Efficient Parallel Cooperative Implementations of GRASP Heuristics

Celso C. Ribeiro

Universidade Federal Fluminense, Department of Computer Science
Niterói, RJ 24210-240, Brazil

Isabel Rosseti

Universidade Federal Fluminense, Department of Computer Science
Niterói, RJ 24210-240, Brazil

Abstract

We propose a parallel cooperative strategy for the implementation of the GRASP metaheuristic and we illustrate it with a GRASP with path-relinking heuristic for the 2-path network design problem. Numerical results illustrating the effectiveness of the approach are reported. We comment in detail the implementation strategies that take most advantage of the algorithm structure. Computational experiments show linear speedups on a Linux cluster with 32 machines.

Key words: Metaheuristics, parallel implementation, parallelization, parallel GRASP, network design, heuristics, combinatorial optimization

1 Introduction

GRASP (*Greedy Randomized Adaptive Search Procedures*) [8] is a multistart metaheuristic for finding approximate solutions to combinatorial optimization problems. It can be thought of as a search method that repeatedly applies local search to different initial solutions. GRASP has experienced continued development and has been applied in a wide range of areas, see e.g. for a detailed annotated bibliography [10]. Resende and Ribeiro [16] reviewed successful implementation techniques and parameter tuning strategies, as well as enhancements, extensions, and hybridizations of the original algorithm.

Email addresses: celso@inf.puc-rio.br (Celso C. Ribeiro), rosseti@ic.uff.br (Isabel Rosseti).

Metaheuristics such as genetic algorithms, tabu search, ant colonies, and GRASP are time consuming search methods that offer a wide range of possibilities for effective parallel algorithms that require efficient implementations. Cung et al. [5] pointed out that parallel implementations of metaheuristics appear quite naturally as an effective alternative to speed up the search for approximate solutions to hard combinatorial optimization problems. They not only allow solving larger problems or finding improved solutions with respect to their sequential counterparts, but they also lead to more robust algorithms. The latter is often reported as one of the main advantages obtained with parallel implementations of metaheuristics: parallel implementations appear to be less-dependent on parameter tuning and their success is not limited to few or small classes of problem instances. However, developing and tuning efficient parallel cooperative implementations of GRASP and other metaheuristics is not easy and requires a thorough programming effort.

We describe an efficient strategy for the implementation of parallel cooperative GRASP heuristics for combinatorial optimization problems. This strategy is illustrated with an application to the context of the 2-path network design problem. We first give a high-level description of a general GRASP with path-relinking heuristic. The 2-path network design problem and the customization of the GRASP with path-relinking heuristic to this problem are described in Section 3. General ideas concerning strategies for the parallel implementation of metaheuristics are reviewed in Section 4. The parallel cooperative GRASP with path-relinking heuristic for the 2-path network design problem is described in Section 5. Computational results obtained on a 32-processor cluster are reported in Section 6. Concluding remarks are drawn in the last section.

2 GRASP with path-relinking

GRASP [8, 16] is a multi-start or iterative metaheuristic, in which each iteration consists of two phases: construction and local search. The construction phase builds a feasible solution, whose neighborhood is investigated until a local minimum is found during the local search phase. The best overall solution is kept as the result. The pseudo-code in Figure 1 illustrates the main blocks of a basic GRASP procedure for minimization, in which `Max_Iterations` iterations are performed and `Seed` is used as the initial seed for the pseudorandom number generator.

The construction and local search phases are problem-dependent and should be customized for each problem. Figure 2 illustrates the construction phase with its pseudo-code. At each iteration, all elements that can be incorporated into the solution under construction without destroying feasibility are evaluated according to a greedy function. The latter represents the increase in

```
procedure GRASP(MaxIterations, Seed)
    ReadInput();
    f* ← ∞;
    for k = 1, ..., MaxIterations do
        Solution ← GreedyRandomizedConstruction(Seed);
        Solution ← LocalSearch(Solution);
        if f(Solution) < f* then do;
            f* ← f(Solution);
            BestSolution ← Solution;
        end;
    end;
    return BestSolution;
end GRASP.
```

Fig. 1. Pseudo-code of the basic GRASP metaheuristic.

the cost function due to the incorporation of each element into the solution under construction. The restricted candidate list (RCL) is formed by the best elements, i.e. those whose incorporation into the current solution results in the smallest incremental costs. The element to be incorporated into the partial solution is randomly selected from those in the RCL. Once the selected element is incorporated into the partial solution, the candidate list is updated and the incremental costs are reevaluated.

```
procedure GreedyRandomizedConstruction(Seed)
    Solution ← ∅;
    Evaluate the incremental costs of the candidate elements e ∈ E;
    while Solution is not a complete solution do
        Build the restricted candidate list (RCL);
        Select an element s from the RCL at random;
        Solution ← Solution ∪ {s};
        Reevaluate the incremental costs;
    end;
    return Solution;
end.
```

Fig. 2. Pseudo-code of the construction phase.

The local search phase attempts to improve the solutions built in the first phase. A local search algorithm works in an iterative fashion by successively replacing the current solution by a better solution within its neighborhood. It terminates when no better solution is found in the neighborhood. The pseudo-code of a local search algorithm starting from the solution constructed in the first phase and using a neighborhood structure $N(.)$ is given in Figure 3.

A shortcoming of the standard GRASP framework is the independence of its

```
procedure LocalSearch(Solution)
while Solution is not locally optimal do
    Find y ∈ N(Solution) with f(y) < f(Solution);
    Solution ← y;
end;
return Solution;
end.
```

Fig. 3. Pseudo-code of the local search phase.

iterations, i.e., it does not learn from the solutions found in previous iterations. The algorithm described in Figure 1 discards information about solutions that do not improve the incumbent. Information gathered from good solutions can be used to implement more effective memory-based heuristics.

Path-relinking was originally proposed by Glover [11] as an intensification strategy exploring trajectories connecting elite solutions obtained by tabu search or scatter search [12, 13]. Starting from one or more elite solutions, path-relinking generates paths leading toward other elite solutions and explores them in the search for better solutions. To generate paths, moves are selected to introduce attributes in the current solution that are present in the elite guiding solution. Path-relinking is a strategy that seeks to incorporate attributes of high quality solutions, by favoring them in the selected moves.

The algorithm in Figure 4 illustrates the pseudo-code of the path-relinking procedure applied to a pair of solutions x_s (starting solution) and x_t (target solution). The procedure starts by computing the symmetric difference $\Delta(x_s, x_t)$ between the two solutions, i.e. the set of moves needed to reach x_t from x_s. The current solution is initialized with x_s and a path of solutions is generated linking x_s and x_t. The best solution $\overline{x}$ in this path is returned by the algorithm. At each step, the procedure examines all moves $m \in \Delta(x, x_t)$ from the current solution x and selects the one which results in the least cost solution, i.e. the one which minimizes $f(x \oplus m)$, where $x \oplus m$ is the solution resulting from applying move m to solution x. The best move $\overline{m}$ is made, producing solution $x \oplus \overline{m}$. The set of available moves is updated. If necessary, the best solution $\overline{x}$ along the path under construction is updated. The procedure terminates when x_t is reached, i.e. when $\Delta(x, x_t)$ is empty.

Path-relinking may also be viewed as a constrained local search strategy applied to the initial solution x_s, in which only a limited set of moves can be performed and where uphill moves are allowed. Several alternatives have been considered and combined in recent implementations of path-relinking: periodical relinking, forward relinking, backward relinking, back and forward relinking, randomized relinking, and truncated relinking, see e.g. Resende and Ribeiro [17] for a recent survey.

procedure `PathRelinking`(x_s, x_t)
Compute the symmetric difference $\Delta(x_s, x_t)$ between x_s and x_t;
$\overline{f} \leftarrow \min\{f(x_s), f(x_t)\}$;
$\overline{x} \leftarrow \text{argmin}\{f(x_s), f(x_t)\}$;
$x \leftarrow x_s$;
while $\Delta(x, x_t) \neq \emptyset$ **do**
    Let $\overline{m}$ such that $f(x \oplus \overline{m}) = \min\{f(x \oplus m) : m \in \Delta(x, x_t)\}$;
    $\Delta(x \oplus \overline{m}, x_t) \leftarrow \Delta(x, x_t) \setminus \{\overline{m}\}$;
    $x \leftarrow x \oplus \overline{m}$;
    if $f(x) < \overline{f}$ **then**
        $\overline{f} \leftarrow f(x)$;
        $\overline{x} \leftarrow x$;
    end;
end;
return $\bar{x}$;
end.

Fig. 4. Pseudo-code of the path-relinking procedure.

Path-relinking may be used an effective enhancement of the basic GRASP procedure. It incorporates a memory-based intensification phase to an otherwise memoryless procedure. This strategy strongly improves solution quality and reduces computation times with respect to memoryless implementations, see e.g. [1, 2, 9, 15, 18, 19]. It is also an effective strategy for the implementation of parallel cooperative heuristics.

In this context, path-relinking is applied to pairs (x_s, x_t) of solutions, in which one of them is a locally optimal solution and the other is randomly chosen from a pool with at most `MaxElite` elite solutions previously found. This pool is originally empty. Since we wish 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 other solution currently 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 replaces the latter. If the pool is not full, the candidate is simply inserted. Resende and Ribeiro [17] reviewed advances and applications of GRASP based on the use of path-relinking.

3 A GRASP heuristic for the 2-path network design problem

Let $G = (V, E)$ be a connected undirected 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

$w : E \rightarrow R_+$ associated with the edges of G and a set D of pairs of origin-destination nodes, the *2-path network design problem* (2PNDP) consists in finding a minimum weighted subset of edges $E' \subseteq E$ containing a 2-path between every origin-destination pair in D. Applications can be found in the design of communication networks, in which paths with few edges are sought to enforce high reliability and small delays. The decision version of 2PNDP has been proved to be NP-complete by Dahl and Johannessen [6]. We customize in the next sections a GRASP heuristic for the 2-path network design problem.

3.1 Construction phase

The greedy randomized algorithm computes one shortest 2-path at-a-time. Figure 5 illustrates the pseudo-code of the construction phase of the GRASP with path-relinking heuristic for 2PNDP. Initializations are performed in lines 1 and 2. Solution x is computed from scratch using edge weights w' that are initially equal to the original weights w. The loop in lines 3–9 is performed until a 2-path has been computed for every origin-destination pair. Each iteration starts by the random selection in line 4 of a pair (a, b) still to be routed. A shortest path P from a to b using the modified weights w' is computed in line 5. To avoid that edge weights be counted more than once, the weights of the edges in P are temporarily set to 0 for the remaining iterations in line 6. Pair (a, b) is removed in line 7 from the set of origin-destination pairs to be routed and in line 8 the edges in P are inserted into the solution under construction.

Since the loop is executed $|D|$ times and each shortest 2-path can be computed in time $O(|V|)$, the complexity of the construction procedure is $O(|V| \cdot |D|)$.

procedure `GreedyRandomizedConstruction2Path`($Seed$);
1 $x \leftarrow \emptyset$;
2 $w' \leftarrow w$;
3 **while** $D \neq \emptyset$ **do**
4   Select at random an yet unrouted origin-destination pair $(a, b) \in D$;
5   Compute the shortest 2-path P from a to b using weights w';
6     $w'_{ij} \leftarrow 0$ for all edges (i, j) in P;
7     $D \leftarrow D \setminus (\{a, b\})$;
8     $x \leftarrow x \cup P$;
9 **end_while**;
10 **return** x;
end;

Fig. 5. Pseudo-code of the construction phase of a GRASP heuristic for 2PNDP.

3.2 Local search

The neighbors of a solution to 2PNDP may be obtained by replacing any of its 2-paths by another 2-path between the same origin-destination pair.

Figure 6 summarizes the pseudo-code of the local search algorithm for 2PNDP. The neighbor solution x' and modified edge weights are initialized respectively in lines 1 and 2. Variable *nochanges* is initialized in line 3 as a flag to indicate that a local optimum was found. A circular permutation of the demand pairs is randomly created in line 4. The loop in lines 5–16 is performed until all 2-paths in the current solution have been consecutively examined and no shorter 2-path has been found (the current solution is locally optimal). Each iteration starts in line 6 by considering the next origin-destination pair (a, b), according with the circular permutation computed in line 4 and attempting to improve this 2-path. The following steps are performed: temporarily reset to zero the modified weights w' of all edges used by other 2-paths (line 7); compute the shortest 2-path from a to b using the modified edge weights w' (line 8); update the incumbent solution x' if the weight of the new 2-path is shorter (lines 9 and 10); and update the number of consecutive 2-paths examined without change in the current solution (lines 11 and 13). Once the iteration is finished, all weights are reset to their original values in line 15. If less than $|D|$ 2-paths have been consecutively examined without improvement in the current solution, then a new iteration resumes. Otherwise, the neighbor solution x' is returned in line 17. The complexity of each local search iteration is $O(n)$.

3.3 Path-relinking

The pseudo-code of the path-relinking procedure in Figure 4 can be easily customized. Each solution to 2PNDP is characterized by a set of $|D|$ 2-paths between the extremities of each origin-destination pair. The symmetric difference $\Delta(x, x_t)$ between the current solution x and the target solution x_t amounts to the set of 2-paths that appear in one of them but not in the other. Each move $m \in \Delta(x, x_t)$ is defined by one 2-path to be removed from and another to be inserted into the current solution x.

The algorithm in Figure 7 illustrates a GRASP with path-relinking procedure for 2PNDP. We denote by $f(x)$ the cost of solution x, given by the sum of the weights w_{ij} of all edges (i, j) in x. Each iteration has now three main steps:

- *Construction phase:* procedure `GreedyRandomizedConstruction2Path` is used to build a feasible solution;
- *Local search phase:* procedure `LocalSearch2Path` is applied to the solution built in the construction phase and a local minimum is found; and

```
procedure LocalSearch2Path;
x' ← x;
w' ← w;
nochanges ← 0;
Create a random circular permutation of the demand pairs in D;
while nochanges < |D| do
    Select the next origin-destination pair (a, b);
    Temporarily reset to 0 the weights w' of all edges appearing
    in the 2-paths connecting the remaining origin-destination pairs in D;
    Compute the shortest 2-path from a to b using the modified weights w';
    if the weight of the new 2-path is smaller then
        Update solution x' by using the new 2-path;
        nochanges ← 0;
    else
        nochanges ← nochanges + 1;
    end_if;
    Reset the weights w of all edge weights temporarily set to 0;
end_while;
return x';
end.
```

Fig. 6. Pseudo-code of the local search phase of a GRASP heuristic for 2PNDP.

- *Path-relinking phase:* path-relinking is applied twice to a locally optimal solution and to a randomly selected solution from the pool (once using the latter as the starting solution and once using the former). The locally optimal solution and the best solution found along each relinking trajectory are considered as candidates for insertion into the pool.

4 Parallel implementations of metaheuristics

Parallel implementations of metaheuristics appear quite naturally as an effective approach to speedup the search for approximate solutions and to solve larger problems.

Cung et al. [5] reviewed parallelization strategies, implementation issues, and applications of parallel metaheuristics; see also [3]. Parallel implementations of metaheuristics use several processors to concurrently explore solution neighborhoods. In single-walk parallelizations, one unique trajectory is traversed in the neighborhood and the search for the best neighbor at each iteration is performed in parallel. A multiple-walk parallelization is characterized by the investigation in parallel of multiple trajectories, each of them by a different processor. A search thread is a process running in each processor traversing a walk in the neighborhood. These threads can be either independent (when

```
procedure GRASPwithPR2Path(MaxIterations, Seed)
P ← ∅;
f* ← ∞;
for k = 1, . . . , MaxIterations do
    x ← GreedyRandomizedConstruction2Path(Seed);
    x ← LocalSearch2Path(x);
    Update the pool of elite solutions P with x;
    if k ≥ 2 then
        Select at random an elite solution y from the pool P;
        x1 ← PathRelinking(x, y);
        Update the pool of elite solutions P with x1;
        x2 ← PathRelinking(y, x);
        Update the pool of elite solutions P with x2;
        Set x ← argmin{f(x), f(x1), f(x2)};
    end;
    if f(x) < f* then
        f* ← f(x);
        x* ← x;
    end;
end;
return x*;
end.
```

Fig. 7. GRASP with path-relinking heuristic for 2PNDP.

they do not exchange any information) or cooperative (when the information collected along each trajectory is disseminated and used by other threads).

Independent parallelizations can be easily implemented. They lead to good speedups and robust implementations can be obtained by using different parameter settings in each processor. However, this model is quite poor and can be simulated in sequential mode by several successive executions with different initializations. The lack of cooperation between the search threads does not allow the use of the information collected by different processors.

The use of cooperative search threads demands more programming efforts and implementation skills. As the threads exchange information collected along each search trajectory, one expects not only to accelerate the convergence to the best solution, but also to find better solutions than those found by independent strategies within the same computation times.

The efficiency of independent multiple-walk parallel implementations of metaheuristics 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. The speedups are linear for metaheuristics such as simulated annealing [7, 14] and tabu search [4, 20]. This observation can be explained if the random variable *time to find a solution within a target value* is exponentially distributed, as indicated by the following propositions [21]:

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 $\lambda \in \mathbb{R}^+$ (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. Then, it is possible to achieve linear speedups in the time to find a solution within a target value by multiple independent processors. A similar proposition can be stated for a two-parameter 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 $\lambda \in \mathbb{R}^+$ and $\mu \in \mathbb{R}^+$ (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 achieve linear speedups in the time to find a solution within a target value using multiple independent processors.

5 Parallel strategies for the GRASP heuristic for 2PNDP

Aiex et al. [2] have shown experimentally that the solution times to find a target solution value with a GRASP heuristic satisfy Proposition 2, i.e. they fit a two-parameter exponential distribution. This result still holds when GRASP is implemented in conjunction with path-relinking [1, 17]. Consequently, GRASP with path-relinking may be implemented in parallel with linear speedups.

Typical parallelizations of GRASP are based on evenly distributing the iterations over the processors [5]. Each processor performs `MaxIterations`/p iterations, where p is the number of processors and `MaxIterations` the total number of iterations. Each processor has a copy of algorithm `GRASPwithPR2Path`, a copy of the data, and its own pool of elite solutions. One of the processors acts as the master, reading and distributing the problem data, generating the seeds which will be used by the pseudo-random number generators, distributing the iterations, and collecting the best solution found by each processor.

In the parallel cooperative strategy described in this work, the master handles a centralized pool of elite solutions, collecting and distributing them upon request. The $p-1$ slaves exchange the elite solutions found along their search trajectories. Cooperation between the processors is implemented via path-relinking using a centralized pool of elite solutions. In the context of the 2PNDP implementation, each slave may send up to three different solutions to the master at each iteration: solution x obtained by local search (line 6), and solutions x_1 and x_2 obtained by path-relinking (lines 10 and 12).

6 Computational results

The sequential and the parallel variants of the GRASP with path-relinking heuristic described in Section 4 were implemented in C, using version egcs-2.91.66 of the gcc compiler. The MPI LAM 6.3.2 implementation was used. Computational experiments were carried out 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 were connected by a 10 Mbits/s IBM 8274 switch.

In the first experiment, we randomly generated 100 test instances with 70 nodes and 35 origin-destination pairs, with the same parameters used in [6]. We compared the results of the sequential GRASP with path-relinking heuristic running 10000 iterations with those obtained by the greedy heuristic in [6], using two samples of solution values and the statistical test t for unpaired observations. The main statistics are summarized in Table 1 and make it possible to conclude with 40% of confidence that GRASP finds better solutions than the other heuristic. The average value of the solutions obtained by the new heuristic was 2.2% smaller than that of the solutions obtained by the other. These values are relatively small due to the fact that the instances generated in [6] were small and possibly easy, leading even a straightforward heuristic to obtain good results.

In the next experiment, we address the behavior of the independent and cooperative parallel implementations of `GRASPwithPR2Path`. The performance

Table 1
Statistics for parallel GRASP (sample A) and the greedy heuristic (sample B)

	Parallel GRASP (sample A)	Greedy (sample B)
Size	$n_A = 100$	$n_B = 30$
Mean	$\mu_A = 443.73$	$\mu_B = 453.67$
Standard deviation	$S_A = 40.64$	$S_B = 61.56$

of the parallel implementation is quite uniform over all problem instances. Therefore, all results illustrated in this section concern one specific standard instance with 100 nodes, 4950 edges, and 1000 origin-destination pairs. We use the methodology proposed in [2] to assess experimentally and compare the behavior of different algorithm versions. 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 a probability $p_i = (i - \frac{1}{2})/200$ with the i-th sorted running time t_i and plot the points $z_i = (t_i, p_i)$, for $i = 1, \ldots, 200$.

The plots in Figure 8 display the empirical probability distribution of the random variable *time to target solution value* for both the independent and the cooperative parallel implementations, for runs with two, four, and eight processors on the above instance using 683 as the target value. The independent strategy performs better when only two processors are used, because it makes use of the two processors to perform GRASP iterations, while the cooperative strategy makes use of one processor to perform iterations and the other to handle the pool. In other words, 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). This behavior changes when the number of processors increases. The computation times are reduced and the plots shift to the left. The reduction in computation times is more significant for the cooperative parallelization. As the number of processors increases, the gains obtained through cooperation are more important. The cooperative implementation is already faster than the independent for eight processors.

The graph in Figure 9 displays another view of this interpretation. In each case (two, four, and eight processors), we compare the i-th smallest *time to target solution value* observed with the independent parallelization with that observed with the cooperative parallelization, for $i = 1, \ldots, 200$, and we count the number of times (in percentage) each of them was faster. Once again, we notice that when the number of processors increases, the cooperative parallelization finds solutions within the target value faster.

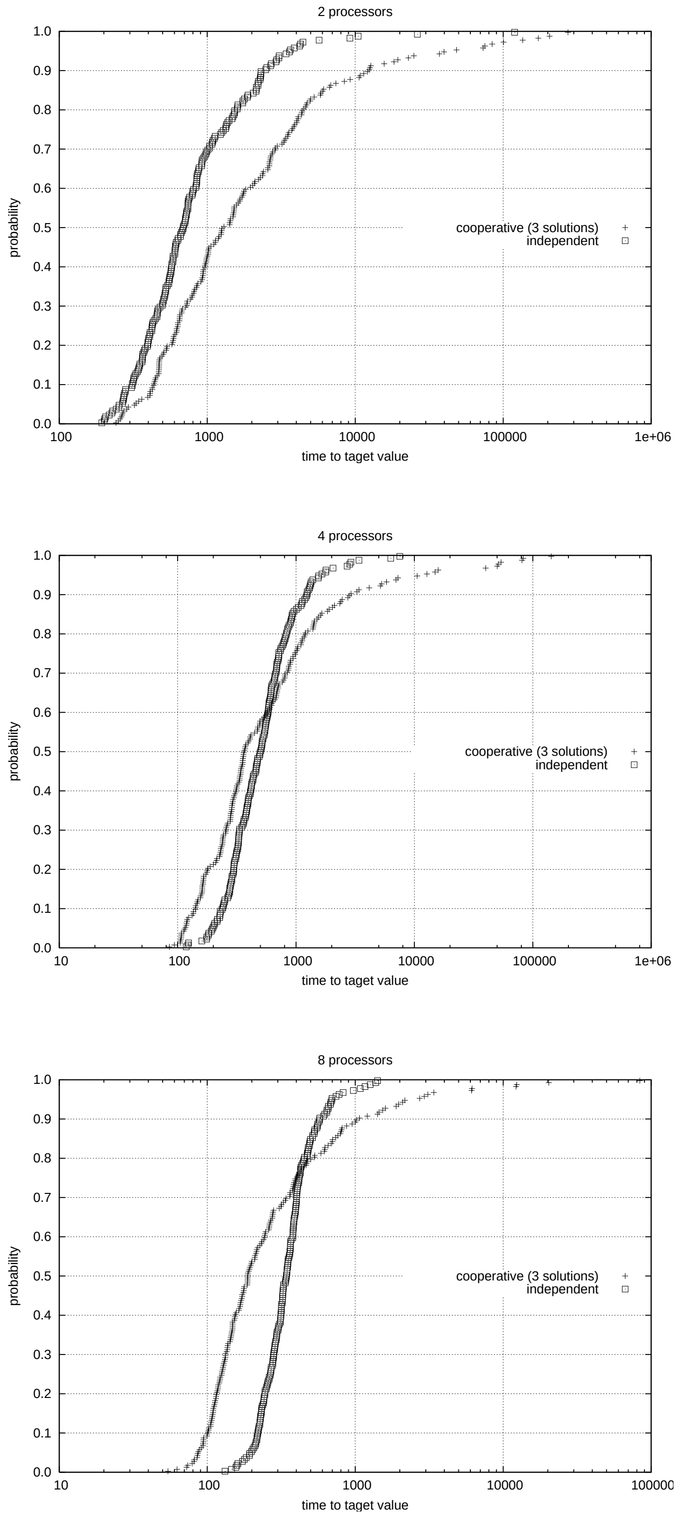

Fig. 8. Empirical distributions of the *time to target solution value* for the independent and the cooperative parallelizations on two, four, and eight processors (target value set at 683).

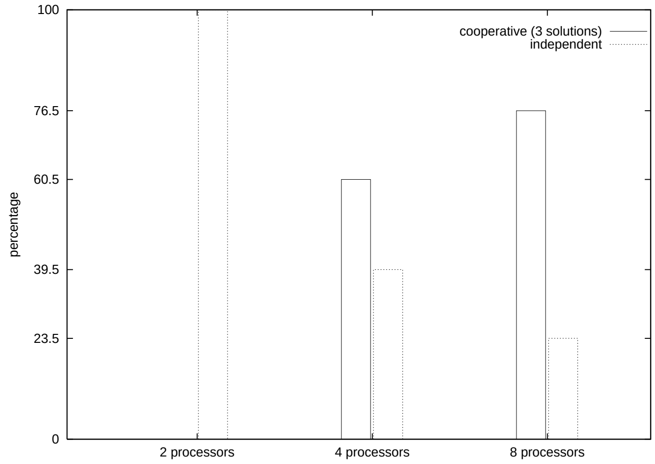

Fig. 9. Pairwise comparisons between the independent and the cooperative parallelizations on two, four, and eight processors (target value set at 683).

However, when the number of processors is further increased to 16, the available memory is not sufficient to handle all messages exchanged through the master processor handling the centralized pool in the cooperative parallelization. In consequence, the system crashes very often and it is not able to make full use of the cooperative parallelization.

Three strategies were investigated to further improve the performance of the cooperative implementation, attempting to reduce the communication cost between the master and the slaves when the number of processors increases:

- 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 (`3 solutions + 3 costs`).
- Only the best solution among x, x_1, and x_2 is sent to the pool at each iteration of the parallel cooperative implementation (`1 solution`).
- 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 other processors and each of them updates its own pool (`distributed`).

Comparative results for the three strategies on the same problem instance are plotted in Figure 10. The first strategy outperformed all others, including the original one based on always sending three full solutions (`3 solutions`). The plots in Figure 11 display the empirical probability distribution of the random variable *time to target solution value* for both the independent and the cooperative parallel implementations, using the new strategy based on sending three costs and three solutions separately, for runs of the same instance on

16 and 32 processors. We recall that the original cooperative strategy sending three full solutions was not even able to run on 16 processors.

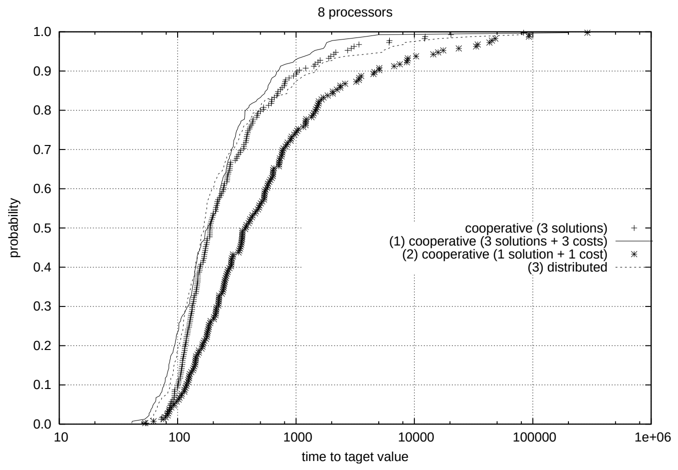

Fig. 10. Strategies for improving the performance of the centralized cooperative parallelization on eight processors.

Table 2 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 `MaxIterations` $= 3200$ iterations. There is a clear degradation in solution quality for the independent 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 guarantee 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 with a large speedup of 17.6 for 32 processors. There results clearly illustrate the relevance of the cooperation between the processors.

Table 2
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

In the last experiment, we performed all runs on a more powerful Linux cluster

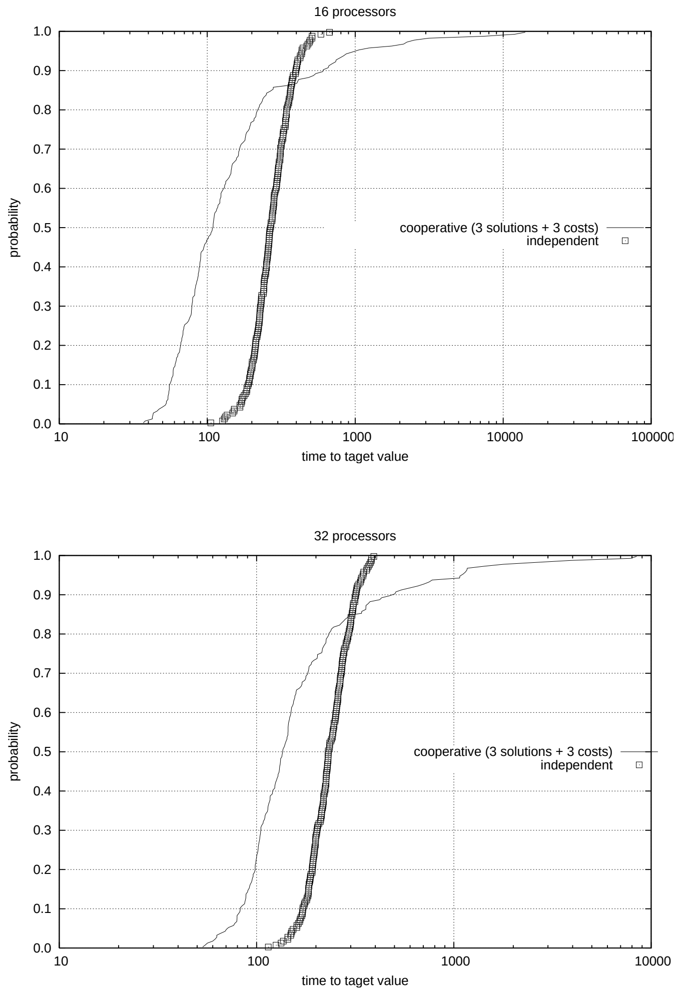

Fig. 11. Empirical distributions of the *time to target solution value* for the independent and the cooperative parallelizations of `GRASPwithPR2Path` on 16 and 32 processors (target value set at 683) when the three costs and the three solutions are sent separately.

of 32 Pentium IV processors with a 1.7 GHz clock and 256 Mbytes of RAM memory each, running under the Red Hat 7.2 implementation of Linux and using a Extreme Networks switch with 48 ports at 10/100 Mbits/s and 2 ports at 1 Gbits/s. The plots displaying the probability distributions of the random variable *time to target solution value* for both the independent and the cooperative parallel implementations using the new strategy that sends costs and solutions separately are given in Figure 12. The cooperative strategy presents smaller computation times and scales even better on the new cluster. The graph in Figure 13 corroborates these conclusions, showing that when the number of processors increases, the cooperative parallelization finds solutions within the target value consistently faster than the independent.

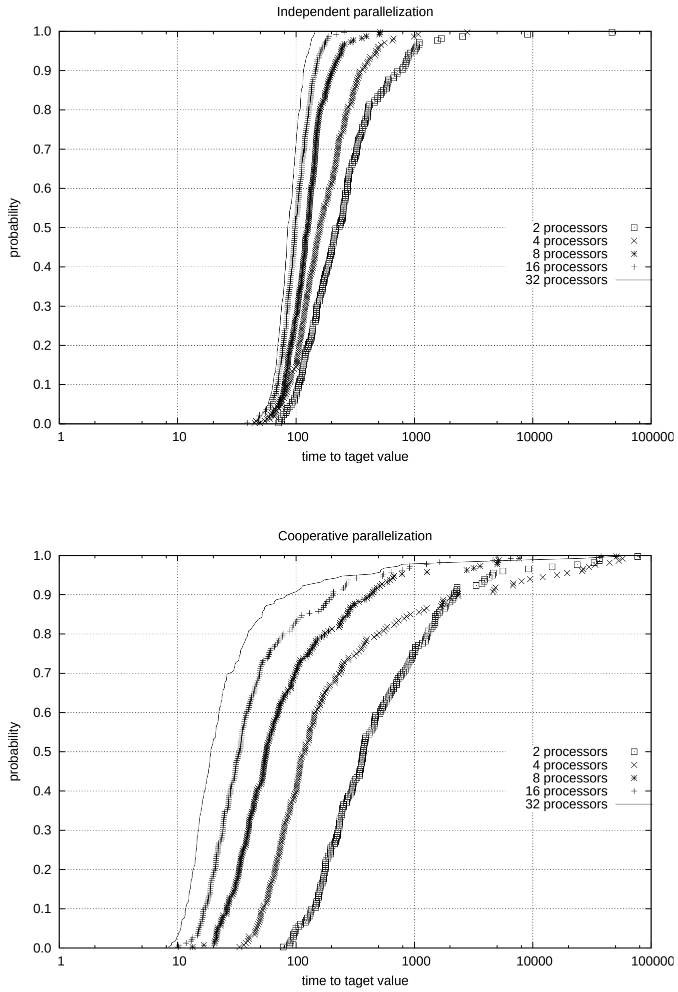

Fig. 12. Empirical distributions on the more powerful cluster of the *time to target solution value* for the independent and the cooperative parallelizations when costs and solutions are sent separately.

7 Concluding remarks

Metaheuristics such as GRASP are powerful tools for finding high-quality solutions to hard combinatorial optimization problems. Although these solution approaches are able to find very good solutions, their computation times are often very large. Parallel cooperative implementations may lead to significant speedups, smaller computation times, and more robust algorithms. However, more programming efforts and implementation skills are necessary.

Path-relinking has been increasingly used to introduce memory in the otherwise memoryless GRASP procedure. The hybridization of GRASP and path-relinking has led to some effective cooperative parallel implementations. Cooperation between the processors is usually achieved by sharing elite solutions,

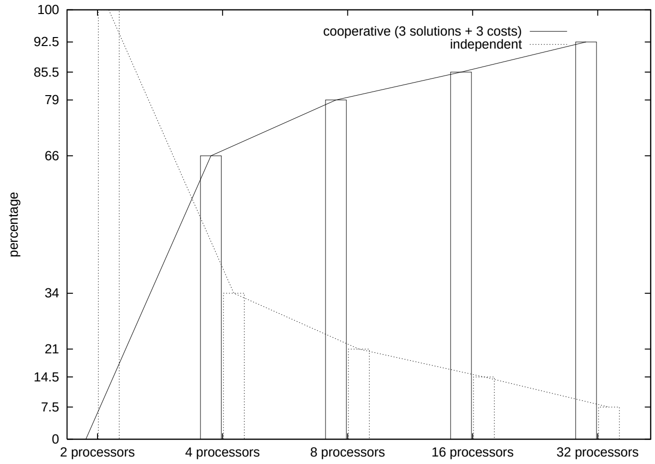

Fig. 13. Pairwise comparisons on the more powerful cluster between the independent and the cooperative parallelizations when costs and solutions are sent separately.

either in a single centralized pool or in distributed pools. Significant speedups are observed in practice for some of these implementations.

We described the 2-path network design problem and a parallel implementation of a GRASP with path-relinking heuristic for its approximate solution. They are used to illustrate the strategies and programming skills involved in the development of robust and efficient parallel cooperative implementations of metaheuristics. Conclusive computational results with speedups of 17.6 on a 32-processor cluster were reported.

References

[1] R.M. Aiex, M.G.C. Resende, P.M. Pardalos, and G. Toraldo. GRASP with path-relinking for the three-index assignment problem. *INFORMS Journal on Computing*, to appear.

[2] 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.

[3] E. Alba. *Parallel Metaheuristics: A New Class of Algorithms.* Wiley, 2005.

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

[5] 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, 2002.

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

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

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

[9] P. Festa, P.M. Pardalos, M.G.C. Resende, and C.C. Ribeiro. Randomized heuristics for the max-cut problem. *Optimization Methods and Software*, 7:1033–1058, 2002.

[10] 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, 2002.

[11] 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.

[12] 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.

[13] F. Glover, M. Laguna, and R. Martí. Scatter search and path relinking: Advances and applications. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 1–35. Kluwer, 2003.

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

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

[16] 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, 2003.

[17] 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*. Kluwer, 2005.

[18] 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.

[19] M.C. Souza, C. Duhamel, and C.C. Ribeiro. A GRASP with path-relinking heuristic for the capacitated minimum spanning tree problem. In M.G.C. Resende and J. Souza, editors, *Metaheuristics: Computer Decision Making*, pages 627–658. Kluwer, 2003.

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

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