. Kluwer Academic, Norwell (2002)

Resende, M.G.C., Ribeiro, C.C.: A GRASP with path-relinking for private virtual circuit routing. Networks **41**(1), 104–114 (2003)

Resende, M.G.C., Ribeiro, C.C.: GRASP with path-relinking: Recent advances and applications. In: Ibaraki, T., Nonobe, K., Yagiura, M. (eds.) Metaheuristics: Progress as Real Problem Solvers, pp. 29–63. Springer-Verlag, Berlin (2005)

Resende, M.G.C., Ribeiro, C.C.: Greedy randomized adaptive search procedures: Advances and applications. In: Gendreau, M., Potvin, J.-Y. (eds.) Handbook of Metaheuristics, 2nd edn. Springer Science+Business Media, Berlin (2010)

Ribeiro, C.C., Rosseti, I.: A parallel GRASP for the 2-path network design problem. In: Lecture Notes in Computer Science, vol. 2004, pp. 922–926. Springer-Verlag, Berlin (2002)

Ribeiro, C.C., Uchoa, E., Werneck, R.F.: A hybrid GRASP with perturbations for the Steiner problem in graphs. INFORMS J. Comput. **14**, 228–246 (2002)

Roupin, F.: From linear to semidefinite programming: An algorithm to obtain semidefinite relaxations for bivalent quadratic problems. J. Combin. Optim. **8**, 469–493 (2004)