PARALLEL GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURES

MAURICIO G.C. RESENDE AND CELSO C. RIBEIRO

ABSTRACT. A GRASP (Greedy Randomized Adaptive Search Procedure) is a metaheuristic for producing good-quality solutions of combinatorial optimization problems. It is usually implemented with a construction procedure based on a greedy randomized algorithm followed by local search. In this Chapter, we survey parallel implementations of GRASP. We describe simple strategies to implement independent parallel GRASP heuristics and more complex cooperative schemes using a pool of elite solutions to intensify the search process. Some applications of independent and cooperative parallelizations are presented in detail.

1. INTRODUCTION

Metaheuristics are high level procedures that coordinate simple heuristics, such as local search, to find solutions that are of better quality than those found by the simple heuristics alone. One such metaheuristic is GRASP (Greedy Randomized Adaptive Search Procedure) [23, 24, 26, 55]. A GRASP is a multi-start procedure, where each iteration usually consists of two phases: construction and local search. The construction phase produces a feasible solution that is used as the starting point for local search. The multi-start procedure returns the best local optimum found.

In the GRASP construction phase, a feasible solution is built, one element at a time. For example, a spanning tree is built one edge at a time; a schedule is built one operation at a time; and a clique is built one vertex 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. When building a spanning tree, for example, the candidate elements are those yet unselected edges whose inclusion in the solution does not result in a cycle. 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 at least as good as 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 function and the set of candidate elements used to determine the next RCL. The construction procedure terminates when the set of candidate elements is empty, obtaining a feasible solution. Algorithm 1 shows a GRASP in pseudo-code form,

Date: December 6, 2004.

Key words and phrases. Combinatorial optimization, local search, GRASP, path-relinking, parallel algorithm.

AT&T Labs Research Technical Report TD-67EKXH. To appear in *Parallel Metaheuristics*, E. Alba (ed.), John Wiley and Sons, 2005.

where the objective function $f(x)$ is minimized over the set X. The GRASP runs for `MaxIterations` iterations. The best solution returned is x^*, with $f(x^*) = f^*$.

Data : Number of iterations `MaxIterations`
Result: Solution $x^* \in X$
$f^* \leftarrow \infty$;
for $i = 1, \ldots, i_{\max}$ **do**
  $x \leftarrow$ `GreedyRandomizedConstruction`();
  $x \leftarrow$ `LocalSearch`(x);
  if $f(x) < f^*$ **then**
    $f^* \leftarrow f(x)$;
    $x^* \leftarrow x$;
  end
end

Algorithm 1: Pseudo-code of a basic GRASP for minimization.

Local search makes use of the concept of solution neighborhood. 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 for a wide range of combinatorial optimization problems [26, 27]. Many extensions and improvements with respect to the GRASP introduced in [23, 24] have been proposed. Many of these extensions consist in the hybridization of the method with other methaheuristics.

Parallel computers have increasingly found their way into metaheuristics [16, 20]. 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 [6, 7, 25, 19, 39, 40, 41, 43, 44, 46, 47, 51]. GRASP is applied to each partition in parallel. These implementations can be categorized as *multiple-walk independent-thread* [16, 67], where the communication among processors during GRASP iterations is limited to the detection of program termination,

Recently, there has been much work on hydridization of GRASP and path-relinking [57]. Parallel approaches for GRASP with path-relinking can be categorized as multiple-walk independent-thread or *multiple-walk cooperative-thread* [16, 67], where processors share information on elite solutions visited during previous GRASP iterations. Examples of parallel GRASP with path-relinking can be found in [2, 4, 14, 42, 60].

In this Chapter, we present a survey of parallel GRASP heuristics. In Section 2, we consider multiple-walk independent-thread strategies. Multiple-walk cooperative-thread strategies are examined in Section 3. Some applications of parallel GRASP and parallel GRASP with path-relinking are surveyed in Section 4. In Section 5, we make some concluding remarks.

2. Multiple-walk independent-thread strategies

Most parallel implementations of GRASP follow the *multiple-walk independent-thread* strategy, based on the distribution of the iterations over the processors.

In general, each search thread has to perform `MaxIterations`/p iterations, where p and `MaxIterations` are, respectively, the number of processors and the total number of iterations. Each processor has a copy of the sequential algorithm, a copy of the problem data, and an independent seed to generate its own pseudorandom number sequence. To avoid that the processors find the same solutions, each of them must use a different sequence of pseudorandom numbers. A single global variable is required to store the best solution found over all processors. One of the processors acts as the master, reading and distributing problem data, generating the seeds which will be used by the pseudorandom number generators at each processor, distributing the iterations, and collecting the best solution found by each processor. Since the iterations are completely independent and very little information is exchanged, linear speedups are easily obtained provided that no major load imbalance problems occur. The iterations may be evenly distributed over the processors or according with their demands, to improve load balancing.

Pardalos, Pitsoulis, and Resende [46] reported on results of a parallel GRASP for the quadratic assignment problem on a Kendall Square Research KSR-1 parallel computer with 128 processors. The implementation used the `pthread` mechanism, a lightweight process that is the fundamental unit of concurrency on the KSR-1 [36]. Each pthread executes on a separate processor and has its own memory. Twenty instances from the QAPLIB [13] were run for 1000 GRASP iterations on each of 64 single processors. For each instance, the best solution found over all processors was used as the stopping criterion for solving the instance on 54, 44, 34, 24, 14, 4, and 1 processors. Speedups were computed by averaging the running times of all instances.

Pardalos, Pitsoulis, and Resende [47] implemented a parallel GRASP for the MAX-SAT problem on a cluster of SUN-SPARC 10 workstations, sharing the same file system, with communication done using the Parallel Virtual Machine (PVM) [30] software package. Each instance was run on a parallel GRASP using 1, 5, 10, and 15 processors, with a maximum number of iterations of 1000, 200, 100, and 66, respectively. The amount of CPU time required to perform the specified number of iterations, and the best solution found were recorded. Since communication was kept to a minimum, linear speedups were expected. Figure 1 shows individual speedups as well as average speedups for these runs. Figure 2 shows that the average quality of the solution found was not greatly affected by the number of processors used.

Martins et al. [43] implemented a parallel GRASP for the Steiner problem in graphs. Parallelization is achieved by the distribution of 512 iterations over the processors, with the value of the RCL parameter α randomly chosen in the interval $[0.0, 0.3]$ at each iteration. The algorithm was tested on an IBM SP-2 machine with 32 processors, using the Message Passing Interface (MPI) library [65] for communication. The 60 problems from series C, D, and E of the OR-Library [10] were used for the computational experiments. The parallel implementation obtained 45 optimal solutions over the 60 test instances. The relative deviation with respect to the optimal value was never larger than 4%. Almost-linear speedups observed for 2, 4, 8, and 16 processors with respect to the sequential implementation are illustrated in Figure 3.

Path-relinking may also be used in conjunction with parallel implementations of GRASP. In the case of the multiple-walk independent-thread implementation

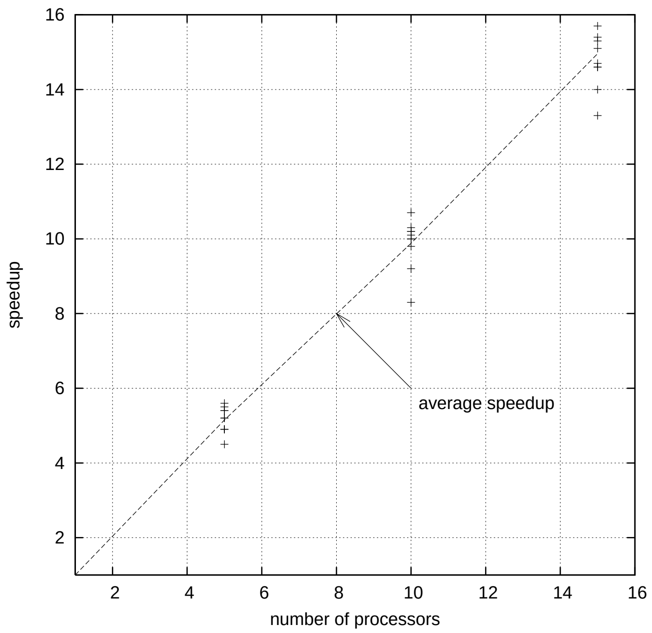


FIGURE 1. Average speedups on 5, 10, and 15 processors for maximum satisfiability problems.

described by Aiex et al. [4] for the 3-index assignment problem and Aiex, Binato, and Resende [2] for the job shop scheduling problem, each processor applies path-relinking to pairs of elite solutions stored in a local pool. Computational results using MPI on an SGI Challenge computer with 28 R10000 processors showed linear speedups for the 3-index assignment problem, but sub-linear for the job shop scheduling problem.

Alvim and Ribeiro [6, 7] showed that multiple-walk independent-thread approaches for the parallelization of GRASP may benefit much from load balancing techniques, whenever heterogeneous processors are used or if the parallel machine is simultaneously shared by several users. In this case, almost-linear speedups may be obtained with a heterogeneous distribution of the iterations over the p processors in $q \geq p$ packets. Each processor starts performing one packet of $\lceil \texttt{MaxIterations}/q \rceil$ iterations and informs the master when it finishes its packet of iterations. The master stops the execution of each slave processor when there are no more iterations to be performed and collects the best solution found. Faster or less loaded processors will perform more iterations than the others. In the case of the parallel GRASP implemented for the problem of traffic assignment described in [49], this dynamic load balancing strategy allowed reductions in the elapsed times of up to 15% with respect to the times observed for the static strategy, in which the iterations were uniformly distributed over the processors.

The efficiency of multiple-walk independent-thread parallel implementations of metaheuristics, based on running multiple copies of the same sequential algorithm,

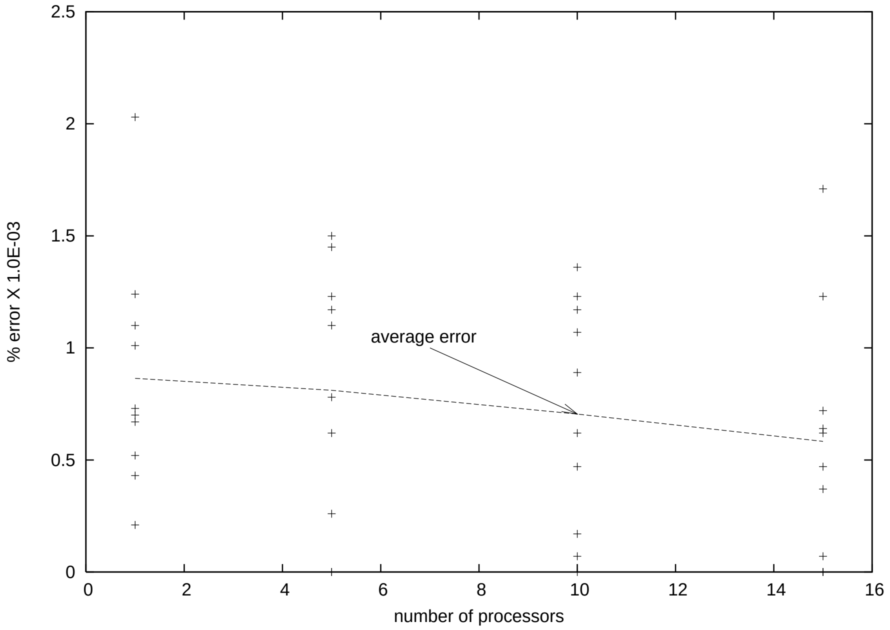


FIGURE 2. Percentage error on 1, 5, 10, and 15 processors for maximum satisfiability problems.

has been addressed by some authors. A given target value τ for the objective function is broadcast to all processors which independently execute the sequential algorithm. All processors halt immediately after one of them finds a solution with value at least as good as τ. The speedup is given by the ratio between the times needed to find a solution with value at least as good as τ, using respectively the sequential algorithm and the parallel implementation with p processors. These speedups are linear for a number of metaheuristics, including simulated annealing [18, 45]; iterated local search algorithms for the traveling salesman problem [21]; tabu search, provided that the search starts from a local optimum [9, 66]; and WalkSAT [64] on hard random 3-SAT problems [35]. This observation can be explained if the random variable *time to find a solution within some target value* is exponentially distributed, as indicated by the following proposition [67]:

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

This proposition follows from the definition of the exponential distribution. It implies that the probability $1 - e^{-\rho t/\lambda}$ of finding a solution within a given target value in time ρt with a sequential algorithm is equal to the probability of finding a solution at least as good as that in time t using ρ independent parallel processors. Hence, it is possible to achieve linear speedups in the time to find a solution within a target value by multiple independent processors. An analogous proposition can be stated for a two parameter (shifted) exponential distribution:

Proposition 2: Let $P_\rho(t)$ be the probability of not having found a given target solution value in t time units with ρ independent processors. If $P_1(t) = e^{-(t-\mu)/\lambda}$ with

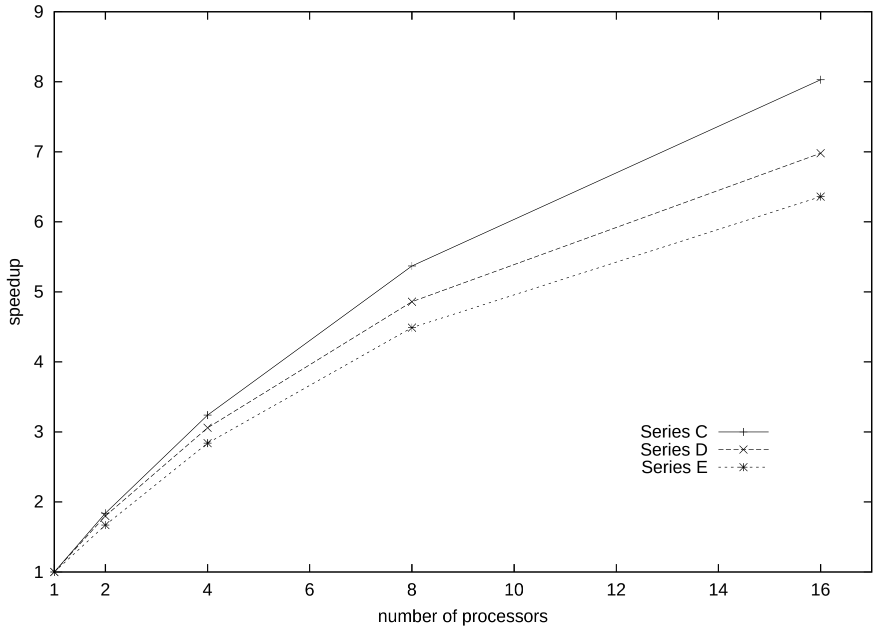


FIGURE 3. Average speedups on 2, 4, 8, and 16 processors on Steiner tree problem in graphs.

$\lambda \in \mathbb{R}^+$ and $\mu \in \mathbb{R}^+$, corresponding to a two parameter exponential distribution, then $P_\rho(t) = e^{-\rho(t-\mu)/\lambda}$.

Analogously, this proposition follows from the definition of the two-parameter exponential distribution. It implies that the probability of finding a solution within a given target value in time ρt with a sequential algorithm is equal to $1-e^{-(\rho t-\mu)/\lambda}$, while the probability of finding a solution at least as good as that in time t using ρ independent parallel processors is $1-e^{-\rho(t-\mu)/\lambda}$. 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 speedups in the time to find a solution within a target value using multiple independent processors.

Aiex, Resende, and Ribeiro [5] showed experimentally that the solution times for GRASP also have this property, i.e. that they fit a two-parameter exponential distribution. Figure 4 illustrates this result, depicting the superimposed empirical and theoretical distributions observed for one of the cases studied along the computational experiments reported by the authors, which involved 2400 runs of GRASP procedures for each of five different problems: maximum independent set [25, 51], quadratic assignment [39, 52], graph planarization [54, 59], maximum weighted satisfiability [53], and maximum covering [50]. We observe that the empirical distribution plots illustrating these conclusions were originally introduced by Feo, Resende, and Smith [25]. Empirical distributions are produced from experimental data and corresponding theoretical distributions are estimated from the empirical distributions. The same result still holds when GRASP is implemented in conjunction with a post-optimization path-relinking procedure [4].

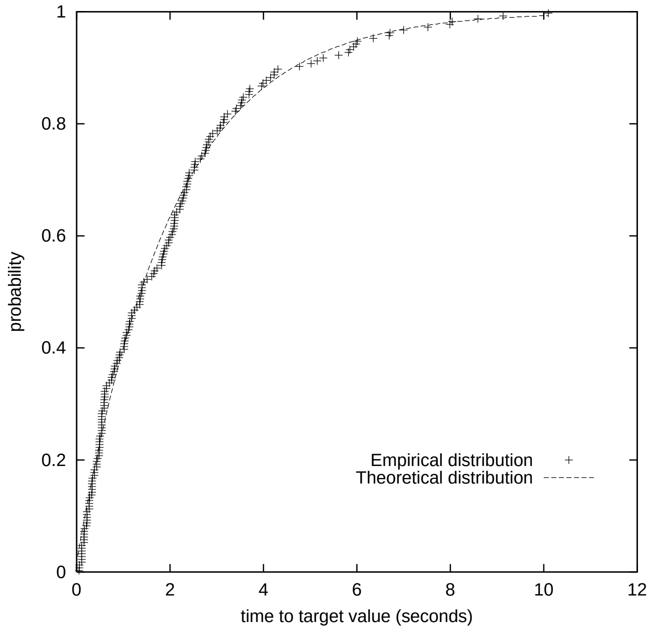


FIGURE 4. Superimposed empirical and theoretical distributions (times to target values measured in seconds on an SGI Challenge computer with 28 processors).

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 the sequential GRASP and GRASP with path-relinking for the three-index assignment problem [4] are shown in Figures 5 and 6, respectively. Analogously, Figures 7 and 8 show the same plots for the job-shop scheduling problem [2]. 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. 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. [15], one determines 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 this 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 the intercept of the line depicted in the Q-Q plot.

To avoid possible distortions caused by outliers, one does not estimate the distribution mean by linear regression on the points of the Q-Q plot. Instead, one estimates 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. Let

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

be 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.

Aiex and Resende [3] proposed a test using a sequential implementation to determine whether it is likely that a parallel implementation using multiple independent processors will be efficient. A parallel implementation is said to be efficient if it achieves linear speedup (with respect to wall time) to find a solution at least as good as a given target value. The test consists in running K (200, for example) independent trials 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.

3. Multiple-walk cooperative-thread strategies

Path-relinking has been implemented with GRASP in multiple-walk independent-thread strategies [4]. In this section, however, we focus on the use of path-relinking as a mechanism for implementing GRASP in the multiple-walk cooperative-thread strategies framework. We first briefly outline path-relinking and its hybridization

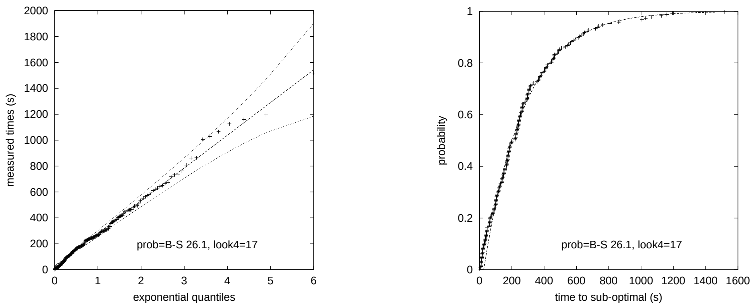


FIGURE 5. Q-Q plot and exponential distribution for GRASP for the three-index assignment problem: instance `B-S 26.1` with target value of 17.

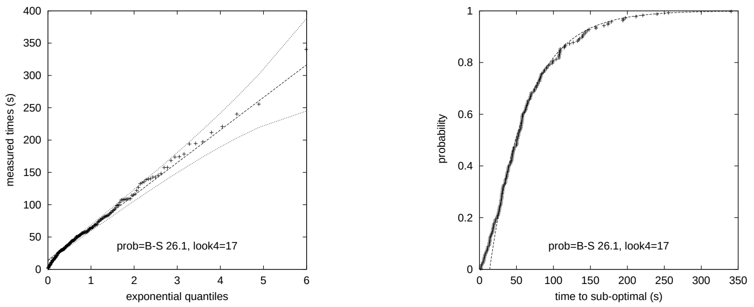


FIGURE 6. Q-Q plot and exponential distribution for GRASP with path-relinking for the three-index assignment problem: instance `B-S 26.1` with target value of 17.

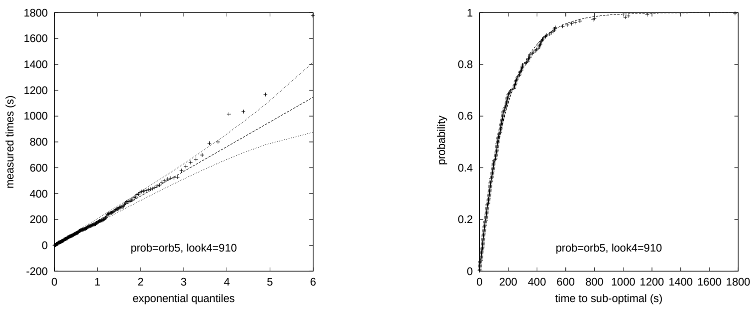


FIGURE 7. Q-Q plot and exponential distribution for GRASP for the job shop scheduling problem: instance `orb5` with target value of 910.

with GRASP. Then, we discuss how cooperation among the threads can be achieved by using path-relinking.

Path-relinking was originally proposed by Glover [31] as a strategy to explore trajectories connecting elite solutions obtained by tabu search or scatter search [32, 33, 34]. Paths in the solution space connecting pairs of elite solutions are explored

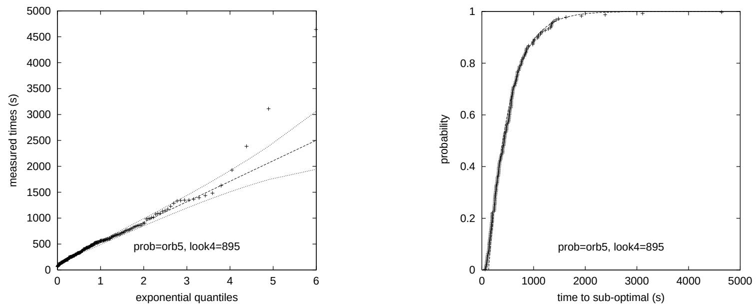


FIGURE 8. Q-Q plot and exponential distribution for GRASP with path-relinking for the job shop scheduling problem: instance `orb5` with target value of 895.

in the search for better solutions. Each pair consists of a starting solution and a guiding solution. Paths emanating from the starting solution are generated by applying moves that introduce in the current solution attributes that are present in the guiding solution.

Data : Starting solution x_s and guiding solution x_t
Result: Best solution x^* in path from x_s to x_t
Compute symmetric difference $\Delta(x_s, x_t)$;
$f^* \leftarrow \min\{f(x_s), f(x_t)\}$;
$x^* \leftarrow \text{argmin}\{f(x_s), f(x_t)\}$;
$x \leftarrow x_s$;
while $\Delta(x, x_t) \neq \emptyset$ **do**
    $m^* \leftarrow \text{argmin}\{f(x \oplus m) : m \in \Delta(x, x_t)\}$;
    $\Delta(x \oplus m^*, x_t) \leftarrow \Delta(x, x_t) \setminus \{m^*\}$;
    $x \leftarrow x \oplus m^*$;
    if $f(x) < f^*$ **then**
        $f^* \leftarrow f(x)$;
        $x^* \leftarrow x$;
    end
end

Algorithm 2: Pseudo-code of path-relinking from starting solution x_s to guiding solution x_t.

Algorithm 2 shows the pseudo-code of the path-relinking procedure applied between the starting and guiding solutions. The procedure first computes the symmetric difference $\Delta(x_s, x_t)$ between the two solutions, which defines the moves needed to reach the guiding solution (x_t) from the initial solution (x_s). A path of neighboring solutions is generated linking x_s and x_t. The best solution x^* in this path is returned. At each step, all moves $m \in \Delta(x, x_t)$ from the current solution x are examined and the one which results in the least cost solution is selected, i.e. the move that minimizes $f(x \oplus m)$, where $x \oplus m$ is the solution resulting from applying move m to solution x. The best move m^* is made, producing solution $x \oplus m^*$. This move is taken out of the set of available moves. If necessary, the best solution x^* is updated. The procedure terminates when x_t is reached, i.e. when $\Delta(x, x_t) = \emptyset$.

The use of path-relinking within a GRASP procedure was first proposed by Laguna and Martí [37]. It was followed by several extensions, improvements, and successful applications [1, 2, 3, 4, 11, 14, 22, 56, 57, 58, 60, 61, 62].

In its hybridization with GRASP, path-relinking is usually applied to pairs (x, y) of solutions, where x is a locally optimal solution produced by each GRASP iteration after local search and y is an elite solution randomly chosen from a pool with a limited number `MaxElite` of elite solutions found along the search. Since the symmetric difference is a measure of the length of the path explored during relinking, a strategy biased toward pool elements y with high symmetric difference with respect to x is often better than one using uniform random selection [58].

The pool is originally empty. To maintain a pool of good but diverse solutions, each locally optimal solution obtained by local search is considered as a candidate to be inserted into the pool if it is sufficiently different from every solution in the pool. If the pool already has `MaxElite` solutions and the candidate is better than the worst of them, then a simple strategy is to have the former replace the latter. Another strategy, which tends to increase the diversity of the pool, is to replace the pool element most similar to the candidate among all pool elements with cost worse than the candidate's. If the pool is not full, the candidate is simply inserted.

```
Data   : Number of iterations MaxIterations
Result: Solution x* ∈ X
P ← ∅;
f* ← ∞;
for i = 1, ..., i_max do
    x ← GreedyRandomizedConstruction();
    x ← LocalSearch(x);
    if i ≥ 2 then
        Randomly select an elite subset 𝒴 ⊆ P to relink with x;
        for y ∈ 𝒴 do
            Set one of solutions x and y as the starting solution;
            Set the other as the guiding solution;
            x_p ← PathRelinking(x_s, x_t);
            Update the elite set P with x_p;
            if f(x_p) < f* then
                f* ← f(x_p);
                x* ← x_p;
            end
        end
    end
end
x* = argmin{f(x), x ∈ P};
```

Algorithm 3: A basic GRASP with path-relinking heuristic for minimization.

Algorithm 3 shows the pseudo-code for a hybrid GRASP with path-relinking. Each GRASP iteration has now three main steps. In the construction phase, a greedy randomized construction procedure is used to build a feasible solution. The local search phase takes the solution built in the first phase and progressively improves it using a neighborhood search strategy, until a local minimum is found. In the path-relinking phase, path-relinking is applied to the solution obtained by local search and to a randomly selected solution from the pool. The best solution found along this trajectory is also considered as a candidate for insertion in the pool and the incumbent is updated.

Two basic mechanisms may be used to implement a multiple-walk cooperative-thread GRASP with path-relinking heuristic. In *distributed strategies* [2, 3], each thread maintains its own pool of elite solutions. Each iteration of each thread consists initially of a GRASP construction, followed by local search. Then, the local optimum is combined with a randomly selected element of the thread's pool using path-relinking. The output of path-relinking is finally tested for insertion into the pool. If accepted for insertion, the solution is sent to the other threads, where it is tested for insertion into the other pools. Collaboration takes place at this point. Though there may be some communication overhead in the early iterations, this tends to ease up as pool insertions become less frequent.

The second mechanism is that used in *centralized strategies* [42, 60], in which a single pool of elite solution is used. As before, each GRASP iteration performed at each thread starts by the construction and local search phases. Next, an elite solution is requested to and received from the centralized pool. Once path-relinking has been performed, the solution obtained as the output is sent to the pool and tested for insertion. Collaboration takes place when elite solutions are sent from the pool to other processors different from the one that originally computed it.

We notice that, in both the distributed and the centralized strategies, each processor has a copy of the sequential algorithm and a copy of the data. One processor acts as the master, reading and distributing the problem data, generating the seeds which will be used by the pseudo-random number generators at each processor, distributing the iterations, and collecting the best solution found by each processor. In the case of a distributed strategy, each processor has its own pool of elite solutions and all available processors perform GRASP iterations. Contrary to the case of a centralized strategy, one particular processor does not perform GRASP iterations and is used exclusively to store the pool and to handle all operations involving communication requests between the pool and the slaves. In the next section, we describe three examples of parallel implementations of GRASP with path-relinking.

4. Some parallel GRASP implementations

In this section, we describe a comparison of multiple-walk independent-thread and multiple-walk cooperative-thread strategies for GRASP with path-relinking for the three-index assignment problem [4], the job shop scheduling problem [2], and the 2-path network design problem [42, 60]. For each problem, we first state the problem and describe the construction, local search, and path-relinking procedures. We then show numerical results comparing the different parallel implementations.

The experiments described in Subsections 4.1 and 4.2 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 Message Passing Toolkit 1.4, which contains a fully compliant implementation of version 1.2 of the Message-Passing Interface (MPI) [65] specification. In the parallel experiments, wall clock times were measured with the MPI function `MPI_WT`. This was 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.

In the experiments described in Subsection 4.3, both variants of the parallel GRASP with path-relining heuristic were implemented in C (version `egcs-2.91.66` of the `gcc` compiler) and the MPI LAM 6.3.2 implementation. Computational experiments were performed on a cluster of 32 Pentium II 400MHz processors with 32 Mbytes of RAM memory each, running under the Red Hat 6.2 implementation of Linux. Processors are connected by a 10 Mbits/s IBM 8274 switch.

4.1. **Three-index assignment.**

4.1.1. *Problem formulation.* The NP-hard [28, 29] three-index assignment problem (AP3) [48] is a straightforward extension of the classical two-dimensional assignment problem and can be formulated as follows. Given three disjoint sets I, J, and K with $|I| = |J| = |K| = n$ 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 $(i, j, k) \in I \times J \times K$. Another way to formulate the AP3 is with permutations. There are n^3 cost elements. The optimal solution consists of the n smallest cost elements, such that the constraints are not violated. The constraints are enforced if one assigns to each set I, J, and K, the numbers $1, 2, \ldots, n$ and none of the chosen triplets (i, j, k) is allowed to have the same value for indices i, j, and k as another. 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 $N = \{1, 2, \ldots, n\}$.

4.1.2. *GRASP construction.* The construction phase selects n triplets, one at a time, to form a three-index assignment S. The usual random choice in the interval $[0, 1]$ for the RCL parameter α is made at each iteration. The value remains constant during the entire construction phase. Construction begins with an empty solution S. The initial set C of candidate triplets consists of the set of all triplets. Let $\underline{c}$ and $\overline{c}$ denote, respectively, the values of the smallest and largest cost triplets in C. All triplets (i, j, k) in the candidate set C having cost $c_{ijk} \leq \underline{c} + \alpha(\overline{c} - \underline{c})$ are placed in the RCL. 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, any triplet $(i, j, k) \in C$ such that $i = i_p$ or $j = j_p$ or $k = k_p$ is removed from C. 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.

4.1.3. *Local search.* If the solution of the AP3 is represented by a pair of permutations (p, q), then the solution space consists of all $(n!)^2$ possible combinations of permutations. If p is a permutation vector, then a 2-exchange permutation of p is a permutation vector that results from swapping two elements in p. 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. In the local search, the cost of each neighbor 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.

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

$$S = \{(p_1^S, p_2^S, \ldots, p_n^S), (q_1^S, q_2^S, \ldots, q_n^S)\}.$$

Path-relinking is done between an initial solution

$$S = \{(p_1^S, p_2^S, \ldots, p_n^S), (q_1^S, q_2^S, \ldots, q_n^S)\}$$

and a guiding solution

$$T = \{(p_1^T, p_2^T, \ldots, p_n^T), (q_1^T, q_2^T, \ldots, q_n^T)\}.$$

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

$$\delta_p^{S,T} = \{i = 1, \ldots, n \mid p_i^S \neq p_i^T\},$$
$$\delta_q^{S,T} = \{i = 1, \ldots, n \mid q_i^S \neq q_i^T\}.$$

During a path-relinking move, a permutation π (p or q) array in S, given by

$$(\ldots, \pi_i^S, \pi_{i+1}^S, \ldots, \pi_{j-1}^S, \pi_j^S, \ldots),$$

is replaced by a permutation array

$$(\ldots, \pi_j^S, \pi_{i+1}^S, \ldots, \pi_{j-1}^S, \pi_i^S, \ldots),$$

by exchanging permutation elements π_i^S and π_j^S, where $i \in \delta_\pi^{S,T}$ and $j \in \{1, 2, \ldots, n\}$ are such that $\pi_j^T = \pi_i^S$.

4.1.5. *Parallel independent-thread GRASP with path-relinking for AP3.* We study the parallel efficiency of the multiple-walk independent-thread GRASP with path-relinking on AP3 instances `B-S 20.1`, `B-S 22.1`, `B-S 24.1`, and `B-S 26.1` of Balas and Saltzman [8] using 7, 8, 7, and 8 as target solution values, respectively. Table 1 shows the estimated exponential distribution parameters for the multiple-walk independent-thread GRASP with path-relinking strategy obtained from 200 independent runs of a sequential variant of the algorithm. In addition to the sequential variant, 60 independent runs of 2-, 4-, 8-, and 16-thread variants were run on the four test problems. Average speedups 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. The average speedups can be seen in Table 2 and Figure 9.

TABLE 1. Estimated exponential distribution parameters μ and λ obtained with 200 independent runs of a sequential GRASP with path-relinking on AP3 instances `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.

Problem	estimated parameter		
	μ	λ	$\lvert\mu\rvert/\lambda$
B-S 20.1	-26.46	1223.80	.021
B-S 22.1	-135.12	3085.32	.043
B-S 24.1	-16.76	4004.11	.004
B-S 26.1	32.12	2255.55	.014
average			**.020**

TABLE 2. Speedups for multiple-walk independent-thread implementations of GRASP with path-relinking on instances `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. Speedups are computed with the average of 60 runs.

Problem	number of processors							
	2		4		8		16	
	speedup	effic.	speedup	effic.	speedup	effic.	speedup	effic.
B-S 20.1	1.67	0.84	3.34	0.84	6.22	0.78	10.82	0.68
B-S 22.1	2.25	1.13	4.57	1.14	9.01	1.13	14.37	0.90
B-S 24.1	1.71	0.86	4.00	1.00	7.87	0.98	12.19	0.76
B-S 26.1	2.11	1.06	3.89	0.97	6.10	0.76	11.49	0.72
average	**1.94**	**0.97**	**3.95**	**0.99**	**7.3**	**0.91**	**12.21**	**0.77**

4.1.6. *Parallel cooperative-thread GRASP with path-relinking for AP3.* We now study the multiple-walk cooperative-thread strategy for GRASP with path-relinking on the AP3. As with the independent-thread GRASP with path-relinking strategy, the target solution values 7, 8, 7, and 8 were used for instances `B-S 20.1`, `B-S 22.1`, `B-S 24.1`, and `B-S 26.1`, respectively. Table 3 and Figure 10 show super-linear speedups on instances `B-S 22.1`, `B-S 24.1`, and `B-S 26.1` and about 90% efficiency for `B-S 20.1`. Super-linear speedups are possible because good elite solutions are shared among the threads and are combined with GRASP solutions, whereas they would not be combined in an independent-thread implementation.

Figure 11 compares average speedup of the two implementations tested in this section, namely the multiple-walk independent-thread and multiple-walk cooperative-thread GRASP with path-relinking implementations using target solution values 7, 8, 7, and 8, on the same instances. The figure shows that the cooperative variant of GRASP with path-relinking achieves the best parallelization.

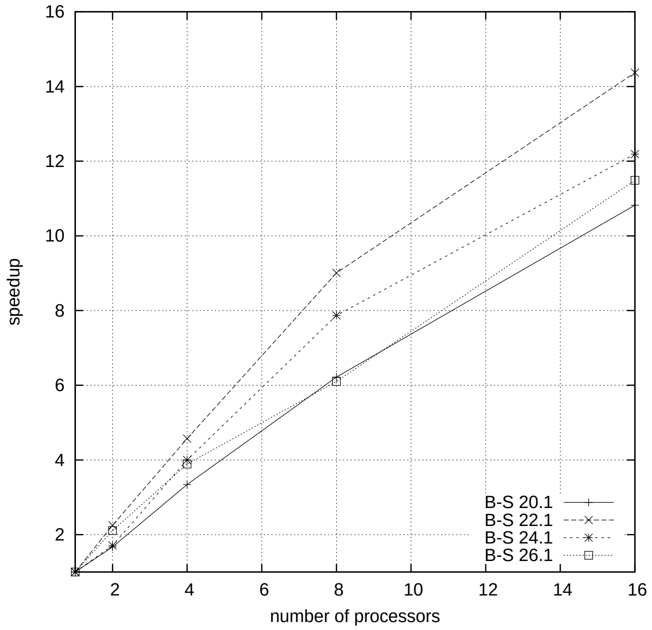


FIGURE 9. Average speedups on 2, 4, 8, and 16 processors for multiple-walk independent-thread parallel GRASP with path-relinking on AP3 instances `B-S 20.1`, `B-S 22.1`, `B-S 24.1`, and `B-S 26.1`.

TABLE 3. Speedups for multiple-walk cooperative-thread implementations of GRASP with path-relinking on instances `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. Average speedups were computed over 60 runs.

	number of processors							
	2		4		8		16	
Problem	speedup	effic.	speedup	effic.	speedup	effic.	speedup	effic.
`B-S 20.1`	1.56	0.78	3.47	0.88	7.37	0.92	14.36	0.90
`B-S 22.1`	1.64	0.82	4.22	1.06	8.83	1.10	18.78	1.04
`B-S 24.1`	2.16	1.10	4.00	1.00	9.38	1.17	19.29	1.21
`B-S 26.1`	2.16	1.08	5.30	1.33	9.55	1.19	16.00	1.00
average	**1.88**	**0.95**	**4.24**	**1.07**	**8.78**	**1.10**	**17.10**	**1.04**

4.2. **Job shop scheduling.**

4.2.1. *Problem formulation.* The job shop scheduling problem (JSP) is an NP-hard [38] combinatorial optimization problem that has long challenged researchers. 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

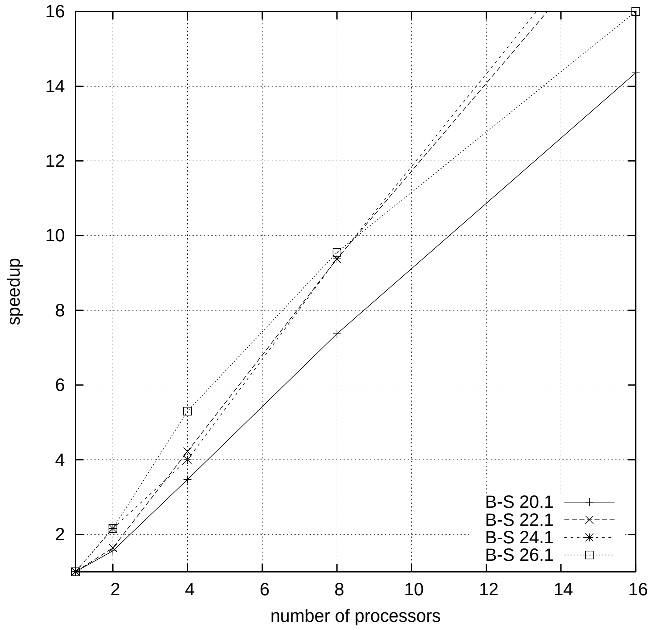


FIGURE 10. Average speedups on 2, 4, 8, and 16 processors for multiple-walk cooperative-thread parallel GRASP with path-relinking on AP3 instances `B-S 20.1`, `B-S 22.1`, `B-S 24.1`, and `B-S 26.1`.

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.

A feasible solution of the JSP can be built from a permutation of the set of jobs $\mathcal{J}$ on each of the machines in the set $\mathcal{M}$, 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 cannot be interrupted 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.

4.2.2. *GRASP construction.* Consider the GRASP construction phase for the JSP, proposed in Binato et al. [12] and Aiex, Binato, and Resende [2], where a single operation is the building block of the construction phase. A feasible schedule is built by scheduling individual operations, one at a time, until all operations have been scheduled.

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 $\mathcal{O}_s$. Denote the value of the greedy function for candidate operation σ_k^j by $h(\sigma_k^j)$.

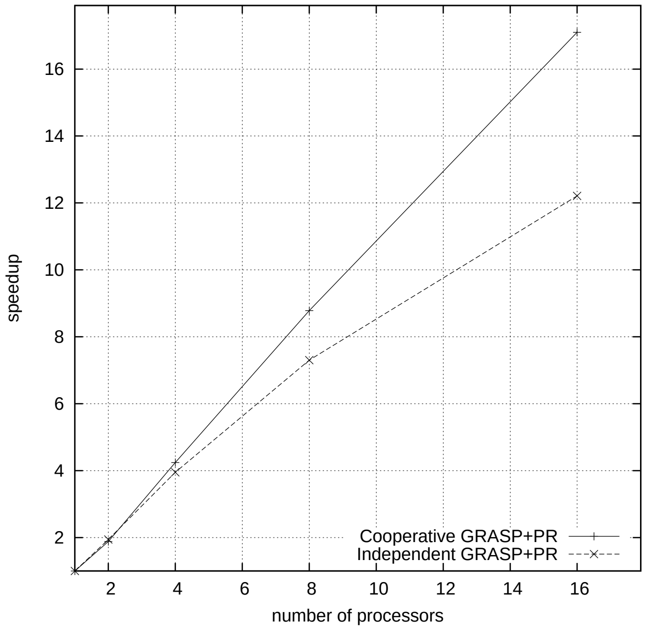


FIGURE 11. Average speedups on 2, 4, 8, and 16 processors for the parallel algorithms tested on instances of AP3: multiple-walk independent-thread GRASP with path-relinking and multiple-walk cooperative-thread GRASP with path-relinking.

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

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

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 into the earliest available feasible time slot on machine $\mathcal{M}_{\sigma^j_k}$. 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^j_k to the already-scheduled operations, i.e. $h(\sigma^j_k) = \mathcal{C}_{max}$ for $\mathcal{O} = \{\mathcal{O}_s \cup \sigma^j_k\}$. On odd numbered iterations, solutions are constructed by favoring operations from jobs having long remaining processing times. The greedy function used is given by $h(\sigma^j_k) = -\sum_{\sigma^j_l \not\in \mathcal{O}_s} p^j_l$, 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.

4.2.3. *Local search.* To attempt to decrease the makespan of the solution produced in the construction phase, we employ the 2-exchange local search used in [2, 12, 66],

TABLE 4. Estimated exponential distribution parameters μ and λ obtained with 200 independent runs of a sequential GRASP with path-relinking on JSP instances abz6, mt10, orb5, and la21, with target values 943, 938, 895, and 1100, respectively.

Problem	estimated parameter μ	λ	$\lvert\mu\rvert/\lambda$
abz6	47.67	756.56	.06
mt10	305.27	524.23	.58
orb5	130.12	395.41	.32
la21	175.20	407.73	.42
average			**.34**

TABLE 5. Speedups for multiple-walk independent-thread implementations of GRASP with path-relinking on instances abz6, mt10, orb5, and la21, with target values 943, 938, 895, and 1100, respectively. Speedups are computed with the average of 60 runs.

	number of processors							
	2		4		8		16	
Problem	speedup	effic.	speedup	effic.	speedup	effic.	speedup	effic.
abz6	2.00	1.00	3.36	0.84	6.44	0.81	10.51	0.66
mt10	1.57	0.79	2.12	0.53	3.03	0.39	4.05	0.25
orb5	1.95	0.98	2.97	0.74	3.99	0.50	5.36	0.34
la21	1.64	0.82	2.25	0.56	3.14	0.39	3.72	0.23
average	**1.79**	**0.90**	**2.67**	**0.67**	**4.15**	**0.52**	**5.91**	**0.37**

based on the disjunctive graph model of Roy and Sussmann [63]. We refer the reader to [2, 12] for a description of the implementation of the local search procedure.

4.2.4. *Path-relinking.* Path-relinking for job shop scheduling is similar to path-relinking for three-index assignment. Where in the case of three-index assignment each solution is represented by two permutation arrays, in the job shop scheduling problem, each solution is made up of $|\mathcal{M}|$ permutation arrays of numbers $1, 2, \ldots, |\mathcal{J}|$.

4.2.5. *Parallel independent-thread GRASP with path-relinking for JSP.* We study the efficiency of the multiple-walk independent-thread GRASP with path-relinking on JSP instances abz6, mt10, orb5, and la21 of ORLib [10] using 943, 938, 895, and 1100 as target solution values, respectively. Table 4 shows the estimated exponential distribution parameters for the multiple-walk independent-thread GRASP with path-relinking strategy obtained from 200 independent runs of a sequential variant of the algorithm. In addition to the sequential variant, 60 independent runs of 2-, 4-, 8-, and 16-thread variants were run on the four test problems. As before, average speedups 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 average speedups can be seen in Table 5 and Figure 12.

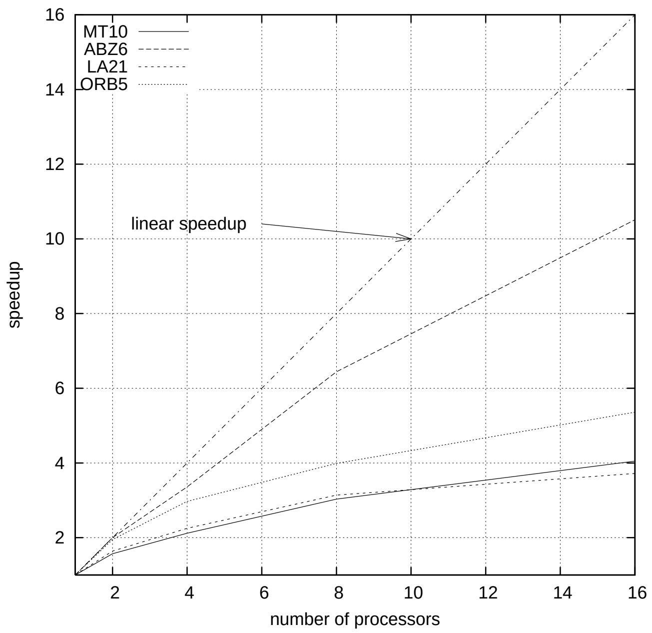


FIGURE 12. Average speedups on 2, 4, 8, and 16 processors for multiple-walk independent-thread parallel GRASP with path-relinking on JSP instances `abz6`, `mt10`, `orb5`, and `la21`.

Compared to the efficiencies observed on the AP3 instances, those for these instances of the JSP were much worse. While with 16 processors average speedups of 12.2 were computed for the AP3, average speedups of only 5.9 were computed for the JSP. This is consistent with the $|\mu|/\lambda$ values, which were on average .34 for the JSP, and 0.02 for the AP3.

4.2.6. *Parallel cooperative-thread GRASP with path-relinking for JSP.* We now study the multiple-walk cooperative-thread strategy for GRASP with path-relinking on the JSP. As with the independent-thread GRASP with path-relinking strategy, the target solution values 943, 938, 895, and 1100 were used for instances `abz6`, `mt10`, `orb5`, and `la21`, respectively. Table 6 and Figure 13 show super-linear speedups on instances `abz6` and `mt10`, linear speedup on `orb5` and about 70% efficiency for `la21`. As before, super-linear speedups are possible because good elite solutions are shared among the threads and these elite solutions are combined with GRASP solutions whereas they would not be combined in an independent-thread implementation.

Figure 14 compares the average speedup of the two implementations tested in this section, namely implementations of the multiple-walk independent-thread and multiple-walk cooperative-thread GRASP with path-relinking using target solution values 943, 938, 895, and 1100, on instances `abz6`, `mt10`, `orb5`, and `la21`, respectively.

The figure shows that the cooperative variant of GRASP with path-relinking achieves the best parallelization.

4.3. **2-path network design problem.**

TABLE 6. Speedups for multiple-walk cooperative-thread implementations of GRASP with path-relinking on instances `abz6`, `mt10`, `orb5`, and `la21`, with target values 943, 938, 895, and 1100, respectively. Average speedups were computed over 60 runs.

	number of processors							
	2		4		8		16	
Problem	speedup	effic.	speedup	effic.	speedup	effic.	speedup	effic.
`abz6`	2.40	1.20	4.21	1.05	11.43	1.43	23.58	1.47
`mt10`	1.75	0.88	4.58	1.15	8.36	1.05	16.97	1.06
`orb5`	2.10	1.05	4.91	1.23	8.89	1.11	15.76	0.99
`la21`	2.23	1.12	4.47	1.12	7.54	0.94	11.41	0.71
average	**2.12**	**1.06**	**4.54**	**1.14**	**9.05**	**1.13**	**16.93**	**1.06**

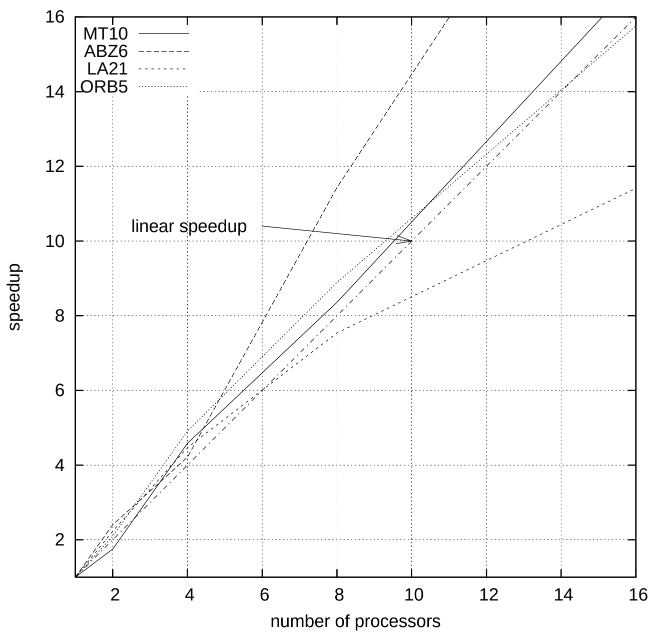


FIGURE 13. Average speedups on 2, 4, 8, and 16 processors for multiple-walk cooperative-thread parallel GRASP with path-relinking on JSP instances `abz6`, `mt10`, `orb5`, and `la21`.

4.3.1. *Problem formulation.* Let $G = (V, E)$ be a connected graph, where V is the set of nodes and E is the set of edges. A k-path between nodes $s, t \in V$ is a sequence of at most k edges connecting them. Given a non-negative weight function $w : E \to R_+$ associated with the edges of G and a set D of pairs of origin-destination nodes, the *2-path network design problem* (2PNDP) consists of finding a minimum weighted subset of edges $E' \subseteq E$ containing a 2-path between every origin-destination pair.

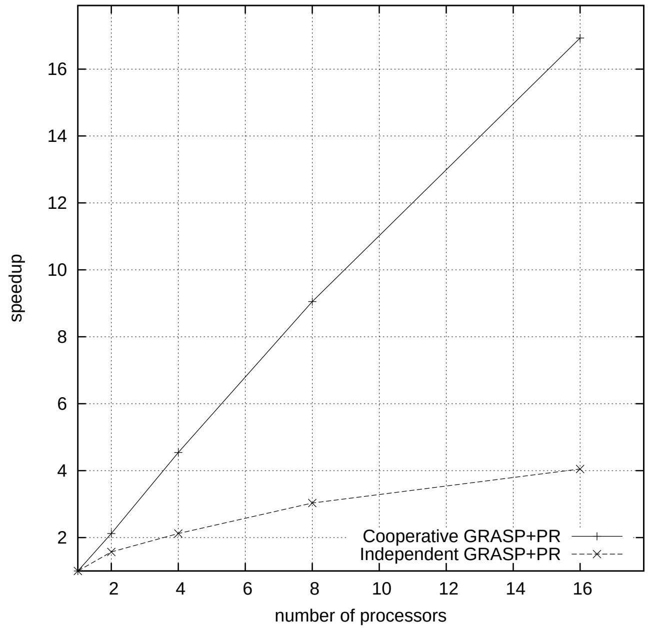


FIGURE 14. Average speedups on 2, 4, 8, and 16 processors for the parallel algorithms tested on instances of JSP: multiple-walk independent-thread GRASP with path-relinking and multiple-walk cooperative-thread GRASP with path-relinking.

Applications of 2PNDP can be found in the design of communications networks, in which paths with few edges are sought to enforce high reliability and small delays. 2PNDP was shown to be NP-hard by Dahl and Johannessen [17].

4.3.2. *GRASP construction.* The construction of a new solution begins by the initialization of modified edge weights with the original edge weights. Each iteration of the construction phase starts by the random selection of an origin-destination pair still in D. A shortest 2-path between the extremities of this pair is computed, using the modified edge weights. The weights of the edges in this 2-path are set to zero until the end of the construction procedure, the origin-destination pair is removed from D, and a new iteration resumes. The construction phase stops when 2-paths have been computed for all origin-destination pairs.

4.3.3. *Local search.* The local search phase seeks to improve each solution built in the construction phase. Each solution may be viewed as a set of 2-paths, one for each origin-destination pair in D. To introduce some diversity by driving different applications of the local search to different local optima, the origin-destination pairs are investigated at each GRASP iteration in a circular order defined by a different random permutation of their original indices.

Each 2-path in the current solution is tentatively eliminated. The weights of the edges used by other 2-paths are temporarily set to zero, while those which are not used by other 2-paths in the current solution are restored to their original values. A new shortest 2-path between the extremities of the origin-destination pair under investigation is computed, using the modified weights. If the new 2-path improves the current solution, then the latter is modified; otherwise the previous 2-path is restored. The search stops if the current solution was not improved after a sequence of $|D|$ iterations along which all 2-paths have been investigated. Otherwise, the next 2-path in the current solution is investigated for substitution and a new iteration resumes.

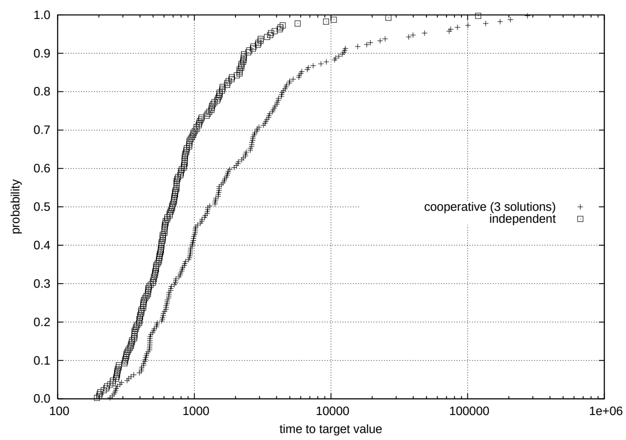
4.3.4. *Path-relinking.* A solution to 2PNDP is represented as a set of 2-paths connecting each origin-destination pair. Path-relinking starts by determining all origin-destination pairs whose associated 2-paths are different in the starting and guiding solutions. These computations amount to determining a set of moves which should be applied to the initial solution to reach the guiding one. Each move is characterized by a pair of 2-paths, one to be inserted and the other to be eliminated from the current solution.

4.3.5. *Parallel implementations of GRASP with path-relinking for 2PNDP.* As for problems AP3 and JSP, in the case of the independent-thread parallel implementation of GRASP with path-relinking for 2PNDP, each processor has a copy of the sequential algorithm, a copy of the data, and its own pool of elite solutions. One processor acts as the master, reading and distributing the problem data, generating the seeds which will be used by the pseudo-random number generators at each processor, distributing the iterations, and collecting the best solution found by each processor. All the p available processors perform GRASP iterations.

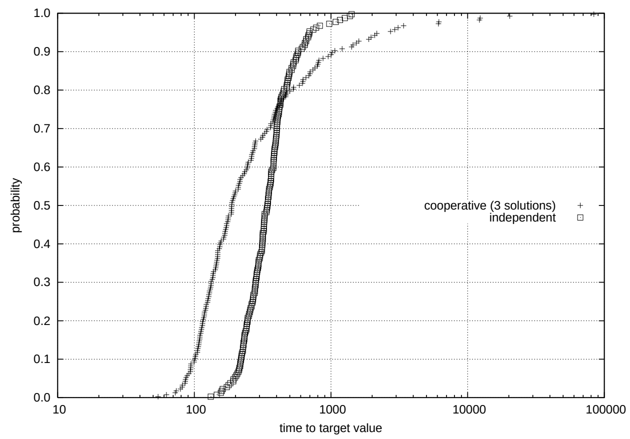
However, in the case of the cooperative-thread parallel implementation of GRASP with path-relinking for 2PNDP, the master handles a centralized pool of elite solutions, collecting and distributing them upon request (recall that in the case of AP3 and JSP each processor had its own pool of elite solutions). The $p-1$ slaves exchange the elite solutions found along their search trajectories. In the proposed implementation for 2PNDP, each slave may send up to three different solutions to the master at each iteration: the solution obtained by local search, and the solutions w^1 and w^2 obtained by forward and backward path-relinking [57] between the same pair of starting and guiding solutions, respectively.

4.3.6. *Computational results.* The results illustrated in this section concern an instance with 100 nodes, 4950 edges, and 1000 origin-destination pairs. We use the methodology proposed in [5] to assess experimentally the behavior of randomized algorithms. This approach is based on plots showing empirical distributions of the random variable *time to target solution value*. To plot the empirical distribution, we fix a solution target value and run each algorithm 200 times, recording the running time when a solution with cost at least as good as the target value is found. For each algorithm, 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 $i = 1, \ldots, 200$.

Results obtained for both the independent-thread and the cooperative-thread parallel implementations of GRASP with path-relinking on the above instance with the target value set at 683 are reported in Figure 15. The cooperative implementation is already faster than the independent one for eight processors. For fewer processors the independent implementation is naturally faster, since it employs all p processors in the search (while only $p-1$ slave processors take part effectively in the computations performed by the cooperative implementation).



(a)



(b)

FIGURE 15. Running times for 200 runs of the multiple-walk independent-thread and the multiple-walk cooperative-thread implementations of GRASP with path-relinking using (a) two processors and (b) eigth processors, with the target solution value set at 683.

Three different strategies were investigated to further improve the performance of the cooperative-thread implementation, by reducing the cost of the communication between the master and the slaves when the number of processors increases:

(1) Each send operation is broken in two parts. First, the slave sends only the cost of the solution to the master. If this solution is better than the worst solution in the pool, then the full solution is sent. The number of messages increases, but most of them will be very small ones with light memory requirements.
(2) Only one solution is sent to the pool at each GRASP iteration.

(3) A distributed implementation, in which each slave handles its own pool of elite solutions. Every time a processor finds a new elite solution, the latter is broadcast to the others.

Comparative results for these three strategies on the same problem instance are plotted in Figure 16. The first strategy outperformed all others.

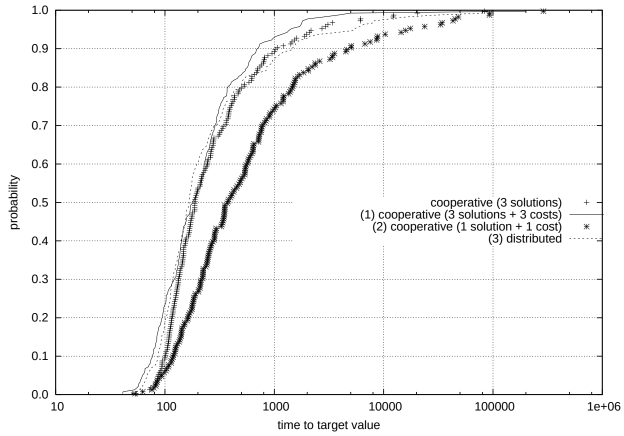


FIGURE 16. Strategies for improving the performance of the centralized multiple-walk cooperative-thread implementation on eight processors.

Table 7 shows the average computation times and the best solutions found over ten runs of each strategy when the total number of GRASP iterations is set at 3200. There is a clear degradation in solution quality for the independent-thread strategy when the number of processors increases. As fewer iterations are performed by each processor, the pool of elite solutions gets poorer with the increase in the number of processors. Since the processors do not communicate, the overall solution quality is worse. In the case of the cooperative strategy, the information shared by the processors guarantees the high quality of the solutions in the pool. The cooperative implementation is more robust. Very good solutions are obtained with no degradation in quality and significant speedups.

TABLE 7. Average times and best solutions over ten runs for 2PNDP.

	independent		cooperative	
processors	best value	avg. time (s)	best value	avg. time (s)
1	673	1310.1	—	—
2	676	686.8	676	1380.9
4	680	332.7	673	464.1
8	687	164.1	676	200.9
16	692	81.7	674	97.5
32	702	41.3	678	74.6

5. Conclusion

Metaheuristics, such as GRASP, have found their way into the standard toolkit of combinatorial optimization methods. Parallel computers have increasingly found their way into metaheuristics.

In this chapter, we surveyed work on the parallelization of GRASP. We first showed that the random variable *time to target solution value* for GRASP heuristics fits a two-parameter (shifted) exponential distribution. Under the mild assumption that the product of the number of processors by the shift in the distribution is small compared to the standard deviation of the distribution, linear speedups can be expected in parallel multiple-walk independent-thread implementations. We illustrated with an application to the maximum satisfiability problem a case where this occurs.

Path-relinking has been increasingly used to introduce memory in the otherwise memoryless original GRASP procedure. The hydridization of GRASP and path-relinking has led to some effective multiple-walk cooperative-thread implementations. Collaboration between the threads is usually achieved by sharing elite solutions, either in a single centralized pool or in distributed pools. In some of these implementations, super-linear speedups are achieved even for cases where little speedup occurs in multiple-walk independent-thread variants.

Parallel cooperative implementations of metaheuristics lead to significant speedups, smaller computation times, and more robust algorithms. However, they demand more programming efforts and implementation skills. The three applications described in this survey illustrate the strategies and programming skills involved in the development of robust and efficient parallel cooperative implementations of GRASP.

References

[1] R.M. Aiex. *Uma investigação experimental da distribuição de probabilidade de tempo de solução em heurísticas GRASP e sua aplicação na análise de implementações paralelas.* PhD thesis, Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil, 2002.

[2] 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.

[3] R.M. Aiex and M.G.C. Resende. Parallel strategies for GRASP with path-relinking. In T. Ibaraki, K. Nonobe, and M. Yagiura, editors, *Metaheuristics: Progress as real problem solvers.* Springer, 2005. To appear.

[4] R.M. Aiex, M.G.C. Resende, P.M. Pardalos, and G. Toraldo. GRASP with path relinking for three-index assignment. *INFORMS Journal on Computing*, 17, 2005. In press.

[5] 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.

[6] 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.

[7] 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.

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

[9] R. Battiti and G. Tecchiolli. Parallel biased search for combinatorial optimization: Genetic algorithms and TABU. *Microprocessors and Microsystems*, 16:351–367, 1992.

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

[11] S. Binato, H. Faria Jr., and M.G.C. Resende. Greedy randomized adaptive path relinking. In J.P. Sousa, editor, *Proceedings of the IV Metaheuristics International Conference*, pages 393–397, 2001.

[12] 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.

[13] R. Burkard, S. Karisch, and F. Rendl. QAPLIB – A quadratic assignment problem library. *European Journal of Operations Research*, 55:115–119, 1991.

[14] 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.

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

[16] V.-D. Cung, S.L. Martins, C.C. Ribeiro, and C. Roucairol. Strategies for the parallel implementation of metaheuristics. In C.C. Ribeiro and P. Hansen, editors, *Essays and surveys in metaheuristics*, pages 263–308. Kluwer Academic Publishers, 2002.

[17] G. Dahl and B. Johannessen. The 2-path network design problem. *Networks*, 43:190–199, 2004.

[18] N. Dodd. Slow annealing versus multiple fast annealing runs: An empirical investigation. *Parallel Computing*, 16:269–272, 1990.

[19] L.M.A. Drummond, L.S. Vianna, M.B. Silva, and L.S. Ochi. Distributed parallel metaheuristics based on GRASP and VNS for solving the traveling purchaser problem. In *Proceedings of the Ninth International Conference on Parallel and Distributed Systems – ICPADS'02*, pages 1–7. IEEE, 2002.

[20] 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.

[21] H.M.M. Ten Eikelder, M.G.A. Verhoeven, T.W.M. Vossen, and E.H.L. Aarts. A probabilistic analysis of local search. In I.H. Osman and J.P. Kelly, editors, *Metaheuristics: Theory & applications*, pages 605–618. Kluwer Academic Publishers, 1996.

[22] H. Faria Jr., S. Binato, M.G.C. Resende, and D.J. Falcão. Transmission network design by a greedy randomized adaptive path relinking approach. *IEEE Transactions on Power Systems*, 20(1), 2005. In press.

[23] 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.

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

[25] 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.

[26] 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.

[27] P. Festa and M.G.C. Resende. An annotated bibliography of GRASP. Technical Report TD-5WYSEW, AT&T Labs Research, Florham Park, NJ 07932, February 2004.

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

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

[30] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Mancheck, and V. Sunderam. *PVM: Parallel virtual machine, A user's guide and tutorial for networked parallel computing*. Scientific and Engineering Computation. MIT Press, Cambridge, MA, 1994.

[31] 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.

[32] 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.

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

[34] 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.

[35] H. Hoos and T. Stützle. Towards a characterisation of the behaviour of stochastic local search algorithms for SAT. *Artificial Intelligence*, 112:213–232, 1999.

[36] Kendall Square Research. *KSR Parallel Programming.* 170 Tracer Lane, Waltham, MA, February 1992.

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

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

[39] 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.

[40] 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.

[41] 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.

[42] S.L. Martins, C.C. Ribeiro, and I. Rosseti. Applications and parallel implementations of metaheuristics in network design and routing. *Lecture Notes in Computer Science*, 3285:205–213, 2004.

[43] 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 IRREGULAR'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.

[44] 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.

[45] L.J. Osborne and B.E. Gillett. A comparison of two simulated annealing algorithms applied to the directed Steiner problem on networks. *ORSA Journal on Computing*, 3:213–225, 1991.

[46] 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.

[47] 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.

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

[49] M. Prais and C.C. Ribeiro. Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing*, 12:164–176, 2000.

[50] M.G.C. Resende. Computing approximate solutions of the maximum covering problem using GRASP. *J. of Heuristics*, 4:161–171, 1998.

[51] 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.

[52] M.G.C. Resende, P.M. Pardalos, and Y. Li. Algorithm 754: Fortran subroutines for approximate solution of dense quadratic assignment problems using GRASP. *ACM Transactions on Mathematical Software*, 22:104–118, 1996.

[53] M.G.C. Resende, L.S. Pitsoulis, and P.M. Pardalos. Fortran subroutines for computing approximate solutions of MAX-SAT problems using GRASP. *Discrete Applied Mathematics*, 100:95–113, 2000.

[54] M.G.C. Resende and C.C. Ribeiro. A GRASP for graph planarization. *Networks*, 29:173–189, 1997.

[55] 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.

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

[57] M.G.C. Resende and C.C. Ribeiro. GRASP with path-relinking: Recent advances and applications. In T. Ibaraki, K. Nonobe, and M. Yagiura, editors, *Metaheuristics: Progress as real problem solvers*. Springer, 2005. To appear.

[58] 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.

[59] C.C. Ribeiro and M.G.C. Resende. Algorithm 797: Fortran subroutines for approximate solution of graph planarization problems using GRASP. *ACM Transactions on Mathematical Software*, 25:341–352, 1999.

[60] 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.

[61] 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.

[62] I. Rosseti. *Heurísticas para o problema de síntese de redes a 2-caminhos*. PhD thesis, Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil, July 2003.

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

[64] B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 337–343, Seattle, 1994. MIT Press.

[65] 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.

[66] E.D. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17:443–455, 1991.

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

(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

(Celso C. Ribeiro) Universidade Federal Fluminense, Department of Computer Science, Niterói, RJ 24210-240 Brazil.

E-mail address, Celso C. Ribeiro: celso@inf.puc-rio.br