INFORMS Journal on Computing
Vol. 17, No. 2, Spring 2005, pp. 224–247
ISSN 1091-9856 | EISSN 1526-5528 | 05 | 1702 | 0224

DOI 10.1287/ijoc.1030.0059

GRASP with Path Relinking for Three-Index Assignment

Renata M. Aiex
Department of Computer Science, Catholic University of Rio de Janeiro, R. Marquês de São Vicente, 225, Rio de Janeiro, RJ 22453-900, Brazil, rma@inf.puc-rio.br

Mauricio G. C. Resende
Internet and Network Systems Research Center, AT&T Labs Research, 180 Park Avenue, Room C241, Florham Park, New Jersey 07932, USA, mgcr@research.att.com

Panos M. Pardalos
Department of Industrial and Systems Engineering, University of Florida, 303 Weil Hall, Gainesville, Florida 32611, USA, pardalos@ufl.edu

Gerardo Toraldo
Department of Agricultural Engineering and Agronomy, University of Naples Federico II, via Università 100, 80055 Portici, Italy, toraldo@unina.it

This paper proposes and tests variants of GRASP (greedy randomized adaptive search procedure) with path relinking for the three-index assignment problem (AP3). GRASP is a multistart metaheuristic for combinatorial optimization. It usually consists of a construction procedure based on a greedy randomized algorithm and of a local search. Path relinking is an intensification strategy that explores trajectories that connect high-quality solutions. Several variants of the heuristic are proposed and tested. Computational results show clearly that this GRASP for AP3 benefits from path relinking and that the variants considered in this paper compare well with previously proposed heuristics for this problem. GRASP with path relinking was able to improve the solution quality of heuristics proposed by Balas and Saltzman (1991), Burkard et al. (1996), and Crama and Spieksma (1992) on all instances proposed in those papers. We show that the random variable "time to target solution," for all proposed GRASP with path-relinking variants, fits a two-parameter exponential distribution. To illustrate the consequence of this, one of the variants of GRASP with path relinking is shown to benefit from parallelization.

Key words: heuristics; GRASP; path relinking; statistical analysis; three-index assignment problem; parallel programming
History: Accepted by Michael Gendreau, Heuristic Search and Learning; received December 2000; revised May 2002, March 2003; accepted June 2003.

1. Introduction

The three-index assignment problem (AP3) was first stated by Pierskalla (1967) as a straightforward extension of the classical two-dimensional assignment problem. It can be viewed as an optimization problem on a complete tripartite graph $K_{n,n,n} = (I \cup J \cup K, (I \times J) \cup (I \times K) \cup (J \times K))$, where I, J, and K are disjoint sets of size n. If a cost $c_{i,j,k}$ is associated with each triangle $(i, j, k) \in I \times J \times K$, then the AP3 consists of finding a subset $A \in I \times J \times K$ of n triangles such that every element of $I \cup J \cup K$ occurs in exactly one triangle of A, and the sum of the costs of the chosen triangles is minimized.

The AP3 has a 0–1 integer programming formulation:

$$\min \sum_{i \in I,\, j \in J,\, k \in K} c_{ijk} x_{ijk}$$

subject to

$$\sum_{j \in J,\, k \in K} x_{ijk} = 1, \quad \forall i \in I,$$

$$\sum_{i \in I,\, k \in K} x_{ijk} = 1, \quad \forall j \in J,$$

$$\sum_{i \in I,\, j \in J} x_{ijk} = 1, \quad \forall k \in K,$$

$$x_{ijk} \in \{0, 1\}, \quad \forall i \in I,\ j \in J,\ k \in K,$$

where $I = J = K = \{1, 2, \ldots, n\}$.

The above formulation models, for example, the problem of assigning jobs to workers to machines at minimum cost. c_{ijk} is the cost of assigning job j to worker i on machine k. The 0–1 decision variable $x_{ijk} = 1$ if, and only if, job j is assigned to worker i on machine k. Each constraint implies that each element

of a set is assigned to exactly one element of each of the other two sets.

The AP3 can be also formulated using permutation functions. There are n^3 cost elements and the optimal solution of the AP3 consists of the n smallest, such that the constraints are not violated. Assign to each set I, J, and K the numbers $1, 2, \ldots, n$. The three sets of constraints can now be seen as one. None of the chosen cost elements c_{ijk} is allowed to have the same value for indices i, j, and k as another. For example, $x_{1,2,4} = x_{3,2,5} = 1$ is infeasible, since these assignments share index $j = 2$. The permutation-based formulation for the AP3 is

$$\min_{p,\, q \in \pi_N} \sum_{i=1}^{n} c_{ip(i)q(i)}$$

where π_N denotes the set of all permutations of the set of integers $N = \{1, 2, \ldots, n\}$. Note that $|\pi_N| = n!$.

An equivalent formulation of the AP3 (often called the *three-dimensional matching problem*) is the following: Given three disjoint sets I, J, and K such that $|I| = |J| = |K| = n$ and a weight c_{ijk} associated with each ordered triplet $(i, j, k) \in I \times J \times K$, find a minimum-weight collection of n disjoint triplets $(i, j, k) \in I \times J \times K$.

The permutation-based formulation has several advantages. Apart from being simple and compact, it facilitates the implementation of a heuristic for the AP3 since the constraints can be taken care of by the objective function itself.

The AP3 is NP-Complete (Frieze 1983, Garey and Johnson 1979). Applications of the AP3 can be found in Pierskalla (1967, 1968), Frieze and Yadegar (1981), and Crama et al. (1990) and include scheduling ingots in soaking pits in a rolling mill, scheduling capital investments, military troop assignment, satellite coverage optimization, scheduling teaching practice, and production of printed circuit boards.

Exact and heuristic algorithms have been proposed for the three-index assignment problem, including Balas and Saltzman (1991), Burkard and Fröhlich (1980), Burkard and Rudolf (1993), Burkard et al. (1996), Crama and Spieksma (1992), Fröhlich (1979), Hansen and Kaufman (1973), Leue (1972), Pardalos and Pitsoulis (2000), Pierskalla (1967, 1968), Vlach (1967), and Voss (2000).

The aim of this paper is to propose a new class of heuristics for the three-index assignment problem, evaluate experimentally these heuristics, compare solutions found by the new heuristics with previously known solutions, and show that the new heuristics can benefit from parallelization.

The remainder of the paper is organized as follows. In §2, the construction and local search phases of a GRASP for the AP3 are described. Path relinking is presented in §3. Section 4 shows how GRASP and path relinking are combined. A parallel implementation using the Message Passing Interface library is shown in §5. In §6, computational results, using the sequential and parallel implementations, are reported. Concluding remarks are made in §7.

2. GRASP Construction and Local Search

2.1. GRASP

A greedy randomized adaptive search procedure (GRASP) (Feo and Resende 1989, 1995; Festa and Resende 2002) is a multistart or iterative process in which each GRASP iteration consists of two phases. In a construction phase, a feasible solution is produced, and in a local search phase, a local optimum in the neighborhood of the constructed solution is sought. The best overall solution is kept as the result.

In the construction phase, a feasible solution is iteratively constructed, one element at a time. The basic GRASP construction phase is similar to the semi-greedy heuristic proposed independently by Hart and Shogan (1987). At each construction iteration, the choice of the next element to be added is determined by ordering all candidate elements (i.e., those that can be added to the solution) in a candidate list C with respect to a greedy function g: $C \to \mathbb{R}$. This function measures the (myopic) benefit of selecting each element. The heuristic is adaptive because the benefits associated with every element are updated at each iteration of the construction phase to reflect the changes brought on by the selection of the previous element. The probabilistic component of a GRASP is characterized by randomly choosing one of the best candidates in the list, but not necessarily the top candidate. The list of best candidates is called the *restricted candidate list* (RCL).

It is almost always beneficial to apply a local search to attempt to improve each constructed solution. A local search algorithm works in an iterative fashion by successively replacing the current solution by a better solution in the neighborhood of the current solution. It terminates when no better solution is found in the neighborhood.

GRASP has been applied to numerous assignment problems (Ahuja et al. 2000; Feo and González-Velarde 1995; Fleurent and Glover 1999; Li et al. 1994; Murphey et al. 1998; Mavridou et al. 1998; Pardalos et al. 1995, 1997; Pardalos and Resende 2002; Pitsoulis 1999; Pitsoulis et al. 2001; Rangel et al. 1999; Resende et al. 1996; Robertson 2001).

2.2. GRASP Construction for AP3

The GRASP construction phase builds a feasible solution S by selecting n triplets, one at a time. Figure 1 illustrates the construction phase in pseudo-code.

```
procedure CONSTRUCT(seed, n, c, S)
1    Select α ∈ [0,1] at random;
2    S = ∅;
3    C = {(i,j,k) ∈ I × J × K};
4    for p = 1,...,n−1 do
5        c̲ = min{c_ijk | (i,j,k) ∈ C};
6        c̄ = max{c_ijk | (i,j,k) ∈ C};
7        C′ = {(i,j,k) ∈ C | c_ijk ≤ c̲ + α(c̄ − c̲)};
8        Select (i_p, j_p, k_p) ∈ C′ at random;
9        S = S ∪ {(i_p, j_p, k_p)};
10       Γ^i(p) = {(i,j,k) ∈ C | i = i_p};
11       Γ^j(p) = {(i,j,k) ∈ C | j = j_p};
12       Γ^k(p) = {(i,j,k) ∈ C | k = k_p};
13       C = C \ {Γ^i(p) ∪ Γ^j(p) ∪ Γ^k(p)};
14   rof;
15   S = S ∪ C;
end CONSTRUCT
```

Figure 1 The GRASP Construction Phase

A restricted candidate list parameter α is selected at random from the interval $[0, 1]$ (line 1). This value is not changed during the construction phase. The solution S is initially empty and the set C of candidate triplets is initially the set of all triplets (lines 2 and 3).

The loop going from line 4 to 14 selects the first $n-1$ triplets. To select the pth triplet to be added to the solution, a restricted candidate list C' is defined (in lines 5 to 7) to include all triplets (i, j, k) in the candidate set C having cost $c_{ijk} \le \underline{c} + \alpha(\bar{c} - \underline{c})$, where

$$\underline{c} = \min\{c_{ijk} \mid (i,j,k) \in C\} \quad \text{and} \quad \bar{c} = \max\{c_{ijk} \mid (i,j,k) \in C\}.$$

Triplet $(i_p, j_p, k_p) \in C'$ is chosen at random in line 8 and is added to the solution, i.e., $S = S \cup \{(i_p, j_p, k_p)\}$ in line 9.

Once (i_p, j_p, k_p) is selected, the set of candidate triplets must be adjusted to take into account that (i_p, j_p, k_p) is part of the solution. Any triplet (i, j, k) such that $i = i_p$ or $j = j_p$ or $k = k_p$ is removed from the current set of candidate triplets in lines 10 to 13. This updating procedure is the computational bottleneck of the construction phase. A straightforward implementation would scan all $O(n^3)$ cost elements $n-1$ times in order to update the candidate list. We make use of four doubly linked lists to implement this process more efficiently, reducing the complexity from $O(n^4)$ to $O(n^3)$.

A doubly linked list $L_c = \{c_{i,j,k}; (i, j, k)\}$ links the ordered set of triplets (i, j, k) in the candidate list C. The elements of L_c appear in increasing order of cost. L_c is used during the construction of the restricted candidate list C'. The minimum and maximum candidate list cost values (computed in lines five and six of the pseudo-code in Figure 1) are kept in the first and last elements of L_c, respectively. These elements are addressed to by pointers. To compute the RCL, L_c is traversed until the cost associated with the current element is greater than the cutoff value $\underline{c} + \alpha(\bar{c} - \underline{c})$. The traversed elements (before the last visited element) make up C'.

A pointer $P_{i,j,k}$ points to each element $\{c_{i,j,k}; (i, j, k)\}$ of L_c. Three other doubly linked lists, L_i, L_j, and L_k, link, respectively, the i, j, and k indices that still appear in elements of C.

To update C after triplet (i_p, j_p, k_p) is chosen (lines 10 to 13 of the pseudo-code in Figure 1), we first remove i_p from L_i, and traverse L_j. For each element $j \in L_j$, list L_k is traversed. This way, all triplets $(i_p, j, k) \in C$ are traversed. For each triplet $(i_p, j, k) \in C$, we remove from L_c the element pointed to by $P_{i_p,j,k}$. Next, we remove j_p from L_j, and traverse L_i. For each element $i \in L_i$, list L_k is traversed. This way, all remaining triplets $(i, j_p, k) \in C$ are traversed. For each triplet $(i, j_p, k) \in C$, we remove from L_c the element pointed to by $P_{i,j_p,k}$. Finally, we remove k_p from L_k, and traverse L_i. For each element $i \in L_i$, list L_j is traversed. This way, all remaining triplets $(i, j, k_p) \in C$ are traversed. For each triplet $(i, j, k_p) \in C$, we remove from L_c the element pointed to by P_{i,j,k_p}.

After $n-1$ triplets have been selected, the set C of candidate triplets contains one last triplet which is added to S in line 15, thus completing the construction phase.

2.3. GRASP Local Search for AP3

In the local search procedure, the current solution is improved by searching its neighborhood for a better solution. If an improvement is detected, the solution is updated and a new neighborhood search is initialized. The definition of the neighborhood $N(s)$ is crucial for the performance of the local search.

The solution of the AP3 can be represented by a pair of permutations (p, q). Therefore, the solution space consists of all $(n!)^2$ possible combinations of permutations.

Let us first define the difference between two permutations s and s' to be

$$\delta(s, s') = \{i \mid s(i) \neq s'(i)\},$$

and the distance between them to be

$$d(s, s') = |\delta(s, s')|.$$

In this local search a 2-exchange neighborhood is adopted. A 2-exchange neighborhood is defined to be

$$N_2(s) = \{s' \mid d(s, s') = 2\}.$$

In general, a k-exchange neighborhood could be used. When defining the neighborhood of a solution (p, q)

in terms of 2-exchanges, one natural choice is to let it be all possible 2-exchange permutations. However, the size of that neighborhood is $\binom{n}{2}^2$, which is large even for small values of n. We propose, instead, a different scheme, in which the neighborhood of a solution (p, q) consists of all 2-exchange permutations of p plus all 2-exchange permutations of q. This means that for a solution $p, q \in \pi_N$, the 2-exchange neighborhood is

$$N_2(p, q) = \{p', q' \mid d(p, p') + d(q, q') = 2\}.$$

Hence, the size of the neighborhood is $|N_2(p)| + |N_2(q)| = 2\binom{n}{2}$. In the local search, each cost of a neighborhood solution is compared with the cost of the current solution. If the cost of the neighbor is lower, then the solution is updated, the search is halted, and a search in the new neighborhood is initialized. The local search ends when no neighbor of the current solution has a lower cost than the current solution.

Figure 2 illustrates this local search using the triplet solution representation. The solution S, built in the construction phase, is used as the starting point for the local search. The double loop from lines 1 to 18 examines the neighboring solutions. The cost of a neighbor associated with permutation p is computed in line 3. If its cost is better than the cost of the current solution, a move to the new solution is done in lines 5 and 6 and the local search is recursively called, starting from this new solution, in line 7. Likewise, the cost of a neighbor associated with permutation q is computed in line 10. If its cost is better than the cost of the current solution, a move to the new solution is done in lines 12 and 13 and the local search is recursively called, starting from this new solution, in line 14. The procedure ends with a solution that is locally optimal with respect to the neighborhood definition.

```
procedure LOCAL(n, c, c_S, S)
1    for p = 1,...,n-1 do
2        for q = p+1,...,n do
3            c_j = c_S - c_{i_p,j_p,k_p} - c_{i_q,j_q,k_q} + c_{i_p,j_q,k_p} + c_{i_q,j_p,k_q};
4            if (c_j < c_S) then
5                S̄ = S \ {(i_p, j_p, k_p)} \ {(i_q, j_q, k_q)};
6                S = S̄ ∪ {(i_p, j_q, k_p)} ∪ {(i_q, j_p, k_q)};
7                LOCAL(n, c, c_j, S);
8                return;
9            fi;
10           c_k = c_S - c_{i_p,j_p,k_p} - c_{i_q,j_q,k_q} + c_{i_p,j_p,k_q} + c_{i_q,j_q,k_p};
11           if (c_k < c_S) then
12               S̄ = S \ {(i_p, j_p, k_p)} \ {(i_q, j_q, k_q)};
13               S = S̄ ∪ {(i_p, j_p, k_q)} ∪ {(i_q, j_q, k_p)};
14               LOCAL(n, c, c_k, S);
15               return;
16           fi;
17       rof;
18   rof;
19   return;
end LOCAL
```

Figure 2 The GRASP Local Search Phase

3. Path Relinking

Path relinking was first introduced in the context of tabu search (Glover and Laguna 1997), as an approach to integrate intensification and diversification strategies in the search. See Glover et al. (2000) for a survey of path relinking. It consists of exploring trajectories that connect high-quality solutions, by starting from an *initial solution* and generating a path in the neighborhood of this solution towards another solution, called the *guiding solution*. This path is generated by selecting moves that introduce in the initial solution attributes of the guiding solution. At each step, all moves that incorporate attributes of the guiding solution are analyzed and the move that best improves (or least deteriorates) the initial solution is chosen. Path relinking in the context of GRASP was first introduced by Laguna and Martí (1999).

For the three-index assignment problem, path relinking is done between an initial solution

$$S = \{(1, j_1^S, k_1^S), (2, j_2^S, k_2^S), \ldots, (n, j_n^S, k_n^S)\}$$

and a guiding solution

$$T = \{(1, j_1^T, k_1^T), (2, j_2^T, k_2^T), \ldots, (n, j_n^T, k_n^T)\}.$$

This path-relinking procedure is summarized in the pseudo-code shown in Figure 3.

Let the symmetric difference between S and T be defined by the following two sets of indices:

$$\delta J = \{i = 1, \ldots, n \mid j_i^S \neq j_i^T\}$$

and

$$\delta K = \{i = 1, \ldots, n \mid k_i^S \neq k_i^T\}.$$

These sets are computed in lines 4 and 5 of the pseudo-code.

An intermediate solution of the path is visited at each step of the loop in lines 6 to 34. Two elementary types of moves can be carried out. In a type-one move, triplets

$$\{(i_1, j_1, k_1), (i_2, j_2, k_2)\}$$

are replaced by triplets

$$\{(i_1, j_2, k_1), (i_2, j_1, k_2)\},$$

while in a type-two move, triplets

$$\{(i_1, j_1, k_1), (i_2, j_2, k_2)\}$$

```
procedure PATHRL(n, c, c_S, S, T)
1    Let S be {(1, j_1^S, k_1^S), (2, j_2^S, k_2^S), ..., (n, j_n^S, k_n^S)};
2    Let T be {(1, j_1^T, k_1^T), (2, j_2^T, k_2^T), ..., (n, j_n^T, k_n^T)});
3    c_gmin = c_S; S_gmin = S;
4    δJ = {i = 1, ..., n | j_i^S ≠ j_i^T};
5    δK = {i = 1, ..., n | k_i^S ≠ k_i^T};
6    for |δJ| + |δK| > 2 do
7        c_min = ∞;
8        for i ∈ δJ do
9            Let q be such that j_q^T == j_i^S;
10           S̄ = S \ {(i, j_i^S, k_i^S)} \ {(q, j_q^S, k_q^S)};
11           S̄ = S̄ ∪ {(i, j_q^S, k_i^S)} ∪ {(q, j_i^S, k_q^S)};
12           c̄ = c_S − c_{i,j_i^S,k_i^S} − c_{q,j_q^S,k_q^S} + c_{i,j_q^S,k_i^S} + c_{q,j_i^S,k_q^S};
13           if c̄ < c_min then
14               c_min = c̄; S_min = S̄; i_min = i;
15               flag = 0;
16           fi;
17       rof;
18       for i ∈ δK do
19           Let q be such that k_q^T == k_i^S;
20           S̄ = S \ {(i, j_i^S, k_i^S)} \ {(q, j_q^S, k_q^S)};
21           S̄ = S̄ ∪ {(i, j_i^S, k_q^S)} ∪ {(q, j_q^S, k_i^S)};
22           c̄ = c_S − c_{i,j_i^S,k_i^S} − c_{q,j_q^S,k_q^S} + c_{i,j_i^S,k_q^S} + c_{q,j_q^S,k_i^S};
23           if c̄ < c_min then
24               c_min = c̄; S_min = S̄; i_min = i;
25               flag = 1;
26           fi;
27       rof;
28       S = S_min; c_S = c_min;
29       if flag == 0 then δJ = δJ \ {i_min};
30       if flag == 1 then δK = δK \ {i_min};
31       if c_S < c_gmin then
32           c_gmin = c_S; S_gmin = S;
33       fi;
34   rof;
35   return (S_gmin);
end PATHRL
```

Figure 3 **Path Relinking Between Initial Solution S and Guiding Solution T**

are replaced by

$$\{(i_1, j_1, k_2), (i_2, j_2, k_1)\}.$$

We use the sets δJ and δK to guide the moves. δJ guides type-one moves, while δK guides type-two moves.

The loop going from line 8 to line 17 considers type-one moves. For all $i \in \delta J$, let q be such that $j_q^T = j_i^S$. The type-one move replaces triplets

$$\{(i, j_i^S, k_i^S), (q, j_q^S, k_q^S)\}$$

by

$$\{(i, j_q^S, k_i^S), (q, j_i^S, k_q^S)\}.$$

Likewise, the loop going from line 18 to line 27 considers type-two moves. For all $i \in \delta K$, let q be such that $k_q^T = k_i^S$. The type-two move replaces triplets

$$\{(i, j_i^S, k_i^S), (q, j_q^S, k_q^S)\}$$

by

$$\{(i, j_i^S, k_q^S), (q, j_q^S, k_i^S)\}.$$

At each step, the move that produces the least costly solution is selected and the corresponding index is deleted from either δJ or δK (line 29 or 30). This process continues until there are only two move indices left in one of the sets δJ or δK. At this stage, any of these two moves results in the *guiding solution* and, therefore, they are not carried out. The best solution found (S_{gmin}) in the path is returned by the procedure.

The hybrid approach proposed here is similar to the strategy proposed in Laguna and Martí (1999). Laguna and Martí maintain a pool of three elite solutions consisting of the three best-quality solutions so far produced. In our implementation, a pool P of elite solutions is formed with the solutions found in the first $|P|$ GRASP iterations.

After this initial phase, each solution s_g produced by the GRASP local search phase is relinked with one or more elite solutions. Laguna and Martí select one elite solution $s_e \in P$ and generate a path from s_g to s_e. In our implementation, given s_g and s_e, we always generate two paths, one from s_g to s_e, and another from s_e to s_g. This is done because these paths often visit different intermediate solutions. We implemented two strategies for selecting s_e. The first is the one proposed by Laguna and Martí, where s_e is selected at random from the pool. The second relinks s_g with all elite solutions in P.

Laguna and Martí update their pool by maintaining in it three best-quality solutions. We use an approach proposed by Fleurent and Glover (1999) for using elite solutions within the GRASP framework. The approach proposed by Fleurent and Glover, used to perform the pool update, is explained below.

Let c_{best} and c_{worst} be the objective function values of the best and the worst solutions in P, respectively. Given two solutions

$$S = \{(1, j_1^S, k_1^S), (2, j_2^S, k_2^S), \ldots, (n, j_n^S, k_n^S)\}$$

and

$$T = \{(1, j_1^T, k_1^T), (2, j_2^T, k_2^T), \ldots, (n, j_n^T, k_n^T)\}),$$

let

$$\Delta(S, T) = \sum_{i=1}^{n} J_i + \sum_{i=1}^{n} K_i,$$

where

$$J_i = \begin{cases} 1 & \text{if } j_i^S \neq j_i^T \\ 0 & \text{otherwise} \end{cases}$$

and

$$K_i = \begin{cases} 1 & \text{if } k_i^S \neq k_i^T \\ 0 & \text{otherwise,} \end{cases}$$

be a measure of dissimilarity of solutions S and T.

Solution S_{gmin} output from the path-relinking procedure is a candidate for insertion into the pool and is accepted if it satisfies one of the following acceptance criteria:

1. $c_{gmin} < c_{\text{best}}$, i.e., S_{gmin} is the best solution found so far;

2. $c_{\text{best}} < c_{gmin} < c_{\text{worst}}$ and for all elite solutions $S_p \in P$, $\Delta(S_{gmin}, S_p) > n$, i.e., S_{gmin} is better than the worst elite solution, and more than half of the elements of the permutation arrays in S_{gmin} differ (according to the measure of dissimilarity explained above) from the corresponding elements in the permutation arrays of each solution in P.

Once accepted for insertion into P, S_{gmin} replaces the worst elite solution, which is discarded from P.

Path relinking can also be used as an intensification phase for the elite set. This is accomplished by applying path relinking to each pair of elite solutions in the pool and updating the pool if necessary. The procedure is repeated until no further change in the pool occurs. This type of intensification can be done in a post-optimization phase (using the final pool of elite solutions), or periodically during the optimization (using the current set of elite solutions).

When applying path relinking as a post-optimization step, after no further change in the elite set occurs, the local search procedure of §2.3 is applied to each elite solution, as the solutions produced by path relinking are not always local optima. The local optima found are candidates for insertion into the elite set. If a change in the elite set occurs, the entire post-processing step is repeated.

4. GRASP with Path Relinking

In this section, we show how the procedures described above are combined in our implementation. Figure 4 presents pseudo-code for this algorithm.

The algorithm uses two stopping criteria. It halts either after `maxitr` iterations are done or if a solution with objective value less than or equal to `look4` is found. Each iteration consists of solution construction (line 5), local search using the constructed solution as the initial solution (line 6), and once the pool of elite solutions is full, a path-relinking phase. Path relinking examines two-way paths (lines 9 to 14) between the solution produced by the local search and a subset

```
procedure GRASP_PR(seed, n, c, look4, maxitr, maxpool, freq)
1    POOL = ∅;
2    for i = 1, ..., maxitr do
3        seed = rand(seed);
4        seedc = seed;
5        CONSTRUCT(seedc, n, c, S);
6        LOCAL(n, c, c_S, S);
7        if |POOL| == maxpool then
8            select SUBPOOL ⊆ POOL;
9            for T ∈ SUBPOOL do
10               S_gmin = PATHRL(n, c, c_S, S, T);
11               UPDATE_POOL(S_gmin, c_gmin, POOL);
12               S_gmin = PATHRL(n, c, c_T, T, S);
13               UPDATE_POOL(S_gmin, c_gmin, POOL);
14           rof;
15       else POOL = POOL ∪ {S} fi;
16       if mod(i, freq) == 0 then INTENSIFY(POOL) fi;
17       S_best = argmin{POOL};
18       if c_best ≤ look4 then break fi;
19   rof;
20   POSTOPT(POOL);
21   S_best = argmin{POOL};
22   return (S_best);
end GRASP_PR;
```

Figure 4 Pseudo-Code for GRASP with Path Relinking

of elite solutions (in `SUBPOOL`). In our implementation, `SUBPOOL` can consist of a single solution, selected at random from the pool, or the entire pool. After each path relinking is done, the best path solution is tested for insertion into the pool (lines 11 and 13). If the pool is not full, the local search solution is simply added to the pool (line 15).

The intensification scheme for the elite set, described in §3, is done every `freq` iterations (line 16). Finally, when one of the stopping criteria is satisfied, the post-optimization path relinking with local search phase, described in §3, is computed (line 20).

5. A Parallel Approach for GRASP with Path Relinking

As running time for a sequential implementation of the algorithm increases super-quadratically with problem dimension, it is natural to consider a parallel implementation to speed up the computations. Figure 5 shows, in log-log scale, CPU times for 10,000 iterations for GRASP with path relinking for instances of increasing dimension. The regression model $T = 10^{-2.56} n^{2.71}$, where T is CPU time in seconds (on a SGI Challenge computer with 196 MHz MIPS R10000 processors) and n is the dimension of the instance, is also shown in the figure as a straight line. The value of the coefficient of multiple determination (R^2) is 0.968.

We next present a basic parallelization scheme for GRASP with path relinking. Figure 6 shows pseudo-code for this scheme, which according to the

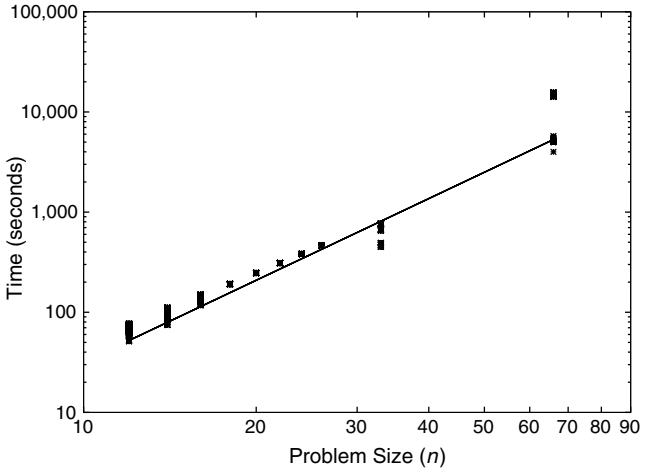

Figure 5 Problem Dimension Against CPU Time for 10,000 Iterations of GRASP with Path Relinking

```
procedure PARALLEL_GRASP_PR(n, c, seed, look4, maxitr, maxpool, freq)
1    my_rank = GetRank(); nprocs = GetNumProcs();
2    for i = 1, ..., maxitr * my_rank do
3        seed = rand(seed);
4    rof;
5    POOL = ∅; num_stop = 0;
6    for i = 1, ..., ∞ do
7        seed = rand(seed);
8        seedc = seed;
9        CONSTRUCT(seedc, n, c, S);
10       LOCAL(n, c, c_S, S);
11       if |POOL| == maxpool then
12           select SUBPOOL ⊆ POOL;
13           for T ∈ SUBPOOL do
14               S_gmin = PATHRL(n, c, c_S, S, T);
15               UPDATE_POOL(S_gmin, c_gmin, POOL);
16               S_gmin = PATHRL(n, c, c_T, T, S);
17               UPDATE_POOL(S_gmin, c_gmin, POOL);
18           rof;
19       else POOL = POOL ∪ {S} fi;
20       if mod(i, freq) = 0 then INTENSIFY(POOL) fi;
21       S_best = argmin{POOL};
22       if c_best ≤ look4 then SendAll(look4_stop) fi;
23       if i == maxitr then
24           num_stop = num_stop + 1;
25           SendAll(maxitr_stop)
26       fi;
27       received = VerifyReceiving(flag);
28       if received then
29           if flag == look4_stop then break;
30           else if flag == maxitr_stop then
31               num_stop = num_stop + 1;
32       fi;
33       if num_stop == nprocs then break fi;
34   rof;
35   POSTOPT(POOL);
36   S_GlobalBest = GetGlobalBest(S_best);
37   return (S_GlobalBest);
end PARALLEL_GRASP_PR;
```

Figure 6 Pseudo-Code for Parallel GRASP with Path Relinking

taxonomy proposed by Verhoeven and Aarts (1995) is a *multiple independent walks* parallelization.

Our implementation uses message passing for communication between processors. This communication is limited to program initialization and termination. A single processor reads the problem data and passes it to the remaining nproc -1 processes. Processes send a message to all others when they either stop upon finding a solution at least as good as the target or complete the maximum number of allotted iterations.

Each processor executes a copy of the program. In line 1 the processor's rank and the number of processors are determined. In the beginning of each GRASP construction phase, the random-number-generator is reinitialized with a different seed. To increase the likelihood of independence of processors, identical seeds of the random-number-generator (rand()) must not be used by more than one processor. The initial seed for processor my_rank is computed in lines 2–4. This way, each processor has a sequence of maxitr initial seeds. Note that using different seeds does not completely guarantee independence of the processors. The problem stems from the fact that the sequences of random numbers used by different processors may "overlap," unless the seeds are sufficiently apart. Obviously, the probability of this occurring with a generator with a very long sequence is not big, and should have a minimal impact on the results. The loop from line 6 to line 34 executes the iterations. The construction phase seed (seedc) is computed according to lines 7–8. If a solution with cost at least as good as the target (look4) is found by a process, a message is sent to all other processes, indicating this occurrence (line 22). Likewise, if a process reaches its maximum number of iterations, it sends a message to all other processes indicating this (lines 23–26) and increments its counter of number of terminated processes (num_stop) in line 24. In line 27, the process verifies if a message has been sent to it and, if so, takes appropriate action in lines 28–33. If the message indicated stopping by solution value, the iterations are terminated in line 29. If the number of maximum iterations has been reached by some other process, the counter of number of terminated processes is incremented in line 31, and if all processes have been terminated, the iterations are stopped in line 33.

Each process, upon completing its iterations, runs the post-optimization phase of pool of elite solutions in line 35. A reduce operator determines the best global solution among all processes in line 36 and returns this solution.

6. Computational Results

In this section, we present computational results using sequential and parallel implementations of the algorithms described in this paper. We describe the computer environment used to conduct the experiments,

the instances selected for each of the seven algorithms, and present results comparing the variants as well as results showing that this procedure can produce near-optimal solutions on instances of the AP3.

6.1. Computer Environment

The experiments were done on an SGI Challenge computer (28 196-MHz MIPS R10000 processors) with 7.6 Gb of memory. Each run of the sequential implementations used a single processor. The parallel implementations were run on 1, 2, 4, 8, and 16 processors. Load on the machine was not uniform throughout the experiments and may have affected processor availability.

The algorithms were coded in Fortran and were compiled with the SGI MIPSpro F77 compiler using flags `-O3 -r4 -64`. The Message-Passing Interface (MPI) specification has become a common standard for message-passing libraries for parallel computations (Snir et al. 1998). The parallel codes used SGI's Message Passing Toolkit 1.4, which contains a fully compliant implementation of the MPI 1.2 specification. CPU times for the sequential implementation were measured with the system function `etime`. In the parallel implementation, times measured are wall-clock time and were done with the MPI function `MPI_WT`. This is also the case for runs with a single processor that are compared to parallel 2, 4, 8, and 16 processor runs. Timing in the parallel runs excludes the time to read the problem data, initialize the random-number-generator seeds, and to output the solution. The pseudo-random-number-generator proposed in Schrage (1979) was used to produce the sequences of random numbers used.

6.2. Test Problems

Three classes of test problems taken from the literature were used in the experiment.

The first class of test problems was described by Balas and Saltzman (1991). Integer cost coefficients $c_{i,j,k}$ for those problems are uniformly generated in the interval $[0, 100]$. We limited our experiments to all instances generated by Balas and Saltzman of size $n = 12, 14, 16, 18, 20, 22, 24,$ and 26. For each size, we considered all of the five instances provided by Balas and Saltzman.

The second class of problems is the class $T\Delta$, described by Crama and Spieksma (1992). The costs in these problems are generated as follows. The AP3 is viewed as the optimization problem on a complete tripartite graph $K_{n,n,n}$ referred to in the introduction of this paper. For class $T\Delta$, a length $d_{u,v} \geq 0$ is assigned to each edge of $K_{n,n,n}$ and the cost $c_{i,j,k}$ of a triangle $(i, j, k) \in I \times J \times K$ is its total length, i.e., $c_{i,j,k} = d_{i,j} + d_{i,k} + d_{j,k}$. Three types of randomly generated problems are considered. They differ in how the lengths $d_{u,v}$ are computed (see Crama and Spieksma 1992 for details). Each type consists of three instances of size $n = 33$ and three instances of size $n = 66$, totaling 18 instances.

The final class of problems is described in Burkard et al. (1996). Problems in this class have decomposable cost coefficients. Let α_i, β_j, and γ_k be the elements of three n-element sequences. The cost coefficient $c_{i,j,k} = \alpha_i \cdot \beta_j \cdot \gamma_k$. The instances considered in our experiment are the largest ones tested by Burkard et al. (1996) (of sizes $n = 12, 14, 16$). Each has integer cost coefficients α_i, β_j, and γ_k uniformly distributed in the interval $[0, 10]$. For each problem size, all of the 100 test instances provided by Burkard et al. were considered.

6.3. Algorithm Variants

We considered seven variants of the GRASP and path-relinking schemes proposed in this paper.

1. `GRASP`: This variant is a pure GRASP with no path relinking.
2. `GPR(RAND)`: This variant adds to `GRASP` a two-way path relinking between the initiating solution and a randomly selected target solution from the elite set.
3. `GPR(ALL)`: This variant adds to `GRASP` a two-way path relinking between the GRASP solution and all `|POOL|` solutions in the elite set `POOL`.
4. `GPR(RAND,POST)`: This variant adds to `GPR(RAND)` a post-optimization phase where two-way path relinking is done between all elite set solutions and the resulting solutions are locally optimized. The post-optimization procedure is reapplied until no further change in the elite set is observed.
5. `GPR(ALL,POST)`: This variant adds to `GPR(ALL)` a post-optimization phase where two-way path relinking is done between all elite set solutions and the resulting solutions are locally optimized. The post-optimization procedure is reapplied until no further change in the elite set is observed.
6. `GPR(RAND,POST,INT)`: This variant adds to `GPR(RAND,POST)` an intensification scheme that is repeated at fixed iteration intervals. In the intensification scheme, two-way path relinking is done between all elite set solutions. The procedure is reapplied until no further change in the elite set is observed.
7. `GPR(ALL,POST,INT)`: This variant adds to `GPR(ALL,POST)` an intensification scheme that is repeated at fixed iteration intervals. In the intensification scheme, two-way path relinking is done between all elite set solutions. The procedure is reapplied until no further change in the elite set is observed.

6.4. The Experiments

Our objective with the experimental part of this paper is to evaluate the effectiveness of path relinking when used in conjunction with GRASP. We aim to answer

three broad questions:

1. Does path relinking improve the performance of GRASP, and what is the trade-off in terms of CPU time?

2. What are the trade-offs between CPU times and solution quality using the different variants of GRASP with path relinking described in this paper?

3. Are the random variables *time to target solution* for the different variants of GRASP with path relinking exponentially distributed, and if so, how does a straightforward parallel implementation perform?

To study the above questions, we considered the test problems described in §6.2 and used the variants of GRASP with path relinking listed in §6.3.

To study the effect of path relinking on GRASP, we compared the pure GRASP variant (GRASP) and the simplest GRASP with path-relinking variant (GPR(RAND)) on problems 20.1, 22.1, 24.1, and 26.1 of Balas and Saltzman (1991). The two variants were run for $r = 200$ times for each of the four problems. Execution was terminated when a solution of value at most equal to look4 was found. look4 values of 19, 20, 17, and 19 were used for problems 20.1, 22.1, 24.1, and 26.1, respectively. These values are far from optimal and can usually be found in few iterations. Empirical probability distributions for time to target solution are plotted in Figures 7 and 8. To plot the empirical distribution, we associate with the ith sorted running time (t_i) a probability

$$p_i = \left(i - \tfrac{1}{2}\right)/r,$$

and plot the points $z_i = (t_i, p_i)$, for $i = 1, \ldots, r$, where $r = 200$. We comment on the choice of p_i later on §6.5. The plots clearly show that path relinking reduces the time needed to find a sub-optimal target solution. For example, on problem 20.1 the probability of finding a solution at least as good as the target solution in at most 50 seconds is 89% for GPR(RAND), while for GRASP it is 44%, and in at most 100 seconds, it is 98% for GPR(RAND) and 66% for GRASP. Table 1 shows these probabilities for this and the other three instances. These results show that even though more computational effort is needed per iteration of GPR(RAND), this is compensated for by the reduced number of iterations needed to produce the solution.

Similar to the plots comparing GRASP and GPR(RAND) in Figures 7 and 8, Figures 9 and 10 show empirical probability distributions for time to target solution for variants GPR(RAND), GPR(RAND,INT), GPR(ALL), and GPR(ALL,INT) on the same problems 20.1, 22.1, 24.1, and 26.1 using the harder-to-find look4 target values 7, 8, 7, and 8, respectively. For example, Figure 9 shows that for problem 20.1, the probability that GPR(RAND) finds a solution of value at most 7 in less than 2063s is 0.5, while with the same probability GPR(RAND,INT), GPR(ALL), and GPR(ALL,INT) find a solution of value at most 7 in less than 1744s, 851s, and 718s, respectively. Table 2 shows times as a function of probability for the four test problems and four variants. The plots and the table show that GPR(ALL) and GPR(ALL,INT) outperform GPR(RAND) and GPR(RAND,INT), finding the target solutions in less time despite the fact that the time per iteration of both GPR(RAND) and GPR(RAND,INT) is significantly less than for GPR(ALL) and GPR(ALL,INT). Note also that the data show that intensification appears to benefit more GPR(RAND,INT) with respect to GPR(RAND) than GPR(ALL,INT) with respect to GPR(ALL), and that as the problem size increases, the benefit of intensification is diminished.

In another type of experiment, the different variants were run by a fixed number of iterations on the entire set of test problems. Two types of runs were done. The first with 100 iterations (quick) and the second

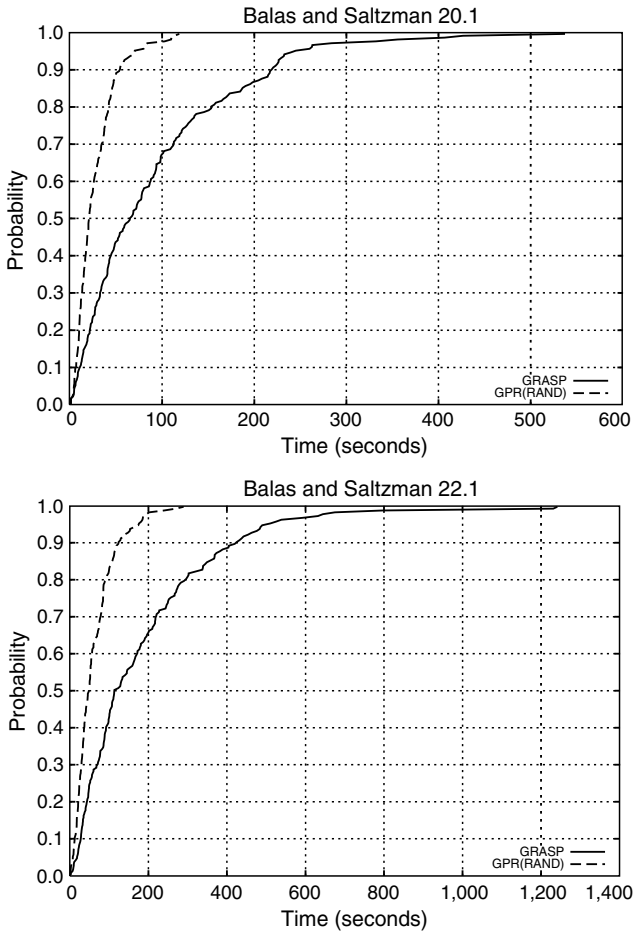

Figure 7 Empirical Probability Distributions of Time to Target Value for GRASP and GPR(RAND) (Balas and Saltzman Test Problems 20.1 and 22.1)

Note. Stopping criteria: look4 = 19 and look4 = 20 for test problems 20.1 and 22.1, respectively.

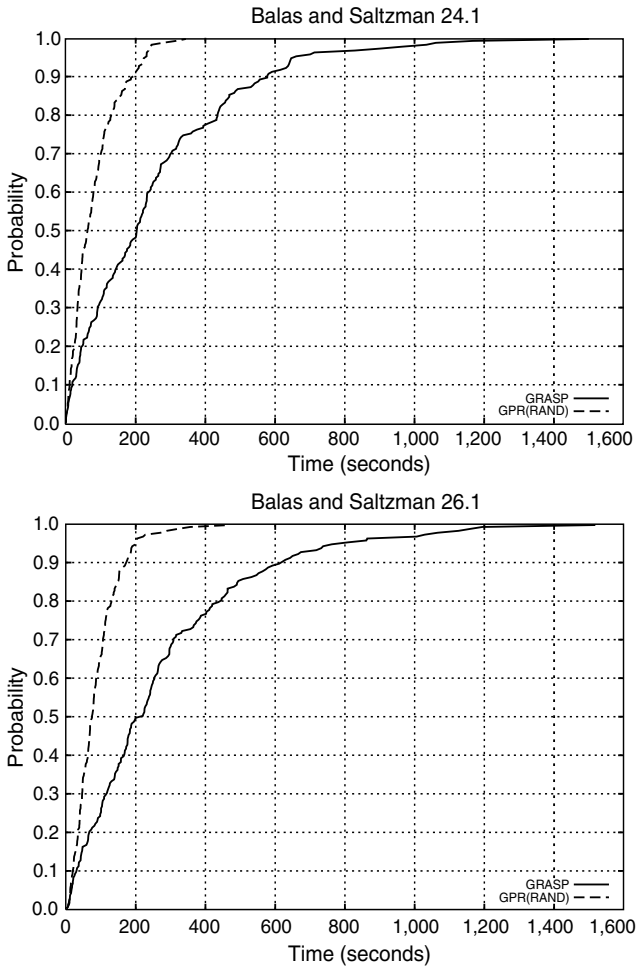

Figure 8 Empirical Probability Distributions of Time to Target Value for GRASP and GPR(RAND) (Balas and Saltzman Test Problems 24.1 and 26.1)

Note. Stopping criteria: look4 = 17 and look4 = 19 for test problems 24.1 and 26.1, respectively.

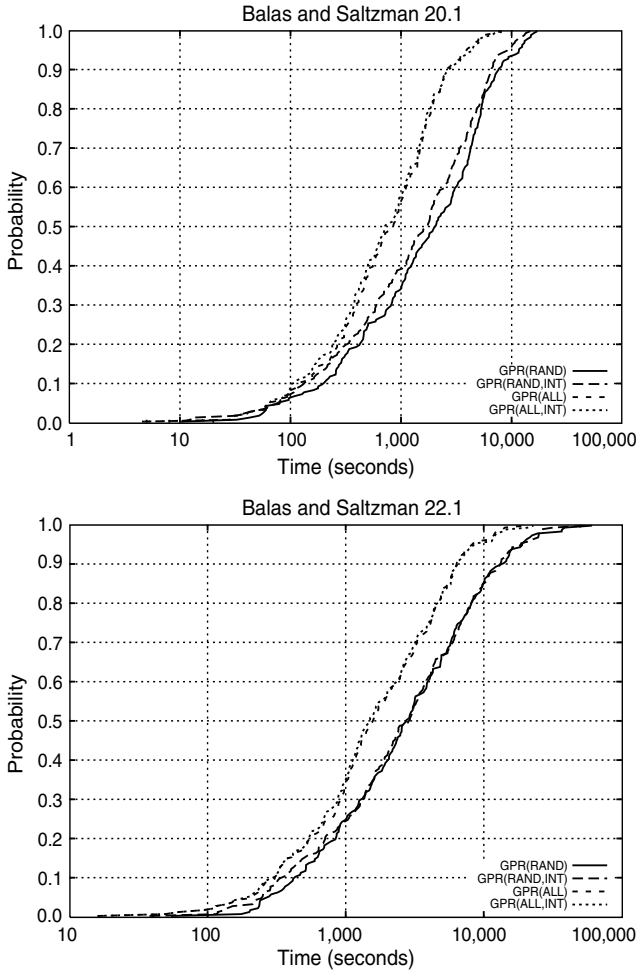

Figure 9 Empirical Probability Distributions of Time to Target Value for Different Variants of GRASP with Path Relinking (Balas and Saltzman Test Problems 20.1 and 22.1)

Note. Stopping criteria: look4 = 7 and look4 = 8 for test problems 20.1 and 22.1, respectively.

with 10,000 iterations (long). Five quick and five long independent runs were done for each variant-instance pair.

Tables 3 and 4 show results for the Balas and Saltzman (1991) test problems with 100 and 10,000 iterations, respectively. Each row shows statistics taken over five instances with five independent runs each, i.e., a total of 25 runs. For example, for $n = 12$, the instances are 12.1, 12.2, 12.3, 12.4, and 12.5. Column entry n is the dimension of the problem. Column entry B-S is the average solution found using the *variable-depth interchange* heuristic of Balas and Saltzman (1991) and column OPT lists the average optimal solution reported by Balas and Saltzman. In these tables, as well as the ones that follow, the GRASP with path-relinking variants GRASP,

Table 1 Probability Estimates of Finding a Solution at Least as Good as the Target Solution as a Function of Maximum Solution Time

	20.1		22.1		24.1		26.1	
Time	GPR(RAND)	GRASP	GPR(RAND)	GRASP	GPR(RAND)	GRASP	GPR(RAND)	GRASP
50s	0.89	0.44	0.52	0.25	0.45	0.22	0.35	0.16
100s	0.98	0.68	0.83	0.43	0.70	0.32	0.66	0.25
150s	1.00	0.80	0.93	0.56	0.84	0.42	0.85	0.37
200s	1.00	0.87	0.98	0.66	0.91	0.49	0.96	0.50

Note. Instances are Balas and Saltzman 20.1, 22.1, 24.1, and 26.1 with target values 19, 20, 17, and 19, respectively.

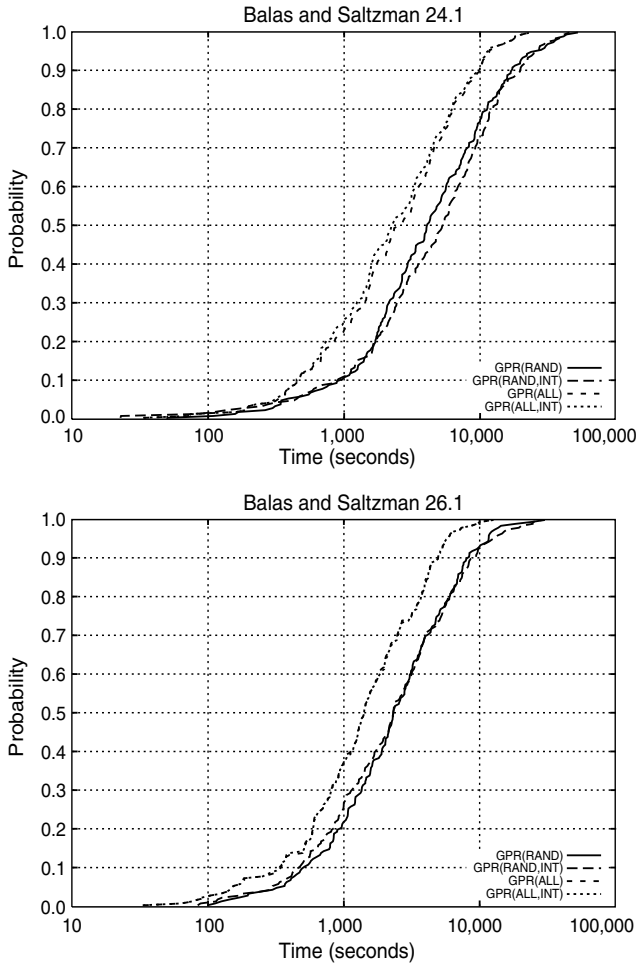

Figure 10 Empirical Probability Distributions of Time to Target Value for Different Variants of GRASP with Path Relinking (Balas and Saltzman Test Problems 24.1 and 26.1)

Note. Stopping criteria: look4 = 7 and look4 = 8 for test problems 24.1 and 26.1, respectively.

GPR(RAND), GPR(RAND,POST), GPR(RAND,POST,INT), GPR(ALL), GPR(ALL,POST), and GPR(ALL,POST,INT) are indicated by the column headings GRASP, GPR(R), GPR(R,P), GPR(R,P,I), GPR(A), GPR(A,P), and GPR(A,P,I), respectively. For each instance size, the highlighted cells in each row correspond to those variants with the smallest average solutions for instances of that size. For each algorithm variant and problem dimension, the corresponding cell in Tables 3 and 4 lists average cost (taken over the best of the five independent runs for each instance), the number of times the variant found the best solution (winner) over the five instances of that dimension (e.g., GPR(ALL,POST) found the best solution for two of the five instances of dimension 12, while GPR(ALL,POST,INT) found the best solution for all of the five instances of dimension 12), and the average CPU time of the 25 runs of that cell. The last row of these tables adds up the winners for each variant.

Table 2 Time to Find a Solution at Least as Good as the Target Solution as a Function of Probability

	Balas and Saltzman 20.1			
Probability	GPR(ALL,INT)	GPR(ALL)	GPR(RAND,INT)	GPR(RAND)
0.2	238s	266s	338s	428s
0.5	718s	851s	1,744s	2,063s
0.8	1,856s	1,918s	4,887s	5,331s
	Balas and Saltzman 22.1			
Probability	GPR(ALL,INT)	GPR(ALL)	GPR(RAND,INT)	GPR(RAND)
0.2	544s	546s	710s	852s
0.5	1,502s	1,584s	2,916s	2,763s
0.8	4,731s	4,780s	8,244s	8,161s
	Balas and Saltzman 24.1			
Probability	GPR(ALL,INT)	GPR(ALL)	GPR(RAND,INT)	GPR(RAND)
0.2	742s	856s	1,714s	1,697s
0.5	2,274s	2,603s	5,378s	4,084s
0.8	6,198s	6,306s	12,467s	11,336s
	Balas and Saltzman 26.1			
Probability	GPR(ALL,INT)	GPR(ALL)	GPR(RAND,INT)	GPR(RAND)
0.2	594s	592s	852s	963s
0.5	1,411s	1,411s	2,315s	2,315s
0.8	3,698s	3,691s	5,941s	6,049s

Note. Instances are Balas and Saltzman 20.1, 22.1, 24.1, and 26.1 with target values 7, 8, 7, and 8, respectively.

Tables 3 and 4 show the following.

- On short runs, GRASP with path relinking finds better solutions than does the variable-depth interchange heuristic of Balas and Saltzman for $n \leq 20$, but worse solutions for $n \geq 22$. On long runs, however, all variants (including pure GRASP) find better solutions than the variable-depth interchange heuristic.
- Optimal solutions are found on long runs for $n \leq 14$.
- On both short and long runs, all GRASP with path-relinking variants find solutions that are, on average, better than pure GRASP.
- On both short and long runs, post-optimization and fixed-interval intensification helps both random path relinking and full elite set path relinking.
- On long runs, solutions improve with algorithm sophistication, i.e., as one moves from the left side to the right side of the table.
- On short runs, random path relinking with either post-optimization or post-optimization and fixed-interval intensification finds better solutions than does full elite set path relinking without intensification. However, full elite set path relinking with post-optimization finds better solutions than does random path relinking with post-optimization. Also, full elite set path relinking with post-optimization

Table 3 Balas and Saltzman Test Problems (100 Iterations)

			GRASP		GPR(R)		GPR(R,P)		GPR(R,P,I)		GPR(A)		GPR(A,P)		GPR(A,P,I)	
n	B-S	OPT	Cost	Wins	Cost	Wins	Cost	Wins	Cost	Wins	Cost	Wins	Cost	Wins	Cost	Wins
12	24.0	15.6	31.4 (0.09*s*)	0	28.8 (0.11*s*)	0	20.0 (0.35*s*)	2	20.0 (0.86*s*)	3	21.2 (0.54*s*)	0	20.0 (0.71*s*)	2	**18.0 (1.24s)**	**5**
14	22.4	10.0	27.8 (0.16*s*)	0	25.4 (0.18*s*)	0	16.8 (0.54*s*)	1	16.2 (1.27*s*)	2	17.8 (0.78*s*)	0	16.4 (1.05*s*)	0	**15.0 (1.81s)**	**4**
16	25.0	10.0	25.0 (0.26*s*)	0	25.0 (0.29*s*)	0	18.8 (0.80*s*)	0	**18.4 (1.72s)**	**3**	21.2 (1.07*s*)	0	20.8 (1.44*s*)	1	**18.4 (2.45s)**	**4**
18	17.6	6.4	26.8 (0.40*s*)	0	26.8 (0.44*s*)	0	19.2 (1.04*s*)	3	18.6 (2.25*s*)	2	21.2 (1.43*s*)	0	**17.2 (1.88s)**	**2**	17.6 (3.19*s*)	3
20	27.4	4.8	27.0 (0.58*s*)	1	27.0 (0.63*s*)	1	**18.6 (1.33s)**	**3**	20.8 (2.80*s*)	2	21.4 (1.87*s*)	2	21.0 (2.39*s*)	2	18.8 (3.97*s*)	2
22	18.8	4.0	24.6 (0.80*s*)	1	24.6 (0.86*s*)	1	22.4 (1.63*s*)	2	**20.8 (3.46s)**	**4**	23.0 (2.37*s*)	3	23.0 (2.99*s*)	3	22.6 (4.85*s*)	4
24	14.0	1.8	31.2 (1.09*s*)	0	31.2 (1.16*s*)	0	25.2 (2.10*s*)	0	20.0 (4.20*s*)	2	23.4 (2.96*s*)	1	20.2 (3.64*s*)	2	**16.8 (5.83s)**	**2**
26	15.7	1.3	28.0 (1.44*s*)	1	28.0 (1.52*s*)	1	**21.2 (2.69s)**	**2**	22.4 (4.79*s*)	2	23.4 (3.64*s*)	2	22.4 (4.40*s*)	2	21.8 (6.71*s*)	3
Total winners				3		3		13		20		8		14		27

and fixed-interval intensification finds better solutions than random path relinking with post-optimization and fixed-interval intensification.

- CPU times increase as post-processing and intensification are added to GRASP with path relinking.

Tables 5, 6, and 7 show results for the Crama and Spieksma type I, II, and III test problems (Crama and Spieksma 1992), respectively, with 100 and 10,000 iterations. Each table shows number of iterations, problem dimension n, the value obtained by H, the best heuristic in Crama and Spieksma, and the lower bound reported by Crama and Spieksma. We refer to the Crama and Spieksma heuristic as C-S. Each cell corresponding to an instance-variant pair shows the value of the best solution found out of the five independent runs, the number times the best solution

Table 4 Balas and Saltzman Test Problems (10,000 Iterations)

			GRASP		GPR(R)		GPR(R,P)		GPR(R,P,I)		GPR(A)		GPR(A,P)		GPR(A,P,I)	
n	B-S	OPT	Cost	Wins	Cost	Wins	Cost	Wins	Cost	Wins	Cost	Wins	Cost	Wins	Cost	Wins
12	24.0	15.6	16.4 (8.37*s*)	3	**15.6 (10.77s)**	**5**	**15.6 (10.85s)**	**5**	**15.6 (14.32s)**	**5**	**15.6 (71.63s)**	**5**	**15.6 (71.70s)**	**5**	**15.6 (74.79s)**	**5**
14	22.4	10.0	12.0 (14.37*s*)	2	10.6 (17.71*s*)	3	10.2 (17.83*s*)	4	10.2 (22.67*s*)	4	**10.0 (102.22s)**	**5**	**10.0 (102.30s)**	**5**	**10.0 (106.55s)**	**5**
16	25.0	10.0	13.2 (23.51*s*)	0	12.8 (27.64*s*)	1	12.8 (27.85*s*)	1	11.0 (34.33*s*)	2	10.6 (138.31*s*)	2	10.6 (138.37*s*)	2	**10.2 (143.89s)**	**4**
18	17.6	6.4	11.8 (36.58*s*)	0	9.8 (42.04*s*)	0	8.0 (42.20*s*)	2	9.8 (50.39*s*)	0	**7.2 (184.86s)**	**3**	**7.2 (183.70s)**	**3**	7.4 (190.88*s*)	2
20	27.4	4.8	14.2 (53.40*s*)	0	9.8 (60.20*s*)	1	9.4 (60.53*s*)	1	10.2 (70.64*s*)	1	7.2 (237.02*s*)	3	6.6 (237.35*s*)	4	**6.4 (246.70s)**	**5**
22	18.8	4.0	13.6 (74.82*s*)	0	12.4 (83.18*s*)	0	10.6 (83.27*s*)	1	10.2 (95.63*s*)	1	**7.2 (298.17s)**	**4**	**7.2 (298.40s)**	**4**	7.8 (309.64*s*)	3
24	14.0	1.8	13.4 (101.66*s*)	0	9.8 (111.14*s*)	1	9.8 (111.98*s*)	1	8.4 (127.09*s*)	3	**7.4 (368.45s)**	**5**	**7.4 (368.82s)**	**5**	**7.4 (382.45s)**	**5**
26	15.7	1.3	13.4 (134.53*s*)	0	9.8 (145.60*s*)	1	9.8 (146.27*s*)	1	9.8 (164.15*s*)	2	**8.4 (448.54s)**	**5**	**8.4 (449.06s)**	**5**	**8.4 (465.20s)**	**5**
Total winners				5		12		16		18		32		33		34

Table 5 Crama and Spieksma Type I Test Problems

				GRASP		GPR(R)		GPR(R,P)		GPR(R,P,I)		GPR(A)		GPR(A,P)		GPR(A,P,I)	
Iterations	n	C-S	LB	Cost	Wins	Cost	Wins	Cost	Wins	Cost	Wins	Cost	Wins	Cost	Wins	Cost	Wins
100	33	1,618	1,607	1,617 (4.88s)	0	1,613 (4.94s)	0	**1,608 (5.45s)**	**1**	**1,608 (6.15s)**	**1**	**1,608 (6.14s)**	**1**	**1,608 (6.41s)**	**1**	**1,608 (7.35s)**	**1**
100	33	1,411	1,395	1,419 (5.08s)	0	1,405 (5.15s)	0	**1,401 (5.73s)**	**1**	**1,401 (6.42s)**	**1**	**1,401 (6.48s)**	**1**	**1,401 (6.76s)**	**1**	**1,401 (7.70s)**	**1**
100	33	1,609	1,604	1,616 (4.70s)	0	1,609 (4.79s)	0	**1,604 (5.32s)**	**1**	**1,604 (6.25s)**	**1**	**1,604 (6.22s)**	**1**	**1,604 (6.49s)**	**1**	**1,604 (7.54s)**	**1**
100	66	2,668	2,654	2,767 (151.34s)	0	2,753 (151.95s)	0	2,687 (154.97s)	0	2,682 (165.08s)	0	2,679 (158.54s)	0	**2,678 (159.69s)**	**1**	2,681 (171.20s)	0
100	66	2,469	2,433	2,515 (147.97s)	0	2,490 (149.14s)	0	2,461 (152.15s)	0	**2,452 (163.34s)**	**1**	2,454 (155.04s)	0	2,454 (156.22s)	0	**2,452 (166.80s)**	**1**
100	66	2,775	2,748	2,822 (149.07s)	0	2,788 (149.53s)	0	**2,766 (151.93s)**	**1**	2,778 (162.79s)	0	2,783 (154.77s)	0	2,779 (155.87s)	0	2,774 (171.40s)	0
Total winners					0		0		4		4		3		4		4
10,000	33	1,618	1,607	1,609 (459.35s)	0	**1,608 (467.50s)**	**1**	**1,608 (466.48s)**	**1**	**1,608 (477.70s)**	**1**	**1,608 (654.70s)**	**1**	**1,608 (652.26s)**	**1**	**1,608 (660.49s)**	**1**
10,000	33	1,411	1,395	**1,401 (474.90s)**	**1**	**1,401 (489.25s)**	**1**	**1,401 (485.71s)**	**1**	**1,401 (495.97s)**	**1**	**1,401 (672.43s)**	**1**	**1,401 (671.76s)**	**1**	**1,401 (680.50s)**	**1**
10,000	33	1,609	1,604	1,606 (443.60s)	0	**1,604 (460.63s)**	**1**	**1,604 (459.33s)**	**1**	**1,604 (465.86s)**	**1**	**1,604 (648.24s)**	**1**	**1,604 (673.43s)**	**1**	**1,604 (676.07s)**	**1**
10,000	66	2,668	2,654	2,714 (14,432.23s)	0	2,670 (14,470.68s)	0	2,670 (14,380.15s)	0	**2,664 (14,440.98s)**	**1**	**2,664 (15,352.03s)**	**1**	**2,664 (15,662.94s)**	**1**	**2,664 (15,470.11s)**	**1**
10,000	66	2,469	2,433	2,484 (14,157.01s)	0	2,454 (14,197.33s)	0	2,454 (14,220.41s)	0	2,454 (14,238.75s)	0	**2,449 (14,986.12s)**	**1**	**2,449 (15,018.97s)**	**1**	**2,449 (15,010.90s)**	**1**
10,000	66	2,775	2,748	2,801 (14,196.91s)	0	**2,758 (14,244.77s)**	**1**	**2,758 (14,265.57s)**	**1**	**2,758 (14,283.52s)**	**1**	2,759 (15,023.19s)	0	2,759 (15,011.29s)	0	2,759 (15,084.57s)	0
Total winners					1		4		4		5		5		5		5

was found, and the average running time over the five runs.

For type I problems, Table 5 shows the following.

• On short runs the GRASP with path-relinking variants found a better solution than did C-S on all but one instance, whereas on the long runs, better solutions were found for all instances.

• On short runs, pure GRASP improved upon the solution found by C-S only on a single instance of size $n = 33$, while on instances of size $n = 66$ C-S found better solutions than GRASP.

• On long runs, GRASP found better solutions for the instances of size $n = 33$, but worse solutions for the instances of size $n = 66$.

• On all short runs and all but one long run, GRASP with random or full elite set path relinking found solutions that are better than pure GRASP.

• On all short runs, post-processing improved GRASP with random path relinking. In two of six short runs, post-processing improved GRASP with full elite set path relinking. However, for long runs, post-processing did not make any difference.

• There is an insignificant difference between the different GRASP with path-relinking variants on the long runs.

• On one instance of size $n = 33$ more than one GRASP with path-relinking variant found an optimal solution.

• The relative error with respect to the lower bound was at most 0.9% for the short runs and 0.65% for the long runs.

For type-II problems, Table 6 shows the following.

• On all short and long runs, the pure GRASP found solutions that were better than those found with C-S.

• On four of six short runs and on all long runs, random path relinking improved pure GRASP. On all short and long runs, full elite set path relinking improved pure GRASP.

• On all short runs, post-processing improved GRASP with random path relinking. However,

Table 6 Crama and Spieksma Type II Test Problems

				GRASP		GPR(R)		GPR(R,P)		GPR(R,P,I)		GPR(A)		GPR(A,P)		GPR(A,P,I)	
Iterations	n	C-S	LB	Cost	Wins	Cost	Wins	Cost	Wins	Cost	Wins	Cost	Wins	Cost	Wins	Cost	Wins
100	33	4,861	4,772	4,805 (4.49*s*)	0	4,805 (4.59*s*)	0	4,799 (5.50*s*)	0	4,799 (7.26*s*)	0	**4,798 (6.89s)**	1	**4,798 (7.50s)**	1	**4,798 (9.50s)**	1
100	33	5,142	5,035	5,086 (4.25*s*)	0	5,081 (4.37*s*)	0	**5,070 (5.49s)**	1	5,072 (7.54*s*)	0	5,075 (6.83*s*)	0	5,071 (7.62*s*)	0	5,071 (9.97*s*)	0
100	33	4,352	4,260	4,306 (4.36*s*)	0	4,301 (4.42*s*)	0	**4,288 (5.54s)**	1	4,290 (7.58*s*)	0	4,290 (6.72*s*)	0	4,290 (7.25*s*)	0	4,289 (9.19*s*)	0
100	66	9,780	9,633	9,728 (133.99*s*)	0	9,728 (134.10*s*)	0	9,709 (141.95*s*)	0	9,707 (154.09*s*)	0	9,710 (145.42*s*)	0	9,709 (149.12*s*)	0	**9,703 (163.67s)**	1
100	66	9,142	8,831	8,990 (138.28*s*)	0	8,981 (139.04*s*)	0	**8,962 (144.25s)**	1	8,966 (155.43*s*)	0	8,966 (149.31*s*)	0	8,966 (150.97*s*)	0	8,964 (167.24*s*)	0
100	66	9,888	9,670	9,803 (133.23*s*)	0	9,791 (133.81*s*)	0	9,768 (139.25*s*)	0	9,770 (154.42*s*)	0	9,766 (145.58*s*)	0	**9,764 (157.98s)**	1	9,767 (161.25*s*)	0
Total winners					0		0		3		0		1		2		2
10,000	33	4,861	4,772	4,804 (419.43*s*)	0	**4,797 (433.18s)**	1	**4,797 (434.92s)**	1	**4,797 (446.75s)**	1	**4,797 (752.91s)**	1	**4,797 (754.34s)**	1	**4,797 (766.06s)**	1
10,000	33	5,142	5,035	5,076 (398.35*s*)	0	**5,067 (414.93s)**	1	**5,067 (413.58s)**	1	5,069 (429.32*s*)	0	5,068 (759.88*s*)	0	5,068 (761.77*s*)	0	5,068 (772.84*s*)	0
10,000	33	4,352	4,260	4,296 (405.03*s*)	0	**4,287 (418.35s)**	1	**4,287 (420.94s)**	1	4,288 (430.96*s*)	0	**4,287 (747.17s)**	1	**4,287 (748.03s)**	1	**4,287 (762.19s)**	1
10,000	66	9,780	9,633	9,720 (13,453.17*s*)	0	9,703 (13,449.01*s*)	0	9,703 (13,387.55*s*)	0	9,699 (13,545.63*s*)	0	**9,694 (14,676.68s)**	1	**9,694 (14,553.60s)**	1	**9,694 (14,629.08s)**	1
10,000	66	9,142	8,831	8,976 (13,238.00*s*)	0	8,957 (13,257.29*s*)	0	8,957 (13,335.41*s*)	0	8,956 (13,412.89*s*)	0	**8,951 (14,871.51s)**	1	**8,951 (14,706.02s)**	1	8,954 (14,922.91*s*)	0
10,000	66	9,888	9,670	9,784 (12,689.67*s*)	0	9,757 (12,739.95*s*)	0	9,756 (12,748.95*s*)	0	9,759 (12,784.51*s*)	0	9,753 (14,446.85*s*)	0	9,753 (14,326.70*s*)	0	**9,751 (14,391.67s)**	1
Total winners					0		3		3		1		4		4		4

post-processing only improved GRASP with random path relinking on a single long run.

• On half of the short runs, post-processing improves GRASP with full elite set path relinking, while on the long runs it has no influence.

• Fixed-interval intensification deteriorates solution quality more than it improves it.

• The relative error with respect to the lower bound was at most 1.5% for the short runs and 1.35% for the long runs.

For type III problems, Table 7 shows the following.

• On all short and long runs, GRASP with path relinking found solutions that were better than those found with C-S.

• On only one short run, pure GRASP found a better solution than did C-S. On the other five, C-S was better.

• On four of six long runs, pure GRASP improved upon C-S, while on the remaining two, both algorithms found solutions of the same quality.

• GRASP with random path relinking and GRASP with full elite set path relinking found better solutions than pure GRASP on all short and long runs.

• On all short runs and two long runs, post-optimization improved GRASP with random path relinking. On the remaining three long runs, both algorithms found solutions of the same quality.

• On two short runs, post-optimization improved GRASP with full elite set path relinking, while on the remaining three short runs and all long runs, both algorithms found solutions of the same quality.

• On short runs, fixed-interval intensification deteriorated the solution quality of GRASP with random path relinking or full elite set path relinking more than it improved it. On long runs, it had no influence with respect to solution quality.

• The relative error with respect to the lower bound was at most 1.8% for the short runs as well as for the long runs.

• Both short and long runs on one instance of size $n = 33$ produced an optimal solution.

Table 7 Crama and Spieksma Type III Test Problems

				GRASP		GPR(R)		GPR(R,P)		GPR(R,P,I)		GPR(A)		GPR(A,P)		GPR(A,P,I)	
Iterations	n	C-S	LB	Cost	Wins	Cost	Wins	Cost	Wins	Cost	Wins	Cost	Wins	Cost	Wins	Cost	Wins
100	33	135	133	137 (1.75*s*)	0	136 (1.87*s*)	0	**133 (2.69s)**	**1**	**133 (4.44s)**	**1**	**133 (4.13s)**	**1**	**133 (4.53s)**	**1**	**133 (6.56s)**	**1**
100	33	137	130	136 (1.73*s*)	0	133 (1.83*s*)	0	**131 (2.58s)**	**1**	132 (4.38*s*)	0	**131 (3.97s)**	**1**	**131 (4.50s)**	**1**	**131 (6.15s)**	**1**
100	33	135	130	136 (1.73*s*)	0	133 (1.84*s*)	0	132 (2.60*s*)	0	**131 (3.89s)**	**1**	**131 (3.82s)**	**1**	**131 (4.24s)**	**1**	**131 (5.84s)**	**1**
100	66	293	283	297 (42.73*s*)	0	291 (43.28*s*)	0	**286 (47.14s)**	**1**	**286 (55.57s)**	**1**	**286 (52.29s)**	**1**	**286 (53.18s)**	**1**	**286 (63.08s)**	**1**
100	66	294	281	295 (42.24*s*)	0	291 (42.24*s*)	0	**286 (45.63s)**	**1**	287 (55.09*s*)	0	287 (50.79*s*)	0	**286 (52.54s)**	**1**	287 (62.16*s*)	0
100	66	293	280	295 (41.77*s*)	0	288 (42.05*s*)	0	**282 (45.37s)**	**1**	283 (54.55*s*)	0	284 (50.73*s*)	0	283 (52.69*s*)	0	283 (61.55*s*)	0
Total winners					0		0		5		3		4		5		4
10,000	33	135	133	135 (149.44*s*)	0	**133 (162.57s)**	**1**	**133 (164.11s)**	**1**	**133 (179.94s)**	**1**	**133 (475.23s)**	**1**	**133 (476.97s)**	**1**	**133 (490.79s)**	**1**
10,000	33	137	130	134 (149.74*s*)	0	**131 (162.42s)**	**1**	**131 (163.33s)**	**1**	**131 (178.10s)**	**1**	**131 (459.16s)**	**1**	**131 (459.70s)**	**1**	**131 (471.21s)**	**1**
10,000	33	135	130	134 (148.27*s*)	0	132 (162.04*s*)	0	**131 (161.64s)**	**1**	**131 (175.95s)**	**1**	**131 (439.93s)**	**1**	**131 (439.06s)**	**1**	**131 (451.72s)**	**1**
10,000	66	293	283	293 (4,280.75*s*)	0	**286 (4,222.88s)**	**1**	**286 (4,082.55s)**	**1**	**286 (4,160.62s)**	**1**	**286 (5,303.95s)**	**1**	**286 (5,243.44s)**	**1**	**286 (5,322.97s)**	**1**
10,000	66	294	281	292 (3,955.23*s*)	0	**286 (3,951.28s)**	**1**	**286 (3,943.24s)**	**1**	**286 (3,967.02s)**	**1**	**286 (5,117.01s)**	**1**	**286 (4,980.35s)**	**1**	**286 (5,126.86s)**	**1**
10,000	66	293	280	292 (4,073.55*s*)	0	283 (3,861.92*s*)	0	**282 (3,865.62s)**	**1**	**282 (3,909.43s)**	**1**	**282 (5,008.44s)**	**1**	**282 (5,015.85s)**	**1**	**282 (5,059.06s)**	**1**
Total winners					0		4		6		6		6		6		6

Tables 8 and 9 show results for the Burkard et al. (1996) test problems, respectively, with 100 and 10,000 iterations. Each table shows the problem dimension n, the value obtained by the Burkard et al. heuristics Simple_LSH on instances of size $n = 12$ and LSH on instances of size $n \geq 14$. We refer to the Burkard et al. heuristics as B-R-W. Each of the 100 instances of each size was solved five times independently with the GRASP with path-relinking variants. Each cell corresponding to an instance-variant pair shows the average value computed over the 100 best solutions (one for each of five independent runs), the number of times the best solution was found, and the average running time over the five hundred runs.

Tables 8 and 9 show the following.

- On all short and long runs, the average solution found by pure GRASP was better than the average solution found by B-R-W.

Table 8 Burkard et al. Test Problems (100 Iterations)

		GRASP		GPR(R)		GPR(R,P)		GPR(R,P,I)		GPR(A)		GPR(A,P)		GPR(A,P,I)	
n	B-R-W	Cost	Wins	Cost	Wins	Cost	Wins	Cost	Wins	Cost	Wins	Cost	Wins	Cost	Wins
12	1,188.02	1,186.92 (0.12*s*)	92	1,186.85 (0.14*s*)	97	1,186.82 (0.31*s*)	99	**1,186.81 (0.73s)**	**100**	**1,186.81 (0.51s)**	**100**	**1,186.81 (0.62s)**	**100**	**1,186.81 (1.03s)**	**100**
14	1,469.19	1,468.18 (0.21*s*)	73	1,467.91 (0.23*s*)	87	1,467.76 (0.49*s*)	98	**1,467.74 (1.10s)**	**100**	**1,467.75 (0.75s)**	**98**	**1,467.75 (0.91s)**	**99**	**1,467.74 (1.52)**	**100**
16	1,476.80	1,475.93 (0.35*s*)	56	1,475.65 (0.38*s*)	64	1,475.17 (0.76*s*)	96	1,475.14 (1.61*s*)	99	1,475.15 (1.07*s*)	98	**1,475.13 (1.33s)**	**100**	**1,475.13 (2.21)**	**100**
Total winners			221		248		293		299		297		299		300

Table 9 Burkard et al. Test Problems (10,000 Iterations)

		GRASP		GPR(R)		GPR(R,P)		GPR(R,P,I)		GPR(A)		GPR(A,P)		GPR(A,P,I)	
n	B-R-W	Cost	Wins	Cost	Wins	Cost	Wins	Cost	Wins	Cost	Wins	Cost	Wins	Cost	Wins
12	1,188.02	**1,186.81 (11.19s)**	**100**	**1,186.81 (13.29s)**	**100**	**1,186.81 (13.35s)**	**100**	**1,186.81 (15.96s)**	**100**	**1,186.81 (65.90s)**	**100**	**1,186.81 (65.96s)**	**100**	**1,186.81 (68.30s)**	**100**
14	1,469.19	1,467.75 (19.09*s*)	99	**1,467.74 (21.92s)**	**100**	**1,467.74 (22.01s)**	**100**	**1,467.74 (25.67s)**	**100**	**1,467.74 (94.70s)**	**100**	**1,467.74 (94.79s)**	**100**	**1,467.74 (98.02s)**	**100**
16	1,476.80	1,475.18 (31.77*s*)	96	**1,475.13 (35.58s)**	**100**	**1,475.13 (35.74s)**	**100**	**1,475.13 (41.08s)**	**100**	**1,475.13 (134.62s)**	**100**	**1,475.13 (134.86s)**	**100**	**1,475.13 (139.30s)**	**100**
Total winners			295		300		300		300		300		300		300

- On short runs, for all problem sizes adding either random path relinking or full elite set path relinking to GRASP, reduced the average cost of the solutions. On long runs, this occurred for problems of size $n \geq 14$.
- On short runs, post-optimization improved GRASP with random and full elite set path relinking, while fixed-interval intensification improved post-optimized GRASP with random and full elite set path relinking. On long runs, all GRASP with path-relinking variants found the same solution.
- On short runs, GRASP with full elite set path relinking was better than GRASP with random path relinking. Also, GRASP with full elite set path relinking and post-optimization was better than GRASP with random path relinking and post-optimization. GRASP with full elite set path relinking, post-optimization, and fixed-interval intensification was better than GRASP with random path relinking, post-optimization, and fixed-interval intensification.
- On all but five long runs, pure GRASP found the best solution averages.

6.5. Experiments with a Parallel Implementation

Aiex et al. (2002) studied the empirical probability distributions of the random variable *time to target solution* in five GRASP implementations. They showed that, given a target solution value, the time it takes GRASP to find a solution at least as good as the target fits a two-parameter exponential distribution. Standard methodology for graphical analysis (Chambers et al. 1983) is used to compute the empirical and theoretical distributions and estimate the parameters of the distributions. We use the same methodology to study the time to target solution for four variants of GRASP with path relinking: `GPR(RAND)`, `GPR(RAND,INT)`, `GPR(ALL)`, and `GPR(ALL,INT)`. We consider four test problems: Balas and Saltzman 20.1, 22.1, 24.1, and 26.1, and `look4` target values 7, 8, 7, and 8, respectively. Our objective is to show that the four variants of GRASP with path relinking have time to target value distributions that fit a two-parameter exponential distribution.

Figures 11, 12, 13, and 14 show empirical and theoretical distributions of the random variable *time to target solution* as well as quantile-quantile plots for `GPR(RAND)`, `GPR(RAND,INT)`, `GPR(ALL)`, and `GPR(ALL,INT)`, respectively. Each figure shows four rows of plots, one for each of the four problems. Each row is made up of two plots. The plot on the left shows empirical and theoretical distributions computed with 200 runs of the GRASP with path-relinking variant. For each of the 200 runs of each combination, the random-number-generator is initialized with a distinct seed. Our description of each plot follows Aiex et al. (2002) closely. For each instance/variant pair, the running times are sorted in increasing order. To plot the empirical distribution, we associate with the ith sorted running time (t_i) a probability $p_i = (i - 1/2)/200$ and plot the points $z_i = (t_i, p_i)$, for $i = 1, \ldots, 200$.

The plot on the right is a quantile-quantile (or Q-Q) plot. To estimate the parameters of the two-parameter exponential distribution, we first draw the theoretical quantile-quantile plot for the data. To describe Q-Q plots, recall that the cumulative distribution function for the two-parameter exponential distribution is given by

$$F(t) = 1 - e^{-(t-\mu)/\lambda},$$

where λ is the mean of the distribution data (and indicates the spread of the data) and μ is the shift of the distribution with respect to the ordinate axis.

For each value p_i, $i = 1, \ldots, 200$, we associate a p_i-quantile $Qt(p_i)$ of the theoretical distribution. For each p_i-quantile we have, by definition, that

$$F(Qt(p_i)) = p_i.$$

Hence, $Qt(p_i) = F^{-1}(p_i)$ and therefore, for the two-parameter exponential distribution, we have

$$Qt(p_i) = -\lambda \ln(1 - p_i) + \mu.$$

The quantiles of the data of an empirical distribution are simply the (sorted) raw data. Note that if we

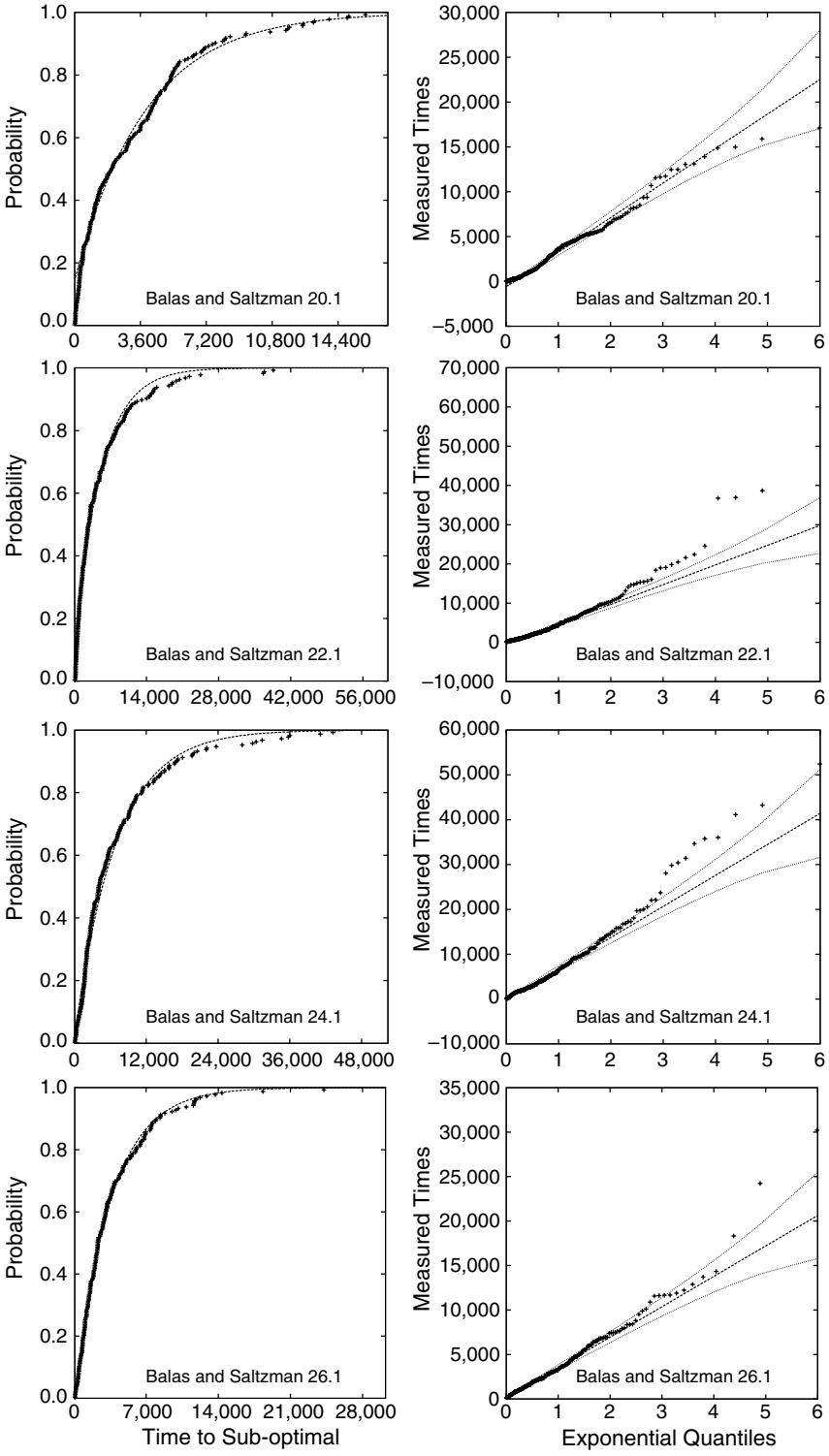

Figure 11 Empirical and Theoretical Time to Target Solution (`look4`) Distributions and Quantile-Quantile Plots for GPR(RAND)
Note. Balas and Saltzman problems 20.1, 22.1, 24.1, and 26.1, using `look4` = 7, 8, 7, and 8, respectively.

were to use $p_i = i/200$, for $i = 1, \ldots, 200$, then $Qt(p_{200})$ would be undefined.

A theoretical quantile-quantile plot (or theoretical Q-Q plot) is obtained by plotting the quantiles of the data of an empirical distribution against the quantiles of a theoretical distribution. This involves three steps. First, the data (in our case, the measured times) are sorted in ascending order. Second, the quantiles of

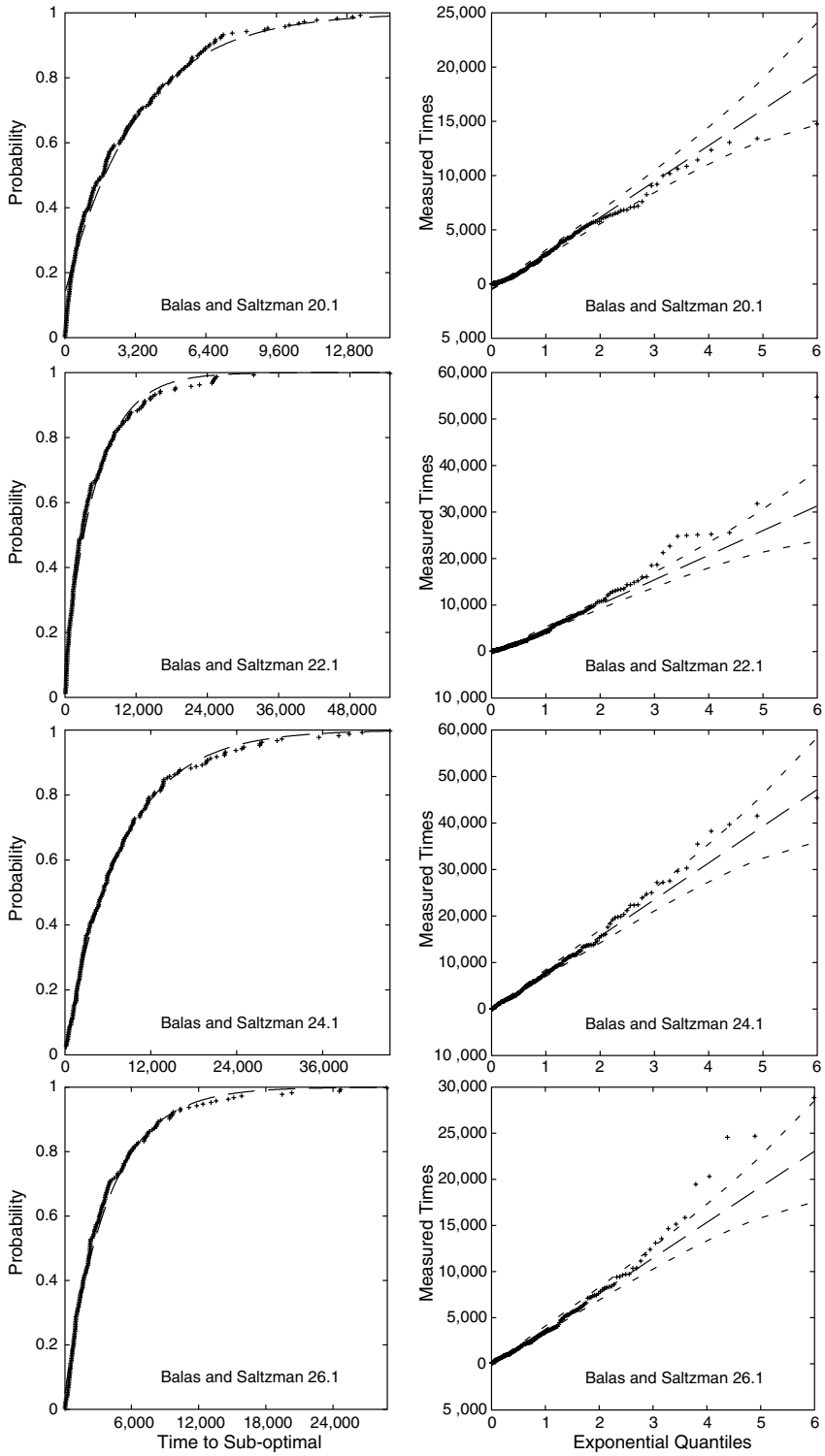

Figure 12 Empirical and Theoretical Time to Target Solution (`look4`) **Distributions and Quantile-Quantile Plots for GPR(RAND,INT)**
Note. Balas and Saltzman problems 20.1, 22.1, 24.1, and 26.1, using `look4` = 7, 8, 7, and 8, respectively.

the theoretical exponential distribution are obtained. Finally, a plot of the data against the theoretical quantiles is made.

In a situation where the theoretical distribution is a close approximation of the empirical distribution, the points in the Q-Q plot will have a nearly straight configuration. If the parameters λ and μ of the theoretical distribution that best fits the measured data could be estimated *a priori*, the points in a Q-Q plot would tend to follow the line $x = y$. Alternatively, in a plot of the

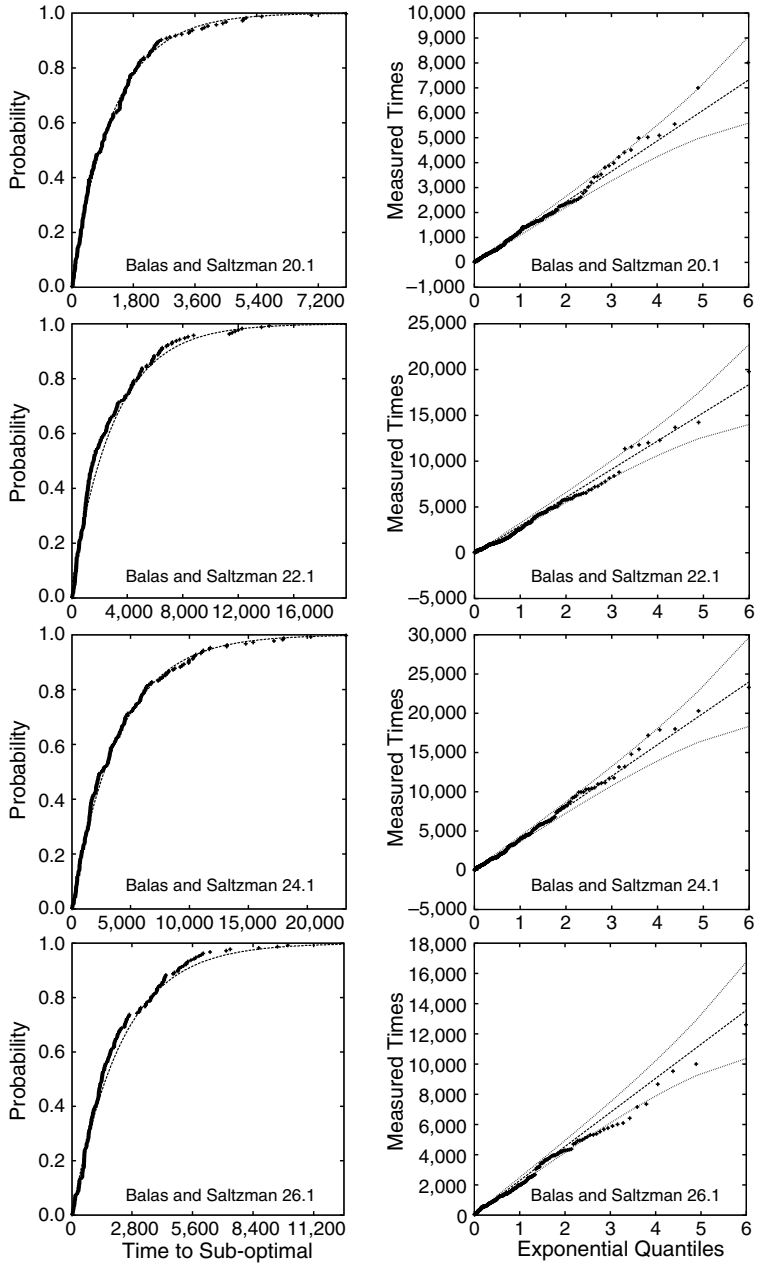

Figure 13 Empirical and Theoretical Time to Target Solution (look4) **Distributions and Quantile-Quantile Plots for GPR(ALL)**
Note. Balas and Saltzman problems 20.1, 22.1, 24.1, and 26.1, using look4 = 7, 8, 7, and 8, respectively.

data against a two-parameter exponential distribution with $\lambda = 1$ and $\mu = 0$, the points would tend to follow the line $y = \lambda x + \mu$. Consequently, parameters λ and μ of the two-parameter exponential distribution can be estimated, respectively, by the slope and intercept of the line depicted in the Q-Q plot.

To avoid possible distortions caused by outliers, we do not estimate the distribution mean with the data mean or by linear regression on the points of the Q-Q plot. Instead, we estimate the slope $\hat{\lambda}$ of line $y = \lambda x + \mu$ using the upper quartile q_u and lower quartile q_l of the data. The upper and lower quartiles are, respectively, the $Q(1/4)$ and $Q(3/4)$ quantiles, respectively. We take

$$\hat{\lambda} = (z_u - z_l)/(q_u - q_l)$$

as an estimate of the slope, where z_u and z_l are the u-th and l-th points of the ordered measured times, respectively. This informal estimation of the distribution of the measured data mean is robust since it will not be distorted by a few outliers (Chambers et al. 1983). These estimates are used to plot the theoretical distributions on the plots on the left side of the figures.

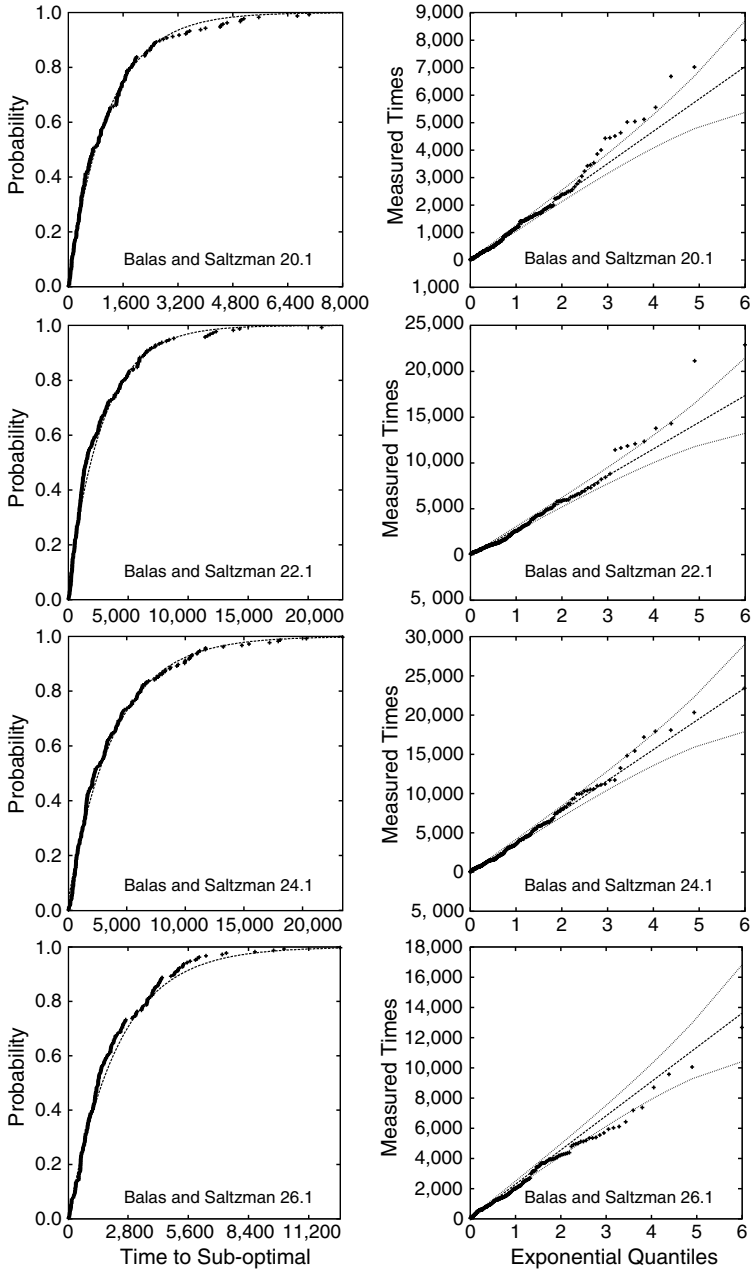

Figure 14 Empirical and Theoretical Time to Target Solution (look4) **Distributions and Quantile-Quantile Plots for GPR(ALL,INT)**
Note. Balas and Saltzman problems 20.1, 22.1, 24.1, and 26.1, using look4 = 7, 8, 7, and 8, respectively.

To analyze the straightness of the Q-Q plots, we superimpose them with variability information. For each plotted point, we show plus and minus one standard deviation in the vertical direction from the line fitted to the plot. An estimate of the standard deviation for point z_i, $i = 1, \ldots, 200$, of the Q-Q plot is

$$\hat{\sigma} = \hat{\lambda}\sqrt{\frac{p_i}{(1-p_i)200}}.$$

Figures 11, 12, 13, and 14 show that there is little departure from straightness in the Q-Q plots and consequently the distributions fit a two-parameter exponential distribution.

The following proposition (Aiex et al. 2002, Verhoeven and Aarts 1995) can be stated for a two-parameter (shifted) exponential distribution.

Proposition 1. *Let $P_\rho(t)$ be the probability of not having found a given (target) solution in t time units with ρ independent processes. If $P_1(t) = e^{-(t-\mu)/\lambda}$ with $\lambda \in \mathbb{R}^+$ and $\mu \in \mathbb{R}$, i.e., P_1 corresponds to a two-parameter exponential distribution, then $P_\rho(t) = e^{-\rho(t-\mu)/\lambda}$.*

This proposition follows from the definition of the two-parameter exponential distribution. It implies that the probability of finding a solution of a given value in time ρt with a sequential process is equal to $1 - e^{-(\rho t-\mu)/\lambda}$, while the probability of finding a solution at least as good as that given value in time t with ρ independent parallel processes is $1 - e^{-\rho(t-\mu)/\lambda}$. Note that if $\mu = 0$, then both probabilities are equal and correspond to the nonshifted exponential distribution. Furthermore, if $\rho\mu \ll \lambda$, then the two probabilities are approximately equal and it is possible to achieve approximate linear speedup in solution time to target solution by multiple independent processes.

We illustrate that GRASP with path relinking can be implemented in a straightforward parallel way and that near-linear speedup can be achieved. In these experiments, we disable stopping due to a maximum number of iterations, i.e., the algorithms terminate only when a solution of value at least as good as `look4` is found. This stopping criterion, although not used in practice for the AP3, is useful to study the behavior of GRASP. In §6.4, it was used to compare the execution times of GRASP variants by fixing their final solution quality. In this section, the stopping criterion using solution quality is applied to a parallel GRASP to study how its computational times vary according to the number of processors used, considering that the final solution quality is fixed among the executions. As a consequence, interrupting a parallel program upon finding a solution of a given quality allows for an understanding of how solution quality degrades along the parallel executions. For example, when a linear speedup is achieved for ρ processors, adding the computational time spent by each processor will equal the time spent during the sequential execution. Thus, there was no degradation of the solution quality along the parallel execution. However, if a sublinear speedup is obtained, then the sum of the computational time spent by each processor is greater than the time spent during the sequential execution, and thus the solution quality degraded along the parallel execution.

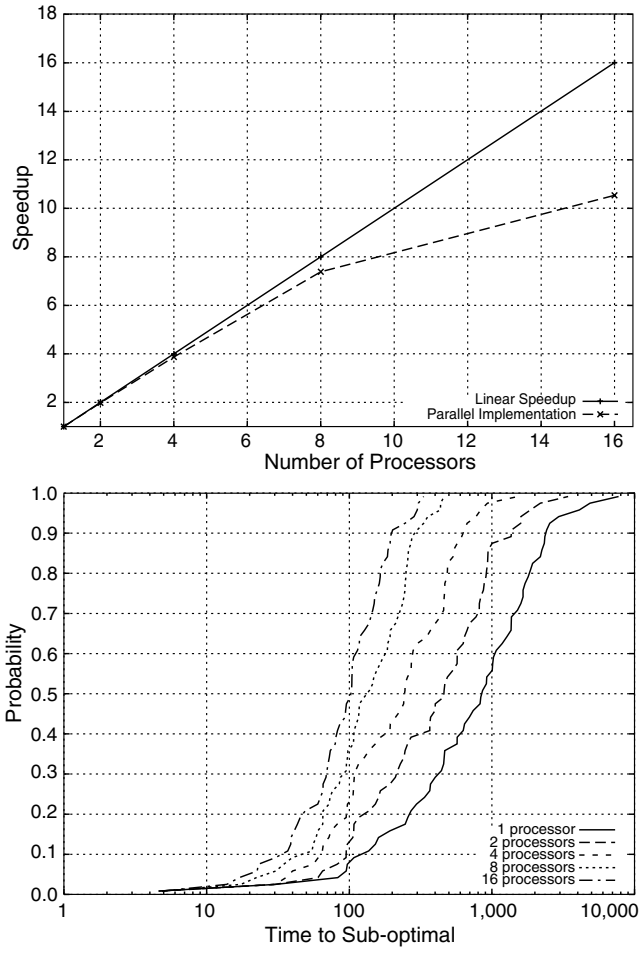

Figure 15 **Speedup and Empirical Distributions for Parallel Implementation of** `GPR(ALL,POST)`

Note. Balas and Saltzman problem 20.1 using `look4` = 7.

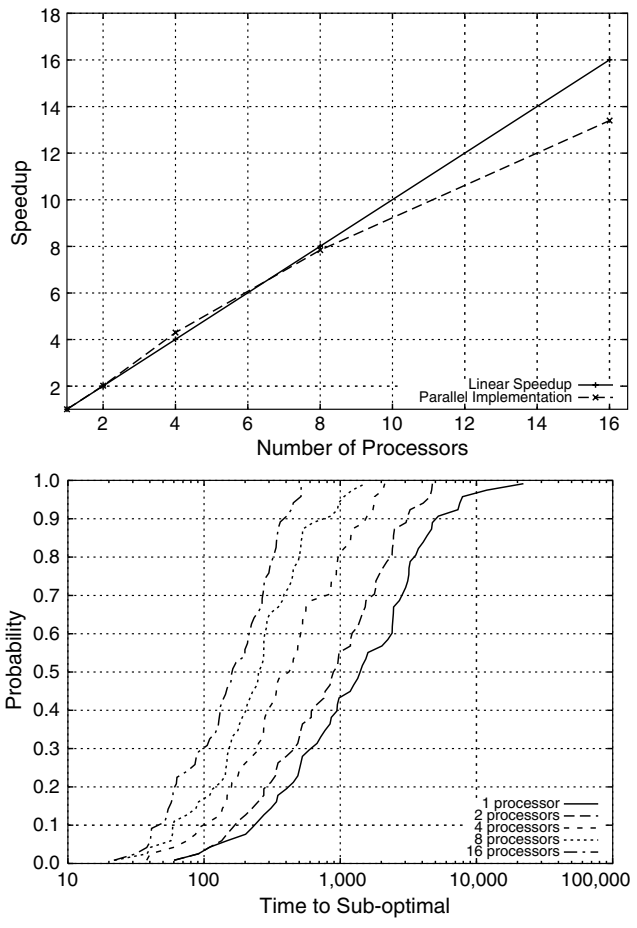

Figure 16 **Speedup and Empirical Distributions for Parallel Implementation of** `GPR(ALL,POST)`

Note. Balas and Saltzman problem 22.1 using `look4` = 8.

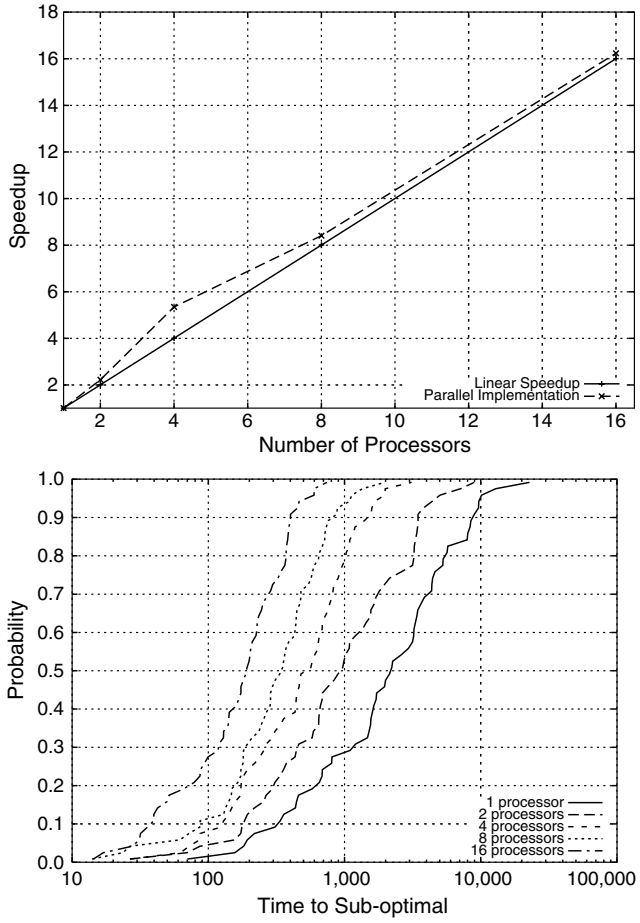

Figure 17 Speedup and Empirical Distributions for Parallel Implementation of GPR(ALL,POST)

Note. Balas and Saltzman problem 24.1 using look4 = 7.

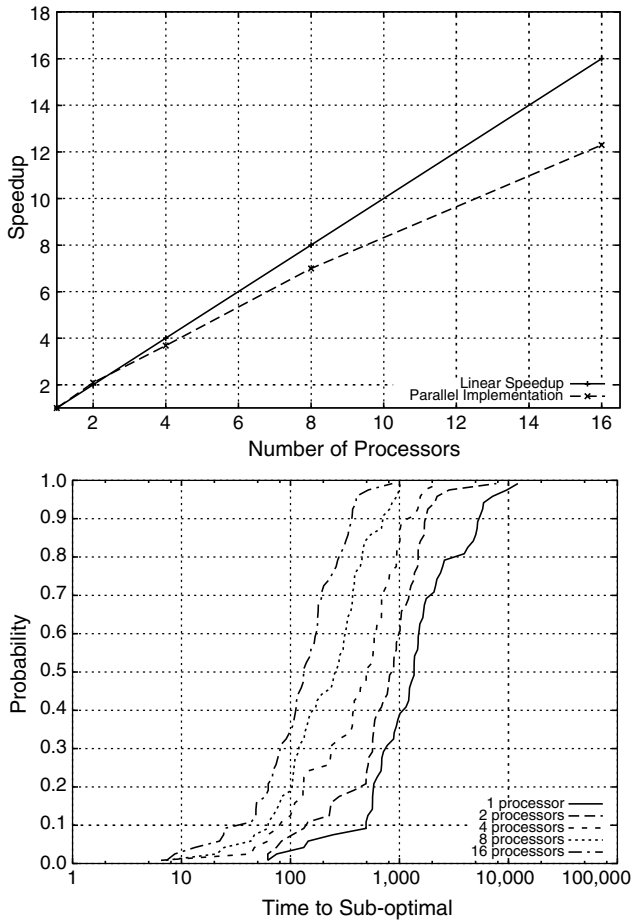

Figure 18 Speedup and Empirical Distributions for Parallel Implementation of GPR(ALL, POST)

Note. Balas and Saltzman problem 26.1 using look4 = 8.

Figures 15, 16, 17, and 18 show speedup and empirical distributions for a parallel implementation of the GRASP with path-relinking variant GPR(ALL,POST), on Balas and Saltzman problems 20.1, 22.1, 24.1, and 26.1, respectively. The plots were generated with 60 independent runs for each parallel run (with 1, 2, 4, 8, and 16 processors).

Figures 15 to 18 show the following.

- Table 10 summarizes the speedups shown in the figures. The table also shows efficiency (speedup divided by number of processors) values.
- Efficiencies are on average superlinear for 2 and 4 processors, slightly sublinear for 8 processors, and sublinear for 16 processors. Since there is very little communication between processors and the experiments were not done on a dedicated machine, the

Table 10 Speedup with Respect to a Single Processor Implementation and Efficiency (Speedup Divided by Number of Processors)

Balas and Saltzman Problem	Number of processors							
	2		4		8		16	
	Speedup	Eff.	Speedup	Eff.	Speedup	Eff.	Speedup	Eff.
20.1	1.98	0.99	3.87	0.97	7.38	0.92	10.53	0.66
22.1	2.01	1.01	4.29	1.07	7.83	0.98	13.39	0.84
24.1	2.22	1.11	5.34	1.34	8.40	1.05	16.23	1.01
26.1	2.09	1.05	3.68	0.92	6.99	0.87	12.29	0.77
Average	**2.08**	**1.04**	**4.30**	**1.08**	**7.65**	**0.96**	**13.11**	**0.69**

Note. Algorithm variant is GPR(ALL,POST). Instances are Balas and Saltzman 20.1, 22.1, 24.1, and 26.1 with target values 7, 8, 7, and 8, respectively.

Table 11 Estimates of Probability of Finding a Solution at Least as Good as the Target Solution in a Given Running Time, as a Function of Number of Processors

Balas and Saltzman Problem	Time	Number of processors 1	2	4	8	16
20.1	100s	0.08	0.13	0.23	0.36	0.49
	500s	0.36	0.54	0.83	1.00	1.00
	1,000s	0.55	0.86	0.98	1.00	1.00
22.1	100s	0.03	0.03	0.10	0.16	0.30
	500s	0.23	0.33	0.58	0.79	0.95
	1,000s	0.43	0.55	0.81	0.95	1.00
24.1	100s	0.01	0.04	0.08	0.11	0.28
	500s	0.18	0.31	0.50	0.71	0.94
	1,000s	0.28	0.53	0.78	0.93	1.00
26.1	100s	0.03	0.06	0.12	0.19	0.34
	500s	0.12	0.24	0.50	0.85	0.96
	1,000s	0.38	0.60	0.83	0.98	1.00

Note. Algorithm variant is `GPR(ALL,POST)`. Instances are Balas and Saltzman 20.1, 22.1, 24.1, and 26.1 with target values 7, 8, 7, and 8, respectively.

falloff in efficiency is probably due to processor availability.

• Table 11 shows, for given running times, the probability of finding a solution at least as good as the target solution in that time, as a function of number of processors. The table shows, for example, that the probability of finding a solution of value at most 8 on Balas and Saltzman instance 24.1 in less than 100 seconds, goes from 1% with one processor to 8% with four processors to 28% with 16 processors.

7. Concluding Remarks

In this paper, we presented a GRASP for the three-index assignment problem (AP3) and showed how path-relinking techniques can be used to improve the performance of the greedy randomized search. New construction and local search procedures were presented. We also described new ways to implement path relinking in a GRASP. Two-way path relinking, full elite set path relinking, path relinking post-optimization, and fixed-interval path-relinking intensification were shown to improve the basic path relinking strategy introduced by Laguna and Martí (1999).

Extensive computational experimentation was done with the different algorithms introduced in this paper.

The GRASP with path-relinking strategies were shown to improve the performance of a pure GRASP (without path relinking), both in terms of finding a solution faster and finding a better solution in a fixed number of iterations.

In general, variants requiring more work per iteration were shown to find solutions of a given quality in less time than variants doing less work per iteration. Also, these more sophisticated variants in general found better solutions in a fixed number of iterations.

We showed that these new GRASP with path-relinking heuristics improved the results for heuristics previously described in Balas and Saltzman (1991), Burkard et al. (1996), and Crama and Spieksma (1992).

We studied the probability distribution of the random variable *time to target solution* on several variants of GRASP with path relinking and concluded that these times can be fitted by a two-parameter exponential distribution. The parameters of these distributions were such that a straightforward parallel implementation of one of the variants was shown to achieve approximate linear speedup.

Acknowledgments

R. M. Aiex was supported by the Brazilian Council for Scientific and Technological Development (CNPq), and this work was done while he was visiting AT&T Labs Research. G. Toraldo was supported by the MURST national project "Algorithms for complex systems optimization" and the M.I.U.R. FIRB project "Large Scale Nonlinear Optimization," n. RBNE01WBBB. The authors thank R. Rudolf, M. J. Saltzman, and F. C. R. Spieksma for making available the test instances used in the experimental part of the paper.

References

Ahuja, R. K., J. B. Orlin, A. Tiwari. 2000. A greedy genetic algorithm for the quadratic assignment problem. *Comput. Oper. Res.* **27** 917–934.

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

Balas, E., M. J. Saltzman. 1991. An algorithm for the three-index assignment problem. *Oper. Res.* **39** 150–161.

Burkard, R. E., K. Fröhlich. 1980. Some remarks on 3-dimensional assignment problems. *Methods Oper. Res.* **36** 31–36.

Burkard, R. E., R. Rudolf. 1993. Computational investigations on 3-dimensional axial assignment problems. *Belgian J. Oper. Res. Statist. Comput. Sci.* **32** 85–98.

Burkard, R. E., R. Rudolf, G. J. Woeginger. 1996. Three-dimensional axial assignment problems with decomposable cost coefficients. *Discrete Appl. Math.* **65** 123–139.

Chambers, J. M., W. S. Cleveland, B. Kleiner, P. A. Tukey. 1983. *Graphical Methods for Data Analysis*. Chapman & Hall, Boston, MA.

Crama, Y., F. C. R. Spieksma. 1992. Approximation algorithms for three-dimensional assignment problems with triangle inequalities. *Eur. J. Oper. Res.* **60** 273–279.

Crama, Y., A. W. J. Kolen, A. G. Oerlemans, F. C. R. Spieksma. 1990. Throughput rate optimization in the automated assembly of printed circuit boards. *Annals Oper. Res.* **26** 455–480.

Feo, T. A., J. L. González-Velarde. 1995. The intermodal trailer assignment problem: Models, algorithms, and heuristics. *Transportation Sci.* **29** 330–341.

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

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

Festa, P., M. G. C. Resende. 2002. GRASP: An annotated bibliography. C. C. Ribeiro, P. Hansen, eds. *Essays and Surveys in Metaheuristics*. Kluwer Academic Publishers, Boston, MA, 325–367.

Fleurent, C., F. Glover. 1999. Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. *INFORMS J. Comput.* **11** 198–204.

Frieze, A. M. 1983. Complexity of a 3-dimensional assignment problem. *Eur. J. Oper. Res.* **13** 161–164.

Frieze, A. M., J. Yadegar. 1981. An algorithm for solving 3-dimensional assignment problems with application to scheduling a teaching practice. *J. Oper. Res. Soc.* **32** 989–995.

Fröhlich, K. 1979. Dreidimensionale Zuordnungsprobleme. Master's thesis, Mathematik Institut, Universität Köln, Köln, Germany.

Garey, M. R., D. S. Johnson. 1979. *Computers and Intractability—A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York.

Glover, F., M. Laguna. 1997. *Tabu Search*. Kluwer Academic Publishers, Boston, MA.

Glover, F., M. Laguna, R. Martí. 2000. Fundamentals of scatter search and path-relinking. *Control Cybernetics* **39** 653–684.

Hansen, P., L. Kaufman. 1973. A primal-dual algorithm for the three-dimensional assignment problem. *Cahiers du CERO* **15** 327–336.

Hart, J. P., A. W. Shogan. 1987. Semi-greedy heuristics: An empirical study. *Oper. Res. Lett.* **6** 107–114.

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

Leue, O. 1972. Methoden zur Lösung dreidimensionaler Zuordnungsprobleme. *Angewandte Informatik* **14** 154–162.

Li, Y., P. M. Pardalos, M. G. C. Resende. 1994. A greedy randomized adaptive search procedure for the quadratic assignment problem. P. M. Pardalos, H. Wolkowicz, eds. *Quadratic Assignment and Related Problems*, Vol. 16. *DIMACS Series on Discrete Mathematics and Theoretical Computer Science.* American Mathematical Society, Providence, RI, 237–261.

Mavridou, T., P. M. Pardalos, L. S. Pitsoulis, M. G. C. Resende. 1998. A GRASP for the biquadratic assignment problem. *Eur. J. Oper. Res.* **105** 613–621.

Murphey, R. A., P. M. Pardalos, L. S. Pitsoulis. 1998. A parallel GRASP for the data association multidimensional assignment problem. P. M. Pardalos, ed. *Parallel Processing of Discrete Problems*, Vol. 106. *The IMA Volumes in Mathematics and Its Applications.* Springer-Verlag, New York, 159–180.

Pardalos, P. M., L. S. Pitsoulis. 2000. *Nonlinear Assignment Problems: Algorithms and Applications*. Kluwer Academic Publishers, Boston, MA.

Pardalos, P. M., M. G. C. Resende, eds. 2002. *Handbook of Applied Optimization.* Oxford University Press, New York.

Pardalos, P. M., L. S. Pitsoulis, M. G. C. Resende. 1995. A parallel GRASP implementation for the quadratic assignment problem. A. Ferreira, J. Rolim, eds. *Parallel Algorithms for Irregularly Structured Problems—Irregular'94.* Kluwer Academic Publishers, Boston, MA, 115–130.

Pardalos, P. M., L. S. Pitsoulis, M. G. C. Resende. 1997. Algorithm 769: Fortran subroutines for approximate solution of sparse quadratic assignment problems using GRASP. *ACM Trans. Math. Software* **23** 196–208.

Pierskalla, W. P. 1967. The tri-substitution method for the three-multidimensional assignment problem. *Canadian Oper. Res. Soc. J.* **5** 71–81.

Pierskalla, W. P. 1968. The multidimensional assignment problem. *Oper. Res.* **16** 422–431.

Pitsoulis, L. S. 1999. *Algorithms for Nonlinear Assignment Problems.* Ph.D. thesis, Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL.

Pitsoulis, L. S., P. M. Pardalos, D. W. Hearn. 2001. Approximate solutions to the turbine balancing problem. *Eur. J. Oper. Res.* **130** 147–155.

Rangel, M. C., N. M. M. Abreu, P. O. Boaventura Netto. 1999. GRASP in the QAP: An acceptance bound for initial solution. *Proc. 3rd Metaheuristics Internat. Conf.*, Catholic University of Rio de Janeiro, Angra dos Reis, Rio de Janeiro, Brazil, 381–386.

Resende, M. G. C., P. M. Pardalos, Y. Li. 1996. Algorithm 754: Fortran subroutines for approximate solution of dense quadratic assignment problems using GRASP. *ACM Trans. Math. Software* **22** 104–118.

Robertson, A. J. 2001. A set of greedy randomized adaptive local search procedure (GRASP) implementations for the multidimensional assignment problem. *Comput. Optim. App.* **19** 145–164.

Schrage, L. 1979. A more portable Fortran random number generator. *ACM Trans. Math. Software* **5** 132–138.

Snir, M., S. Otto, S. Huss-Lederman, D. Walker, J. Dongarra. 1998. *MPI—The Complete Reference, Volume 1—The MPI Core*. The MIT Press, Cambridge, MA.

Verhoeven, M. G. A., E. H. L. Aarts. 1995. Parallel local search. *J. Heuristics* **1** 43–66.

Vlach, M. 1967. Branch and bound method for the three-index assignment problem. *Ekonomicko-Mathematický Obzor* **3** 181–191.

Voss, S. 2000. Heuristics for nonlinear assignment problems. P. M. Pardalos, L. S. Pitsoulis, eds. *Nonlinear Assignment Problems: Algorithms and Applications.* Kluwer Academic Publishers, Boston, MA, 175–215.