PARALLEL STRATEGIES FOR GRASP WITH PATH-RELINKING

RENATA M. AIEX AND MAURICIO G.C. RESENDE

ABSTRACT. A Greedy Randomized Adaptive Search Procedure (GRASP) is a metaheuristic for combinatorial optimization. It usually consists of a construction procedure based on a greedy randomized algorithm and local search. Path-relinking is an intensification strategy that explores trajectories that connect high quality solutions. We analyze two parallel strategies for GRASP with path-relinking and propose a criterion to predict parallel efficiency based on experiments with a sequential implementation of the algorithm. Independent and cooperative parallel strategies are described and implemented for the 3-index assignment problem and the job-shop scheduling problem. The computational results for independent parallel strategies are shown to qualitatively behave as predicted by the criterion.

1. INTRODUCTION

GRASP (Greedy Randomized Adaptive Search Procedure) is a metaheuristic for combinatorial optimization [12, 13, 15, 33]. A GRASP is an iterative process, where each iteration usually consists of two phases: construction and local search. The construction phase builds a feasible solution that is used as the starting solution for local search. The best solution over all GRASP iterations is returned as the result.

In the construction phase, a feasible solution is built, one element at a time. The set of candidate elements is made up of those elements that can be added to the current solution under construction without causing infeasibilities. A candidate element is evaluated by a greedy function that measures the local benefit of including that element in the partially constructed solution. The value-based restricted candidate list (RCL) is made up of candidate elements having a greedy function value above a specified threshold. The next element to be included in the solution is selected at random from the RCL. Its inclusion in the solution alters the greedy functions and the set of candidate elements used to determine the next RCL. The construction procedure terminates when the set of candidate elements is empty.

A local search algorithm successively replaces the current solution by a better solution in its neighborhood, if one exists. It terminates with a locally optimal solution when there is no better solution in the neighborhood. Since the solutions generated by a GRASP construction phase are usually sub-optimal, local search almost always improves the constructed solution.

GRASP has been used to find quality solutions to a wide range of combinatorial optimization problems [15]. Furthermore, many extensions and improvements have been proposed for GRASP. Many of these extensions consist in the hybridization of the method with other methaheuristics. We observe that hybrid strategies usually find better solutions

Date: October 27, 2003.

Key words and phrases. Combinatorial optimization, job shop scheduling, 3-index assignment, local search, GRASP, path-relinking, parallel algorithm.

AT&T Labs Research Technical Report TD-5SQKM9.

than those obtained using the pure GRASP, having in some cases improved the solution for open problems in the literature [35, 36, 38].

Hybrid strategies of GRASP with path-relinking have been leading to significant improvements in solution quality when compared to the solutions obtained by the pure method. Path-relinking was originally proposed by Glover [18] as an intensification strategy exploring trajectories connecting elite solutions obtained by tabu search or scatter search [19, 20, 21]. The use of path-relinking within a GRASP procedure, as an intensification strategy applied to each locally optimal solution, was first proposed by Laguna and Martí [22]. It was followed by several extensions, improvements, and successful applications [1, 2, 9, 34, 38].

The path-relinking approach consists in exploring trajectories that connect an *initial solution* and a *guiding solution*. This is done by introducing 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 best move is chosen. In GRASP with path-relinking, one of these solutions is obtained by the local search phase of GRASP and the other is chosen among the solutions kept in an elite set of solutions found during the iterations of the algorithm.

Parallel computers have increasingly found their way into metaheuristics [11]. A frequent question raised during the development of a parallel strategy is how the algorithm will behave when implemented in parallel as compared to a sequential implementation. The ideal situation is achieving linear speedup, i.e. when the execution time of the sequential program divided by the execution time of the parallel program running in ρ processors is equal to ρ. Methodologies to help analyze the behavior of parallel strategies are significant in the development of a parallel application.

Most of the parallel implementations of GRASP found in the literature consist in either partitioning the search space or partitioning the GRASP iterations and assigning each partition to a processor. GRASP is applied to each partition in parallel. Examples of these strategies can be found in [4, 5, 14, 24, 25, 26, 27, 28, 29, 30, 32].

For hybrid strategies of GRASP with path-relinking, two general parallelization approaches have been proposed. In the first one, named the *independent approach*, the communication among processors during GRASP iterations is limited to the detection of program termination. In the second approach, called the *cooperative approach*, processors share information on elite solutions visited during GRASP iterations. These strategies are classified according to Verhoeven and Aarts [43] as *multiple independent trajectories* and *multiple cooperative trajectories*, respectively. Examples of parallel GRASP with path-relinking can be found in [1, 2, 9, 37].

In this paper, we analyze two parallel strategies for GRASP with path-relinking and propose a criterion to predict parallel efficiency based on experiments with a sequential implementation of the algorithm. Independent and cooperative parallel strategies are described and implemented for the 3-index assignment problem (AP3) and the job-shop scheduling problem (JSP).

The remainder of this paper is organized as follows. In Section 2, the sequential implementations of GRASP and of path-relinking are described for both the AP3 and the JSP. Section 3 shows how GRASP and path-relinking are combined. The parallel approaches are discussed in Section 4. The computational results are reported and analyzed in Section 5. Concluding remarks are made in Section 6.

2. SEQUENTIAL GRASP

In this section, the GRASP implementations developed for the 3-index assignment problem in Aiex et al. [2] and for the job-shop scheduling problem in Aiex, Binato, and Resende [1] are reviewed in Subsections 2.1 and 2.2, respectively. Path-relinking is generalized for these problems in Subsection 2.3.

2.1. **GRASP for the AP3.**

The three-index assignment problem (AP3) was first stated by Pierskalla [31] as a straightforward extension of the classical two-dimensional assignment problem. The AP3 is NP-Hard [16, 17].

A 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 AP3 can also be 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 $1, 2, \ldots, n$. None of the chosen triplets (i, j, k) is allowed to have the same value for indices i, j, and k as another. For example, the choice of triplets $(1, 2, 4)$ and $(3, 2, 5)$ is infeasible, since these triplets 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 $N = \{1, 2, \ldots, n\}$.

2.1.1. *GRASP for AP3 – Construction phase.*

The GRASP construction phase builds a feasible solution S by selecting n triplets, one at a time.

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

To select the p-th ($1 \leq p \leq n-1$) triplet to be added to the solution, a restricted candidate list C' is defined to include all triplets (i, j, k) in the candidate set C having cost $c_{ijk} \leq \underline{c} + \alpha(\overline{c} - \underline{c})$, where

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

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

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 $k = k_p$ must be removed from the current set of candidate triplets. 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. In Aiex et al. [2], four doubly linked lists are used to implement this process more efficiently, reducing the complexity from $O(n^4)$ to $O(n^3)$.

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

2.1.2. *GRASP for AP3 – 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

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

The definition of the neighborhood $N(s)$ is crucial for the performance of the local search. In the 2-exchange neighborhood scheme used in this local search, 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.

2.2. **GRASP for the JSP.** The job shop scheduling problem (JSP) is a well-studied problem in combinatorial optimization. It consists in processing a finite set of jobs on a finite set of machines. Each job is required to complete a set of operations in a fixed order. Each operation is processed on a specific machine for a fixed duration. Each machine can process at most one job at a time and once a job initiates processing on a given machine it must complete processing on that machine without interruption. A schedule is a mapping of operations to time slots on the machines. The makespan is the maximum completion time of the jobs. The objective of the JSP is to find a schedule that minimizes the makespan. The JSP is NP-hard [23] and has also proven to be computationally challenging.

Mathematically, the JSP can be stated as follows. Given a set $\mathcal{M}$ of machines (where we denote the size of $\mathcal{M}$ by $|\mathcal{M}|$) and a set $\mathcal{J}$ of jobs (where the size of $\mathcal{J}$ is denoted by $|\mathcal{J}|$), let $\sigma_1^j \prec \sigma_2^j \prec \cdots \prec \sigma_{|\mathcal{M}|}^j$ be the ordered set of $|\mathcal{M}|$ operations of job j, where $\sigma_k^j \prec \sigma_{k+1}^j$ indicates that operation σ_{k+1}^j can only start processing after the completion of operation σ_k^j. Let O be the set of operations. Each operation σ_k^j is defined by two parameters: $\mathcal{M}_k^j$ is the machine on which σ_k^j is processed and $p_k^j = p(\sigma_k^j)$ is the processing time of operation σ_k^j. Defining $t(\sigma_k^j)$ to be the starting time of the k-th operation $\sigma_k^j \in O$, the JSP can be formulated as follows:

$$\begin{aligned}
&\text{minimize } C_{\max} \\
&\text{subject to: } C_{\max} \geq t(\sigma_k^j) + p(\sigma_k^j), \text{ for all } \sigma_k^j \in O, \\
&\qquad t(\sigma_k^j) \geq t(\sigma_l^j) + p(\sigma_l^j), \text{ for all } \sigma_l^j \prec \sigma_k^j, && \text{(1a)} \\
&\qquad t(\sigma_k^j) \geq t(\sigma_l^i) + p(\sigma_l^i) \vee && \text{(1b)} \\
&\qquad t(\sigma_l^i) \geq t(\sigma_k^j) + p(\sigma_k^j), \text{ for all } \sigma_l^i, \sigma_k^j \in O \text{ such that } \mathcal{M}_{\sigma_l^i} = \mathcal{M}_{\sigma_k^j}, \\
&\qquad t(\sigma_k^j) \geq 0, \text{ for all } \sigma_k^j \in O,
\end{aligned}$$

where C_{max} is the makespan to be minimized.

A feasible solution of the JSP can be built from a permutation of $\mathcal{J}$ on each of the machines in $\mathcal{M}$, observing the precedence constraints, the restriction that a machine can process only one operation at a time, and requiring that once started, processing of an operation must be uninterrupted until its completion. Since each set of feasible permutations has a corresponding schedule, the objective of the JSP is to find, among the feasible permutations, the one with the smallest makespan.

2.2.1. *GRASP for JSP – Construction phase.* Consider the GRASP construction phase for JSP, proposed in Binato et al. [8] and Aiex, Binato, and Resende [1], where a single operation is the building block of the construction phase. That is, a feasible schedule is built by scheduling individual operations, one at a time, until all operations have been scheduled.

Recall that σ_k^j denotes the k-th operation of job j and is defined by the pair $(\mathcal{M}_k^j, p_k^j)$, where $\mathcal{M}_k^j$ is the machine on which operation σ_k^j is performed and p_k^j is the processing time of operation σ_k^j. While constructing a feasible schedule, not all operations can be selected at a given stage of the construction. An operation σ_k^j can only be scheduled if all prior operations of job j have already been scheduled. Therefore, at each construction phase iteration, at most $|\mathcal{J}|$ operations are candidates to be scheduled. Let this set of candidate operations be denoted by O_c and the set of already scheduled operations by O_s and denote the value of the greedy function for candidate operation σ_k^j by $h(\sigma_k^j)$.

The greedy choice is to next schedule operation $\underline{\sigma_k^j} = \text{argmin}(h(\sigma_k^j) \mid \sigma_k^j \in O_c)$. Let $\overline{\sigma_k^j} = \text{argmax}(h(\sigma_k^j) \mid \sigma_k^j \in O_c)$, $\underline{h} = h(\underline{\sigma_k^j})$, and $\overline{h} = h(\overline{\sigma_k^j})$. Then, the GRASP restricted candidate list (RCL) is defined as

$$\text{RCL} = \{\sigma_k^j \in O_c \mid \underline{h} \leq h(\sigma_k^j) \leq \underline{h} + \alpha(\overline{h} - \underline{h})\},$$

where α is a parameter such that $0 \leq \alpha \leq 1$.

A typical iteration of the GRASP construction is summarized as follows: a partial schedule (which is initially empty) is on hand, the next operation to be scheduled is selected from the RCL and is added to the partial schedule, resulting in a new partial schedule. The selected operation is inserted in the earliest available feasible time slot on machine $\mathcal{M}_{\sigma_k^j}$. Construction ends when the partial schedule is complete, i.e. all operations have been scheduled.

The algorithm uses two greedy functions. Even numbered iterations use a greedy function based on the makespan resulting from the inclusion of operation σ_k^j to the already-scheduled operations, i.e. $h(\sigma_k^j) = \mathcal{C}_{max}$ for $O = \{O_s \cup \sigma_k^j\}$. In odd numbered iterations, solutions are constructed by favoring operations from jobs having long remaining processing times. The greedy function used is given by $h(\sigma_k^j) = -\sum_{\sigma_l^j \notin O_s} p_l^j$, which measures the remaining processing time for job j. The use of two different greedy functions produce a greater diversity of initial solutions to be used by the local search.

2.2.2. *GRASP for JSP – Local search phase.* To attempt to decrease the makespan of the solution produced in the construction phase, we employ the 2-exchange local search used in [1, 8, 42], that is based on the disjunctive graph model of Roy and Sussmann [39].

The disjunctive graph $G = (V, A, E)$ is defined such that

$$V = \{O \cup \{0, |\mathcal{J}| \cdot |\mathcal{M}| + 1\}\}$$

is the set of nodes, where $\{0\}$ and $\{|\mathcal{J}|\cdot|\mathcal{M}|+1\}$ are artificial source and sink nodes, respectively,

$$A = \{(v,w) \mid v,w \in O, v \prec w\} \cup \{(0,w) \mid w \in O, \nexists v \in O \ni v \prec w\} \cup \{(v,|\mathcal{J}|\cdot|\mathcal{M}|+1) \mid v \in O, \nexists w \in O \ni v \prec w\}$$

is the set of directed arcs connecting consecutive operations of the same job, and

$$E = \{(v,w) \mid \mathcal{M}_v = \mathcal{M}_w\}$$

is the set of edges that connect operations on the same machine. Vertices in the disjunctive graph model are weighted. Vertices 0 and $|\mathcal{J}|\cdot|\mathcal{M}|+1$ have weight zero, while the weight of vertex $i \in \{1,\ldots,|\mathcal{J}|\cdot|\mathcal{M}|\}$ is the processing time of the operation corresponding to vertex i. Notice that the edges of A and E correspond, respectively, to constraints (1a) and (1b) of the disjunctive programming formulation of the JSP.

An orientation for the edges in E corresponds to a feasible schedule. Given an orientation of E, one can compute the earliest start time of each operation by computing the longest (weighted) path from node 0 to the node corresponding to the operation. Consequently, the makespan of the schedule can be computed by finding the critical (longest) path from node 0 to node $|\mathcal{J}|\cdot|\mathcal{M}|+1$. The objective of the JSP is to find an orientation of E such that the longest path in G is minimized.

Taillard [41] describes an $O(|\mathcal{J}|\cdot|\mathcal{M}|)$ algorithm to compute the longest path on G and an $O(|\mathcal{J}|\cdot|\mathcal{M}|)$ procedure to recompute the makespan when two consecutive operations in the critical path (on the same machine) are swapped. He also shows that the entire neighborhood of a given schedule, where the neighborhood is defined by the swap of two consecutive operations in the critical path, can be examined, i.e. have their makespan computed, in time $O(|\mathcal{J}|\cdot|\mathcal{M}|)$ given that the longest path of G was evaluated.

Given the schedule produced in the construction phase, the local search procedure initially identifies the critical path in the disjunctive graph corresponding to that schedule. All pairs of consecutive operations sharing the same machine in the critical path are tentatively exchanged. If the exchange improves the makespan, the move is accepted. Otherwise, the exchange is undone. Once an exchange is accepted, the critical path may change and a new critical path must be identified. If no pairwise exchange of consecutive operations in the critical path improves the makespan, the current schedule is locally optimal and the local search ends.

2.3. **Path-relinking.** Using permutation arrays, we generalize path-relinking in this subsection for both the AP3 and the JSP.

A solution of the AP3 can be represented by two permutation arrays of numbers $1,2,\ldots,n$ in sets J and K, respectively, as follows:

$$S = \{(j^S_{1,1}, j^S_{1,2}, \ldots, j^S_{1,n}), (j^S_{2,1}, j^S_{2,2}, \ldots, j^S_{2,n})\},$$

where $j^S_{i,k}$ is the k-th number assigned to permutation i in solution S.

Analogously, for the JSP, a schedule can be represented by the permutation of operations in $\mathcal{J}$ on the machines in $\mathcal{M}$. The schedule is represented by $|\mathcal{M}|$ permutation arrays, each with $|\mathcal{J}|$ operations. Each permutation implies an ordering of the operations. A solution of the JSP is represented as follows:

$$S = \{(j^S_{1,1}, j^S_{1,2}, \ldots, j^S_{1,|\mathcal{J}|}), (j^S_{2,1}, j^S_{2,2}, \ldots, j^S_{2,|\mathcal{J}|}), \ldots, (j^S_{|\mathcal{M}|,1}, j^S_{|\mathcal{M}|,2}, \ldots, j^S_{|\mathcal{M}|,|\mathcal{J}|})\},$$

where $j^S_{i,k}$ is the k-th operation executed on machine i in solution S.

A path-relinking strategy for permutation arrays is carried out as follows. For a problem represented with R permutation arrays of elements from a set E, path-relinking is done between an initial solution

$$S = \{(j^S_{1,1}, j^S_{1,2}, \ldots, j^S_{1,|E|}), (j^S_{2,1}, j^S_{2,2}, \ldots, j^S_{2,|E|}), \ldots, (j^S_{R,1}, j^S_{R,2}, \ldots, j^S_{R,|E|})\}$$

and a guiding solution

$$T = \{(j^T_{1,1}, j^T_{1,2}, \ldots, j^T_{1,|E|}), (j^T_{2,1}, j^T_{2,2}, \ldots, j^T_{2,|E|}), \ldots, (j^T_{R,1}, j^T_{R,2}, \ldots, j^T_{R,|E|})\},$$

where $j^{S'}_{i,k}$ is the k-th element of permutation i in solution S'.

Let the difference between S and T be defi ned by the R sets of indices

$$\delta^{S,T}_k = \{i = 1, \ldots, |E| \mid j^S_{k,i} \neq j^T_{k,i}\}, k = 1, \ldots, R.$$

During a path-relinking move, a permutation array in S, given by

$$(\ldots, j^S_{k,i}, j^S_{k,i+1}, \ldots, j^S_{k,q-1}, j^S_{k,q}, \ldots),$$

is replaced by a permutation array

$$(\ldots, j^S_{k,q}, j^S_{k,i+1}, \ldots, j^S_{k,q-1}, j^S_{k,i}, \ldots),$$

by exchanging permutation elements $j^S_{k,i}$ and $j^S_{k,q}$, where $i \in \delta^{S,T}_k$ and q are such that $j^T_{k,q} = j^S_{k,i}$.

At each step of the algorithm, the move that produces the lowest cost solution is selected and its index is removed from the corresponding set $\delta^{S,T}_k$. This continues until there are only two move indices left in one of the sets $\delta^{S,T}_k$. At this point, the move obtained by exchanging these elements will produce the guiding solution. The best solution found during the path traversal is returned by the procedure.

3. GRASP WITH PATH-RELINKING

This section describes how path-relinking and GRASP can be combined to form a hybrid GRASP with path-relinking. We limit this discussion to single processor implementations and consider parallel strategies in the next section. Pseudo-code for the GRASP with path-relinking is presented in Figure 1. Let $|P|$ be the size of the current elite set and let *maxpool* be the the elite set's maximum size. The fi rst *maxpool* GRASP iterations contribute one solution to the elite set per GRASP iteration (line 16). Path-relinking is not done until the pool of elite solutions is full.

In lines 3 and 4, GRASP construction and local search phases are carried out. Each of these phases can be replaced by more elaborated mechanisms, as for example, the construction phase developed for the JSP that alternates between two different greedy functions.

Once the pool of elite solutions is full, solution S produced by the local search phase of GRASP is tested to verify its quality (line 6). This is done to avoid relinking low-quality solutions. If S passes the quality test used, bidirectional path-relinking is done between S and all elements of a subset $P' \subseteq P$ (lines 7 to 15). In bidirectional path-relinking [2], two paths are analyzed: one from the GRASP solution to the selected solution from the pool; another from the selected pool solution to the GRASP solution. The degree of restrictiveness of the quality test is a function of the computational time necessary to do path-relinking. For the AP3, the cost of a solution in the neighborhood of S can be computed from the cost of S in $O(1)$ time and therefore, path-relinking is applied to all solutions obtained in the local search phase of GRASP. For the JSP, on the other hand, the cost of each solution visited by path-relinking is computed in $O(|\mathcal{J}| \cdot |\mathcal{M}|)$ using the

```
procedure GRASP_PR(seed, look4, maxitr, maxpool, ifreq, problem_data)
1     P = ∅;
2     for i = 1,...,maxitr do
3           CONSTRUCTION(seed, S, problem_data);
4           LOCAL(S, problem_data);
5           if |P| == maxpool then
6                 accepted = VERIFY_QUALITY(S);
7                 if accepted then
8                       select P' ⊆ P;
9                       for T ∈ P' do
10                            S_gmin = PATH_RELINKING(c_S, S, T, problem_data);
11                            UPDATE_POOL(S_gmin, c_gmin, P);
12                            S_gmin = PATH_RELINKING(c_T, T, S, problem_data);
13                            UPDATE_POOL(S_gmin, c_gmin, P);
14                      rof;
15                fi;
16          else P = P ∪ {S} fi;
17          if  mod (i, ifreq) == 0 then
18                INTENSIFY(P);
19          fi;
20          S_best = POOLMIN(P);
21          if c_best ≤ look4 then
22                break;
23          fi;
24    rof;
25    POSTOPT(P);
26    S_best = POOLMIN(P);
27    return (S_best);
end GRASP_PR;
```

FIGURE 1. GRASP with path-relinking.

algorithm proposed in Taillard [41]. Therefore, it is computationally expensive to apply path-relinking after each GRASP iteration, and path-relinking is applied only when the GRASP solution satisfies a given quality criterion [1]. After each path-relinking phase, the best solution traversed during path-relinking is tested for inclusion in the elite pool (lines 11 and 13).

Every *ifreq* GRASP iterations, an intensification procedure is carried out (lines 17 to 19). The intensification procedure [2] is accomplished by applying path-relinking to each pair of elite solutions in P and updating the pool when necessary. The procedure is repeated until no further change in P occurs.

The GRASP with path-relinking loop from line 2 to 24 continues for at most *maxitr* iterations, but can be terminated when a solution having a cost less than or equal to $look4$ is found (lines 21 to 23).

Finally, a path-relinking post optimization is done on the elite set (line 25). Path-relinking as a post-optimization step was introduced in Aiex et al. [2] and Ribeiro, Uchoa, and Werneck [38] and has been also used in Resende and Werneck [35, 36]. After applying path-relinking between all pairs of elite solutions without any change in the elite set, the local search procedure 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. PARALLEL GRASP STRATEGIES

In this section, two parallel strategies for the GRASP with path-relinking algorithm shown in Figure 1 are described. The first scheme, called *independent*, limits communication between processors only to problem input, detection of process termination, and determination of best overall solution. In addition to the communication allowed in the independent scheme, the second scheme, called *cooperative*, allows processes to exchange information about their elite sets.

4.1. **Independent parallel strategy.**

We revisit a basic parallelization scheme for GRASP with path-relinking proposed in Aiex et al. [2]. Figure 2 shows pseudo-code for this *multiple independent walks* scheme.

Our implementations use message passing for communication among processors. This communication is limited to program initialization and termination. When ρ processors are used, a single process reads the problem data and passes it to the remaining $\rho - 1$ processes. Processes send a message to all others when they either stop upon finding a solution at least as good as the target value $look4$ or complete the maximum number of allotted iterations.

The independent parallel GRASP with path-relinking is built upon the sequential algorithm of Figure 1. Each process executes a copy of the program. We discuss the differences between the sequential algorithm and this parallel variant. In line 1 of Figure 2, the rank *my_rank* of the process and the number ρ of processes are determined. Each GRASP construction phase is initialized with a random number generator seed. To assure independence of processes, identical seeds of the random number generator (`rand()`) must not be used by more than one process to initiate a construction phase. The initial seed for process *my_rank* is computed in lines 2 to 4. We attempt, this way, to increase the likelihood that each process has a disjunct sequence of $maxitr/\rho$ initial seeds.

The **for** loop from line 6 to line 36 executes the iterations. The construction, local search, and path-relinking phases are identical to the those of the sequential algorithm. In line 23, if a process finds a solution with cost less than or equal to $look4$, it sends a flag to each of the other processes indicating that it has found the solution. Likewise, when a process completes $maxitr/\rho$ iterations, it sends a different flag to each of the other processes indicating that it has completed the preset number of iterations (lines 24 to 27). In line 28, the process checks if there are any status flags to be received. If there is a flag indicating that a solution with cost not greater than $look4$ has been found, the iterations are terminated in line 30. If a flag indicating that some process has completed the preset number of iterations has been received, then a counter num_proc_stop of the number of processes that are ready to be terminated is incremented (line 32). If all processes have completed their iterations, the execution of the main **for** loop is terminated (line 35).

Each process, upon terminating the **for** loop going from line 6 to line 36, runs the post-optimization phase on the pool of elite solutions (line 37). A reduce operator (`GET_GLOBAL_BEST`) determines the global best solution among all processes in line 38.

A pure GRASP parallel algorithm can be obtained from the algorithm in Figure 2 by skipping the execution of lines 9 to 19. As in a basic GRASP, it is necessary to keep track of the best solution found and no pool handling operations are necessary. Therefore, intensification and post-optimization are not defined in a parallel implementation of pure GRASP.

4.2. **Cooperative parallel strategy.**

In the cooperative parallel GRASP with path-relinking, processes share elite set information. We now describe this scheme, whose pseudo-code is presented in Figure 3. This algorithm is built on top of the independent scheme presented

```
procedure INDEPENDENT_GRASP_PR(seed, look4, maxitr, maxpool, ifreq, problem_data)
1   my_rank = GET_RANK(); ρ = GET_NUM_PROCS();
2   for i = 1,...,(maxitr/ρ) * my_rank do
3       seed = rand(seed);
4   rof;
5   P = ∅; num_proc_stop = 0;
6   for i = 1,...,∞ do
7       CONSTRUCTION(seed, S, problem_data);
8       LOCAL(S, problem_data);
9       if |P| == maxpool then
10          accepted = VERIFY_QUALITY(S);
11          if accepted then
12              select P' ⊆ P;
13              for T ∈ P' do
14                  S_gmin = PATH_RELINKING(c_S, S, T, problem_data);
15                  UPDATE_POOL(S_gmin, c_gmin, P);
16                  S_gmin = PATH_RELINKING(c_T, T, S, problem_data);
17                  UPDATE_POOL(S_gmin, c_gmin, P);
18              rof;
19          fi;
20      else P = P ∪ {S} fi;
21      if  mod (i,ifreq) == 0 then INTENSIFY(P); fi;
22      S_best = POOLMIN(P);
23      if c_best ≤ look4 then SEND_ALL(look4_stop) fi;
24      if i == maxitr/ρ then
25          num_proc_stop = num_proc_stop + 1;
26          SEND_ALL(maxitr_stop);
27      fi;
28      received = VERIFY_RECEIVING(flag);
29      if received then
30          if flag == look4_stop then break;
31          else if flag == maxitr_stop then
32              num_proc_stop = num_proc_stop + 1;
33          fi;
34      fi;
35      if num_proc_stop == ρ then break fi;
36  rof;
37  POSTOPT(P);
38  S_GlobalBest = GET_GLOBAL_BEST(S_best);
39  return (S_GlobalBest);
end INDEPENDENT_GRASP_PR;
```

FIGURE 2. Pseudo-code for the independent parallel GRASP with path-relinking.

in the previous subsection. We limit our discussion to the differences between the two schemes, which occur in the path-relinking phase.

Before doing path-relinking between solutions S and T, each process checks if one or more other processes have sent it new elite solutions. If there are new elite solutions to be received, RECEIVE_SOLUTIONS (in lines 14 and 18) receives the elite solutions, tests if each elite solution can be accepted for insertion into its local elite set, and inserts any accepted elite solution. Upon termination of each path-relinking leg, if the local elite set is updated, then (in lines 17 and 21) the process writes the new elite solutions to a local send buffer. In line 23, if the local send buffer is not empty, the process sends the buffer's contents to the other processes.

```
procedure COOPERATIVE_GRASP_PR(seed, look4, maxitr, maxpool, ifreq, problem_data)
1  my_rank = GET_RANK(); ρ = GET_NUM_PROCS();
2  for i = 1,...,(maxitr/ρ) * my_rank do
3      seed = rand(seed);
4  rof;
5  P = ∅; num_proc_stop = 0;
6  for i = 1,...,∞ do
7      CONSTRUCTION(seed, S, problem_data);
8      LOCAL(S, problem_data);
9      if |P| == maxpool then
10         accepted = VERIFY_QUALITY(S);
11         if accepted then
12             select P' ⊆ P;
13             for T ∈ P' do
14                 RECEIVE_SOLUTIONS(P);
15                 S_gmin = PATH_RELINKING(c_S, S, T, problem_data);
16                 updated = UPDATE_POOL(S_gmin, c_gmin, P);
17                 if (updated) then INSERT_SEND_BUFFER(S_gmin, c_gmin, buffer) fi;
18                 RECEIVE_SOLUTIONS(P);
19                 S_gmin = PATH_RELINKING(c_T, T, S, problem_data);
20                 updated = UPDATE_POOL(S_gmin, c_gmin, P);
21                 if (updated) then INSERT_SEND_BUFFER(S_gmin, c_gmin, buffer) fi;
22             rof;
23             SEND_SOLUTIONS(buffer);
24         fi;
25     else P = P ∪ {S} fi;
26     if  mod (i,ifreq) == 0 then INTENSIFY(P) fi;
27     S_best = POOLMIN(P);
28     if c_best ≤ look4 then SEND_ALL(look4_stop) fi;
29     if i == maxitr/ρ then
30         num_proc_stop = num_proc_stop + 1;
31         SEND_ALL(maxitr_stop)
32     fi;
33     received = VERIFY_RECEIVING(flag);
34     if received then
35         if flag == look4_stop then break;
36         else if flag == maxitr_stop then
37             num_proc_stop = num_proc_stop + 1;
38         fi;
39     fi;
40     if num_proc_stop == ρ then break fi;
41 rof;
42 POSTOPT(P);
43 S_GlobalBest = GET_GLOBAL_BEST(S_best);
44 return (S_GlobalBest);
end COOPERATIVE_GRASP_PR;
```

FIGURE 3. Pseudo-code for the cooperative parallel GRASP with path-relinking.

Another difference between the independent and the cooperative schemes concerns the INTENSIFY procedure. In the cooperative scheme, whenever the local elite set pool is updated, the new elite solutions are written to the send buffer. These bufferized solutions will be sent to the other processes the next time that procedure SEND_SOLUTIONS is invoked.

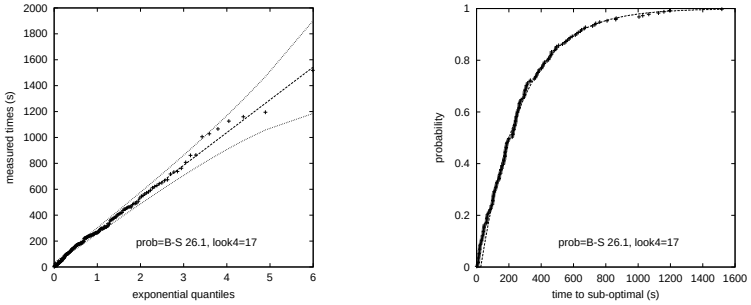

FIGURE 4. Exponential distribution and Q-Q plot for GRASP: problem B-S 26.1.

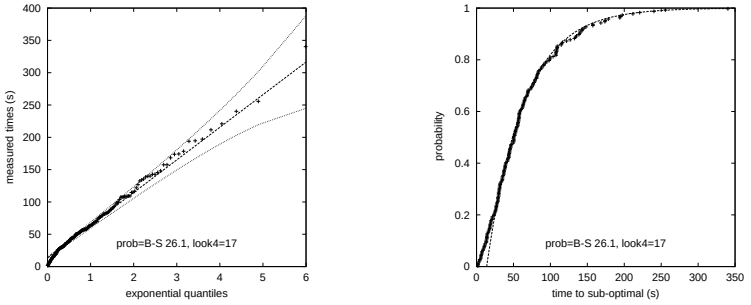

FIGURE 5. Exponential distribution and Q-Q plot for GRASP with path-relinking: problem B-S 26.1.

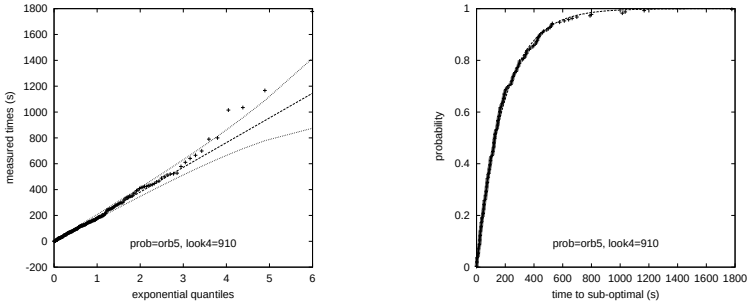

FIGURE 6. Exponential distribution and Q-Q plot for GRASP with path-relinking: problem `orb5`.

5. COMPUTATIONAL RESULTS

This section reports on computational experiments with the parallel versions of the pure GRASP and GRASP with path-relinking proposed in Section 4. The parallel strategies have been implemented for the both the AP3 and JSP described in Section 2.

5.1. **Computer environment.** The experiments were done on an SGI Challenge computer (16 196-MHz MIPS R10000 processors and 12 194-MHz MIPS R10000 processors) with 7.6 Gb of memory. The algorithms were coded in Fortran and were compiled with the SGI MIPSpro F77 compiler using flags `-O3 -static -u`. The parallel codes used SGI's

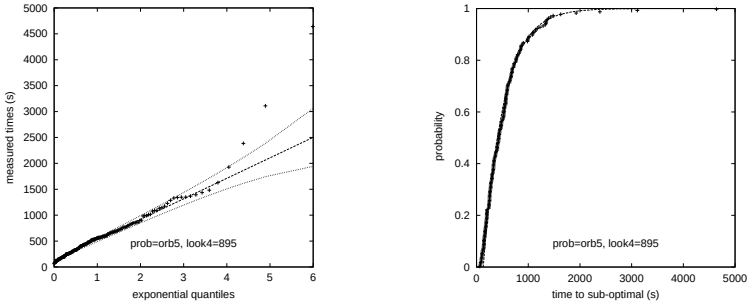

FIGURE 7. Exponential distribution and Q-Q plot for GRASP with path-relinking: problem orb5.

Message Passing Toolkit 1.4, which contains a fully compliant implementation of version 1.2 of the Message-Passing Interface (MPI) [40] specification. In the parallel experiments, times measured were wall clock times, and were done with the MPI function MPI_WT. This is also the case for runs with a single processor that are compared to multiple-processor runs. Timing in the parallel runs excludes the time to read the problem data, to initialize the random number generator seeds, and to output the solution.

The parallel implementations were run on 2, 4, 8, and 16 processors. Load on the machine was low throughout the experiments, therefore processors were always available. The average speedups (execution time of the sequential program divided by the execution time of the parallel program) were computed dividing the sum of the execution times of the independent parallel program executing on one processor by the sum of the execution times of the parallel program on 2, 4, 8, and 16 processors, for 60 runs. The execution times of the independent parallel program executing on one processor and the execution times of the sequential program are approximately the same.

5.2. **The parallel GRASP strategies.** The following parallel algorithms were studied in these experiments:

(1) pure GRASP;
(2) independent GRASP with path-relinking;
(3) cooperative GRASP with path-relinking.

Path-relinking was always applied between the GRASP solution and all solutions in the elite set.

The parallel GRASP as well as the parallel GRASP with path-relinking used in the experiments are named GRASP(*prob*) and GRASP + PR(*prob*), where *prob* indicates the problem type (AP3 or JSP). The parameters of the procedures used in the parallel approaches of GRASP for the AP3 and for the JSP are the same used for testing the sequential algorithms in Aiex et al. [2] and Aiex, Binato, and Resende [1], respectively. Intensification and post-optimization are not carried out during the experiments with the parallel implementations.

5.3. **Test problems.** For the AP3, we tested one problem of each size $n = 20, 22, 24, 26$, generated by Balas and Saltzman [6]. We named these problems B-S 20.1, B-S 22.1, B-S 24.1 and B-S 26.1. For the JSP we tested problems abz6, mt10, orb5, and la21 from four classes of standard problems for the JSP. These problems were obtained from Beasley's OR-Library [1] [7].

[1] http://mscmga.ms.ic.ac.uk/jeb/orlib/jobshopinfo.html

5.4. **Probability distribution of solution time to target value.** Aiex, Resende, and Ribeiro [3] studied the empirical probability distributions of the random variable *time to target value* in GRASP. They showed that, given a target solution value, the time GRASP takes to find a solution with cost at least as good as the target fits a two-parameter exponential distribution. Empirical distributions are produced from experimental data and corresponding theoretical distributions are estimated from the empirical distributions.

A quantile-quantile plot (Q-Q plot) and a plot showing the empirical and the theoretical distributions of the random variable time to target value for sequential GRASP and GRASP with path-relinking for AP3 are shown in Figures 4 and 5, respectively. Analogously, Figures 6 and 7 show the same plots for JSP. These plots are computed by running the algorithms for 200 independent runs. Each run ends when the algorithm finds a solution with value less than or equal to a specified target value (*look4*). Each running time is recorded and the times are sorted in increasing order. 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 $i = 1, \ldots, 200$ as the empirical distribution.

Following Chambers et al. [10], we determine the theoretical quantile-quantile plot for the data to estimate the parameters of the two-parameter exponential distribution. 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 and standard deviation of the distribution 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.

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.

When 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 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. These estimates are used to plot the theoretical distributions on the plots on the right side of the figures.

The lines above and below the estimated line on the Q-Q plots correspond to plus and minus one standard deviation in the vertical direction from the line fitted to the plot. This superimposed variability information is used to analyze the straightness of the Q-Q plots.

The following can be stated for a two parameter (shifted) exponential distribution [3, 43]. 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 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, since $\rho \geq 1$, 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 value using multiple independent processes.

5.5. **A test to predict speedup of parallel implementations.** The observation above suggests a test using a sequential implementation to determine whether it is likely that a parallel implementation using multiple independent processors will be efficient. We say a parallel implementation is efficient if it achieves linear speedup (with respect to wall time) to find a solution at least as good as a given target value (*look4*). The test consists in K (200, for example) independent runs of the sequential program to build a Q-Q plot and estimate the parameters μ and λ of the shifted exponential distribution. If $\rho|\mu| \ll \lambda$, then we predict that the parallel implementation will be efficient.

5.6. **The parallel experiments.** The goals of the experiments in this subsection are threefold. First, we attempt to verify computationally the validity of the test to predict speedup of parallel implementations proposed above. Second, we contrast independent parallel implementations of pure GRASP with GRASP with path-relinking. Finally, we compare independent parallel implementations of GRASP with path-relinking with cooperative implementations. Because of the nature of these experiments, the stopping criterion for maximum number of iterations was disabled, and the programs terminated only when a solution with cost as good as the target value was found.

In Aiex et al. [2] and Aiex, Binato, and Resende [1], we verified that the times to target solution in the GRASP variants for the AP3 and the JSP fit a two-parameter exponential distribution.

5.6.1. *Parallel results for the AP3.* To study the parallel implementation of GRASP(*AP3*), we tested problems `B-S 20.1`, `B-S 22.1`, `B-S 24.1`, and `B-S 26.1` with target values 16, 16, 16, and 17, respectively. The independent and cooperative parallel implementations of GRASP + PR(*AP3*) were studied for problems `B-S 20.1`, `B-S 22.1`, `B-S 24.1`, and `B-S 26.1`, with target values 7, 8, 7, and 8, respectively. The speedups for the independent and cooperative parallel approaches of GRASP + PR(*AP3*) are shown in Figure 8. These plots were generated with 60 independent runs for each number of processors used (1, 2, 4, 8, and 16 processors).

The speedups and efficiencies (speedup divided by the number of processors used) observed for GRASP(*AP3*) are shown in Table 1. We also show the estimates for parameters μ

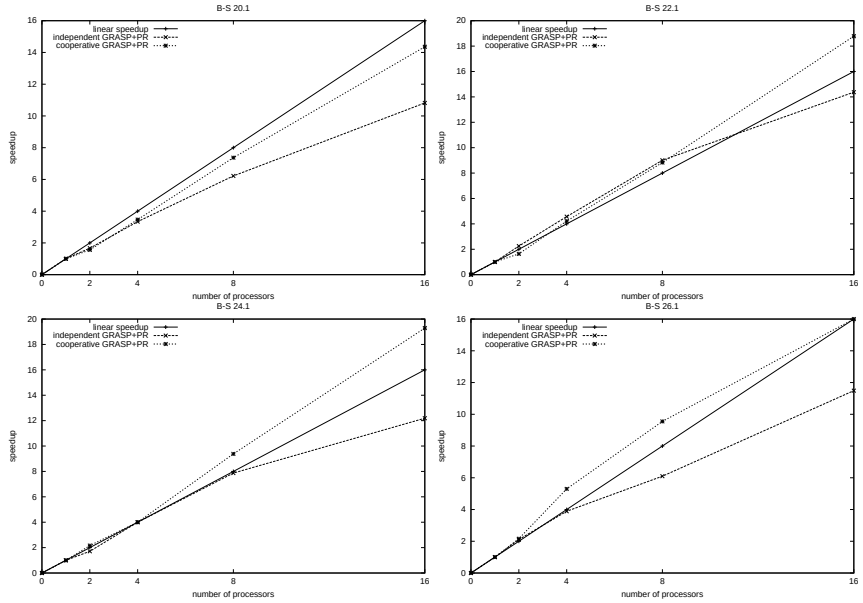

FIGURE 8. Speedups for the parallel implementations of independent and cooperative GRASP with path-relinking for the AP3: problems B-S 20.1 B-S 22.1 B-S 24.1 and B-S 26.1 with target values 7, 8, 7, and 8, respectively.

TABLE 1. Speedup and efficiency for instances of the AP3. Algorithm is the parallel implementation of GRASP. Instances are B-S 20.1, B-S 22.1, B-S 24.1 and B-S 26.1, with target values 16, 16, 16, and 17, respectively. The estimated parameters for the exponential distributions are shown for each pair of instance/target value.

	estimated parameter			number of processors							
				2		4		8		16	
prob.	μ	λ	$\lvert\mu\rvert/\lambda$	spdup	effic.	spdup	effic.	spdup	effic.	spdup	effic.
B-S 20.1	1.28	90.18	.014	2.02	1.01	3.66	.91	6.05	.75	16.30	1.01
B-S 22.1	-2.607	185.21	.014	2.03	1.01	4.58	1.14	10.33	1.29	17.88	1.11
B-S 24.1	-2.890	246.55	.011	2.16	1.08	4.27	1.06	7.89	.98	13.91	.86
B-S 26.1	26.36	252.90	.104	1.62	.81	3.22	.80	6.23	.77	11.72	.73
average:			**.034**	**1.95**	**.97**	**3.93**	**.97**	**7.62**	**.95**	**14.95**	**.93**

and λ of the two-parameter exponential distribution, as well as the value of $|\mu|/\lambda$ for each pair of instance/target value tested.

By examining the parameters estimated for GRASP(*AP3*) in Table 1, we can group the instances into two categories: { B-S 20.1, B-S 22.1, B-S 24.1 } and { B-S 26.1 }. For instances B-S 20.1, B-S 22.1, and B-S 24.1, the ratio $|\mu|/\lambda$ is approximately 0.01, while for runs on problem B-S 26.1, it is about 10 times greater. By our proposed criterion, we would expect that the runs in the first category present better efficiency than the one in the other category, which indeed is what one observes. The super-linear speedups observed in problem B-S 22.1 are probably explained by statistical fluctuation.

TABLE 2. Speedups for instances of the AP3. Algorithms are independent and cooperative implementation GRASP with path-relinking. Instances are B-S 20.1, B-S 22.1, B-S 24.1, and B-S 26.1, with target values 7, 8, 7, and 8, respectively.

	estimated parameter			speedup independent (number of processors)				speedup cooperative (number of processors)			
prob.	μ	λ	$\lvert\mu\rvert/\lambda$	2	4	8	16	2	4	8	16
B-S 20.1	-26.46	1223.80	.021	1.67	3.34	6.22	10.82	1.56	3.47	7.37	14.36
B-S 22.1	-135.12	3085.32	.043	2.25	4.57	9.01	14.37	1.64	4.22	8.83	18.78
B-S 24.1	-16.76	4004.11	.004	1.71	4.00	7.87	12.19	2.16	4.00	9.38	19.29
B-S 26.1	32.12	2255.55	.014	2.11	3.89	6.10	11.49	2.16	5.30	9.55	16.00
average:			**.020**	**1.935**	**3.95**	**7.3**	**12.21**	**1.88**	**4.24**	**8.78**	**17.10**

Some negative estimates for μ observed in Table 1 should be expected whenever the shift in the exponential distribution is small, since μ is the intercept of the ordinate axis of the line defined by the first and third quartiles of the Q-Q plot.

In Table 2, the speedups observed in the plots of Figure 8 for the independent and cooperative parallel GRASP + PR(*AP3*) are summarized. The estimated values of μ, λ, and $|\mu|/\lambda$, using 200 independent runs of the sequential GRASP + PR(*AP3*) are also shown. An approximately linear speedup is observed for the independent GRASP + PR(*AP3*) for up to 8 processors. With 16 processors, we observe a reduction in the speedup, although we can still consider that the program scales well for up to 16 processors. The gradual degradation in speedup as the number of processors increases is expected and is due to the fact that the number of processors (ρ) offsets the ratio $|\mu|/\lambda$, i.e. $|\mu|/\lambda \leq 2|\mu|/\lambda \leq 4|\mu|/\lambda \leq 8|\mu|/\lambda \leq 16|\mu|/\lambda$.

For the cooperative parallel GRASP + PR(*AP3*), we cannot make any prediction based on the ratio $|\mu|/\lambda$, since the processors share information and are therefore not independent. We observe that the cooperative approach benefited more than the independent approach from the increase in the number of processors. The increase in sharing of elite solutions compensated for the increase in inter-process communication. In fact, super-linear speedups can occur for the cooperative approach.

5.6.2. *Parallel results for the JSP.* The parallel GRASP(*JSP*) was tested on instances abz6, mt10, orb5, and la21, with target values 960, 960, 920, and 1120, respectively. The independent and cooperative parallel GRASP + PR(*JSP*), were also tested on instances abz6, mt10, orb5, and la21, but with more difficult target values 943, 938, 895, and 1100, respectively. The plots in Figure 9 show speedup for both parallel implementations of GRASP + PR(*JSP*), The plots were generated with 60 independent runs for each number of processors considered.

Table 3 lists the values of μ, λ, and $|\mu|/\lambda$ for each tested pair of instance and target value, as well as the speedups and efficiencies for the 2, 4, 8, and 16-processor runs. Speedups are, on average, approximately linear, in accordance with the low values observed for $|\mu|/\lambda$ in the table.

Table 4 summarizes the speedups for the independent and the cooperative parallel approaches of GRASP + PR(*JSP*). The values of μ, λ, and $|\mu|/\lambda$ are also shown. In accordance with the speedup prediction test, sub-linear speedups are observed for the independent approach of GRASP + PR(*JSP*). We notice that the ratios $|\mu|/\lambda$ are much higher than those in Table 3 for GRASP(*JSP*), as well as for ratios computed for the AP3 instances. On the other hand, the table shows linear and super-linear speedups for most of the instances tested with

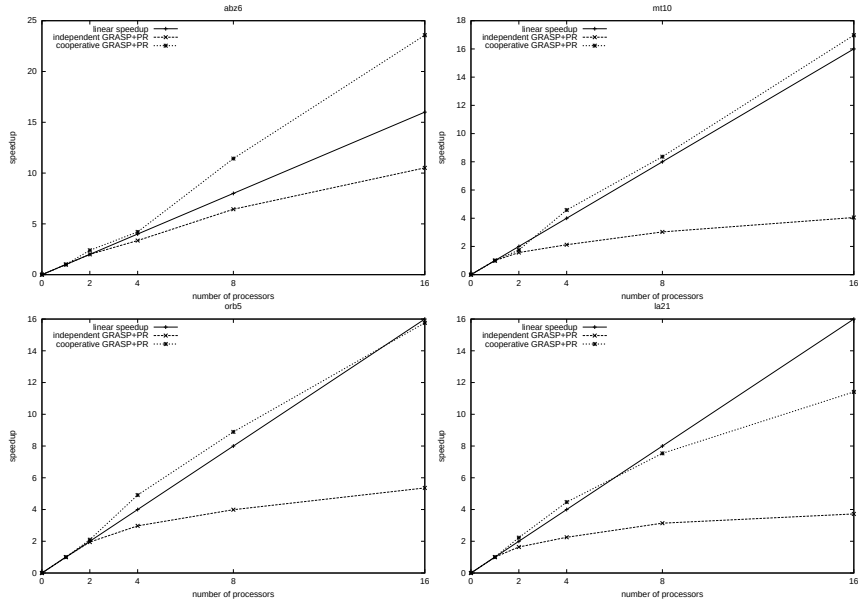

FIGURE 9. Speedups for the parallel implementations of independent and cooperative GRASP with path-relinking for the JSP: problems abz6 mt10 orb5 and la21 with target values 943, 938, 895, and 1100, respectively.

TABLE 3. Speedup, and effi ciency for instances of the JSP. Algorithm is the parallel implementation of GRASP. Instances are abz6, mt10, orb5, and la21, with target values 960, 960, 920, and 1120, respectively. The estimated parameters for the exponential distributions are shown for each pair of instance/target value.

	estimated parameter			number of processors							
				2		4		8		16	
prob.	μ	λ	$\lvert\mu\rvert/\lambda$	spdup	effic.	spdup	effic.	spdup	effic.	spdup	effic.
abz6	.42	15.56	.027	2.04	1.02	4.75	1.18	8.87	1.10	19.17	1.19
mt10	11.72	885.03	013	1.62	.81	4.07	1.01	7.34	.91	14.81	.92
orb5	1.24	38.27	.032	2.12	1.06	3.97	.99	7.63	.95	14.10	.88
la21	-1.01	206.83	.0049	1.94	.97	4.98	1.24	8.13	1.01	19.63	1.22
average:			**.019**	**1.93**	**.96**	**4.44**	**1.10**	**7.99**	**.99**	**16.92**	**1.05**

the cooperative approach. These speedups are considerably higher than the speedups observed for the independent approach. For example, for 16 processors, the average speedups for the cooperative approach are almost three times higher than those of the independent approach. These results show that cooperation is more critical in JSP than AP3. This perhaps is because path-relinking is applied less frequently in the JSP runs than in the AP3 runs, due to the criterion used to avoid applying path-relinking to poor quality solutions in JSP.

Figure 10 shows the empirical distributions for the parallel GRASP(JSP), and the parallel independent and cooperative GRASP + PR(JSP), using 16 processors. The programs were tested for problems abz6, mt10, orb5 and la21, with target values 943, 938, 895,

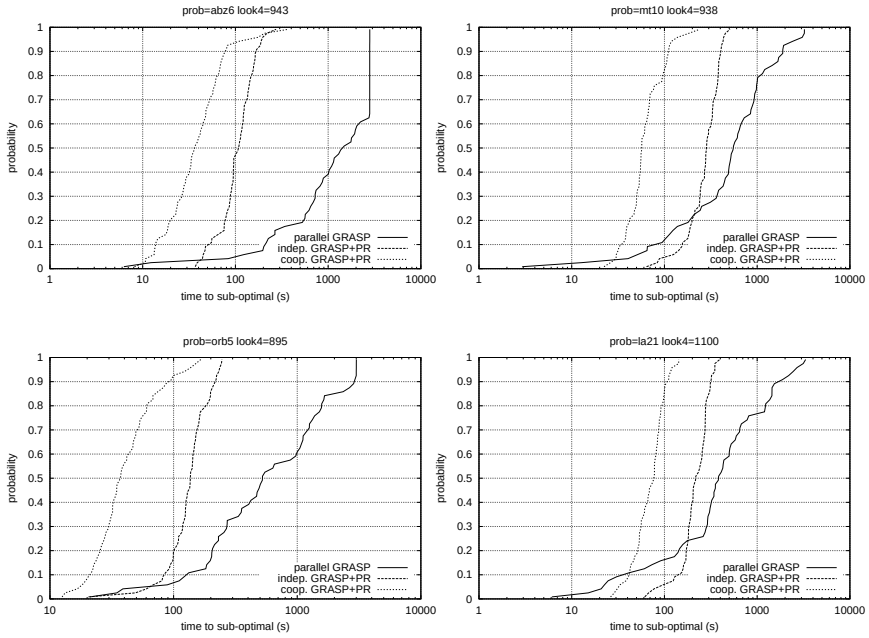

FIGURE 10. Empirical distributions for 16 processor parallel implementations of pure GRASP, independent GRASP with path-relinking, and cooperative GRASP with path-relinking: problems `abz6`, `mt10`, `orb5` and `la21`, for target values 943, 938, 895, and 1100, respectively.

TABLE 4. Speedups for instances of the JSP. Algorithms are independent, and cooperative implementation of GRASP with path-relinking. Instances are `abz6`, `mt10`, `orb5`, and `la21`, with target values 943, 938, 895, and 1100, respectively. The estimated parameters for the exponential distributions are shown for each pair of instance/target value.

	estimated parameter			speedup independent (number of processors)				speedup cooperative (number of processors)			
prob.	μ	λ	$\lvert\mu\rvert/\lambda$	2	4	8	16	2	4	8	16
`abz6`	47.67	756.56	.06	2.00	3.36	6.44	10.51	2.40	4.21	11.43	23.58
`mt10`	305.27	524.23	.58	1.57	2.12	3.03	4.05	1.75	4.58	8.36	16.97
`orb5`	130.12	395.41	.32	1.95	2.97	3.99	5.36	2.10	4.91	8.89	15.76
`la21`	175.20	407.73	.42	1.64	2.25	3.14	3.72	2.23	4.47	7.54	11.41
average:			**.34**	**1.79**	**2.67**	**4.15**	**5.91**	**2.12**	**4.54**	**9.05**	**16.93**

and 1100, respectively. We notice that for 16 processors, although parallel $\texttt{GRASP}(JSP)$ scales better than the independent $\texttt{GRASP}+\texttt{PR}(JSP)$, the execution times of the latter are in general, lower than the execution times of the former. We also notice that the execution times of the cooperative $\texttt{GRASP}+\texttt{PR}(JSP)$ are considerably lower than the execution times of the parallel $\texttt{GRASP}(JSP)$ and of the independent $\texttt{GRASP}+\texttt{PR}(JSP)$.

6. CONCLUSION

Parallel computers have increasingly found their way into metaheuristics. In particular, many parallel implementations of GRASP have been described in the literature. Recently, parallel implementations of GRASP with path-relinking have been proposed. In this paper, we contrast independent and cooperative parallel schemes for GRASP with path-relinking with a parallel scheme for GRASP. We also propose a test using a sequential GRASP implementation to predict the speedup of independent parallel implementations.

Implementations for the 3-index assignment problem and the job-shop scheduling problem are described. Computational experiments are done on four instances from each problem.

We conclude that the proposed test is useful for predicting the degree of speedup expected for parallel implementations of GRASP and GRASP with path-relinking. The results also show that parallel GRASP with path-relinking outperforms parallel GRASP and that the cooperative scheme outperforms the independent approach, often achieving superlinear speedups.

REFERENCES

[1] R.M. Aiex, S. Binato, and M.G.C. Resende. Parallel GRASP with path-relinking for job shop scheduling. *Parallel Computing*, 29:393–430, 2003.

[2] R.M. Aiex, M.G.C. Resende, P.M. Pardalos, and G. Toraldo. GRASP with path relinking for the three-index assignment problem. Technical report, Information Sciences Research Center, AT&T Labs Research, Florham Park, NJ 07932 USA, 2000. To appear in *INFORMS Journal on Computing*.

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

[4] A. Alvim and C.C. Ribeiro. Balanceamento de carga na paralelização da meta-heurística GRASP. In *X Simpósio Brasileiro de Arquiteturas de Computadores*, pages 279–282. Sociedade Brasileira de Computação, 1998.

[5] A.C.F. Alvim. Estratégias de paralelização da metaheurística GRASP. Master's thesis, Departamento de Informática, PUC-Rio, Rio de Janeiro, RJ 22453-900 Brazil, April 1998.

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

[7] J. E. Beasley. OR-Library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41:1069–1072, 1990.

[8] S. Binato, W.J. Hery, D.M. Loewenstern, and M.G.C. Resende. A GRASP for job shop scheduling. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys on Metaheuristics*, pages 58–79. Kluwer Academic Publishers, 2002.

[9] S.A. Canuto, M.G.C. Resende, and C.C. Ribeiro. Local search with perturbations for the prize-collecting Steiner tree problem in graphs. *Networks*, 38:50–58, 2001.

[10] J. M. Chambers, W. S. Cleveland, B. Kleiner, and P. A. Tukey. *Graphical Methods for Data Analysis*. Chapman & Hall, 1983.

[11] S. Duni, P.M. Pardalos, and M.G.C. Resende. Parallel metaheuristics for combinatorial optimization. In R. Corrêa, I. Dutra, M. Fiallos, and F. Gomes, editors, *Models for Parallel and Distributed Computation – Theory, Algorithmic Techniques and Applications*, pages 179–206. Kluwer Academic Publishers, 2002.

[12] 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.

[13] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[14] T.A. Feo, M.G.C. Resende, and S.H. Smith. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, 42:860–878, 1994.

[15] P. Festa and M.G.C. Resende. GRASP: An annotated bibliography. In C.C. Ribeiro and P. Hansen, editors, *Essays and surveys in metaheuristics*, pages 325–367. Kluwer Academic Publishers, 2002.

[16] A.M. Frieze. Complexity of a 3-dimensional assignment problem. *European Journal of Operational Research*, 13:161–164, 1983.

[17] M.R. Garey and D.S. Johnson. *Computers and intractability - A guide to the theory of NP-completeness*. W.H. Freeman and Company, 1979.

[18] F. Glover. Tabu search and adaptive memory programing – Advances, applications and challenges. In R.S. Barr, R.V. Helgason, and J.L. Kennington, editors, *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer, 1996.

[19] F. Glover. Multi-start and strategic oscillation methods – Principles to exploit adaptive memory. In M. Laguna and J.L. Gonzáles-Velarde, editors, *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research*, pages 1–24. Kluwer, 2000.

[20] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[21] 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.

[22] M. Laguna and R. Martí. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11:44–52, 1999.

[23] J. K. Lenstra and A. H. G. Rinnooy Kan. Computational complexity of discrete optimization problems. *Annals of Discrete Mathematics*, 4:121–140, 1979.

[24] 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.

[25] S.L. Martins, P.M. Pardalos, M.G.C. Resende, and C.C. Ribeiro. Greedy randomized adaptive search procedures for the Steiner problem in graphs. In P.M. Pardalos, S. Rajasejaran, and J. Rolim, editors, *Randomization methods in algorithmic design*, volume 43 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 133–145. American Mathematical Society, 1999.

[26] S.L. Martins, M.G.C. Resende, C.C. Ribeiro, and P.M. Pardalos. A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy. *Journal of Global Optimization*, pages 267–283, 2000.

[27] S.L. Martins, C.C. Ribeiro, and M.C. Souza. A parallel GRASP for the Steiner problem in graphs. In A. Ferreira and J. Rolim, editors, *Proceedings of IRREGUALAR'98 – 5th International Symposium on Solving Irregularly Structured Problems in Parallel*, volume 1457 of *Lecture Notes in Computer Science*, pages 285–297. Springer-Verlag, 1998.

[28] 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.

[29] 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 115–133. Kluwer Academic Publishers, 1995.

[30] P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A parallel GRASP for MAX-SAT problems. *Lecture Notes in Computer Science*, 1184:575–585, 1996.

[31] W.P. Pierskalla. The tri-subsitution method for the three-multidimensional assignment problem. *CORS Journal*, 5:71–81, 1967.

[32] M.G.C. Resende, T.A. Feo, and S.H. Smith. Algorithm 787: Fortran subroutines for approximate solution of maximum independent set problems using GRASP. *ACM Trans. Math. Software*, 24:386–394, 1998.

[33] M.G.C. Resende and C.C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 219–249. Kluwer Academic Publishers, 2002.

[34] M.G.C. Resende and C.C. Ribeiro. A GRASP with path-relinking for private virtual circuit routing. *Networks*, 41:104–114, 2003.

[35] M.G.C. Resende and R.F. Werneck. A hybrid heuristic for the *p*-median problem. Technical report, Internet and Network Systems Research Center, AT&T Labs Research, Florham Park, NJ, 2002.

[36] M.G.C. Resende and R.F. Werneck. A hybrid multistart heuristic for the uncapacitated facility location problem. Technical report, Internet and Network Systems Research Center, AT&T Labs Research, Florham Park, NJ, 2003.

[37] C.C. Ribeiro and I. Rosseti. A parallel GRASP for the 2-path network design problem. *Lecture Notes in Computer Science*, 2004:922–926, 2002.

[38] C.C. Ribeiro, E. Uchoa, and R.F. Werneck. A hybrid GRASP with perturbations for the Steiner problem in graphs. *INFORMS Journal on Computing*, 14:228–246, 2002.

[39] B. Roy and B. Sussmann. Les problèmes d'ordonnancement avec contraintes disjonctives, 1964.

[40] 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.

[41] E. D. Taillard. Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing*, 6:108–117, 1994.
[42] E.D. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17:443–455, 1991.
[43] M.G.A. Verhoeven and E.H.L. Aarts. Parallel local search. *Journal of Heuristics*, 1:43–66, 1995.

(Renata M. Aiex) CATHOLIC UNIVERSITY OF RIO DE JANEIRO, RIO DE JANEIRO, RJ 22451 BRAZIL.
E-mail address, Renata M. Aiex: rma@inf.puc-rio.br

(Mauricio G.C. Resende) INTERNET AND NETWORK SYSTEMS RESEARCH CENTER, AT&T LABS RESEARCH, FLORHAM PARK, NJ 07932 USA.
E-mail address, Mauricio G.C. Resende: mgcr@research.att.com