PARALLEL GRASP WITH PATH-RELINKING FOR JOB SHOP SCHEDULING

R.M. AIEX, S. BINATO, AND M.G.C. RESENDE

Abstract. In the job shop scheduling problem (JSP), a finite set of jobs is processed 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. A machine can process no more than one job at a time and once a job initiates processing on a given machine it must complete processing without interruption. A schedule is an assignment of operations to time slots on the machines. The objective of the JSP is to find a schedule that minimizes the maximum completion time, or makespan, of the jobs. In this paper, we describe a parallel greedy randomized adaptive search procedure (GRASP) with path-relinking for the JSP. A GRASP is a 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. Independent and cooperative parallelization strategies are described and implemented. Computational experience on a large set of standard test problems indicates that the parallel GRASP with path-relinking finds good-quality approximate solutions of the job shop scheduling problem.

1. Introduction

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.

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 $\mathcal{O}$ 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

Date: November 2001.

Key words and phrases. Combinatorial optimization, job shop scheduling, local search, GRASP, path-relinking, probabilistic algorithm.

AT&T Labs Research Technical Report.

$p_k^j = p(\sigma_k^j)$ is the processing time of operation σ_k^j. Defining $t(\sigma_k^j)$ to be the starting time of the k-th operation $\sigma_k^j \in \mathcal{O}$, the JSP can be formulated as follows:

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

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. Once the permutation of $\mathcal{J}$ is given, its feasibility status can be determined in $O(|\mathcal{J}| \cdot |\mathcal{M}|)$ time. The feasibility-checking procedure determines the makespan $\mathcal{C}_{max}$ for feasible schedules [36]. 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.

The JSP is NP-hard [26] and has also proven to be computationally challenging. Exact methods [4, 7, 9, 10, 19] have been successful in solving small instances, including the notorious 10×10 instance of Fisher and Thompson [16], proposed in 1963 and only solved twenty years later. Problems of dimension 15×15 are still considered to be beyond the reach of today's exact methods. For such problems there is a need for good heuristics. Surveys of heuristic methods for the JSP are given in [30, 37]. These include dispatching rules reviewed in [18], the shifting bottleneck approach [1, 4], local search [27, 28, 37], simulated annealing [27, 38], tabu search [28,29,36], and genetic algorithms [12]. Recently, Binato et al. [6] described a greedy randomized adaptive search procedure (GRASP) for the JSP. A comprehensive survey of job shop scheduling techniques can be found in Jain and Meeran [24]. In this paper, we present a new parallel GRASP with path-relinking for the job shop scheduling problem.

The remainder of the paper is organized as follows. In Section 2, we describe the new GRASP, describing two construction mechanisms and a local search algorithm. Path-relinking for the JSP and its incorporation to a GRASP are described in Section 3. Two parallelization schemes are presented in Section 4. Computational results are reported in Section 5 and concluding remarks are made in Section 6.

2. GRASP for JSP

GRASP [13–15, 32] is an iterative process, where each GRASP iteration usually consists of two phases: construction and local search. The construction phase builds a feasible solution, whose neighborhood is explored by 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 which measures the local benefit of including that element in the constructed solution. The restricted candidate list (RCL) is made up of candidate elements with 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.

Since the solutions generated by a GRASP construction phase are not guaranteed to be locally optimal, it is almost always beneficial to apply a local search to attempt to improve each constructed solution. A local search algorithm successively replaces the current solution by a better solution from its neighborhood. It terminates when there is no better solution found in the neighborhood.

In the remainder of this section, we describe two construction procedures and a commonly used local search strategy.

2.1. **Construction procedures.** For the JSP, we consider a single operation to be the building block of the construction phase. That is, we build a feasible schedule by scheduling individual operations, one at a time, until all operations are 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 $\mathcal{O}_c$ and the set of already scheduled operations by $\mathcal{O}_s$ 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 $\underline{\sigma_k^j} = \text{argmin}(h(\sigma_k^j) \mid \sigma_k^j \in \mathcal{O}_c)$. Let $\overline{\sigma_k^j} = \text{argmax}(h(\sigma_k^j) \mid \sigma_k^j \in \mathcal{O}_c)$, $\underline{h} = h(\underline{\sigma_k^j})$, and $\overline{h} = h(\overline{\sigma_k^j})$. Then, the GRASP restricted candidate list (RCL) is defined as

$$\text{RCL} = \{\sigma_k^j \in \mathcal{O}_c \mid \underline{h} \le h(\sigma_k^j) \le \underline{h} + \alpha(\overline{h} - \underline{h})\},$$

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

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 $\mathcal{M}_{\sigma_k^j}$. Let a and b denote the start and end times of an available time slot on $\mathcal{M}_{\sigma_k^j}$ and let $e = t(\sigma_{k-1}^j) + p_{k-1}^j$ denote the completion time of operation σ_{k-1}^j. Insertion in this time slot at time point $\max\{a, e\}$ is feasible if and only if $b - \max\{a, e\} \ge p_k^j$. Construction ends when the partial schedule is complete, i.e. all operations have been scheduled.

In Binato et al. [6], the greedy function $h(\sigma_k^j)$ is 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 $\mathcal{O} = \{\mathcal{O}_s \cup \sigma_k^j\}$. We will refer to this greedy function as the *makespan* greedy function.

In this paper, we propose another greedy function, which, as we will see later, is used in conjunction with the makespan greedy function. This function favors operations from jobs having long remaining processing times by using the greedy function $h(\sigma_k^j) = -\sum_{\sigma_l^j \notin \mathcal{O}_s} p_l^j$, which measures the remaining processing time for job j. We refer to it as the *time-remaining* greedy function.

2.2. **Local search phase.** Since there is no guarantee that the schedule obtained in the construction phase is optimal, local search should be applied to attempt to decrease its makespan.

We employ the two exchange local search, based on the disjunctive graph model of Roy and Sussmann [34], and used in Binato et al. [6]. The disjunctive graph $G = (V, A, E)$ is defined such that

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

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

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

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 $|\mathcal{J}| \cdot |\mathcal{M}| + 1$. Thus, the objective of the JSP is to find an orientation of E such that the longest path in G is minimized.

Taillard [36] 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 on the same machine 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 complexity $O(|\mathcal{J} \cdot \mathcal{M}|)$ given that the longest path of G was evaluated. These procedures were implemented in Binato et al. [6] and are borrowed in our implementation.

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

```
procedure PATH_RELINKING ($\mathcal{M}, \mathcal{J}, \mathcal{O}, p, Makespan, S, T$)
1  $S = \{(j^S_{1,1}, j^S_{1,2}, \ldots, j^S_{1,|\mathcal{J}|}), (j^S_{2,1}, j^S_{2,2}, \ldots, j^S_{2,|\mathcal{J}|}), \ldots,$
           $(j^S_{|\mathcal{M}|,1}, j^S_{|\mathcal{M}|,2}, \ldots, j^S_{|\mathcal{M}|,|\mathcal{J}|})\};$
2  $T = \{(j^T_{1,1}, j^T_{1,2}, \ldots, j^T_{1,|\mathcal{J}|}), (j^T_{2,1}, j^T_{2,2}, \ldots, j^T_{2,|\mathcal{J}|}), \ldots,$
           $(j^T_{|\mathcal{M}|,1}, j^T_{|\mathcal{M}|,2}, \ldots, j^T_{|\mathcal{M}|,|\mathcal{J}|})\};$
3  $c_{gmin} = Makespan$; $S_{gmin} = S$;
4  for $k = 1, \ldots, |\mathcal{M}|$ do
5      $\delta^{S,T}_k = \{i = 1, \ldots, |\mathcal{J}| \mid j^S_{k,i} \neq j^T_{k,i}\}$;
6  od;
7  while $(\sum_{k=1}^{|\mathcal{M}|} |\delta^{S,T}_k| > 2)$ do
8      $c_{min} = \infty$;
9      for $k = 1, \ldots, |\mathcal{M}|$ do
10         for $i \in \delta^{S,T}_k$ do
11             Let $q$ be such that $j^T_{k,q} == j^S_{k,i}$;
12             $\bar{S} = S \setminus \{(\ldots, j^S_{k,i}, j^S_{k,i+1}, \ldots, j^S_{k,q-1}, j^S_{k,q}, \ldots)\}$;
13             $\bar{S} = \bar{S} \cup \{(\ldots, j^S_{k,q}, j^S_{k,i+1}, \ldots, j^S_{k,q-1}, j^S_{k,i}, \ldots)\}$;
14             $\bar{c}$= CALCULATE_MAKESPAN ($\bar{S}$);
15             if $\bar{c} \leq c_{min}$ then
16                 $c_{min} = \bar{c}$;
17                 $S_{min} = \bar{S}$;
18                 $i_{min} = i$;
19                 $k_{min} = k$;
20             fi;
21         rof;
22     rof;
23     $S = S_{min}$; $Makespan = c_{min}$;
24     $\delta^{S,T}_{k_{min}} = \delta^{S,T}_{k_{min}} \setminus \{i_{min}\}$;
25     if $Makespan \leq c_{gmin}$ then
26         $c_{gmin} = Makespan$;
27         $S_{gmin} = S$;
28     fi;
29 elihw;
30 return ($S_{gmin}$);
end PATH_RELINKING;
```

FIGURE 1. Path-relinking between initial solution S and guiding solution T.

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.

3. Path-relinking for JSP

Path-relinking is an enhancement to the basic GRASP procedure, leading to significant improvements in solution quality. Path-relinking was originally proposed by

```
procedure GRASP_PR ($\mathcal{M}, \mathcal{J}, \mathcal{O}, p$, look4, maxitr, maxpool, freq)
1  $P = \emptyset$;
2  for $i = 1, \ldots,$ maxitr do
3      if  mod $(i, 2) == 0$ then
4          GREEDY_MASKESPAN($S, \mathcal{M}, p, |\mathcal{M}|, |\mathcal{J}|, Makespan$);
5      else
6          GREEDY_TIME_REMAINING($S, \mathcal{M}, p, |\mathcal{M}|, |\mathcal{J}|, Makespan$);
7      fi;
8      LOCAL($S, \mathcal{M}, p, |\mathcal{M}|, |\mathcal{J}|, Makespan$);
9      if $|P| ==$ maxpool then
10         $accepted =$ VERIFY_QUALITY($S, i$);
11         if $accepted$ then
12             for $T \in P' \subseteq P$ do
13                 $S_{gmin} =$ PATH_RELINKING($\mathcal{M}, \mathcal{J}, \mathcal{O}, p, Makespan, S, T$);
14                 UPDATE_POOL($S_{gmin}, c_{gmin}, P$);
15                 $S_{gmin} =$ PATH_RELINKING($\mathcal{M}, \mathcal{J}, \mathcal{O}, p, Makespan, T, S$);
16                 UPDATE_POOL($S_{gmin}, c_{gmin}, P$);
17             rof;
18         fi;
19     else $P = P \cup \{S\}$ fi;
20     if  mod $(i,$ ifreq$) == 0$ then INTENSIFY($P$) fi;
21     $S_{best} =$ POOLMIN($P$);
22     if MAKESPAN($S_{best}$) $\leq$ look4 then return ($S_{best}$) fi;
23 rof;
24 POST_OPTIMIZATION($P$);
25 $S_{best} =$ POOLMIN($P$);
26 return ($S_{best}$);
end GRASP_PR;
```

FIGURE 2. GRASP with bidirectional path-relinking for JSP.

Glover [20] as an intensification strategy exploring trajectories connecting elite solutions obtained by tabu search or scatter search [21–23]. 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 [25]. It was followed by several extensions, improvements, and successful applications [2, 8, 31, 33].

In this section, we propose a path-relinking strategy for the JSP. In our description, a schedule is 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.

The path-relinking approach consists in exploring trajectories that connect a *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. Usually

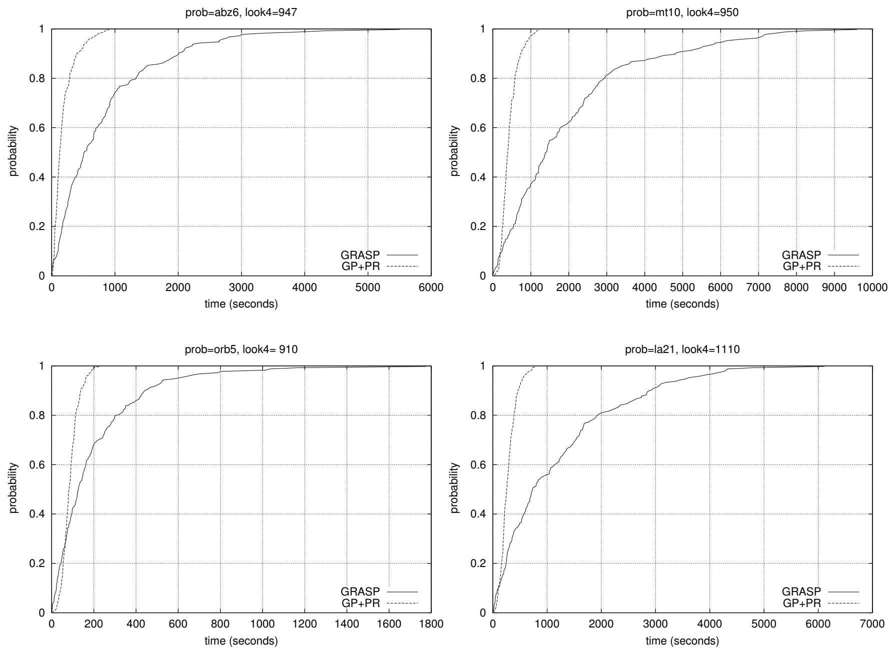

FIGURE 3. Empirical probability distributions of time to target value for `GRASP` and `GP+PR`: problems `abz6`, `mt10`, `orb5` and `la21`.

the guiding solution is of high quality. For the JSP, path-relinking is done between an initial solution

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

and a guiding solution

$$T = \{(j^T_{1,1}, j^T_{1,2}, \dots, j^T_{1,|\mathcal{J}|}), (j^T_{2,1}, j^T_{2,2}, \dots, j^T_{2,|\mathcal{J}|}), \dots, (j^T_{|\mathcal{M}|,1}, j^T_{|\mathcal{M}|,2}, \dots, j^T_{|\mathcal{M}|,|\mathcal{J}|})\}.$$

Pseudo-code for this procedure is shown in Figure 1.

Let the symmetric difference between S and T be defined by the $|\mathcal{M}|$ sets of indices

$$\delta^{S,T}_k = \{i = 1, \dots, |\mathcal{J}| \mid j^S_{k,i} \neq j^T_{k,i}\}, k = 1, \dots, |\mathcal{M}|.$$

These sets are computed in lines 4 to 6 in the pseudo-code.

An intermediate solution of the path-relinking trajectory is visited at each step of the loop from line 7 to 29. During a move, a permutation array in S, given by

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

is replaced by a permutation array

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

by exchanging operations $j^S_{k,i}$ and $j^S_{k,q}$, where $i \in \delta^{S,T}_k$ and q are such that $j^T_{k,q} = j^S_{k,i}$. Note that solutions that violate the precedence constraints can be produced by these moves. The feasibility of solution S is verified during procedure *Calculate_Makespan*(S) (line 14), which consists in computing the critical path in the disjunctive graph presented in Section 2.2, using the algorithm proposed in [36].

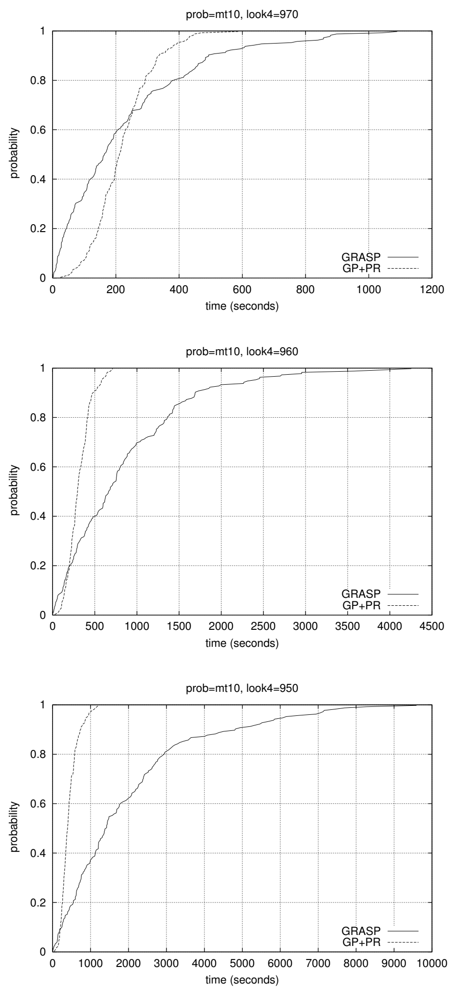

FIGURE 4. Empirical probability distributions of time to target value for `GRASP` and `GP+PR`: problem `mt10` and target values 970, 960 and 950.

An infeasible schedule is detected when a cycle is found in the corresponding graph. The makespan of an infeasible solution is defined to be infinite so as to bias the search toward feasible regions.

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 $\delta_k^{S,T}$ (line 24). This continues until there are only two move indices left in one of the sets $\delta_k^{S,T}$. At this point, the move obtained by exchanging these elements will produce the guiding solution. The best solution (S_{gmin}) found during the path traversal is returned by the procedure.

In the implementation proposed for the JSP, a *pool* P of elite solutions is built with the GRASP solutions produced during the first $|P|$ GRASP iterations. After this initial phase, a solution s_g produced by GRASP is relinked with one or more elite solutions s_e in P. Path-relinking can be applied from GRASP solution s_g to

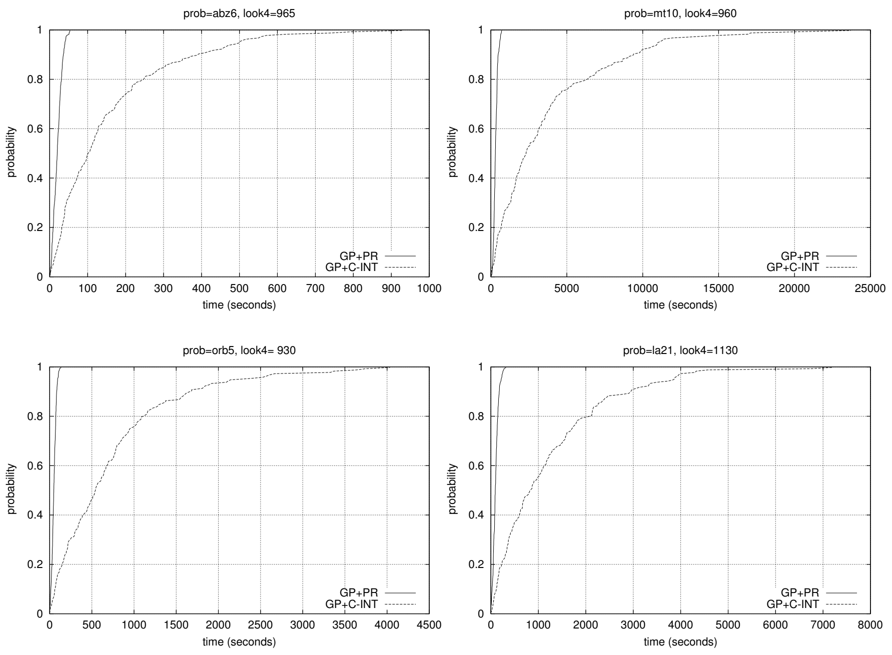

FIGURE 5. Empirical probability distributions of time to target value for `GP+C-INT` and `GP+PR`: problems `abz6`, `mt10`, `orb5` and `la21`.

pool solution s_e, from pool solution s_e to GRASP solution s_g, or in both directions. These two trajectories very often visit different intermediate solutions.

The hybrid strategy proposed uses an approach developed by Fleurent and Glover [17] to incorporate elite solutions to a GRASP. Let c_{best} and c_{worst} be the values of the objective functions of the best and the worst solution in P, respectively. Given two solutions

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

and

$$T = \{(j^T_{1,1}, j^T_{1,2}, \dots, j^T_{1,|\mathcal{J}|}), (j^T_{2,1}, j^T_{2,2}, \dots, j^T_{2,|\mathcal{J}|}), \dots, (j^T_{|\mathcal{M}|,1}, j^T_{|\mathcal{M}|,2}, \dots, j^T_{|\mathcal{M}|,|\mathcal{J}|})\},$$

let

$$\Delta(S,T) = \sum_{k=1}^{|\mathcal{M}|} |\delta^{S,T}_k|$$

be a measure of the non-similarity between S and T.

A solution S_{gmin} produced by path-relinking is a candidate for insertion in the pool. S_{gmin} will be 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. $c_{best} \leq c_{gmin} < c_{worst}$ and for all elite solutions $S_p \in P$, $\Delta(S_{gmin}, S_p) > \Delta_{\min}$, i.e., S_{gmin} is better then the worst solution in P and differs significantly from all elite solutions.

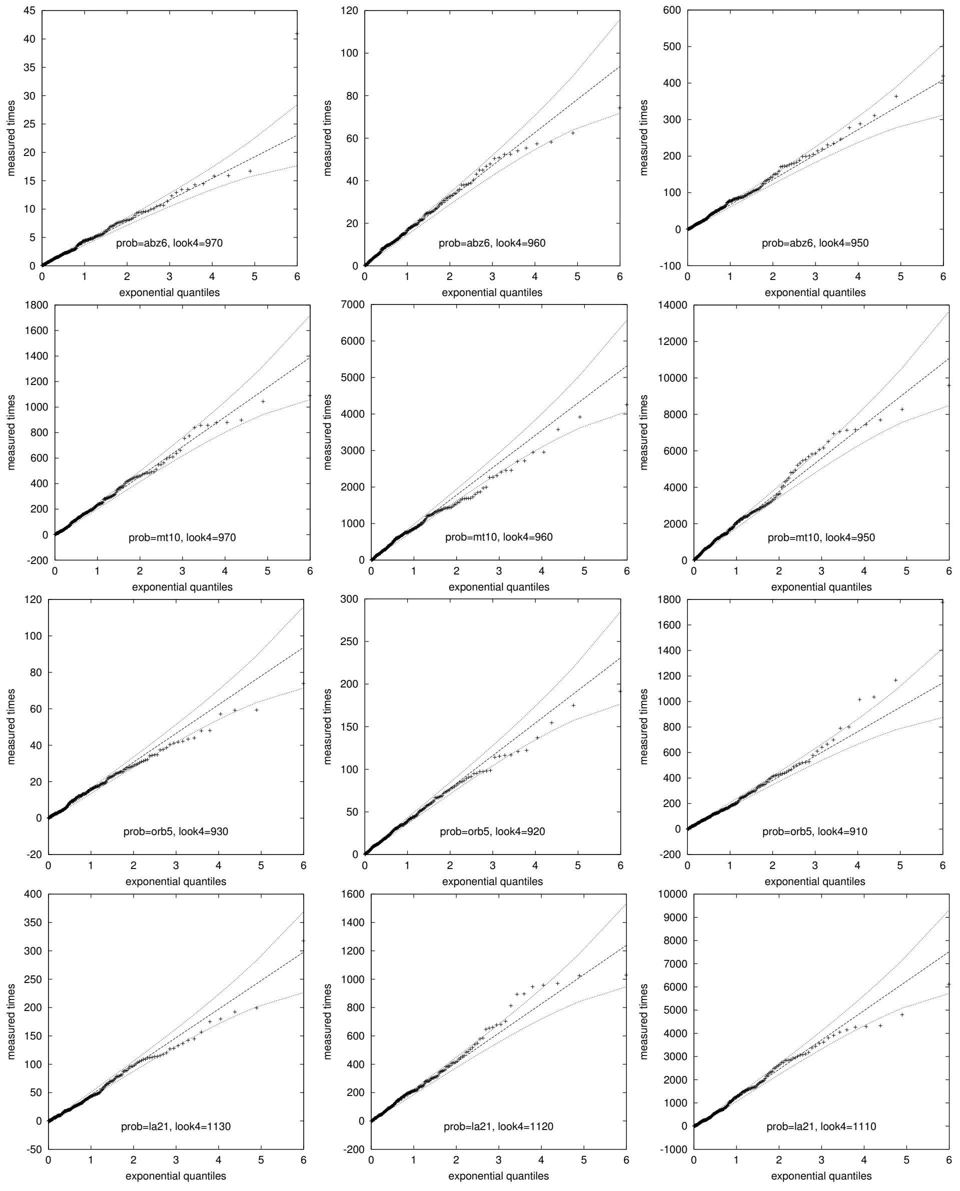

FIGURE 6. Q-Q plots for `GRASP`: problems `abz6`, `mt10`, `orb5` and `la21`.

Once accepted for insertion in P, S_{gmin} will replace the worst elite solution, which will be discarded from P.

Note that if the number of moves needed to traverse the path from $S \in P$ to S_{gmin} is at most $\Delta_{\min}/2$, then path solution S_{gmin} must satisfy $\Delta(S, S_{gmin}) \leq \Delta_{\min}$ and there is no need to compute the symmetric difference since S_{gmin} can only be included in the elite set if it is better than the best elite solution.

In the GRASP with path-relinking for the 3-index assignment problem [2], the cost of a solution in the neighborhood of S can be computed from the cost of S

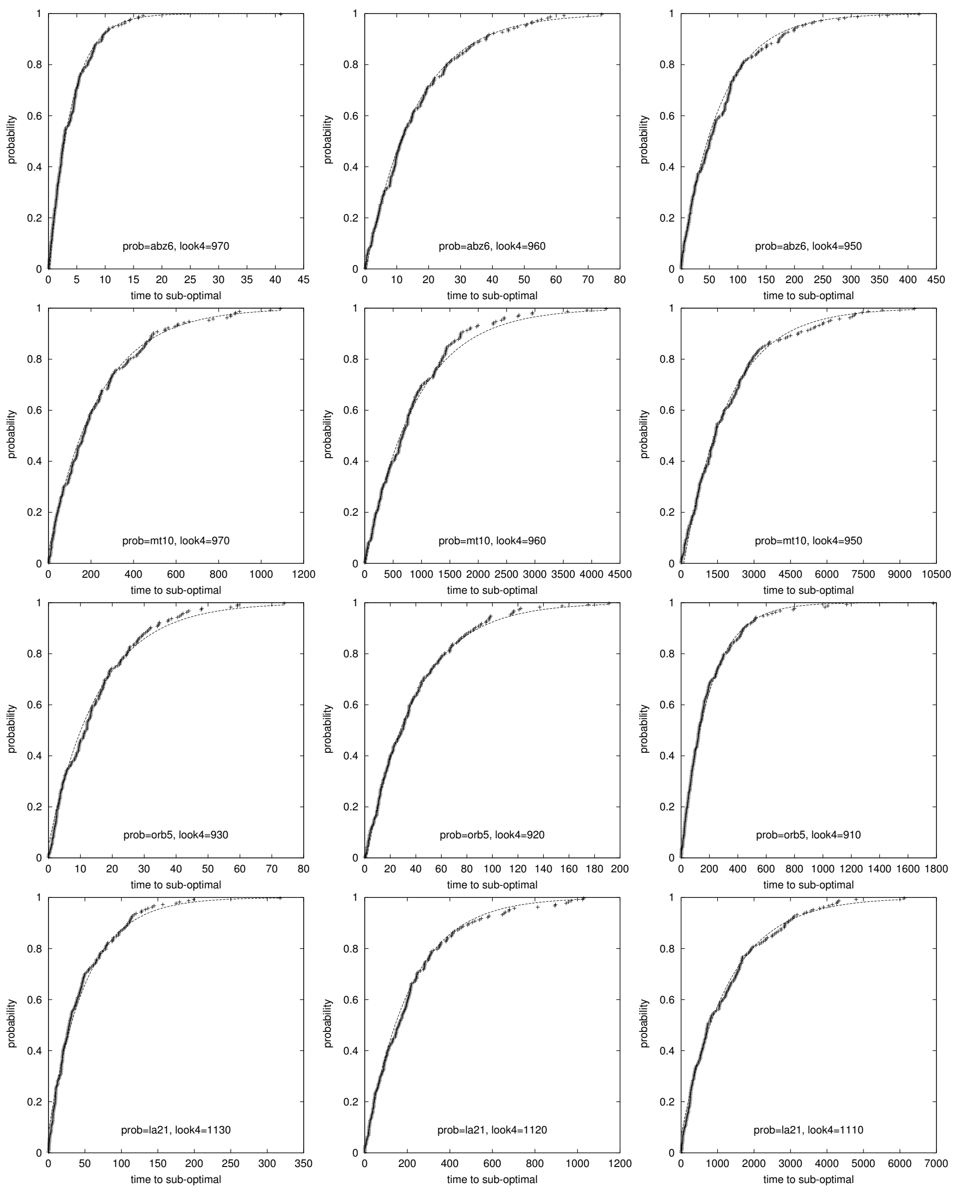

FIGURE 7. Empirical and theoretical probability distributions of time to target for `GRASP`: problems `abz6`, `mt10`, `orb5` and `la21`.

in O(1) time. For the JSP, the cost of each solution visited by path-relinking is computed in $O(|\mathcal{J}|\cdot|\mathcal{M}|)$, using the algorithm proposed in [36]. Therefore, it is computationally expensive to apply path-relinking after each iteration of the GRASP. Instead, we propose to apply path-relinking only when the GRASP solution satisfies a given quality criteria.

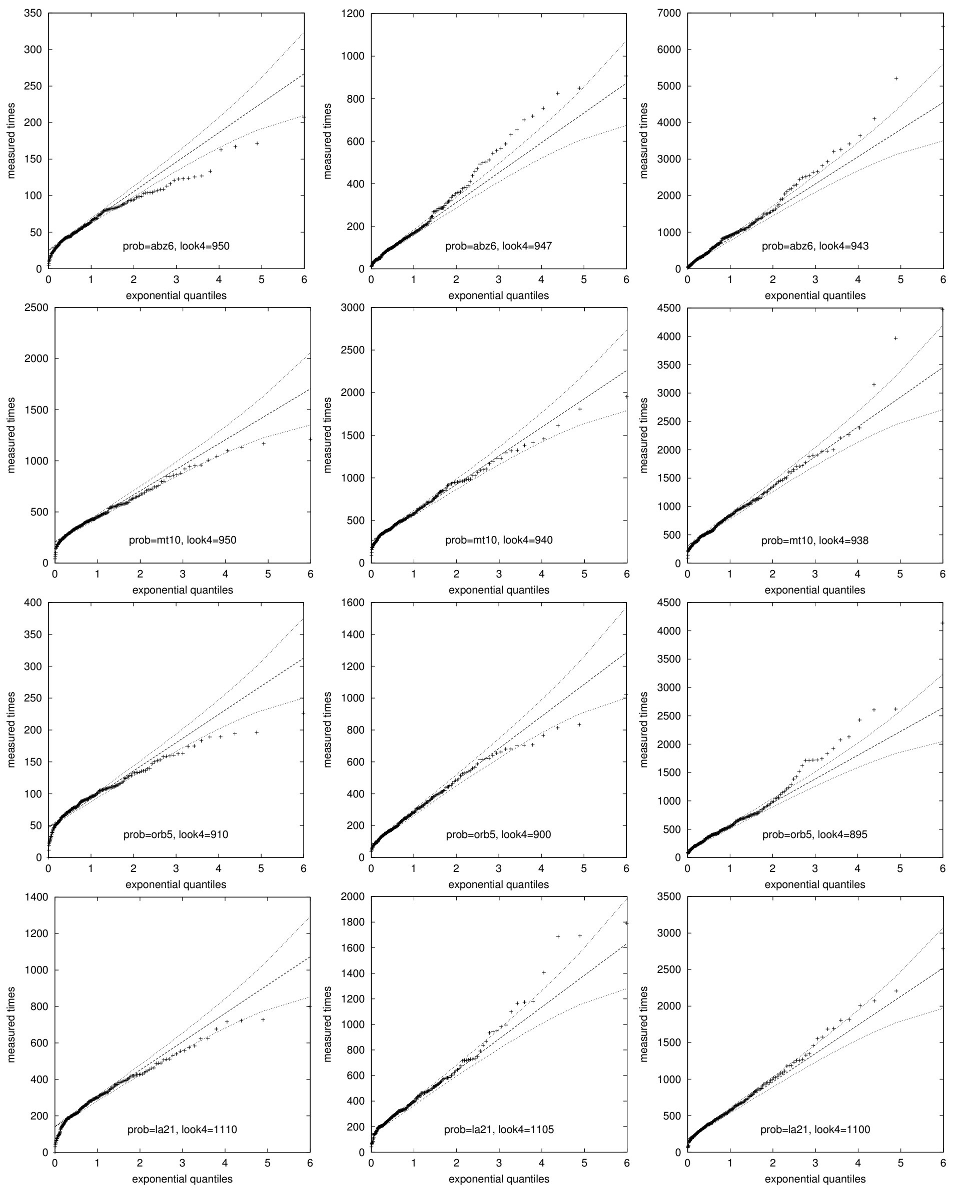

FIGURE 8. Q-Q plots for `GP+PR`: problems `abz6`, `mt10`, `orb5` and `la21`.

The quality criteria proposed uses the mean value μ_n and the standard deviation σ_n of the costs of the GRASP solutions produced during the first n iterations. A solution S_i takes part in path-relinking if

1. $c(S_i) \leq c_{worst}$, for $i \leq n$;
2. $c(S_i) \leq \max(c_{worst}, \mu_n - 2 * \sigma_n)$, for $i > n$.

Path-relinking can also be used in an intensification scheme for the elite set [2]. This 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

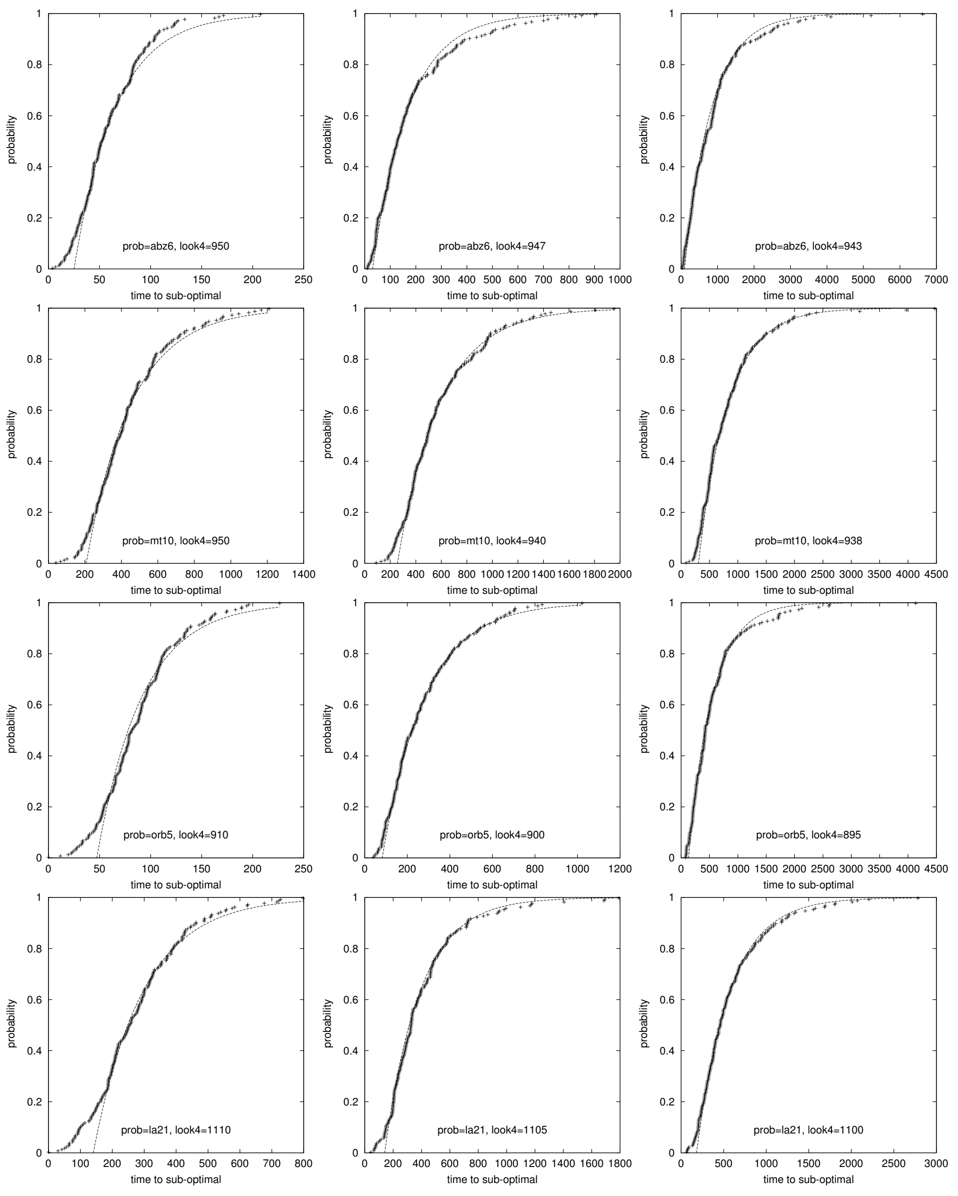

FIGURE 9. Empirical and theoretical probability distributions of time to target for `GP+PR`: problems `abz6`, `mt10`, `orb5` and `la21`.

change in P 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).

The intensification procedure is executed after each interval of `ifreq` iterations during the optimization. After each intensification phase, if no change in P occurs for at least `ifreq` iterations, the costs of the $|P|/2$ worst solutions in P are set to infinity. This is done to guarantee that the solutions in P are renewed. The

Table 1. Probability estimates of finding a solution at least as good as the target solution, as a function of maximum solution time for `GRASP` and `GP+PR`. Instances are `abz6`, `mt10`, `orb5` and `la21`, with target values 947, 950, 910 and 1110, respectively.

	`abz6`		`mt10`		`orb5`		`la21`	
time	`GRASP`	`GP+PR`	`GRASP`	`GP+PR`	`GRASP`	`GP+PR`	`GRASP`	`GP+PR`
100s	.09	.39	.03	.01	.42	.67	.10	.10
500s	.48	.93	.19	.71	.92	1.00	.36	.92
1000s	.74	1.00	.37	.97	.98	1.00	.56	1.00
1500s	.84	1.00	.54	1.00	.99	1.00	.69	1.00

$|P|/2$ worst solutions are eventually replaced by solutions generated in the following GRASP with path-relinking iterations. Hence, solutions with a high makespan, but sufficiently different from the solutions in P, are accepted for insertion in the pool.

Path-relinking as a post-optimization step was introduced in [2]. After applying path-relinking between all pairs of elite solutions and no further change in the elite set occurs, the local search procedure of Subsection 2.2 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.

3.1. **GRASP with path-relinking.** We describe how we combined path-relinking and GRASP to form a hybrid GRASP with path-relinking. Pseudo-code for the GRASP with path-relinking for JSP is presented in Figure 2. Let `maxpool` be the size of the elite set. The first `maxpool` GRASP iterations contribute one solution to the elite set per GRASP iteration (line 19). Path-relinking is not done until the pool of elite solutions is full.

GRASP alternates between the two construction procedures described in Section 2. Odd numbered iterations use the randomized time-remaining greedy function (line 6), while even iterations use randomized makespan greedy (line 4). The local search used is the one proposed by Taillard [36] (line 8).

Once the pool of elite solutions is full, the solution S produced by the local search phase of GRASP is tested to verify its quality (line 10), using the quality criteria described in Section 3. If S passes the quality test, bidirectional path-relinking is done between S and all elements of a subset $P' \subseteq P$ (lines 11 to 18). After each path-relinking phase, the best solution obtained by path-relinking is tested for inclusion in the elite pool (lines 14 and 16).

Every `ifreq` GRASP iterations the path-relinking intensification process is carried out (lines 20).

The GRASP with path-relinking loop from line 2 to 23 continues for at most `maxitr` iterations, but can be terminated when a schedule having a makespan of at most `look4` is found (line 22).

Finally, path-relinking post optimization is done on the elite set (line 24).

4. Parallel GRASP with path-relinking for the JSP

In this section, we describe two parallel implementations of GRASP with path-relinking for the JSP. The first scheme (called non-collaborative) limits communication between processors only for problem input, detection of process termination,

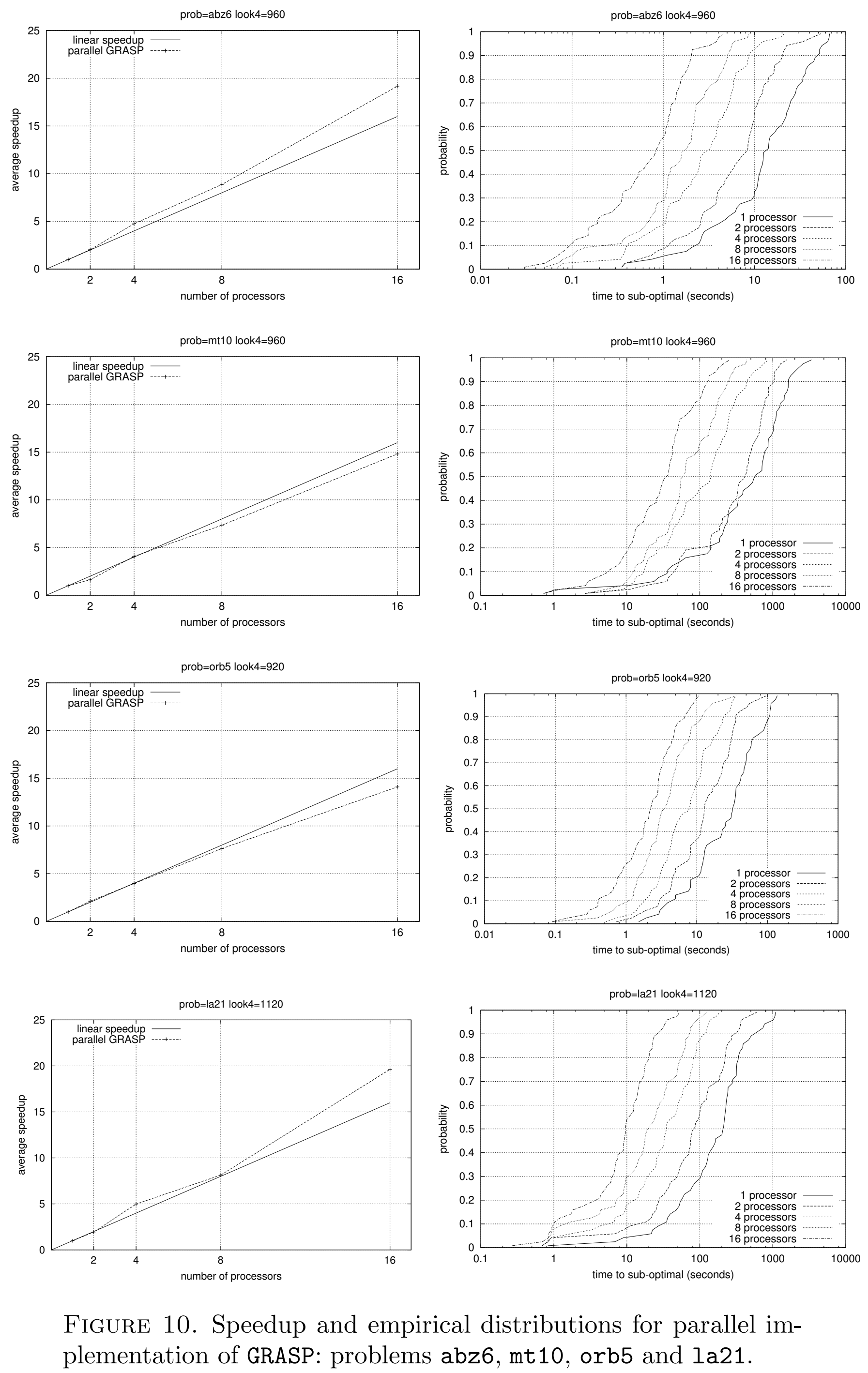

FIGURE 10. Speedup and empirical distributions for parallel implementation of `GRASP`: problems `abz6`, `mt10`, `orb5` and `la21`.

and determination of best overall solution. In addition to the communication allowed in the non-collaborative scheme, the second scheme (called collaborative) allows processes to exchange information regarding their elite sets.

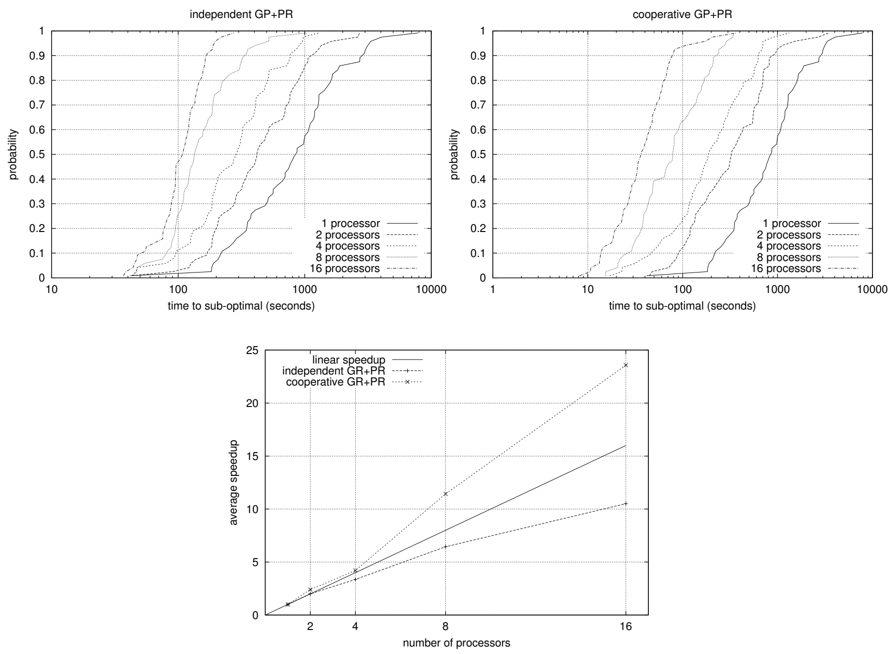

FIGURE 11. Speedup and empirical distributions for parallel implementations of `GP+PR`: problem `abz6` with target value 943.

4.1. **Non-collaborative scheme.** We revisit a basic parallelization scheme for GRASP with path-relinking proposed in [2]. Figure 15 shows pseudo-code for this *multiple independent walks* scheme [39].

Our implementation uses message passing for communication between processors. This communication is limited to program initialization and termination. A single process 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.

The non-collaborative parallel GRASP with path-relinking is built upon the sequential algorithm of Figure 2. Each process executes a copy of the program. We discuss the differences between the sequential algorithm and this parallel algorithm. In line 1 of Figure 15, the rank of the process and the number 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. The initial seed for process `my_rank` is computed in lines 2 to 4. This way, each process has a sequence of `maxitr` initial seeds.

The **for** loop from line 6 to line 37 executes the iterations. The construction, local search, and path-relinking phases are identical to the those of the sequential algorithm. In line 26, if a process finds a schedule with makespan not greater than `look4`, it sends a flag to each of the other processes indicating that it has found the solution. Likewise, when a process completes `maxitr` iterations, it sends a flag

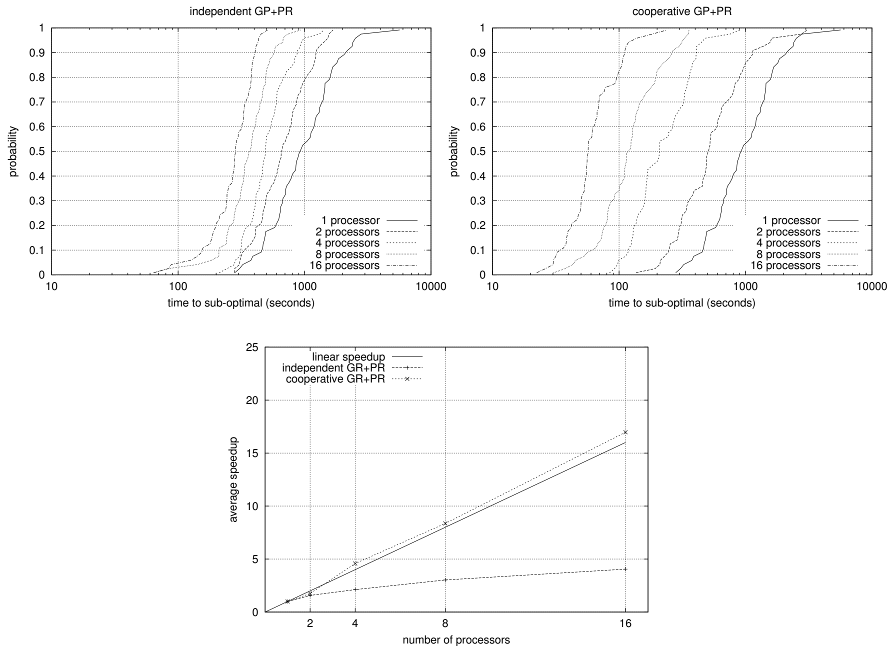

FIGURE 12. Speedup and empirical distributions for parallel implementations of `GP+PR`: problem `mt10` with target value 938.

to each of the other processes indicating that it has completed the preset number of iterations (lines 27 to 30).

In line 31, the process checks if there are any status flags to be received. If there is a flag indicating that a schedule with makespan not greater than `look4` has been found, then execution of **for** loop from line 6 to line 37 is terminated (line 33). If a flag indicating that some process has completed the preset number of iterations, a counter (*num_stop*) of the number of processes that have completed the iterations is incremented (line 34). If all processes have completed their iterations, the execution of the **for** loop is terminated (line 36).

Each process, upon terminating **for** loop going from line 6 to line 37, runs the post-optimization phase on the pool of elite solutions (line 38). A reduce operator (`GET_GLOBAL_BEST`) determines the global best solution among all processes in line 39 and returns this solution.

A parallel pure GRASP can be obtained from the algorithm in Figure 15 by skipping the execution of lines 13 to 23. 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 carried out during a pure GRASP parallel approach.

4.2. **Collaborative scheme.** In the collaborative parallel GRASP with path-relinking, processes share elite set information. We now describe this scheme, whose pseudo-code is presented in Figure 16. This algorithm is built on top of the non-collaborative scheme presented in the previous subsection. We limit our discussion to the differences between the two schemes.

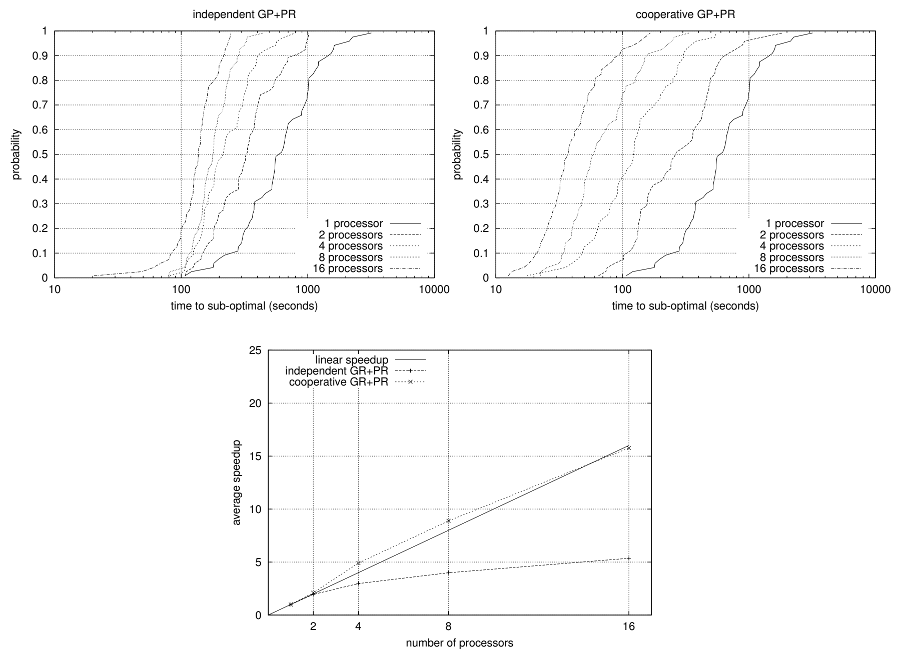

FIGURE 13. Speedup and empirical distributions for parallel implementations of GP+PR: problem orb5 with target value 895.

The differences between the non-collaborative and collaborative schemes 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 new elite solutions to it. If there are new elite solutions to be received, RECEIVE_SOLUTIONS (in lines 17 and 21) 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 20 and 24) the process writes the new elite set solutions to a local send buffer. In line 26, if the local send buffer is not empty, the process sends the buffer contents to the other processes.

Another difference between the non-collaborative and the collaborative schemes concerns the INTENSIFY procedure. In the collaborative scheme, whenever the local elite set pool is updated, the new elite set 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.

5. Computational results

This section reports on results of computational experiments done with sequential and parallel versions of the pure GRASP and GRASP with path-relinking heuristics proposed in this paper.

5.1. **Computer environment.** The experiments were done on an SGI Challenge computer (16 196-MHz MIPS R10000 processors and 12 194-MHz R10000 processors) with 7.6 Gb of memory. Each run of the sequential implementations used a

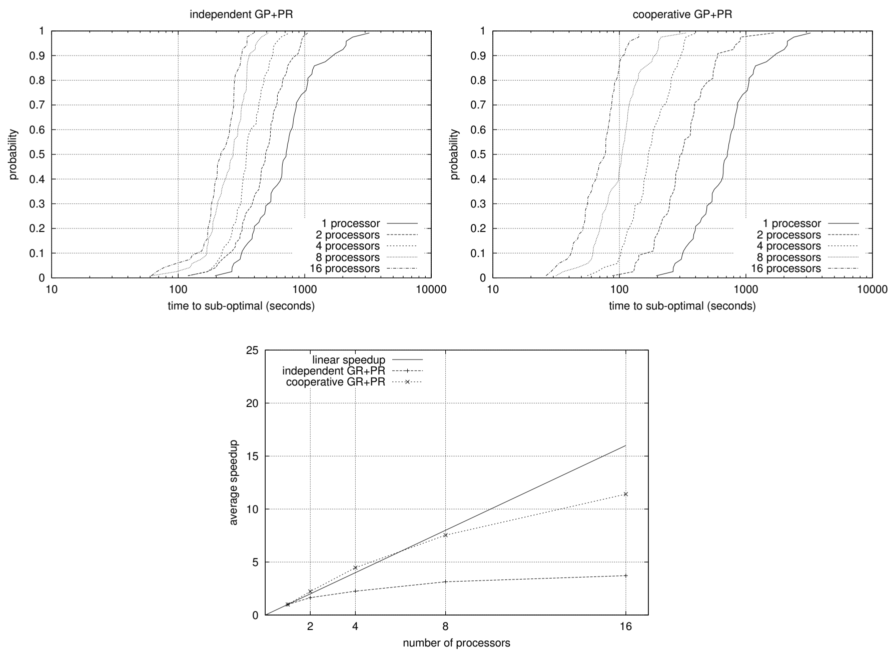

Figure 14. Speedup and empirical distributions for parallel implementations of `GP+PR`: problem `la21` with target value 1100.

Table 2. Time to find a solution with cost at least as good as the target value, as a function of probability. Problem `mt10` was tested for target values 970, 960 and 950. The percentage reduction in solution time of `GP+PR` with respect of `GRASP` is shown for each target value.

	`look4=970`			`look4=960`		
prob.	`GRASP`	`GP+PR`	red.(%)	`GRASP`	`GP+PR`	red.(%)
0.2	44.92s	144.61s	-221.92	214.94s	198.75s	7.53
0.5	163.93s	211.66s	-29.11	667.41s	295.71s	55.69
0.8	386.94s	292.98s	24.28	1362.65s	416.35s	69.44
	`look4=950`					
prob.	`GRASP`	`GP+PR`	red.(%)			
0.2	546.45s	263.02s	51.86			
0.5	1422.20s	394.51s	72.26			
0.8	2951.01s	578.53s	80.39			

single processor. The parallel implementations were run on 1, 2, 4, 8, and 16 processors. Load on the machine was low throughout the experiments and therefore processors were always available.

The algorithms were coded in Fortran and were compiled with the SGI MIPSpro F77 compiler using flags `-O3 -r4 -64`. The Message-Passing Interface (MPI) specification has become a common standard for message-passing libraries for parallel computations [35]. 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

```
procedure NON-COLLAB_GRASP_PR($\mathcal{M}, \mathcal{J}, \mathcal{O}, p$, seed, look4, maxitr, maxpool, freq)
1  $my\_rank$ = GET_RANK(); $nprocs$ = GET_NUM_PROCS();
2  for $i = 1, \ldots, (maxitr/nprocs) * my\_rank$ do
3      seed = rand(seed);
4  rof;
5  $P = \emptyset$; $num\_stop = 0$;
6  for $i = 1, \ldots, \infty$ do
7      if  mod $(i, 2) == 0$ then
8          GREEDY_MASKESPAN(s$eed, S, \mathcal{M}, p, |\mathcal{M}|, |\mathcal{J}|, Makespan$);
9      else
10         GREEDY_TIME_REMAINING(s$eed, S, \mathcal{M}, p, |\mathcal{M}|, |\mathcal{J}|, Makespan$);
11     fi;
12     LOCAL($S, \mathcal{M}, p, |\mathcal{M}|, |\mathcal{J}|, Makespan$);
13     if $|P| ==$ maxpool then
14         $accepted$ = VERIFY_QUALITY($S, i$);
15         if $accepted$ then
16             for $T \in P' \subseteq P$ do
17                 $S_{gmin}$ = PATH_RELINKING($\mathcal{M}, \mathcal{J}, \mathcal{O}, p, Makespan, S, T$);
18                 UPDATE_POOL($S_{gmin}, c_{gmin}, P$);
19                 $S_{gmin}$ = PATH_RELINKING($\mathcal{M}, \mathcal{J}, \mathcal{O}, p, Makespan, T, S$);
20                 UPDATE_POOL($S_{gmin}, c_{gmin}, P$);
21             rof;
22         fi;
23     else $P = P \cup \{S\}$ fi;
24     if  mod $(i$, ifreq$) == 0$ then INTENSIFY($P$) fi;
25     $S_{best}$ = POOLMIN($P$);
26     if MAKESPAN($S_{best}$) $\leq$ look4 then SEND_ALL($look4\_stop$) fi;
27     if $i == maxitr$ then
28         $num\_stop = num\_stop + 1$;
29         SEND_ALL($maxitr\_stop$);
30     fi;
31     $received$ = VERIFY_RECEIVING($flag$);
32     if $received$ then
33         if $flag == look4\_stop$ then break;
34         else if $flag == maxitr\_stop$ then $num\_stop = num\_stop + 1$ fi;
35     fi;
36     if $num\_stop == nprocs$ then break fi;
37 rof;
38 POSTOPT(POOL);
39 $S_{GlobalBest}$ = GET_GLOBAL_BEST($S_{best}$);
40 return ($S_{GlobalBest}$);
end NON-COLLAB_GRASP_PR;
```

FIGURE 15. Pseudo-code for the non-collaborative parallel GRASP with path-relinking.

```
procedure COLLAB_GRASP_PR($\mathcal{M}, \mathcal{J}, \mathcal{O}, p$, seed, look4, maxitr, maxpool, freq)
1  $my\_rank$ = GET_RANK(); $nprocs$ = GET_NUM_PROCS();
2  for $i = 1, \ldots, (maxitr/nprocs) * my\_rank$ do
3      seed = rand(seed);
4  rof;
5  $P = \emptyset$; $num\_stop = 0$;
6  for $i = 1, \ldots, \infty$ do
7      if  mod $(i, 2) == 0$ then
8          GREEDY_MASKESPAN($seed, S, \mathcal{M}, p, |\mathcal{M}|, |\mathcal{J}|, Makespan$);
9      else
10         GREEDY_TIME_REMAINING($seed, S, \mathcal{M}, p, |\mathcal{M}|, |\mathcal{J}|, Makespan$);
11     fi;
12     LOCAL($S, \mathcal{M}, p, |\mathcal{M}|, |\mathcal{J}|, Makespan$);
13     if $|P| ==$ maxpool then
14         $accepted$ = VERIFY_QUALITY($S, i$);
15         if $accepted$ then
16             for $T \in P' \subseteq P$ do
17                 RECEIVE_SOLUTIONS($P$);
18                 $S_{gmin}$ = PATH_RELINKING($\mathcal{M}, \mathcal{J}, \mathcal{O}, p, Makespan, S, T$);
19                 $updated$ = UPDATE_POOL($S_{gmin}, c_{gmin}, P$);
20                 if ($updated$) then INSERT_SEND_BUFFER($S_{gmin}, c_{gmin}, buffer$) fi;
21                 RECEIVE_SOLUTIONS($P$);
22                 $S_{gmin}$ = PATH_RELINKING($\mathcal{M}, \mathcal{J}, \mathcal{O}, p, Makespan, T, S$);
23                 $updated$ = UPDATE_POOL($S_{gmin}, c_{gmin}, P$);
24                 if ($updated$) then INSERT_SEND_BUFFER($S_{gmin}, c_{gmin}, buffer$) fi;
25             rof;
26             SEND_SOLUTIONS($buffer$);
27         fi;
28     else $P = P \cup \{S\}$ fi;
29     if  mod $(i$, ifreq$) == 0$ then INTENSIFY($P$) fi;
30     $S_{best}$ = POOLMIN($P$);
31     if MAKESPAN($S_{best}$) $\leq$ look4 then SEND_ALL($look4\_stop$) fi;
32     if $i == maxitr$ then
33         $num\_stop = num\_stop + 1$;
34         SEND_ALL($maxitr\_stop$)
35     fi;
36     $received$ = VERIFY_RECEIVING($flag$);
37     if $received$ then
38         if $flag == look4\_stop$ then break;
39         else if $flag == maxitr\_stop$ then $num\_stop = num\_stop + 1$ fi;
40     fi;
41     if $num\_stop == nprocs$ then break fi;
42 rof;
43 POSTOPT(POOL);
44 $S_{GlobalBest}$ = GET_GLOBAL_BEST($S_{best}$);
45 return ($S_{GlobalBest}$);
end COLLAB_GRASP_PR;
```

FIGURE 16. Pseudo-code for collaborative parallel GRASP with path-relinking.

TABLE 3. Probability estimates of finding a solution at least as good as the target solution, as a function of maximum solution time for `GP+C-INT` and `GP+PR`. Instances are `abz6`, `mt10`, `orb5` and `la21`, with target values 965, 960, 930 and 1130, respectively.

	`abz6`		`mt10`		`orb5`		`la21`	
time	`GP+C-INT`	`GP+PR`	`GP+C-INT`	`GP+PR`	`GP+C-INT`	`GP+PR`	`GP+C-INT`	`GP+PR`
100s	.49	1.00	.04	.02	.16	.95	.09	.53
500s	.94	1.00	.17	.90	.46	1.00	.37	1.00
1000s	1.00	1.00	.27	1.00	.75	1.00	.55	1.00
1500s	1.00	1.00	.34	1.00	.86	1.00	.69	1.00

TABLE 4. Experimental results on problem classes `abz`, `car`, `mt`, and `orb`. Table shows problem name, problem dimension (jobs and machines), the best known solution (BKS), the best solution found by `GP+C-INT`, total number of `GP+PR` iterations performed, CPU time per 1000 `GP+PR` iterations, the best solution found by `GP+PR`, and the relative percentage error of the `GP+PR` solution with respect to the BKS.

					`GP+PR`			
problem	$\lvert\mathcal{J}\rvert$	$\lvert\mathcal{M}\rvert$	BKS	`GP+C-INT`	iterations ($\times 10^6$)	time (10^3 iter)	solution	error (%)
`abz5`	10	10	1234	1238	1.0	2.53s	1234	0.0
`abz6`	10	10	943	947	0.3	2.26s	943	0.0
`abz7`	15	20	665	723	50.0	11.66s	692	4.1
`abz8`	15	20	670	729	10.0	49.78s	705	5.2
`abz9`	15	20	691	758	1.0	875.92s	740	7.1
`car1`	11	5	7038	7038	0.001	15.31s	7038	0.0
`car2`	13	4	7166	7166	0.001	52.14s	7166	0.0
`car3`	12	5	7312	7366	50.7	2.08s	7312	0.0
`car4`	14	4	8003	8003	0.01	2.79s	8003	0.0
`car5`	10	6	7702	7702	0.5	4.40s	7702	0.0
`car6`	8	9	8313	8313	0.01	11.85s	8313	0.0
`car7`	7	7	6558	6558	0.001	16.05s	6558	0.0
`car8`	7	7	8264	8264	0.02	4.66s	8264	0.0
`mt06`	6	6	55	55	0.00001	1.74s	55	0.0
`mt10`	10	10	930	938	2.5	4.05s	930	0.0
`mt20`	20	5	1165	1169	4.5	46.48s	1165	0.0
`orb1`	10	10	1059	1070	1.2	46.75s	1059	0.0
`orb2`	10	10	888	889	1.1	11.45s	888	0.0
`orb3`	10	10	1005	1021	6.5	33.32s	1005	0.0
`orb4`	10	10	1005	1031	100.0	1.94s	1011	0.6
`orb5`	10	10	887	891	20.0	14.61s	889	0.2
`orb6`	10	10	1010	1013	3.5	43.75s	1012	0.2
`orb7`	10	10	397	397	0.03	18.72s	397	0.0
`orb8`	10	10	899	909	1.6	24.26s	899	0.0
`orb9`	10	10	934	945	11.1	4.38s	934	0.0
`orb10`	10	10	944	953	0.3	33.25s	944	0.0

`etime`. 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 2, 4, 8, and 16 parallel processor runs. Timing in

TABLE 5. Experimental results on problem class `la`. Table shows problem name, problem dimension (jobs and machines), the best known solution (BKS), the best solution found by `GP+C-INT`, total number of `GP+PR` iterations performed, CPU time per 1000 `GP+PR` iterations, the best solution found by `GP+PR`, and the relative percentage error of the `GP+PR` solution with respect to the BKS.

					`GP+PR`			
					iterations	time		error
problem	$\lvert\mathcal{J}\rvert$	$\lvert\mathcal{M}\rvert$	BKS	`GP+C-INT`	$(\times 10^6)$	(10^3 iter)	solution	(%)
`la01`	10	5	666	666	0.0001	0.82s	666	0.0
`la02`	10	5	655	655	0.004	2.74s	655	0.0
`la03`	10	5	597	604	0.01	2.95s	597	0.0
`la04`	10	5	590	590	0.001	15.71s	590	0.0
`la05`	10	5	593	593	0.0001	1.09s	593	0.0
`la06`	15	5	926	926	0.0001	8.00s	926	0.0
`la07`	15	5	890	890	0.0001	1.58s	890	0.0
`la08`	15	5	863	863	0.0003	5.49s	863	0.0
`la09`	15	5	951	951	0.0001	3.40s	951	0.0
`la10`	15	5	958	958	0.0001	11.50s	958	0.0
`la11`	20	5	1222	1222	0.0001	3.14s	1222	0.0
`la12`	20	5	1039	1039	0.0001	2.89s	1039	0.0
`la13`	20	5	1150	1150	0.0001	3.20s	1150	0.0
`la14`	20	5	1292	1292	0.0001	5.70s	1292	0.0
`la15`	20	5	1207	1207	0.0002	122.75s	1207	0.0
`la16`	10	10	945	946	1.3	2.27s	945	0.0
`la17`	10	10	784	784	0.02	3.29s	784	0.0
`la18`	10	10	848	848	0.05	9.07s	848	0.0
`la19`	10	10	842	842	0.02	15.14s	842	0.0
`la20`	10	10	902	907	17.0	1.63s	902	0.0
`la21`	15	10	1047	1091	100.0	3.51s	1057	1.0
`la22`	15	10	927	960	26.0	3.45s	927	0.0
`la23`	15	10	1032	1032	0.01	39.39s	1032	0.0
`la24`	15	10	935	978	125.0	3.26s	954	2.0
`la25`	15	10	977	1028	32.0	3.29s	984	0.7
`la26`	20	10	1218	1271	3.5	6.35s	1218	0.0
`la27`	20	10	1235	1320	10.5	27.51s	1269	2.8
`la28`	20	10	1216	1293	20.0	17.77s	1225	0.7
`la29`	20	10	1157	1293	50.0	6.17s	1203	4.0
`la30`	20	10	1355	1368	3.0	7.61s	1355	0.0
`la31`	30	10	1784	1784	0.01	267.60s	1784	0.0
`la32`	30	10	1850	1850	0.0001	12.66s	1850	0.0
`la33`	30	10	1719	1719	0.001	875.11s	1719	0.0
`la34`	30	10	1721	1753	0.05	80.33s	1721	0.0
`la35`	30	10	1888	1888	0.01	348.32s	1888	0.0
`la36`	15	15	1268	1334	51.0	5.54s	1287	1.5
`la37`	15	15	1397	1457	20.0	12.51s	1410	0.9
`la38`	15	15	1196	1267	20.0	32.87s	1218	1.8
`la39`	15	15	1233	1290	6.0	59.06s	1248	1.2
`la40`	15	15	1222	1259	2.0	104.18s	1244	1.8

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

Table 6. Experimental results: Overall solution quality by problem class. Sum of all best known solutions (BKS) for each class is compared with sum of best `GP+PR` solutions. Relative error is of `GP+PR` solution with respect to BKS.

problem	sum of BKS	sum of `GP+PR` sol.	relative error (%)
`abz`	4203	4314	2.64
`car`	60356	60356	0.00
`mt`	2150	2150	0.00
`orb`	9028	9038	0.11
`la`	44297	44513	0.49

Table 7. Experimental results: Percentage of `GP+PR` solutions within a tolerance of the best known solution (BKS).

	percentage of `GP+PR` solutions within					
problem	0% of BKS	.5% of BKS	1% of BKS	2% of BKS	5% of BKS	10% of BKS
`abz`	40.0	40.0	40.0	40.0	60.0	100.0
`car`	100.0	100.0	100.0	100.0	100.0	100.0
`mt`	100.0	100.0	100.0	100.0	100.0	100.0
`orb`	70.0	90.0	100.0	100.0	100.0	100.0
`la`	72.5	72.5	82.5	95.0	100.0	100.0

5.2. **Test Problems.** The experiments were done on 66 instances from five classes of standard JSP test problems: `abz`, `car`, `la`, `mt`, and `orb`. The problem dimensions vary from 6 to 30 jobs and from 4 to 20 machines. All instances tested were downloaded from Beasley's OR-Library [1] [5].

5.3. **The sequential experiments.** The goal of the sequential experiments was to observe the general behavior of the implementations of the proposed algorithms. In these experiments, we present results comparing the following heuristics:

1. `GRASP`: Pure GRASP alternating between makespan and time-remaining randomized greedy constructions;
2. `GP+PR`: GRASP with path-relinking described in Section 3;
3. `GP+C-INT`: The GRASP with construction intensification and POP, described in Binato et al. [6].

We aim to verify how the solutions obtained by `GP+PR` compare to the best known solutions for a set of standard test problems. To illustrate the effectiveness of the proposed hybrid GRASP, the solution times to target solution of `GP+PR` are compared to the solution times of `GRASP` and `GP+C-INT`.

On all sequential (and parallel) implementations tested in this paper, the restricted candidate list parameter α is chosen at random from the uniform distribution in the interval $[0, 1]$ at each GRASP iteration and remains fixed throughout the iteration.

For the experiments performed with `GP+PR`, we used a pool of size $|P|$=30 and a differentiation factor for insertion into the pool of *dif*=25%. In all experiments done with `GP+PR`, path-relinking was applied between the solution obtained by GRASP and all solutions in the pool. The standard deviation used to verify if a solution

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

obtained by the local search will take part in path-relinking is computed from the costs of the first n=10,000 GRASP solutions. In the runs of `GP+PR`, used to generate the plots shown in this paper, intensification and post-optimization are not applied. In the runs of `GP+PR` shown in Tables 4 and 5, the intensification is applied after each interval of *freq*=500,000 iterations.

In all experiments performed with `GP+C-INT`, the program was configured to use the same parameter values used in the tests reported in [6]. Therefore, a pool of 30 elite solutions was used in the intensification phase and a differentiation factor of *dif*=80% was used to control the insertion into the pool. POP was activated with a parameter `freq` = 40, i.e., it was applied after the construction of 40% and 80% of the partial solution. A linear distribution function was used to bias the selection of candidate elements in the RCL.

To study the effect of path-relinking on GRASP, we compared `GRASP` and `GP+PR` on problems `abz6`, `mt10`, `orb5`, and `la21`. Two hundred independent runs of the two heuristics were done for each of the four problems. Execution was interrupted when a solution with cost at least as good as `look4` was found. The `look4` values used for problems `abz6`, `mt10`, `orb5`, and `la21` were 947, 950, 910, and 1110, respectively. These values are far from the optimal values, and in general, can be obtained after a few iterations. Figure 3 shows the empirical distributions for solution time of `GRASP` and `GP+PR`. To plot these empirical distributions, we associate with the i-th sorted running time (t_i) a probability $p_i = (i - \frac{1}{2})/200$. The points $z_i = (t_i, p_i)$ are then plotted for $i = 1, \ldots, 200$. We observe in the plots that `GP+PR` finds the target solution faster than `GRASP`. Table 1 shows estimated probabilities of finding the target solution as a function of CPU time for the four instances tested. For example, for a computational time of at most 500 seconds, the estimated probability of finding a solution at least as good as the target solution for problem `abz6` is 93% for `GP+PR`, while for `GRASP` it is 48%. On problem `la21`, the estimated probability of finding a solution at least as good as the target solution in time at most 1000 seconds is 56% for `GRASP` and 100% for `GP+PR`. These results illustrate for the JSP, the fact observed in [2], that although each iteration of a hybrid approach of GRASP with path-relinking takes more computational time when compared to an iteration of a pure GRASP, it is compensated by the reduced number of iterations needed to find the target solution.

Figure 4 shows a comparison between `GRASP` and `GP+PR` for problem `mt10`, using three target values: 970, 960, and 950. These target values are 4.3%, 3.2%, and 2.1% away from the best known value, respectively. The figure is composed of three plots, where the difficulty of obtaining the target solution grows from top to bottom. The empirical distributions are plotted the same way as in the plots of Figure 3. For the topmost plot, we observe that `GRASP` finds the target solution before `GP+PR` for probabilities below 68%. This occurs because the target value sought is easy to find and `GRASP` iterations are faster than `GP+PR` iterations. By gradually increasing the difficulty to find the target values on the two remaining plots (middle and bottom), we observe that the probabilities for which `GRASP` still finds the target solution faster than `GP+PR` decrease to 19% and 9%, respectively. Table 2 shows the computational times for `GRASP` and `GP+PR` to find each of the three target values with probabilities 20%, 50%, and 80%. For each pair of GRASP variant and target value, the table also shows the percentage reduction in solution time of `GP+PR` with respect to the solution time of `GRASP`, as a function of the probability. The difficulty to obtain the target solution grows from left to right and top to bottom in the table. We observe, that as the target value approaches the best known value, the percentage reduction in solution time of GP+PR grows. For example, for a target value of 960, and a probability of 50%, the percentage reduction in solution time is 55.6%. Decreasing the target value to 950, the percentage reduction in solution time increases to 72.3% for the same probability of 50%.

Variants GP+PR and GP+C-INT are compared in Figure 5. The empirical distributions are plotted for these GRASPs using the same methodology used to plot the empirical distributions in Figures 3 and 4. The same test problems, abz6, mt10, orb5, and la21, are used in this experiment, with target values 965, 960, 930, and 1130, respectively. Notice that the target values used in this experiment are easier to obtain than the target values used to compare GRASP and GP+PR for the same four instances. This was necessary because of the high computational time needed for GP+C-INT to obtain target solutions of quality comparable to the quality of the solutions found in the first experiment. Table 3 shows, for GP+C-INT and GP+PR, estimates of probabilities of finding a solution with cost at least as good as the target value, as a function of maximum solution time. For example, for problem la21, we observe that the estimated probability for GP+C-INT to obtain a solution with cost at most 1130 in less than 500 seconds is 37%, while for GP+PR this probability is 100%. For problem mt10, we observe that the estimated probabilities for GP+C-INT and GP+PR to find the target solution in less than 1000 seconds are 27% and 100%, respectively. Therefore, we verify that the use of intensification in a GRASP shows better results when this phase is carried out after the local search phase, i.e., after a pure GRASP iteration. This happens because a premature intensification, i.e., an intensification phase done during the GRASP construction phase, might reduce drastically the number of local minima visited during a run of GRASP.

To verify the behavior of the proposed algorithm in terms of solution quality, GP+PR was extensively executed for all test problems considered. The number of GRASP iterations was frequently in the millions (where each GRASP iteration uses a different seed of the random number generator). These results are shown in Tables 4 and 5. Each table shows problem name, problem dimension (number of jobs and machines), the best known solution (BKS), the cost of the solution found by GP+C-INT, and, for GP+PR, the total number of GRASP iterations executed, CPU time in seconds to run 1000 GRASP iterations, the cost of the best solution found, and the percentage relative error of the GP+PR solution with respect to the BKS.

Of the 66 tested instances, GP+PR found the BKS in 49 cases (74.2%). It found a solution within 0.5% of the BKS for 50 instances (75.7%). In 56 instances (84.8%), GP+PR solution was within 1% of the BKS and in 61 cases (92.4%) it was within 2% of the BKS. GP+PR solution was within 5% of the BKS in 64 instances (97%), while for all other cases, the solution found was within 7.5% of the BKS.

Tables 6 and 7 summarizes the results for each problem class. Table 6 shows, for each problem class, its name, the sum of the BKS values, the sum of the values of the best solutions found by GP+PR, and the percentage relative error of the sum of the values of the best GP+PR solutions with respect to the sum of the BKS values. Table 7 shows, for each problem class, its name, and the percentage of instances for which a GP+PR solution within 0%, 0.5%, 1%, 2%, 5%, and 10% of the BKS was produced. From these tables, one can conclude that the easiest classes are car and mt, for which GP+PR obtained the BKS for all instances. For classes orb and la, the average relative errors are within 0.5% of the BKS and therefore, GP+PR was capable of producing solutions of high quality for most problems in these classes.

The most difficult class was `abz`, where the average relative error with respect to the BKS achieved 2.64%.

5.4. **Probability distribution for solution time.** Aiex, Resende, and Ribeiro [3] 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 [11] was used to compute the empirical and theoretical distributions and to estimate the parameters of the distributions. We use the same methodology to study *time to target value* for `GRASP` and `GP+PR`. Our objective is to show that these variants of GRASP have time to target value distributions that fit a two-parameter exponential distribution.

The quantile-quantile plots (Q-Q plots) and the plots showing the empirical and theoretical distributions of the random variable *time to target solution* for `GRASP` are shown in Figures 6 and 7, respectively. Analogously, Figures 8 and 9 show the Q-Q plots and the plots with the empirical and theoretical distributions of the random variable *time to target solution* for `GP+PR`. Three target values are considered for each of the test problems, `abz6`, `mt10`, `orb5`, and `la21`, for the two GRASP variants. All plots are computed with 200 runs of the GRASP 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.

Figures 6 and 8 are made up of 12 quantile-quantile plots, one for each pair of problem instance/target value for `GRASP` and `GP+PR`, respectively. Analogously, Figures 7 and 9 are made up of 12 plots showing the empirical and theoretical distributions of the random variable *time to target solution*, each corresponding to a pair of problem instance/target value for `GRASP` and `GP+PR`, respectively. Each figure is made up of four rows, each corresponding to a different problem. Each row of the figure depicts three plots, where the difficulty to find the target value increases from left to right. Our description of each plot follows [3] 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 $i = 1, \dots, 200$.

Tables 9 and 12 show the target values and the parameters estimated by the methodology for `GRASP` and `GP+PR`, respectively. Following the methodology proposed in [11], we first draw 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 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 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 [11]. 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, $i = 1, \ldots, 200$, of the Q-Q plot is

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

Figures 6 and 8 show that there is little departure from straightness in the Q-Q plots for `GRASP`, as well as for `GP+PR`. We also observe that as the difficulty of finding the target value increases, the plotted points become more fitted to the estimated line. Therefore, we verify that the distributions fit a two-parameter exponential distribution.

Binato et al. [6] show that the probability distribution of solution time of `GP+C-INT` fits a two-parameter exponential distribution. In this section, we show that the probability distributions of solution time of a GRASP where the construction phase is computed alternating between two greedy functions (`GRASP`) and of a GRASP

TABLE 8. Speedup with respect to a single processor implementation and efficiency (speedup divided by number of processors). Algorithm is the parallel implementation of `GRASP`. Instances are `abz6`, `mt10`, `orb5`, and `la21`, with target values 960, 960, 920, and 1120, respectively.

	number of processors							
	2		4		8		16	
problem	speedup	eff.	speedup	eff.	speedup	eff.	speedup	eff.
`abz6`	2.04	1.02	4.75	1.18	8.87	1.10	19.17	1.19
`mt10`	1.62	.81	4.07	1.01	7.34	.91	14.81	.92
`orb5`	2.12	1.06	3.97	.99	7.63	.95	14.10	.88
`la21`	1.94	.97	4.98	1.24	8.13	1.01	19.63	1.22
average:	**1.93**	**.96**	**4.44**	**1.10**	**7.99**	**.99**	**16.92**	**1.05**

with path-relinking restricted to iterations where the local search obtained high quality solutions (`GP+PR`) also fit a two-parameter exponential distribution. These results reinforce the conclusions drawn in [3] for pure GRASPs.

The following can be stated for a two parameter (shifted) exponential distribution [3, 39]. 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, 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.

5.5. **The parallel experiments.**

The parallel algorithms used in these experiments are:

1. the pure GRASP;
2. the non-collaborative GRASP with path-relinking;
3. the collaborative GRASP with path-relinking.

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. The parallel `GRASP` was studied for problems `abz6`, `mt10`, `orb5`, and `la21`, with `look4` values 960, 960, 920, and 1120, respectively. The independent and cooperative parallel implementations of `GP+PR` were also tested for problems `abz6`, `mt10`, `orb5`, and `la21`, but with more difficult `look4` values 943, 938, 895, and 1100, respectively. The parameters of the procedures used in the parallel approaches were the same used for testing the sequential algorithm. Intensification and post-optimization are not carried out during the experiments with the parallel implementations. Figure 10 shows speedup and empirical distributions for the parallel implementations of `GRASP`. Analogously, Figures 11, 12, 13, and 14 show speedup and empirical distributions for both parallel implementations of `GP+PR`.

Table 9. Test problems used to study the empirical probability distributions of the random variable time to target solution of `GRASP`. Table shows for each tested problem, cost of the BKS, target value and estimated parameters.

			estimated parameters	
problem	BKS	target	$\hat{\mu}$	$\hat{\lambda}$
`abz6`	943	970	0.203	3.804
		960	0.428	15.567
		950	-1.323	68.490
`mt10`	930	970	-9.403	233.092
		960	11.723	885.034
		950	109.528	1827.882
`orb5`	887	930	-0.783	15.757
		920	1.249	38.273
		910	-1.011	191.111
`la21`	1047	1130	-4.115	50.343
		1120	-1.015	206.836
		1110	-87.594	1268.081

Table 10. 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 is the parallel implementation of `GRASP`. Instances are `abz6`, `mt10`, `orb5`, and `la21`, with target values 960, 960, 920, and 1120, respectively.

		probab. parallel GRASP number of processors				
problem	time	1	2	4	8	16
`abz6`	10s	.34	.67	.93	1.00	1.00
	20s	.61	.90	.98	1.00	1.00
	50s	.90	.98	1.00	1.00	1.00
`mt10`	10s	.04	.02	.04	.07	.19
	100s	.16	.20	.45	.64	.82
	500s	.46	.60	.92	1.00	1.00
`orb5`	10s	.20	.37	.62	.87	.99
	20s	.36	.62	.84	.96	1.00
	50s	.70	.94	1.00	1.00	1.00
`la21`	10s	.04	.08	.18	.30	.54
	100s	.29	.57	.87	.96	1.00
	500s	.89	.97	1.00	1.00	1.00

The plots were generated with 60 independent runs for each number of processors considered (1, 2, 4, 8, and 16 processors).

Table 8 summarizes the speedups shown in the plots. The table also shows efficiency (speedup divided by number of processors) values. Speedups are on average approximately linear. Table 9 shows the values of the parameters μ and λ of the two-parameter exponential distributions plotted for the pairs of instance/target values used to study the behavior of `GRASP`. The parameter μ is an estimate of the minimum time needed for `GRASP` to find the target value for the instances. The parameter λ is an estimate of the spread of the measured times for pair of instance/target value. The sum $\mu + \lambda$ is an estimate of the average solution time for

Table 11. Speedup with respect to a single processor implementation. Algorithms are independent and cooperative implementations of GP+PR. Instances are abz6, mt10, orb5, and la21, with target values 943, 938, 895, and 1100, respectively.

	speedup independent (number of processors)				speedup cooperative (number of processors)			
problem	2	4	8	16	2	4	8	16
abz6	2.00	3.36	6.44	10.51	2.40	4.21	11.43	23.58
mt10	1.57	2.12	3.03	4.05	1.75	4.58	8.36	16.97
orb5	1.95	2.97	3.99	5.36	2.10	4.91	8.89	15.76
la21	1.64	2.25	3.14	3.72	2.23	4.47	7.54	11.41
average:	**1.79**	**2.67**	**4.15**	**5.91**	**2.12**	**4.54**	**9.05**	**16.93**

Table 12. Test problems used to study the empirical probability distributions of the random variable time to target solution of GP+PR. Table shows for each tested problem, cost of the BKS, target value and estimated parameters.

			estimated parameters	
problem	BKS	target	$\hat{\mu}$	$\hat{\lambda}$
abz6	943	950	25.067	40.348
		947	30.652	140.487
		943	92.220	744.247
mt10	930	950	206.950	249.865
		940	255.666	334.774
		938	305.281	524.236
orb5	887	910	47.322	44.268
		900	82.006	200.680
		895	131.435	418.053
la21	1047	1110	140.530	155.441
		1105	140.399	248.812
		1100	181.539	390.571

pair of instance/target value. For a small value of parameter μ, a two-parameter exponential distributions can be approximated by a simple exponential distribution. Therefore, approximate linear speedups were expected for the parallel GRASP on this set of instance/target values.

Table 10 shows, for given running times, the estimated 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 920 on problem orb5 in at most 10 seconds, goes from 20% with one processor, to 62% with four processors, and to 99% with sixteen processors.

Table 11 summarizes the speedups shown in the plots for the independent and cooperative parallel approaches of GP+PR. Sublinear speedups are observed for the independent approach. Table 12 shows the values of parameters μ and λ of the two-parameter exponential distributions plotted for the pairs of instance/target values used to study GP+PR. We notice that the ratios λ/μ computed with the parameters in this table are much lower than the values of λ/μ, for the parameters estimated for the pairs of instance/target values used to study GRASP. As stated

TABLE 13. 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. Algorithms are independent and cooperative implementations of `GP+PR`. Instances are `abz6`, `mt10`, `orb5`, and `la21`, with target values 943, 938, 895, and 1100, respectively.

		probab. independent (number of processors)					probab. cooperative (number of processors)				
problem	time	1	2	4	8	16	1	2	4	8	16
`abz6`	100s	.01	.03	.12	.25	.47	.01	.12	.24	.64	.94
	500s	.30	.59	.79	.95	1.00	.30	.61	.79	1.00	1.00
	1000s	.57	.85	.97	.99	1.00	.57	.92	.98	1.00	1.00
`mt10`	100s	0.0	0.0	0.0	.02	0.05	0.0	0.0	.05	.34	.82
	500s	.17	.32	.55	.82	.98	.17	.49	.95	1.00	1.00
	1000s	.54	.79	.95	1.00	1.00	.54	.85	1.00	1.00	1.00
`orb5`	100s	0.0	0.0	.02	.03	.19	0.0	.07	.42	.74	.92
	500s	.35	.75	.93	1.00	1.00	.35	.80	.97	1.00	1.00
	1000s	.75	.97	1.00	1.00	1.00	.75	.95	1.00	1.00	1.00
`la21`	100s	0.0	0.0	0.0	.02	.06	0.0	.02	.08	.44	.87
	500s	.29	.52	.82	.98	1.00	.29	.79	1.00	1.00	1.00
	1000s	.75	.98	1.00	1.00	1.00	.75	.98	1.00	1.00	1.00

before, although `GP+PR` finds the target solution faster than `GRASP`, its iterations need higher CPU times, which corresponds to higher values of μ. Path-relinking also speedups GRASP, reducing the spread of the solution time, i.e., the parameter λ. Therefore, μ values are higher and λ values are lower for `GP+PR` with respect to `GRASP` parameters. For these reasons, the distributions plotted for `GP+PR` cannot be approximated by a simple exponential distribution. As noted in the observation about the two-parameter exponential distribution, as the number of used processors ρ increases, the speedup of the algorithm degrades. That observation does not take into account sharing of information by the processes. Therefore, no conclusions from the distributions plotted for the sequential `GP+PR` can be drawn for the cooperative approach. However, we observe an approximate linear speedup for all instances tested for the cooperative approach, outperforming the independent variant.

In Table 13, the estimated probability of finding a solution at least as good as the target solution before a specified time is shown as a function of number of processors. For example, the table shows, for problem `mt10`, that the probability of finding a solution of value at least as good as 938 in at most 500 seconds, goes from 32% with two processor, to 55% with four processors, and to 98% with 16 processors, on the independent approach. For the cooperative approach, these values increase to 49%, 95% and 100%, for two, four and sixteen processors, respectively.

6. CONCLUDING REMARKS

We describe a new algorithm for finding approximate solutions to the job shop scheduling problem. This GRASP uses some of the ideas proposed in the GRASP of Binato et al. [6]. That GRASP applies an intensification strategy during the construction phase that uses information obtained from "good" solutions to implement a memory-based procedure to influence the construction phase. In the hybrid GRASP proposed in this paper, the intensification phase is moved to the end of each GRASP iteration and is done using path-relinking. Due to the high computational requirements of path-relinking, only solutions accepted by a quality criteria undergo this procedure. Furthermore, the new GRASP alternates between two semi-greedy algorithms to construct solutions, which produces a higher variety of initial solutions for the local search. The algorithm was evaluated on 66 standard test problems and was shown to produce optimal or near-optimal solutions on all instances.

We observe that the hybrid GRASP with path-relinking obtains a solution of a given quality faster than the pure GRASP. Therefore, the increase in the computational time of each GRASP iteration due to the computation of path-relinking is compensated by an increase in the method's robustness. We also verify that the intensification applied after each GRASP iteration using path-relinking outperforms the intensification strategy used in Binato et al., which is applied during the construction phase.

We verify that the time to target sub-optimal solution of the proposed GRASPs fit well a two-parameter exponential distribution. Two parallelization strategies were proposed for the GRASP with path-relinking: an independent and a cooperative. The independent parallel strategy, as expected, shows a sub-linear speedup. The cooperative approach shows an approximate linear speedup for all instances tested, thus attesting that the extra time spent in communication among processes is compensated by an increase in solution quality.

References

[1] J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34:391–401, 1988.

[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, AT&T Labs-Research, 2000.

[3] 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.

[4] D. Applegate and W.Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3:149–156, 1991.

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

[6] S. Binato, W.J. Hery, D. 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 59–79. Kluwer Academic Publishers, 2001.

[7] P. Brucker, B. Jurisch, and B. Sievers. A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49:105–127, 1994.

[8] 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, 2001.

[9] J. Carlier and E. Pinson. An algorithm for solving the job-shop problem. *Management Science*, 35:164–176, 1989.

[10] J. Carlier and E. Pinson. A practical use of Jackson's preemptive schedule for solving the job-shop problem. *Annals of Operations Research*, 26:269–287, 1990.

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

[12] L. Davis. Job shop scheduling with genetic algorithms. In *Proceedings of the First International Conference on Genetic Algorithms and their Applications*, pages 136–140. Morgan Kaufmann, 1985.

[13] 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.

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

[15] P. Festa and M.G.C. Resende. GRASP: An annotated bibliography. In C.C. Ribeiro and P. Hansen, editors, *Essays and surveys on metaheuristics*, pages 325–367. Kluwer Academic Publishers, 2001.

[16] H. Fisher and G. L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In J. F. Muth and G. L. Thompson, editors, *Industrial Scheduling*, pages 225–251. Prentice Hall, Englewood Cliffs, NJ, 1963.

[17] 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.

[18] S. French. *Sequencing and scheduling:An introduction to the mathematics of the job-shop*. Horwood, 1982.

[19] B. Giffler and G. L. Thompson. Algorithms for solving production scheduling problems. *Operations Research*, 8:487–503, 1960.

[20] 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.

[21] 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.

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

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

[24] A. S. Jain and S. Meeran. A state-of-the-art review of job-shop scheduling techniques. Technical report, Department of Applied Physics, Electronic and Mechanical Engineering, University of Dundee, Dundee, Scotland, 1998.

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

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

[27] H.R. Lourenço. Local optimization and the job-shop scheduling problem. *European Journal of Operational Research*, 83:347–364, 1995.

[28] H.R. Lourenço and M. Zwijnenburg. Combining the large-step optimization with tabu-search: Application to the job-shop scheduling problem. In I.H. Osman and J.P. Kelly, editors, *Meta-Heuristics: Theory and Apllications*, pages 219–236. Kluwer Academic Publishers, 1996.

[29] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42:797–813, 1996.

[30] E. Pinson. The job shop scheduling problem: A concise survey and some recent developments. In P. Chrétienne, E.G. Coffman Jr., J.K. Lenstra, and Z. Liu, editors, *Scheduling theory and its application*, pages 277–293. John Wiley and Sons, 1995.

[31] M.G.C. Resende and C.C. Ribeiro. A grasp with path-relinking for permanent virtual circuit routing. Technical report, AT&T Labs Research, 2001.

[32] M.G.C. Resende and C.C. Ribeiro. Greedy randomized adaptive search procedures. Technical report, AT&T Labs Research, Florham Park, NJ, 2001. To appear in *State-of-the-Art Handbook in Metaheuristics*, F. Glover and G. Kochenberger, eds., Kluwer Academic Publishers.

[33] C.C. Ribeiro, E. Uchoa, and R.F. Werneck. A hybrid GRASP with perturbations for the Steiner problem in graphs. Technical report, Computer Science Department, Catholic University of Rio de Janeiro, 2001.

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

[35] 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.

[36] E. D. Taillard. Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing*, 6:108–117, 1994.

[37] R. J. M. Vaessens, E. H. L. Aarts, and J. K. Lenstra. Job shop scheduling by local search. *INFORMS Journal on Computing*, 8:302–317, 1996.

[38] P. J. M. Van Laarhoven, E. H. L. Aarts, and J. K. Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40:113–125, 1992.

[39] M.G.A. Verhoeven and E.H.L. Aarts. Parallel local search. *J. of Heuristics*, 1:43–66, 1995.

(R.M. Aiex) Catholic University of Rio de Janeiro, Rio de Janeiro, RJ 22451 Brazil.
E-mail address, R.M. Aiex: `rma@inf.puc-rio.br`

(S. Binato) Electrical Energy Research Center (CEPEL), P.O. Box 68007, Rio de Janeiro, RJ 21944-970 Brazil.
E-mail address, S. Binato: `silvio@cepel.br`

(M.G.C. Resende) Information Sciences Research, AT&T Labs Research, Florham Park, NJ 07932 USA.
E-mail address, M.G.C. Resende: `mgcr@research.att.com`