

GRASP WITH PATH RELINKING FOR THE THREE-INDEX ASSIGNMENT PROBLEM

R.M. AIEX, M.G.C. RESENDE, P.M. PARDALOS, AND G. TORALDO

ABSTRACT. This paper describes variants of GRASP (greedy randomized adaptive search procedure) with path relinking for the three index assignment problem (AP3). GRASP is a multi-start 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, Rudolf, and Woeginger (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.

1. INTRODUCTION

The three-index assignment problem (AP3) was first stated by Pierskalla [31] 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 in 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

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

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

Date: February 20, 2001.

Key words and phrases. Metaheuristics, GRASP, path relinking, three-index assignment, parallel computing.

AT&T Labs Research Technical Report.

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

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

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

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 $1, 2, \dots, n$. 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 $N = \{1, 2, \dots, n\}$. 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 [15, 18] since it is a special case of the set partitioning problem. Applications of the AP3 can be found in Pierskalla [30, 31], Frieze and Yeagar [16], and Crama et al. [8] 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 board.

Exact and heuristic algorithms have been proposed for the 3-index assignment problem, including [3, 4, 5, 6, 9, 17, 21, 24, 30, 31, 40].

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

2. GRASP CONSTRUCTION AND LOCAL SEARCH

2.1. GRASP. A greedy randomized adaptive search procedure (GRASP) [11, 12, 13] is a multi-start or iterative process, in which each GRASP iteration consists of

```

procedure CONSTRUCT(seed, n, c, S)
1   Select  $\alpha \in [0, 1]$  at random;
2    $S = \emptyset$ ;
3    $C = \{(i, j, k) \in I \times J \times K\}$ ;
4   for  $p = 1, \dots, n - 1$  do
5        $\underline{c} = \min\{c_{ijk} \mid (i, j, k) \in C\}$ ;
6        $\bar{c} = \max\{c_{ijk} \mid (i, j, k) \in C\}$ ;
7        $C' = \{(i, j, k) \in C \mid c_{ijk} \leq \underline{c} + \alpha(\bar{c} - \underline{c})\}$ ;
8       Select  $(i_p, j_p, k_p) \in C'$  at random;
9        $S = S \cup \{(i_p, j_p, k_p)\}$ ;
10       $\Gamma^i(p) = \{(i, j, k) \in C \mid i = i_p\}$ ;
11       $\Gamma^j(p) = \{(i, j, k) \in C \mid j = j_p\}$ ;
12       $\Gamma^k(p) = \{(i, j, k) \in C \mid k = k_p\}$ ;
13       $C = C \setminus \{\Gamma^i(p) \cup \Gamma^j(p) \cup \Gamma^k(p)\}$ ;
14  rof;
15   $S = S \cup C$ ;
end CONSTRUCT

```

FIGURE 1. The GRASP construction phase

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 [22]. 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 $g : C \rightarrow \mathbb{R}$. 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 [1, 10, 14, 25, 27, 26, 28, 29, 32, 33, 34, 35, 36, 37].

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.

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 p -th 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 $c_{ijk} \leq \underline{c} + \alpha(\bar{c} - \underline{c})$, where

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

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 5 and 6 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.

```

procedure LOCAL( $n, c, c_S, S$ )
1   for  $p = 1, \dots, n - 1$  do
2     for  $q = p + 1, \dots, n$  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          $\bar{S} = S \setminus \{(i_p, j_p, k_p)\} \setminus \{(i_q, j_q, k_q)\}$ ;
6          $S = \bar{S} \cup \{(i_p, j_q, k_p)\} \cup \{(i_q, j_p, k_q)\}$ ;
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         $\bar{S} = S \setminus \{(i_p, j_p, k_p)\} \setminus \{(i_q, j_q, k_q)\}$ ;
13         $S = \bar{S} \cup \{(i_p, j_p, k_q)\} \cup \{(i_q, j_q, k_p)\}$ ;
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

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 line 1 to 18 examines the neighboring solutions. The cost of a neighbor associated with the p permutation 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 the q permutation 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.

3. PATH RELINKING

Path relinking was first introduced in the context of tabu search [19] as an approach to integrate intensification and diversification strategies in the search. See [20] for a survey of path relinking. It consists in 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 [23].

For the 3-index assignment problem, path relinking is done between an initial solution $S = \{(1, j_1^S, k_1^S), (2, j_2^S, k_2^S), \dots, (n, j_n^S, k_n^S)\}$ and a guiding solution $T = \{(1, j_1^T, k_1^T), (2, j_2^T, k_2^T), \dots, (n, j_n^T, k_n^T)\}$. 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:

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

and

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

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

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

```

procedure PATHRL( $n, c, c_S, S, T$ )
1  Let  $S$  be  $\{(1, j_1^S, k_1^S), (2, j_2^S, k_2^S), \dots, (n, j_n^S, k_n^S)\}$ ;
2  Let  $T$  be  $\{(1, j_1^T, k_1^T), (2, j_2^T, k_2^T), \dots, (n, j_n^T, k_n^T)\}$ ;
3   $c_{gmin} = c_S$ ;  $S_{gmin} = S$ ;
4   $\delta J = \{i = 1, \dots, n \mid j_i^S \neq j_i^T\}$ ;
5   $\delta K = \{i = 1, \dots, n \mid k_i^S \neq k_i^T\}$ ;
6  for  $|\delta J| + |\delta K| > 2$  do
7       $c_{min} = \infty$ ;
8      for  $i \in \delta J$  do
9          Let  $q$  be such that  $j_q^T == j_i^S$ ;
10          $\bar{S} = S \setminus \{(i, j_i^S, k_i^S)\} \setminus \{(q, j_q^S, k_q^S)\}$ ;
11          $\bar{S} = \bar{S} \cup \{(i, j_q^S, k_i^S)\} \cup \{(q, j_i^S, k_q^S)\}$ ;
12          $\bar{c} = c_S - c_{i, j_i^S, k_i^S} - c_{q, j_q^S, k_q^S} + c_{i, j_q^S, k_i^S} + c_{q, j_i^S, k_q^S}$ ;
13         if  $\bar{c} < c_{min}$  then
14              $c_{min} = \bar{c}$ ;  $S_{min} = \bar{S}$ ;  $i_{min} = i$ ;
15             flag = 0;
16         fi;
17     rof;
18     for  $i \in \delta K$  do
19         Let  $q$  be such that  $k_q^T == k_i^S$ ;
20          $\bar{S} = S \setminus \{(i, j_i^S, k_i^S)\} \setminus \{(q, j_q^S, k_q^S)\}$ ;
21          $\bar{S} = \bar{S} \cup \{(i, j_i^S, k_q^S)\} \cup \{(q, j_q^S, k_i^S)\}$ ;
22          $\bar{c} = c_S - c_{i, j_i^S, k_i^S} - c_{q, j_q^S, k_q^S} + c_{i, j_i^S, k_q^S} + c_{q, j_q^S, k_i^S}$ ;
23         if  $\bar{c} < c_{min}$  then
24              $c_{min} = \bar{c}$ ;  $S_{min} = \bar{S}$ ;  $i_{min} = i$ ;
25             flag = 1;
26         fi;
27     rof;
28      $S = S_{min}$ ;  $c_S = c_{min}$ ;
29     if flag == 0 then  $\delta J = \delta J \setminus \{i_{min}\}$ ;
30     if flag == 1 then  $\delta K = \delta K \setminus \{i_{min}\}$ ;
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 .

while in a type-two move, triplets

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

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 [23]. 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 [14] for using elite solutions within the GRASP framework.

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

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

and

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

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. $c_{gmin} < c_{best}$, i.e. S_{gmin} is the best solution found so far;
2. $c_{best} < c_{gmin} < c_{worst}$ 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 differs significantly from all elite solutions.

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 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 Subsection 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 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 Section 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 Section 3, is computed (line 20).

```

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, cS, S);
7      if |POOL| == maxpool then
8          select SUBPOOL ⊆ POOL;
9          for T ∈ SUBPOOL do
10             Sgmin = PATHRL(n, c, cS, S, T);
11             UPDATE_POOL(Sgmin, cgmin, POOL);
12             Sgmin = PATHRL(n, c, cT, T, S);
13             UPDATE_POOL(Sgmin, cgmin, POOL);
14         rof;
15     else POOL = POOL ∪ {S} fi;
16     if mod(i, freq) == 0 then INTENSIFY(POOL) fi;
17     Sbest = argmin{POOL};
18     if cbest ≤ look4 then break fi;
19 rof;
20 POSTOPT(POOL);
21 Sbest = argmin{POOL};
22 return (Sbest);
end GRASP_PR;

```

FIGURE 4. Pseudo-code for GRASP with path relinking.

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 taxonomy proposed by Verhoeven and Aarts [39] 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. Each GRASP construction phase

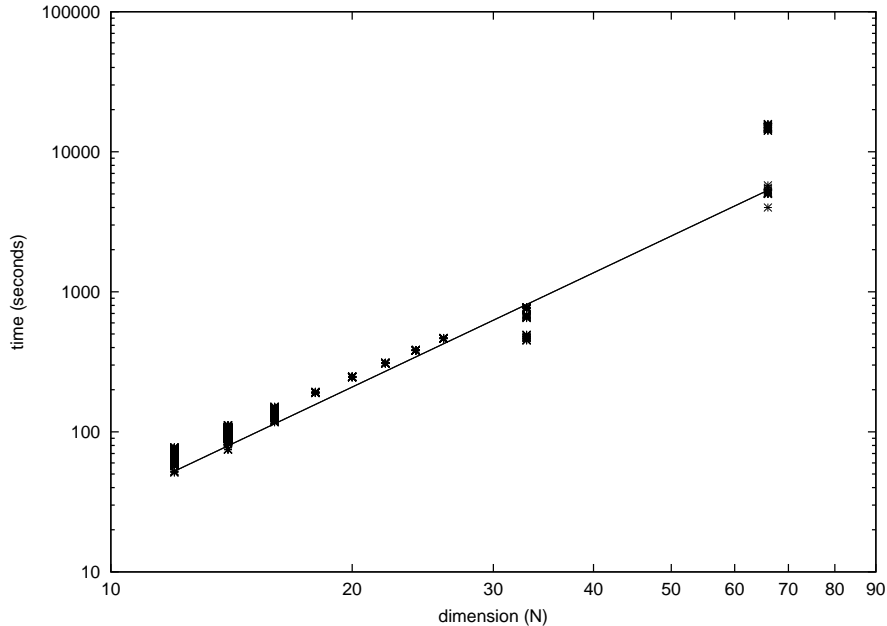


FIGURE 5. Problem dimension against CPU time for 10,000 iterations of GRASP with path relinking.

is initialized with a random number generator seed. To assure independence of processors, identical seeds of the random number generator (`rand()`) must not be used by more than one processor. The initial seed for process `my_rank` is computed in lines 2–4. This way, each process has a sequence of `maxitr` initial seeds. The loop from line 6 to line 34 executes the iterations. The construction phase seed (`seedc`) is computed 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 any 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 global best 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

```

procedure PARALLEL_GRASP_PR( $n, c, \text{seed}, \text{look4}, \text{maxitr}, \text{maxpool}, \text{freq}$ )
1  my_rank = GetRank(); nprocs = GetNumProcs();
2  for  $i = 1, \dots, \text{maxitr} * \text{my\_rank}$  do
3      seed = rand(seed);
4  rof;
5  POOL =  $\emptyset$ ; num_stop = 0;
6  for  $i = 1, \dots, \infty$  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  $\subseteq$  POOL;
13         for  $T \in \text{SUBPOOL}$  do
14              $S_{gmin} = \text{PATHRL}(n, c, c_S, S, T)$ ;
15             UPDATE_POOL( $S_{gmin}, c_{gmin}, \text{POOL}$ );
16              $S_{gmin} = \text{PATHRL}(n, c, c_T, T, S)$ ;
17             UPDATE_POOL( $S_{gmin}, c_{gmin}, \text{POOL}$ );
18         rof;
19     else POOL = POOL  $\cup$  { $S$ } fi;
20     if mod( $i, \text{freq}$ ) = 0 then INTENSIFY(POOL) fi;
21      $S_{best} = \text{argmin}\{\text{POOL}\}$ ;
22     if  $c_{best} \leq \text{look4}$  then SendAll( $\text{look4\_stop}$ ) fi;
23     if  $i == \text{maxitr}$  then
24         num_stop = num_stop + 1;
25         SendAll( $\text{maxitr\_stop}$ )
26     fi;
27     received = VerifyReceiving(flag);
28     if received then
29         if flag ==  $\text{look4\_stop}$  then break;
30         else if flag ==  $\text{maxitr\_stop}$  then
31             num_stop = num_stop + 1;
32         fi;
33     if num_stop == nprocs then break fi;
34 rof;
35 POSTOPT(POOL);
36  $S_{GlobalBest} = \text{GetGlobalBest}(S_{best})$ ;
37 return ( $S_{GlobalBest}$ );
end PARALLEL_GRASP_PR;

```

FIGURE 6. Pseudo-code for parallel GRASP with path relinking.

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

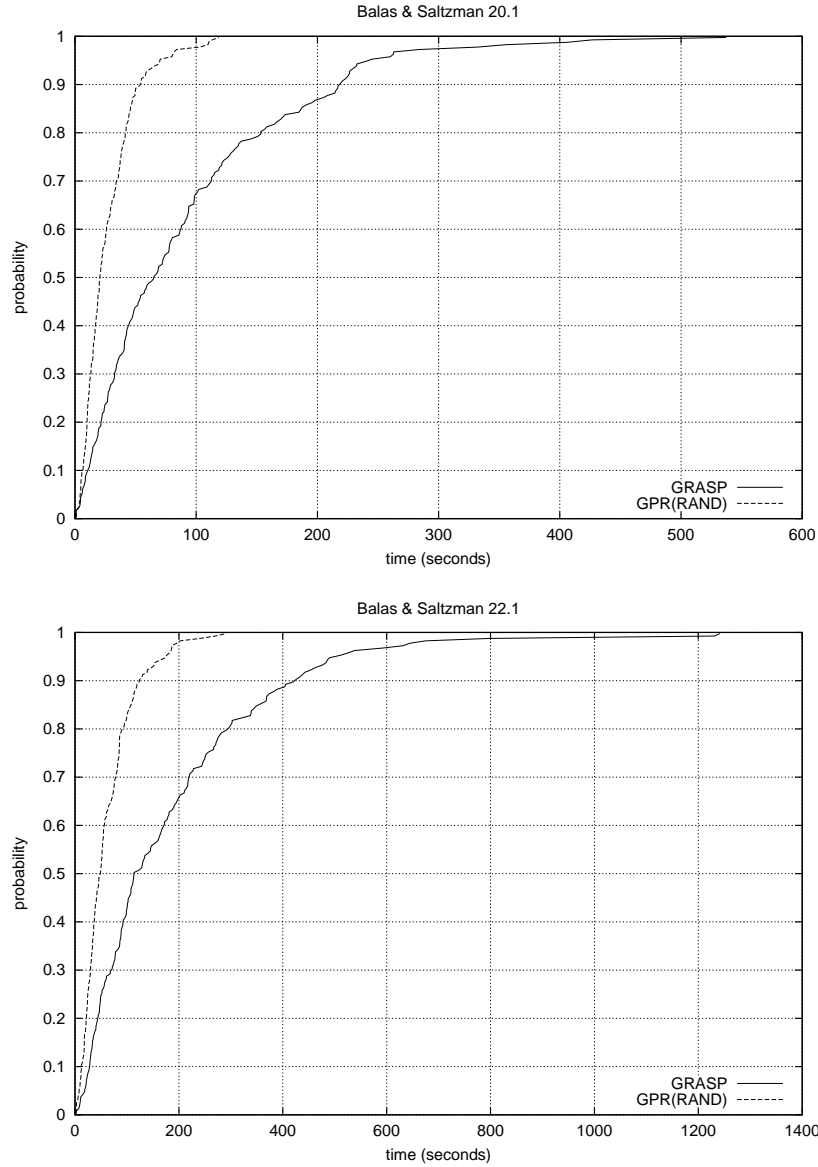


FIGURE 7. Empirical probability distributions of time to target value for GRASP and GPR(RAND) (Balas & Saltzman test problems 20.1 and 22.1). Stopping criteria: `look4 = 19` and `look4 = 20` for test problems 20.1 and 22.1, respectively.

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 MIP-Spro F77 compiler using flags `-O3 -r4 -64`. The Message-Passing Interface (MPI)

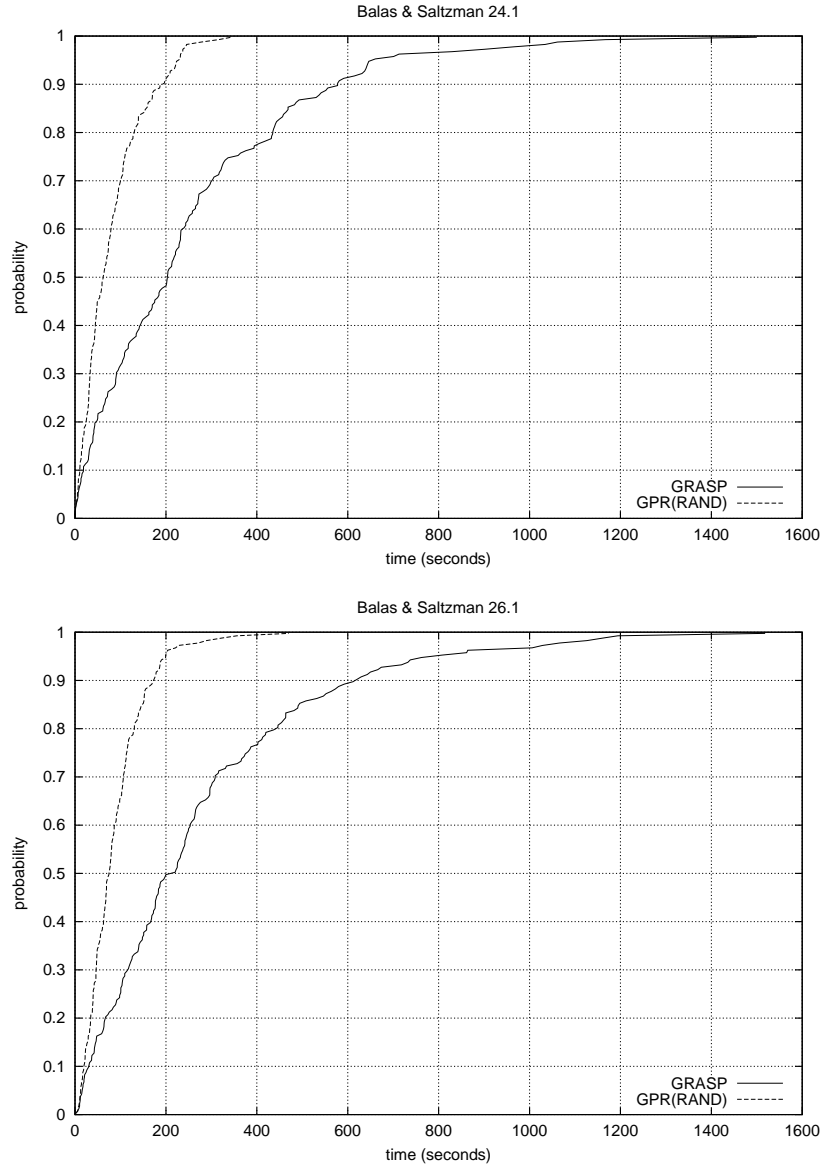


FIGURE 8. Empirical probability distributions of time to target value for GRASP and GPR(RAND) (Balas & Saltzman test problems 24.1 and 26.1). Stopping criteria: `look4 = 17` and `look4 = 19` for test problems 24.1 and 26.1, respectively.

specification has become a common standard for message-passing libraries for parallel computations [38]. 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

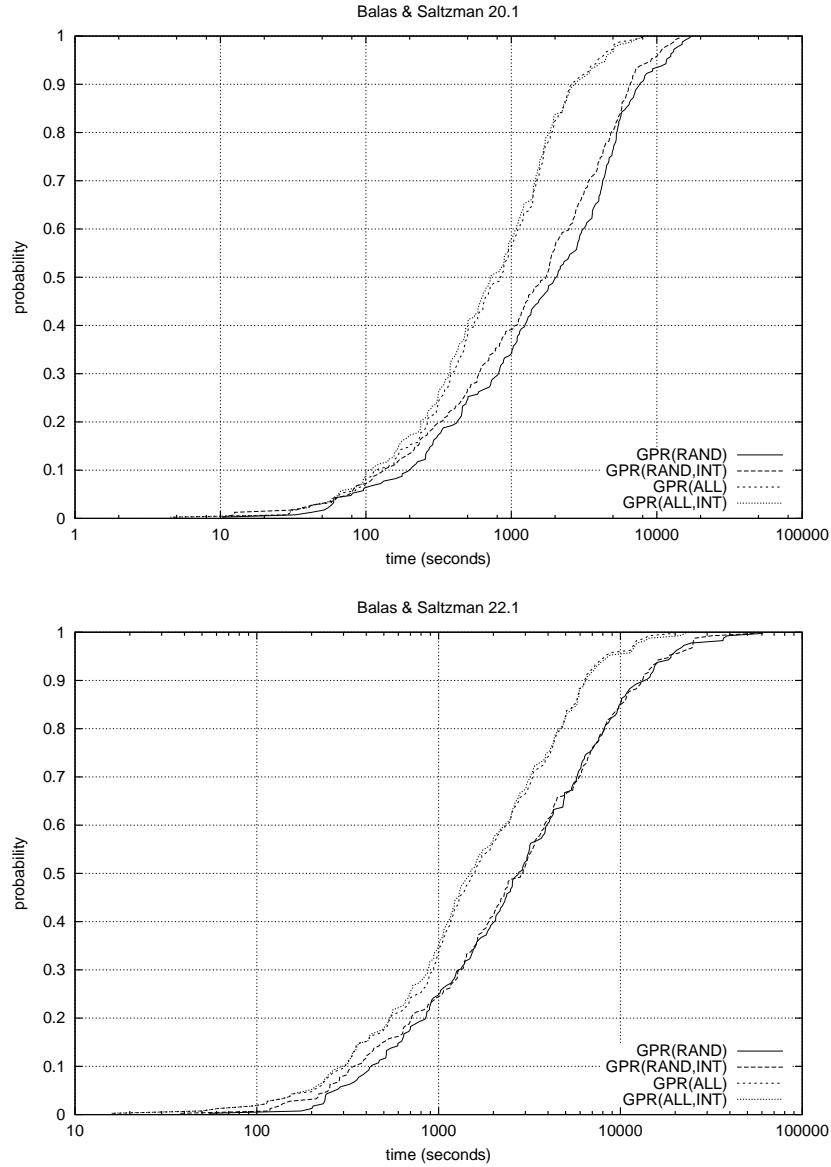


FIGURE 9. Empirical probability distributions of time to target value for different variants of GRASP with path relinking (Balas & Saltzman test problems 20.1 and 22.1). Stopping criteria: `look4 = 7` and `look4 = 8` for test problems 20.1 and 22.1, respectively.

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 exclude the time to read the problem data, initialize the random number generator seeds, and to output the solution.

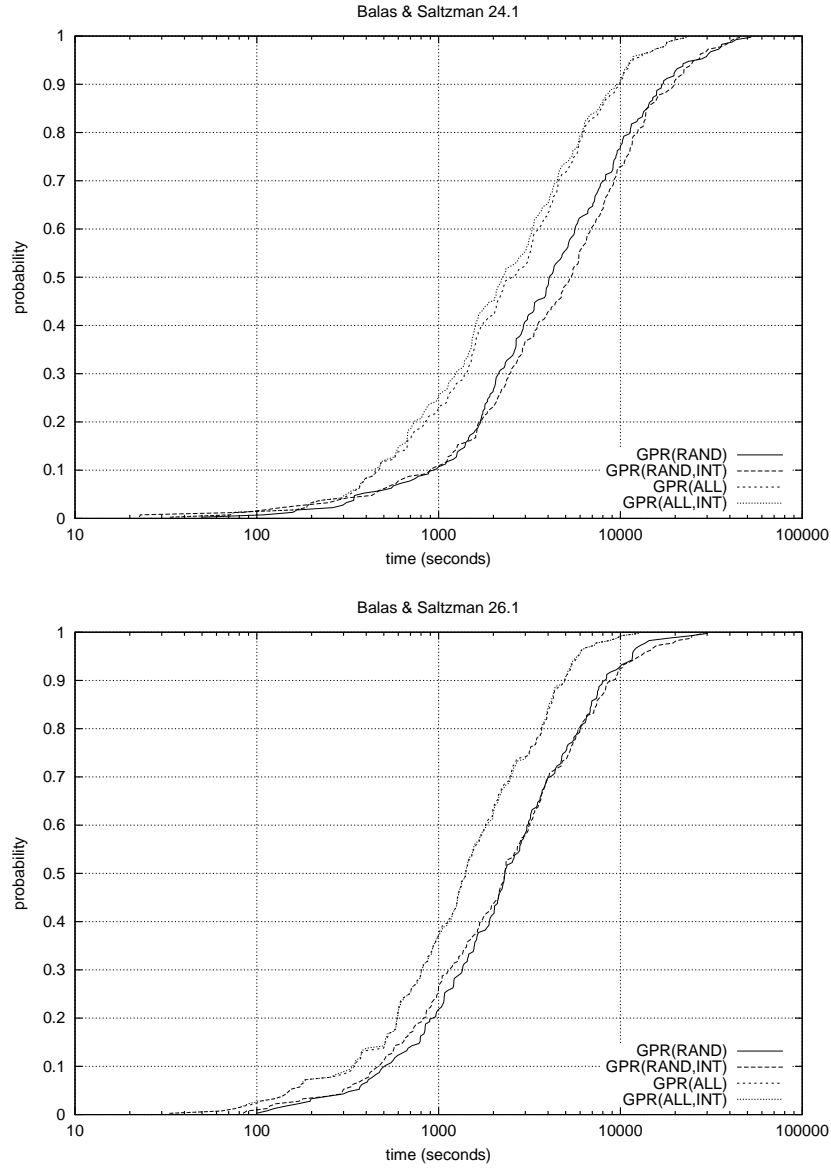


FIGURE 10. Empirical probability distributions of time to target value for different variants of GRASP with path relinking (Balas & Saltzman test problems 24.1 and 26.1). Stopping criteria: `look4 = 7` and `look4 = 8` for test problems 24.1 and 26.1, respectively.

TABLE 1. Probability estimates of finding a solution at least as good as the target solution as a function of maximum solution time. Instances are Balas & Saltzman 20.1, 22.1, 24.1, and 26.1 with target values 19, 20, 17, and 19, respectively.

time	20.1		22.1		24.1		26.1	
	GPR(RAND)	GRASP	GPR(RAND)	GRASP	GPR(RAND)	GRASP	GPR(RAND)	GRASP
50s	.89	.44	.52	.25	.45	.22	.35	.16
100s	.98	.68	.83	.43	.70	.32	.66	.25
150s	1.00	.80	.93	.56	.84	.42	.85	.37
200s	1.00	.87	.98	.66	.91	.49	.96	.50

TABLE 2. Time to find a solution at least as good as the target solution as a function of probability. Instances are Balas & Saltzman 20.1, 22.1, 24.1, and 26.1 with target values 7, 8, 7, and 8, respectively.

probability	Balas & Saltzman 20.1			
	GPR(ALL, INT)	GPR(ALL)	GPR(RAND, INT)	GPR(RAND)
0.2	238s	266s	338s	428s
0.5	718s	851s	1744s	2063s
0.8	1856s	1918s	4887s	5331s

probability	Balas & Saltzman 22.1			
	GPR(ALL, INT)	GPR(ALL)	GPR(RAND, INT)	GPR(RAND)
0.2	544s	546s	710s	852s
0.5	1502s	1584s	2916s	2763s
0.8	4731s	4780s	8244s	8161s

probability	Balas & Saltzman 24.1			
	GPR(ALL, INT)	GPR(ALL)	GPR(RAND, INT)	GPR(RAND)
0.2	742s	856s	1714s	1697s
0.5	2274s	2603s	5378s	4084s
0.8	6198s	6306s	12467s	11336s

probability	Balas & Saltzman 26.1			
	GPR(ALL, INT)	GPR(ALL)	GPR(RAND, INT)	GPR(RAND)
0.2	594s	592s	852s	963s
0.5	1411s	1411s	2315s	2315s
0.8	3698s	3691s	5941s	6049s

TABLE 3. Balas & Saltzman test problems (100 iterations)

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

TABLE 4. Balas & Saltzman test problems (10,000 iterations)

n	B-S	OPT	GRASP cost	GRASP wins	GPR(R) cost	GPR(R) wins	GPR(R,P) cost	GPR(R,P) wins	GPR(R,P,l) cost	GPR(R,P,l) wins	GPR(A) cost	GPR(A) wins	GPR(A,P) cost	GPR(A,P) wins	GPR(A,P,l) cost	GPR(A,P,l) wins							
12	24.0	15.6	16.4 (8.37s)	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 (14.37s)	2	10.6 (17.71s)	3	10.2 (17.83s)	4	10.2 (22.67s)	4	10.0 (102.22s)	5	10.0 (102.30s)	5	10.0 (106.55s)	5							
16	25.0	10.0	13.2 (23.51s)	0	12.8 (27.64s)	1	12.8 (27.85s)	1	11.0 (34.33s)	2	10.6 (138.31s)	2	10.6 (138.37s)	2	10.2 (143.89s)	4							
18	17.6	6.4	11.8 (36.58s)	0	9.8 (42.04s)	0	8.0 (42.20s)	2	9.8 (50.39s)	0	7.2 (184.86s)	3	7.2 (183.70s)	3	7.4 (190.88s)	2							
20	27.4	4.8	14.2 (53.40s)	0	9.8 (60.20s)	1	9.4 (60.53s)	1	10.2 (70.64s)	1	7.2 (237.02s)	3	6.6 (237.35s)	4	6.4 (246.70s)	5							
22	18.8	4.0	13.6 (74.82s)	0	12.4 (83.18s)	0	10.6 (83.27s)	1	10.2 (95.63s)	1	7.2 (298.17s)	4	7.2 (298.40s)	4	7.8 (309.64s)	3							
24	14.0	1.8	13.4 (101.66s)	0	9.8 (111.14s)	1	9.8 (111.98s)	1	8.4 (127.09s)	3	7.4 (368.45s)	5	7.4 (368.82s)	5	7.4 (382.45s)	5							
26	15.7	1.3	13.4 (134.53s)	0	9.8 (145.60s)	1	9.8 (146.27s)	1	9.8 (164.15s)	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 6. Crama & Spieksma type II test problems

Iterations	n	C-S	LB	GRASP cost wins	GPR(R) cost wins	GPR(R,P) cost wins	GPR(R,P,I) cost wins	GPR(A) cost wins	GPR(A,P) cost wins	GPR(A,P,I) cost wins
100	33	4861	4772	4805 (4.49s)	4805 (4.59s)	4799 (5.50s)	4799 (7.26s)	4798 (6.89s)	4798 (7.50s)	4798 (9.50s)
100	33	5142	5035	5086 (4.25s)	5081 (4.37s)	5070 (5.49s)	5072 (7.54s)	5075 (6.83s)	5071 (7.62s)	5071 (9.97s)
100	33	4352	4260	4306 (4.36s)	4301 (4.42s)	4288 (5.54s)	4290 (7.58s)	4290 (6.72s)	4290 (7.25s)	4289 (9.19s)
100	66	9780	9633	9728 (133.99s)	9728 (134.10s)	9709 (141.95s)	9707 (154.09s)	9710 (145.42s)	9709 (149.12s)	9703 (163.67s)
100	66	9142	8831	8990 (138.28s)	8981 (139.04s)	8962 (144.25s)	8966 (155.43s)	8966 (149.31s)	8966 (150.97s)	8964 (167.24s)
100	66	9888	9670	9803 (133.23s)	9791 (133.81s)	9768 (139.25s)	9770 (154.42s)	9766 (145.58s)	9764 (157.98s)	9767 (161.25s)
total winners				0	0	3	0	1	2	2

Iterations	n	C-S	LB	GRASP cost wins	GPR(R) cost wins	GPR(R,P) cost wins	GPR(R,P,I) cost wins	GPR(A) cost wins	GPR(A,P) cost wins	GPR(A,P,I) cost wins
10,000	33	4861	4772	4804 (419.43s)	4797 (433.18s)	4797 (434.92s)	4797 (446.75s)	4797 (752.91s)	4797 (754.34s)	4797 (766.06s)
10,000	33	5142	5035	5076 (398.35s)	5067 (414.93s)	5067 (413.58s)	5069 (429.32s)	5068 (759.88s)	5068 (761.77s)	5068 (772.84s)
10,000	33	4352	4260	4296 (405.03s)	4287 (418.35s)	4287 (420.94s)	4288 (430.96s)	4287 (747.17s)	4287 (748.03s)	4287 (762.19s)
10,000	66	9780	9633	9720 (13453.17s)	9703 (13449.01s)	9703 (13387.55s)	9699 (13545.63s)	9694 (14676.68s)	9694 (14553.60s)	9694 (14629.08s)
10,000	66	9142	8831	8976 (13238.00s)	8957 (13257.29s)	8957 (13335.41s)	8956 (13412.89s)	8951 (14871.51s)	8951 (14706.02s)	8954 (14922.91s)
10,000	66	9888	9670	9784 (12689.67s)	9757 (12739.95s)	9756 (12748.95s)	9759 (12784.51s)	9753 (14446.85s)	9753 (14326.70s)	9751 (14391.67s)
total winners				0	3	3	1	4	4	4

TABLE 8. Burkard, Rudolf, & Woeginger test problems (100 iterations)

n	B-R-W	GRASP cost wins	GPR(R) cost wins	GPR(R,P) cost wins	GPR(R,P,I) cost wins	GPR(A) cost wins	GPR(A,P) cost wins	GPR(A,P,I) cost wins
12	1188.02	1186.92 (0.12s)	1186.85 (0.14s)	1186.82 (0.31s)	1186.81 (0.73s)	1186.81 (0.51s)	1186.81 (0.62s)	1186.81 (1.03s)
14	1469.19	1468.18 (0.21s)	1467.91 (0.23s)	1467.76 (0.49s)	1467.74 (1.10s)	1467.75 (0.75s)	1467.75 (0.91s)	1467.74 (1.52)
16	1476.80	1475.93 (0.35s)	1475.65 (0.38s)	1475.17 (0.76s)	1475.14 (1.61s)	1475.15 (1.07s)	1475.13 (1.33s)	1475.13 (2.21)
total winners		221	248	293	299	297	299	300

TABLE 9. Burkard, Rudolf, & Woeginger test problems (10,000 iterations)

n	B-R-W	GRASP cost wins	GPR(R) cost wins	GPR(R,P) cost wins	GPR(R,P,I) cost wins	GPR(A) cost wins	GPR(A,P) cost wins	GPR(A,P,I) cost wins
12	1188.02	1186.81 (11.19s)	1186.81 (13.29s)	1186.81 (13.35s)	1186.81 (15.96s)	1186.81 (65.90s)	1186.81 (65.96s)	1186.81 (68.30s)
14	1469.19	1467.75 (19.09s)	1467.74 (21.92s)	1467.74 (22.01s)	1467.74 (25.67s)	1467.74 (94.70s)	1467.74 (94.79s)	1467.74 (98.02s)
16	1476.80	1475.18 (31.77s)	1475.13 (35.58s)	1475.13 (35.74s)	1475.13 (41.08s)	1475.13 (134.62s)	1475.13 (134.86s)	1475.13 (139.30s)
total winners		295	300	300	300	300	300	300

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 [3]. 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 [9]. 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 [9] 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 by Burkard, Rudolf, and Woeginger [6]. 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, Rudolf, and Woeginger [6] (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, Rudolf, and Woeginger 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 $|\text{POOL}|$ 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 which 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 which 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 tradeoff in terms of CPU time?
2. What are the tradeoffs 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 Subsection 6.2 and used the variants of GRASP with path relinking listed in Subsection 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 [3]. Two hundred independent runs for each of the two variants were done 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 i -th sorted running time (t_i) a probability $p_i = (i - \frac{1}{2})/200$, and plot the points $z_i = (t_i, p_i)$, for $i = 1, \dots, 200$. 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 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 with 10,000 iterations (long). Five quick and five long independent runs for each variant-instance pair were done.

Tables 3 and 4 show results for the Balas and Saltzman test problems [3] 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 [3] 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, 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 that 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.

Tables 3 and 4 show the following:

- On short runs, GRASP with path relinking finds better solutions than 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 full elite set path relinking without intensification, but full elite set path relinking with post optimization finds better solutions than random path relinking with post optimization and full elite set path relinking with post optimization 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 [9], 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 in [9]. We refer to the Crama and Spieksma heuristic by 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 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 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, 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 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 same quality solutions.
- 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 same quality solutions.
- On short runs, fixed interval intensification deteriorated 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.

Tables 8 and 9 show results for the Burkard, Rudolf, and Woeginger test problems [6], respectively, with 100 and 10,000 iterations. Each table shows problem dimension n , the value obtained by the Burkard, Rudolf, and Woeginger heuristics Simple_LSH on instances of size $n = 12$ and LSH on instances of size $n \geq 14$. We refer to the Burkard, Rudolf, and Woeginger heuristics as B-R-W. Each of the 100 instances of each size were 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 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.
- 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, GRASP with full elite set path relinking and post optimization was better than GRASP with random path relinking and post optimization, and 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, Resende, and Ribeiro [2] 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 [7] 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 and therefore the runs are independent. Our description of each plot follows [2] closely. For each instance/variant pair, the running times are sorted in increasing order. To plot the empirical distribution, we associate with the i -th sorted running time (t_i) a probability $p_i = (i - \frac{1}{2})/200$, and plot the points $z_i = (t_i, p_i)$, for $i = 1, \dots, 200$.

The plot on the right is a quantile-quantile plot (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 , $i = 1, \dots, 200$, 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.

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 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 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 $Q(\frac{1}{4})$ and $Q(\frac{3}{4})$ 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 [7]. These estimates are used to plot the theoretical distributions on the plots on the left side of the figures.

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 , $i = 1, \dots, 200$, 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 [2, 39] 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 non-shifted exponential distribution. Furthermore, if $\rho\mu \ll \lambda$, then the two probabilities are approximately equal and it is possible to approximately achieve linear speed-up 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 maximum number of iterations, i.e. the algorithms terminate only when a solution of value at least as good as `look4` is found. Figures 15, 16, 17, and 18 show speedup and empirical distributions for a parallel implementation of GRASP with path relinking variant `GPR(ALL, POST)`, on

TABLE 10. Speedup with respect to a single processor implementation and efficiency (speedup divided by number of processors). Algorithm variant is GPR(ALL,POST). Instances are Balas & Saltzman 20.1, 22.1, 24.1, and 26.1 with target values 7, 8, 7, and 8, respectively.

Balas & Saltzman problem	number of processors							
	2		4		8		16	
	speedup	eff.	speedup	eff.	speedup	eff.	speedup	eff.
20.1	1.98	.99	3.87	.97	7.38	.92	10.53	.66
22.1	2.01	1.01	4.29	1.07	7.83	.98	13.39	.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	.92	6.99	.87	12.29	.77
average:	2.08	1.04	4.30	1.08	7.65	.96	13.11	.69

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

Balas & Saltzman problem	time	number of processors				
		1	2	4	8	16
20.1	100s	.08	.13	.23	.36	.49
	500s	.36	.54	.83	1.00	1.00
	1000s	.55	.86	.98	1.00	1.00
22.1	100s	.03	.03	.10	.16	.30
	500s	.23	.33	.58	.79	.95
	1000s	.43	.55	.81	.95	1.00
24.1	100s	.01	.04	.08	.11	.28
	500s	.18	.31	.50	.71	.94
	1000s	.28	.53	.78	.93	1.00
26.1	100s	.03	.06	.12	.19	.34
	500s	.12	.24	.50	.85	.96
	1000s	.38	.60	.83	.98	1.00

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 fall off 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 & Saltzman instance 24.1 in less than

100 seconds, goes from 1% with one processor to 8% with four processes to 28% with 16 processors.

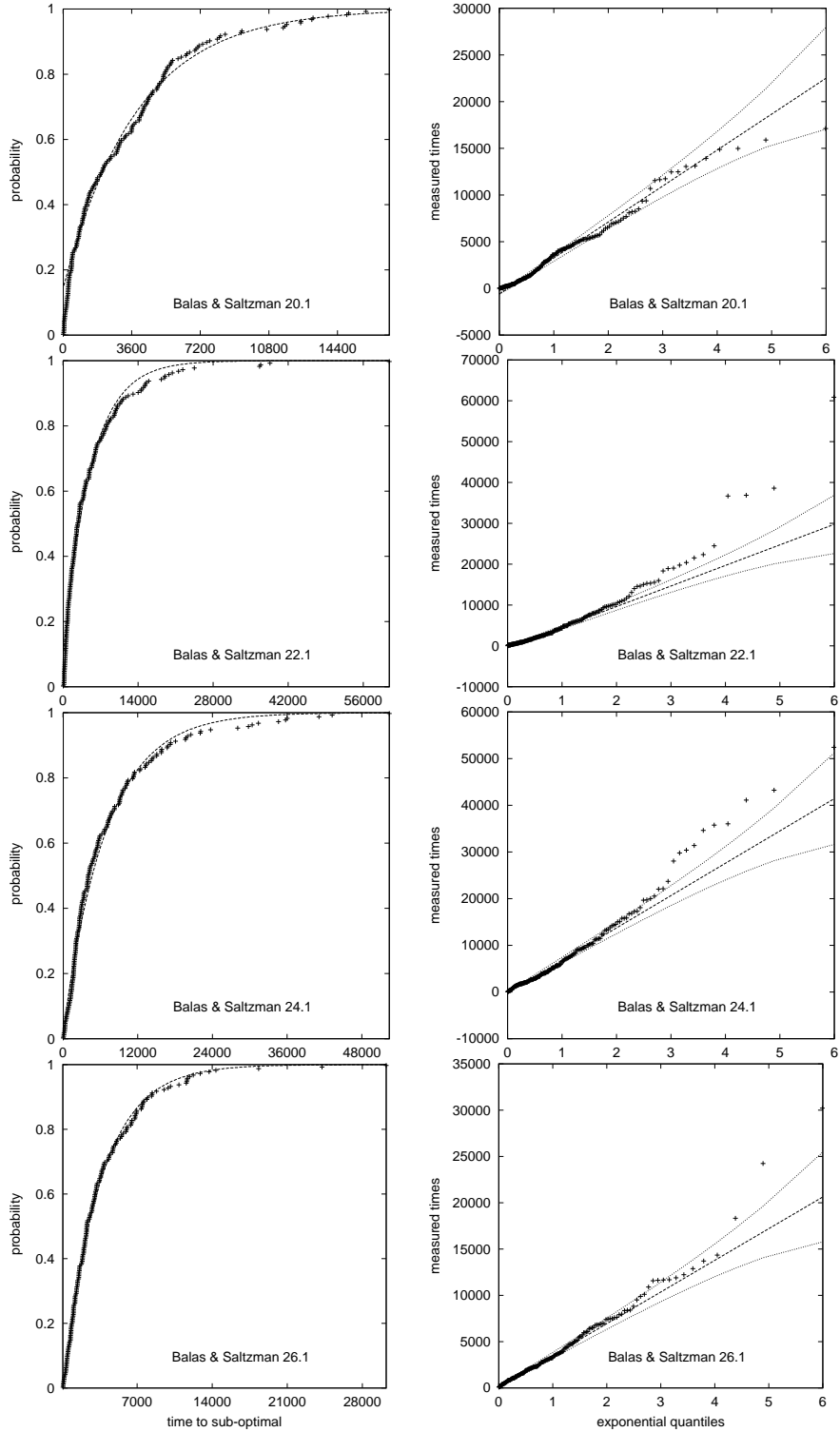


FIGURE 11. Empirical and theoretical time to target solution (look4) distributions and quantile-quantile plots for GPR(RAND): Balas & Saltzman problems 20.1, 22.1, 24.1, and 26.1, using look4 = 7, 8, 7, and 8, respectively.

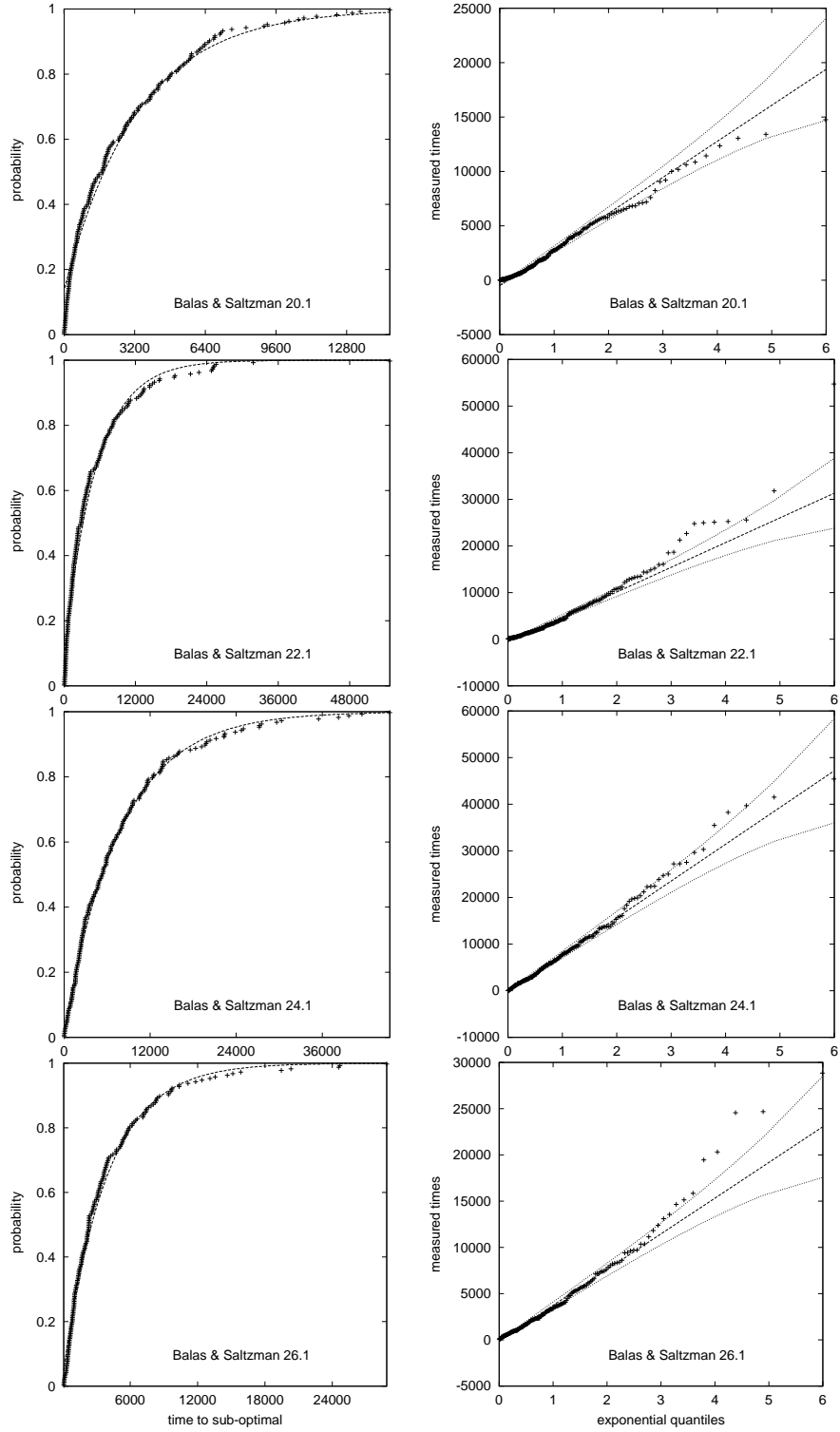


FIGURE 12. Empirical and theoretical time to target solution (look4) distributions and quantile-quantile plots for GPR(RAND,INT): Balas & Saltzman problems 20.1, 22.1, 24.1, and 26.1, using look4 = 7, 8, 7, and 8, respectively.

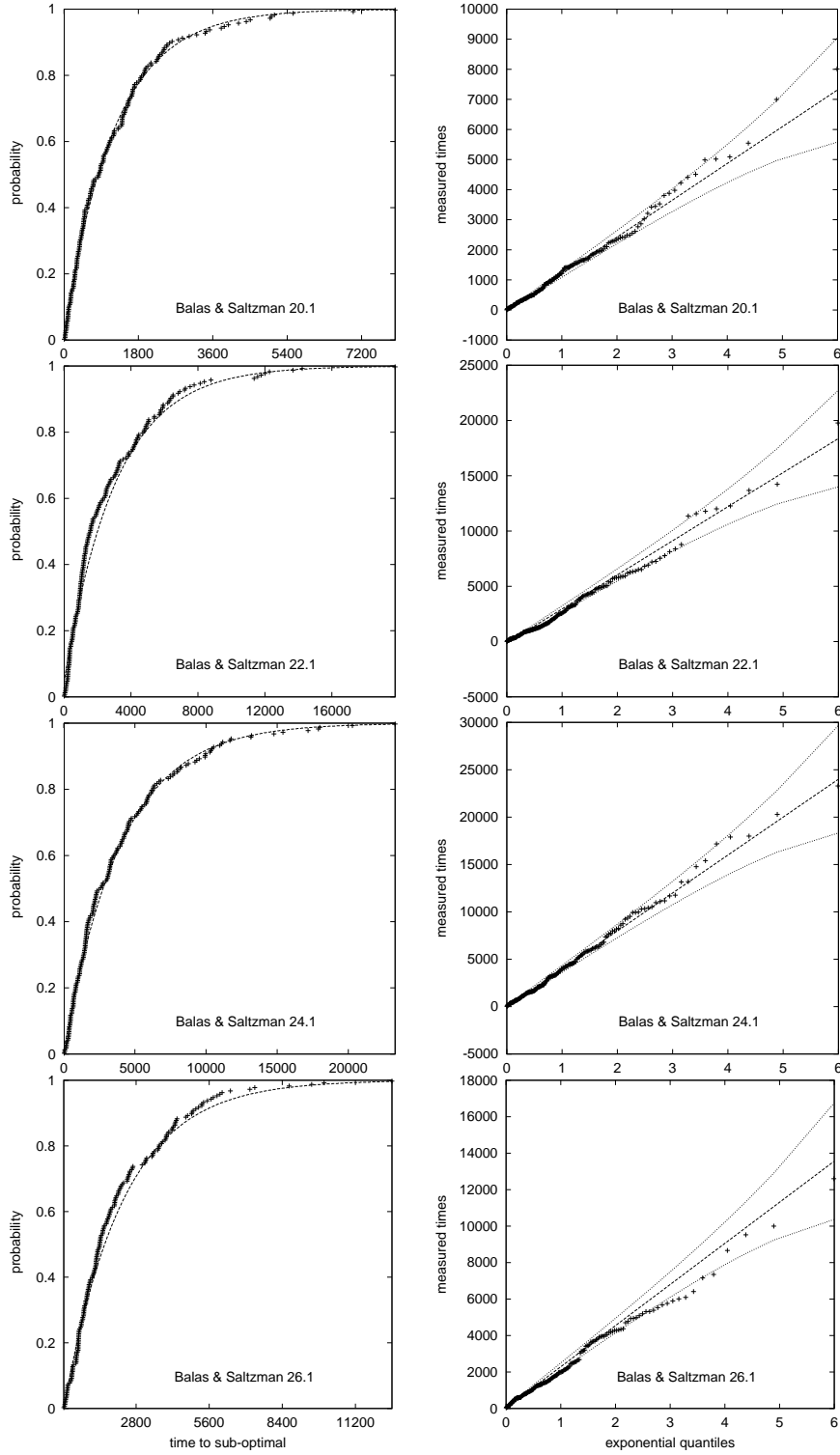


FIGURE 13. Empirical and theoretical time to target solution (look4) distributions and quantile-quantile plots for GPR(ALL): Balas & Saltzman problems 20.1, 22.1, 24.1, and 26.1, using look4 = 7, 8, 7, and 8, respectively.

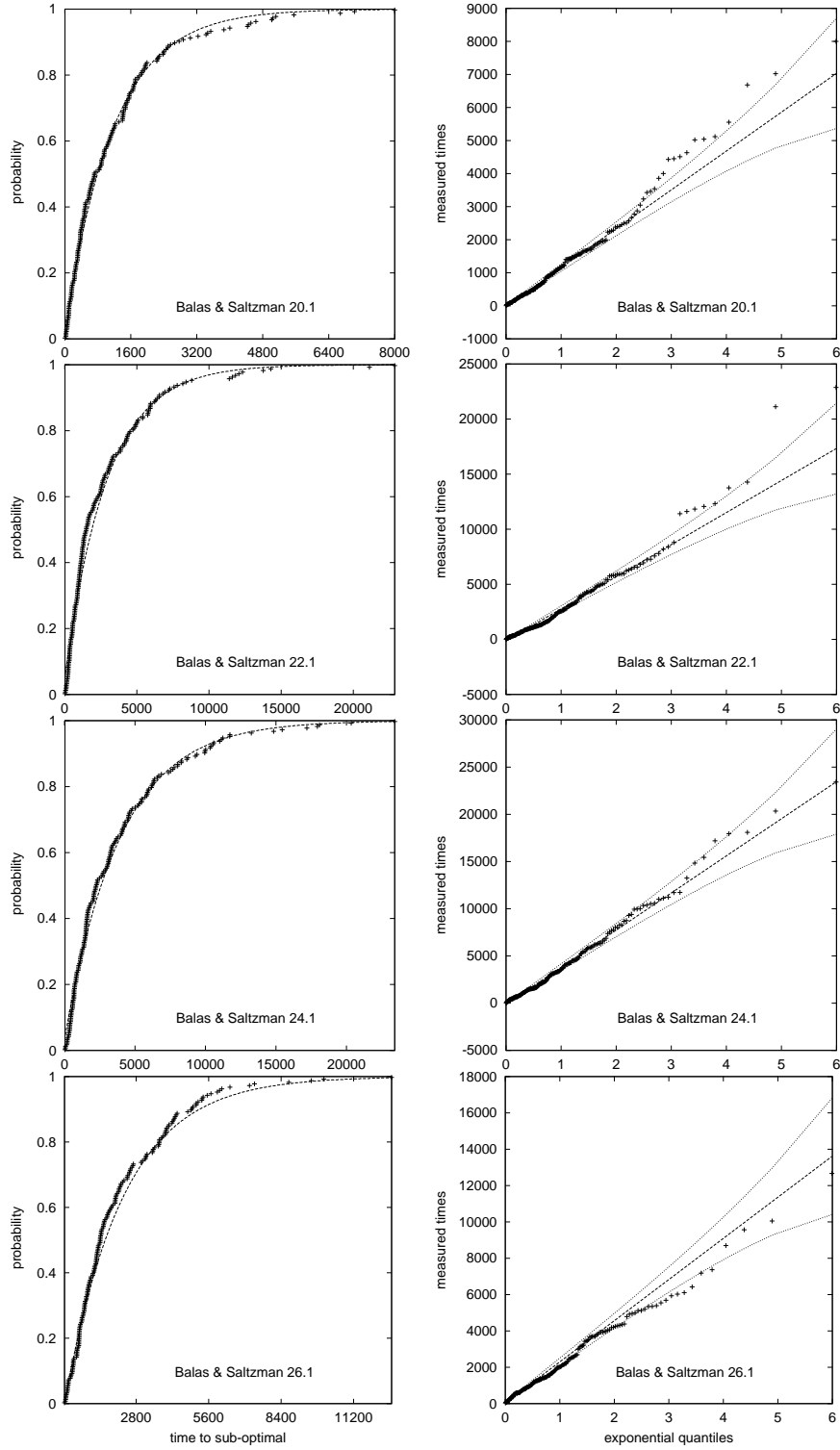


FIGURE 14. Empirical and theoretical time to target solution (look4) distributions and quantile-quantile plots for GPR(ALL,INT): Balas & Saltzman problems 20.1, 22.1, 24.1, and 26.1, using look4 = 7, 8, 7, and 8, respectively.

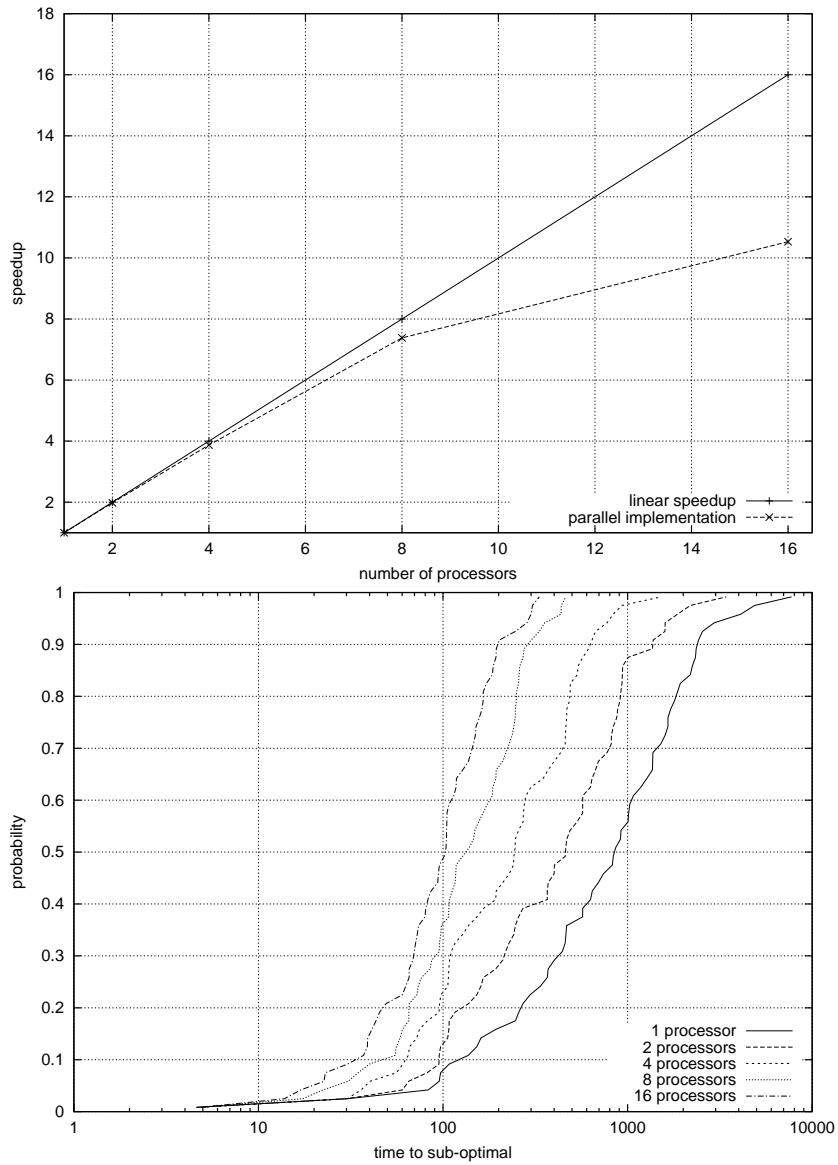


FIGURE 15. Speedup and empirical distributions for parallel implementation of GPR(ALL,POST): Balas & Saltzman problem 20.1 using look4 = 7.

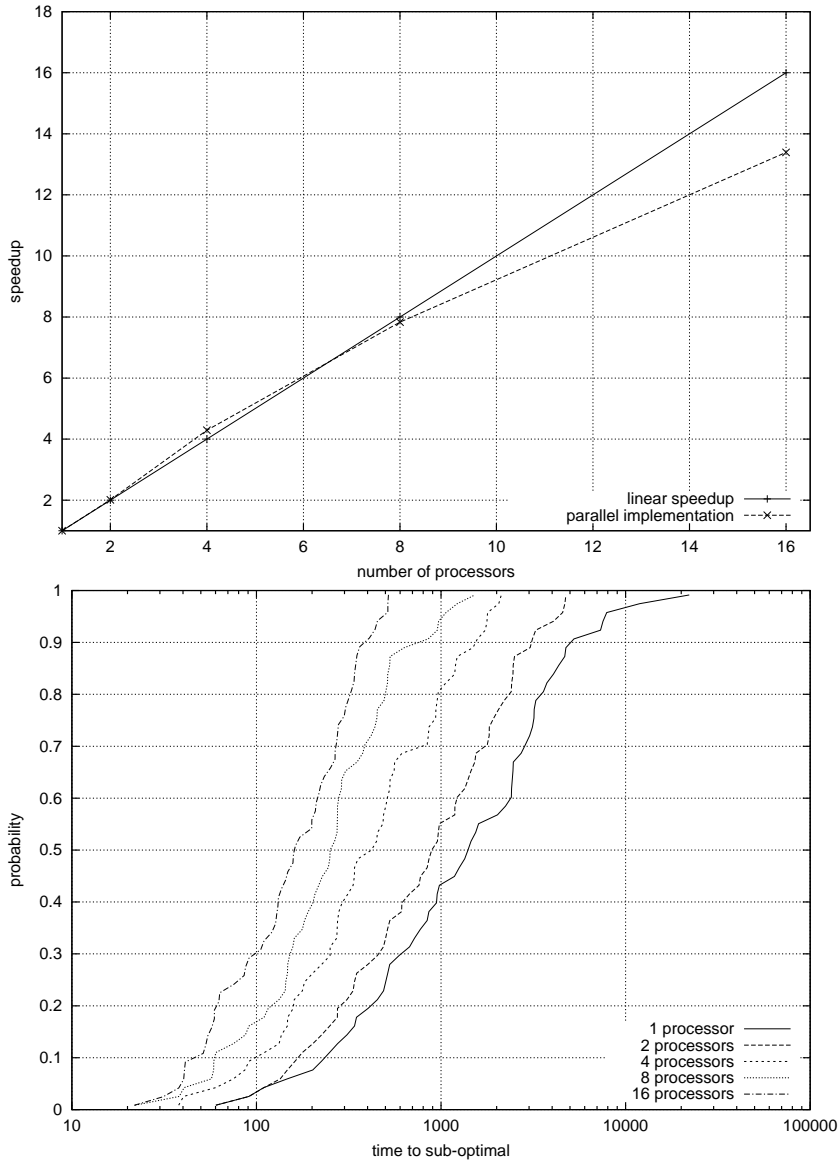


FIGURE 16. Speedup and empirical distributions for parallel implementation of GPR(ALL,POST): Balas & Saltzman problem 22.1 using look4 = 8.

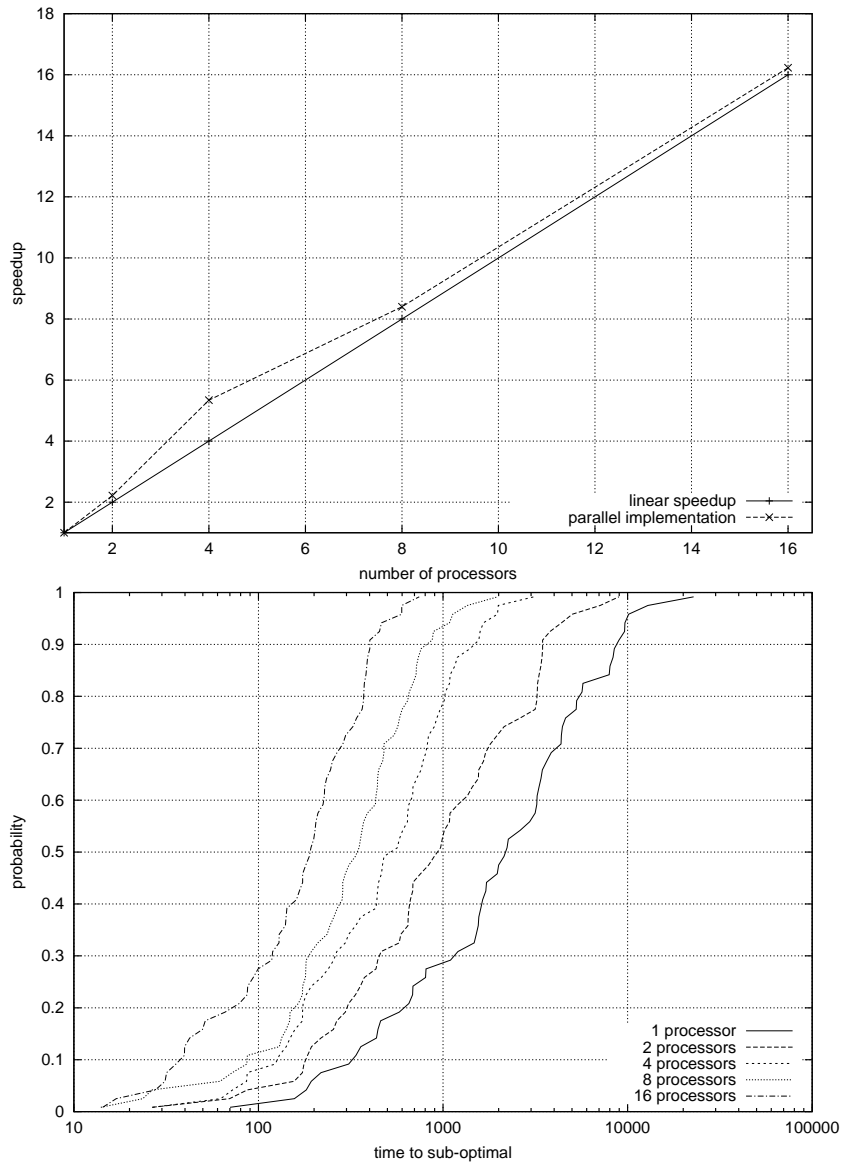


FIGURE 17. Speedup and empirical distributions for parallel implementation of GPR(ALL,POST): Balas & Saltzman problem 24.1 using look4 = 7.

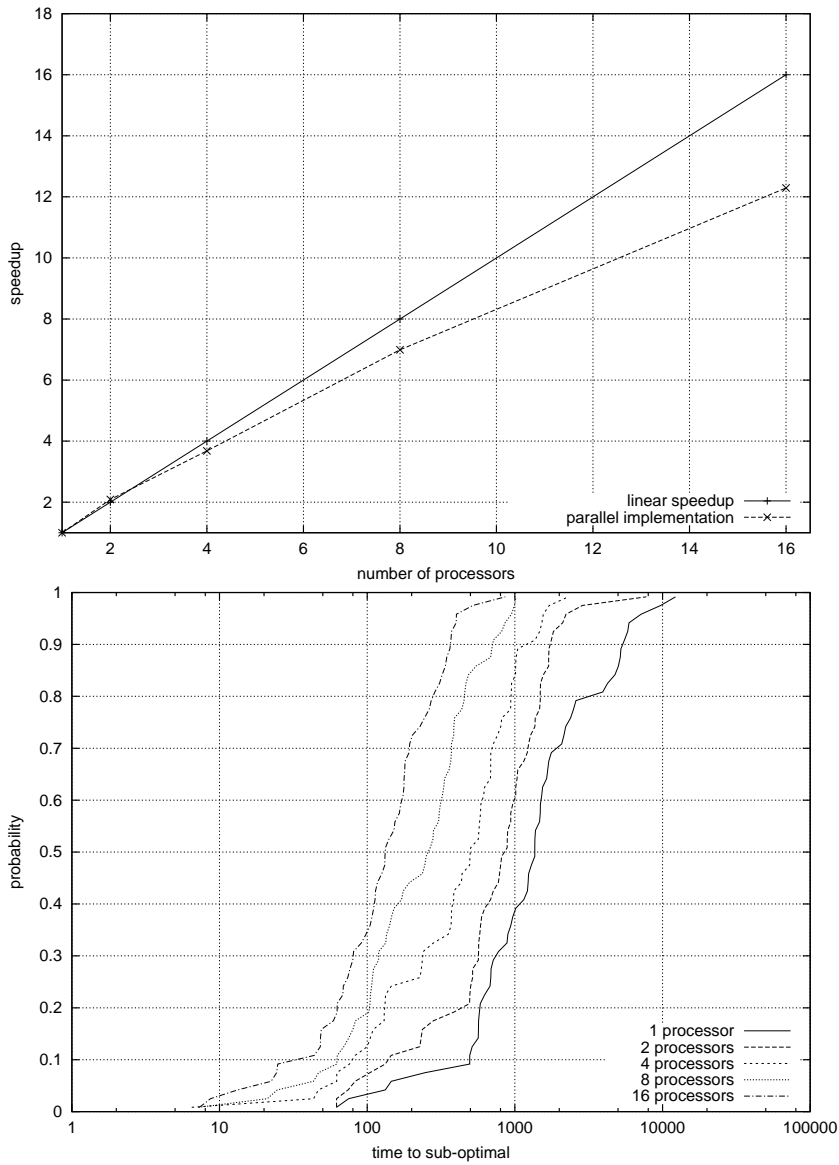


FIGURE 18. Speedup and empirical distributions for parallel implementation of $GPR(ALL,POST)$: Balas & Saltzman problem 26.1 using $look4 = 8$.

7. CONCLUDING REMARKS

In this paper, we presented a GRASP for the 3-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 [23].

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 iterations 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 [3, 6, 9].

We studied the probability distribution of random variable *time to target solution* on several variants of GRASP with path relinking and concluded that these times fit 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 approximately achieve linear speedup.

ACKNOWLEDGMENT

R.M. Aiex was supported by the Brazilian Council for Scientific and Technological Development (CNPq) and was done while this author was visiting AT&T Labs Research. G. Toraldo was supported by the MURST national project “Algorithms for complex systems optimization.” 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

- [1] R.K. Ahuja, J.B. Orlin, and A. Tiwari. A greedy genetic algorithm for the quadratic assignment problem. *Computers and Operations Research*, 27:917–934, 2000.
- [2] R.M. Aiex, M.G.C. Resende, and C.C. Ribeiro. Probability distribution of solution time in GRASP: An experimental investigation. Technical report, AT&T Labs Research, Florham Park, NJ 07733, 2000.
- [3] E. Balas and M.J. Saltzman. An algorithm for the three-index assignment problem. *Oper. Res.*, 39:150–161, 1991.
- [4] R.E. Burkard and K. Fröhlich. Some remarks on 3-dimensional assignment problems. *Methods Oper. Res.*, 36:31–36, 1980.
- [5] R.E. Burkard and R. Rudolf. Computational investigations on 3-dimensional axial assignment problems. *Belgian J. Oper. Res. Statist. Comput. Sci.*, 32:85–98, 1993.
- [6] R.E. Burkard, R. Rudolf, and G.J. Woeginger. Three-dimensional axial assignment problems with decomposable cost coefficients. *Discrete Applied Mathematics*, 65:123–139, 1996.
- [7] J. M. Chambers, W. S. Cleveland, B. Kleiner, and P. A. Tukey. *Graphical Methods for Data Analysis*. Chapman & Hall, 1983.

- [8] Y. Crama, A.W.J. Kolen, A.G. Oerlemans, and F.C.R. Spieksma. Throughput rate optimization in the automated assembly of printed circuit boards. *Annals of Operations Research*, 26:455–480, 1990.
- [9] Y. Crama and F.C.R. Spieksma. Approximation algorithms for three-dimensional assignment problems with triangle inequalities. *European Journal of Operational Research*, 60:273–279, 1992.
- [10] T.A. Feo and J.L. González-Velarde. The intermodal trailer assignment problem: Models, Algorithms, and Heuristics. *Transportation Science*, 29:330–341, 1995.
- [11] T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.
- [12] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.
- [13] P. Festa and M.G.C. Resende. GRASP: An annotated bibliography. Technical report, AT&T Labs Research, Florham Park, NJ 07733, 2000.
- [14] C. Fleurent and F. Glover. Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. *INFORMS Journal on Computing*, 11:198–204, 1999.
- [15] A.M. Frieze. Complexity of a 3-dimensional assignment problem. *European Journal of Operational Research*, 13:161–164, 1983.
- [16] A.M. Frieze and J. Yadegar. An algorithm for solving 3-dimensional assignment problems with application to scheduling a teaching practice. *Journal of Operational Research Society*, 32:989–995, 1981.
- [17] K. Fröhlich. Dreidimensionale Zuordnungsprobleme. Master’s thesis, Math. Institut, Universität Köln, 1979.
- [18] M.R. Garey and D.S. Johnson. *Computers and intractability - A guide to the theory of NP-completeness*. W.H. Freeman and Company, 1979.
- [19] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.
- [20] F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. Technical report, Graduate School of Business and Administration, University of Colorado, Boulder, CO 80309-0419, 2000.
- [21] P. Hansen and L. Kaufman. A primal-dual algorithm for the three-dimensional assignment problem. *Cahiers du CERO*, 15:327–336, 1973.
- [22] J.P. Hart and A.W. Shogan. Semi-greedy heuristics: An empirical study. *Operations Research Letters*, 6:107–114, 1987.
- [23] M. Laguna and R. Martí. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS J. on Computing*, 11:44–52, 1999.
- [24] O. Leue. Methoden zur Lösung dreidimensionaler Zuordnungsprobleme. *Angewandte Informatik*, 14:154–162, 1972.
- [25] Y. Li, P.M. Pardalos, and M.G.C. Resende. A greedy randomized adaptive search procedure for the quadratic assignment problem. In P.M. Pardalos and H. Wolkowicz, editors, *Quadratic assignment and related problems*, volume 16 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 237–261. American Mathematical Society, 1994.
- [26] T. Mavridou, P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A GRASP for the bi-quadratic assignment problem. *European J. of Operational Research*, 105:613–621, 1998.
- [27] R.A. Murphey, P.M. Pardalos, and L.S. Pitsoulis. A parallel GRASP for the data association multidimensional assignment problem. In P.M. Pardalos, editor, *Parallel processing of discrete problems*, volume 106 of *The IMA Volumes in Mathematics and Its Applications*, pages 159–180. Springer-Verlag, 1998.
- [28] P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A parallel GRASP implementation for the quadratic assignment problem. In A. Ferreira and J. Rolim, editors, *Parallel Algorithms for Irregularly Structured Problems – Irregular’94*, pages 115–130. Kluwer Academic Publishers, 1995.
- [29] P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. Algorithm 769: Fortran subroutines for approximate solution of sparse quadratic assignment problems using GRASP. *ACM Transactions on Mathematical Software*, 23:196–208, 1997.
- [30] W.P. Pierskalla. The multidimensional assignment problem. *Opera J.*, 5:71–81, 1967.
- [31] W.P. Pierskalla. The tri-substitution method for the three-multidimensional assignment problem. *CORS J.*, 5:71–81, 1967.

- [32] L.S. Pitsoulis. *Algorithms for nonlinear assignment problems*. PhD thesis, Department of Industrial and Systems Engineering, University of Florida, 1999.
- [33] L.S. Pitsoulis, P.M. Pardalos, and D.W. Hearn. Approximate solutions to the turbine balancing problem. Technical report, Center for Applied Optimization, Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL, 32611 USA, 1998.
- [34] M.C. Rangel, N.M.M. Abreu, and P.O. Boaventura Netto. GRASP in the QAP: An acceptance bound for initial solution. In *Proc. of the Third Metaheuristics International Conference*, pages 381–386, July 1999.
- [35] M.C. Rangel, N.M.M. de Abreu, P.O. Boaventura Netto, and M.C.S. Boeres. A modified local search for GRASP in the quadratic assignment problem. Technical report, Production Engineering Program, COPPE, Federal University of Rio de Janeiro, Rio de Janeiro, RJ Brazil, 1998.
- [36] M.G.C. Resende, P.M. Pardalos, and Y. Li. Algorithm 754: Fortran subroutines for approximate solution of dense quadratic assignment problems using GRASP. *ACM Transactions on Mathematical Software*, 22:104–118, 1996.
- [37] A.J. Robertson. A set of greedy randomized adaptive local search procedure (GRASP) implementations for the multidimensional assignment problem, 1998.
- [38] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI – The complete reference, Volume 1 – The MPI Core*. The MIT Press, 1998.
- [39] M.G.A. Verhoeven and E.H.L. Aarts. Parallel local search. *J. of Heuristics*, 1:43–66, 1995.
- [40] M. Vlach. Branch and bound method for the three index assignment problem. *Ekonomicko-Matematický Obzor*, 3:181–191, 1967.

(R.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
E-mail address, R.M. Aiex: rma@inf.puc-rio.br

(M.G.C. Resende) INFORMATION SCIENCES RESEARCH, AT&T LABS RESEARCH, 180 PARK AVENUE, ROOM C241, FLORHAM PARK, NJ 07932 USA.
E-mail address, M.G.C. Resende: mgcr@research.att.com

(P.M. Pardalos) DEPARTMENT OF INDUSTRIAL AND SYSTEMS ENGINEERING, UNIVERSITY OF FLORIDA, GAINESVILLE, FL 32611 USA.
E-mail address, P.M. Pardalos: pardalos@ufl.edu

(G. Toraldo) DEPARTMENT OF AGRICULTURAL ENGINEERING AND AGRONOMY, UNIVERSITY OF NAPLES “FEDERICO II” AND CPS (CENTER FOR RESEARCH ON PARALLEL COMPUTING AND SUPERCOMPUTERS), NAPLES, ITALY
E-mail address, G. Toraldo: toraldo@unina.it