Metaheuristics for optimization problems in computer communications

Celso C. Ribeiro

Universidade Federal Fluminense, Department of Computer Science, Rua Passo da Pátria 156, Niterói, RJ 22410-240, Brazil.

Simone L. Martins

Universidade Federal Fluminense, Department of Computer Science, Rua Passo da Pátria 156, Niterói, RJ 22410-240, Brazil.

Isabel Rosseti

ONS, Rua da Quitanda 196, Rio de Janeiro, RJ 20091-000, Brazil.

Abstract

Recent years have witnessed huge advances in computer technology and communication networks, entailing hard optimization problems in areas such as network design and routing. Metaheuristics are general high-level procedures that coordinate simple heuristics and rules to find good approximate solutions to computationally difficult combinatorial optimization problems. They are among the most effective solution strategies for solving optimization problems in practice and have been applied to a very large variety of problems in telecommunications, computer communications, and network design and routing. In this paper, we review the principles associated with some of the main metaheuristics and we give templates for basic implementations of them: simulated annealing, tabu search, GRASP, VNS, genetic algorithms, and path-relinking. The main strategies underlying the development of parallel implementations of metaheuristics are also reviewed. Finally, we present an account of some successful applications of metaheuristics to optimization problems in telecommunications, computer communications, and network design and routing.

Key words: Metaheuristics, telecommunications, computers, communications, networks, network design, network routing

Email addresses: `celso@ic.uff.br` (Celso C. Ribeiro), `simone@ic.uff.br` (Simone L. Martins), `rosseti@ons.org.br` (Isabel Rosseti).

1 Introduction

Combinatorial optimization problems involve finding optimal solutions from a discrete set of feasible solutions. However, even with the advent of new computer technologies and parallel processing, many of these problems cannot be solved to optimality in reasonable computation times, due to their inner nature or to their size. Moreover, reaching optimal solutions is meaningless in many practical situations, since we are often dealing with rough simplifications of reality and the available data is not precise. The goal of approximate algorithms (or heuristics) is to quickly produce good approximate solutions, without necessarily providing any guarantee of solution quality.

Metaheuristics are general high-level procedures that coordinate simple heuristics and rules to find good (often optimal) approximate solutions to computationally difficult combinatorial optimization problems. Among them, we find simulated annealing, tabu search, GRASP, genetic algorithms, scatter search, VNS, ant colonies, and others. They are based on distinct paradigms and offer different mechanisms to escape from locally optimal solutions, contrarily to greedy algorithms or local search methods. Metaheuristics are among the most effective solution strategies for solving combinatorial optimization problems in practice and they have been applied to a very large variety of areas and situations. The customization (or instantiation) of some metaheuristic to a given problem yields a heuristic to the latter.

The main principles underlying the most widely used metaheuristics are reviewed in the next section. Templates of these main metaheuristics are also given. Parallel implementations of metaheuristics entail more robust algorithms and appear quite naturally as an effective approach to speedup the search for good solutions. Strategies for the parallelization of metaheuristics are reviewed in Section 3. Applications of metaheuristics to several problems in telecommunications, computer communications, network design, and network routing are reviewed in Section 4.

2 Metaheuristics and templates

In this section, we review the main principles and ideas involved with greedy algorithms, local search procedures, and five widely and successfully used metaheuristics: simulated annealing, tabu search, GRASP, VNS, and genetic algorithms. We also discuss the main ideas underlining path-relinking, an intensification or post-optimization technique that can be used together with different metaheuristics.

In terms of notation, we consider a combinatorial optimization problem formulated as to minimize $f(S)$ subject to $S \in X$, defined by a ground set $E = \{e_1, \ldots, e_n\}$, a set of feasible solutions $X \subseteq 2^E$, and an objective function $f : 2^E \rightarrow \mathbb{R}$. We seek an optimal solution $S^* \in X$ such that $f(S^*) \leq f(S)$, $\forall S \in X$. The ground set E, the objective function f, and the feasible set X are specific to each problem.

2.1 Greedy algorithms

In a greedy algorithm, solutions are progressively built from scratch. At each iteration, a new element from the ground set E is incorporated into the partial solution under construction, until a complete feasible solution is obtained. The selection of the next element to be incorporated is determined by the evaluation of all candidate elements according to a greedy evaluation function. This greedy function usually represents the incremental increase in the cost function due to the incorporation of this element into the partial solution under construction. The greediness criterion establishes that the element with the smallest incremental increase is selected, with ties being arbitrarily broken. Figure 1 provides a template for a greedy algorithm for a minimization problem.

procedure `GreedyAlgorithm`()
1. $S \leftarrow \emptyset$;
2. Evaluate the incremental cost of each element $e \in E$;
3. **while** S is not a complete feasible solution **do**
4.   Select the element $s \in E$ with the smallest incremental cost;
5. $S \leftarrow S \cup \{s\}$;
6.   Update the incremental cost of each element $e \in E \setminus S$;
7. **end_while**;
8. **return** S;

end.

Fig. 1. Greedy algorithm for minimization.

The solutions obtained by greedy algorithms are not necessarily optimal. Greedy algorithms are often used to build initial solutions to be explored by local search or metaheuristics.

Greedy randomized algorithms are based on the same principle of pure greedy algorithms, but make use of randomization to build different solutions at different runs. Figure 2 illustrates the pseudo-code of a greedy randomized algorithm for minimization. At each iteration, the set of candidate elements is formed by all elements that can be incorporated into the partial solution under construction without destroying feasibility. As before, the selection of

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

Fig. 2. Greedy randomized algorithm for minimization.

the next element is determined by the evaluation of all candidate elements according to a greedy evaluation function. The evaluation of the elements by this function leads to the creation of a restricted candidate list (RCL) formed by the best elements, i.e. those whose incorporation into the current partial 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 has been incorporated into the partial solution, the incremental costs are reevaluated.

2.2 Local search

A neighborhood of a solution S is a set $N(S) \subseteq X$. Each solution $S' \in N(S)$ is reached from S by an operation called a *move*. Normally, two neighbor solutions S and $S' \in N(S)$ differ by only a few elements. Neighborhoods may also eventually contain infeasible solutions not in X. Metaheuristics such as VNS make use of multiple neighborhood structures, see Section 2.6.

A solution S^* is a local optimum with respect to a given neighborhood $N()$ if $f(S^*) \leq f(S), \forall S \in N(S^*)$. Local search methods are based on the exploration of solution neighborhoods. The current solution is successively replaced by a better solution in its neighborhood, until a local optimum is found, i.e. when no better solution can be found in the neighborhood of the current solution. The pseudo-code of a basic local search algorithm for a minimization problem is given in Figure 3. It starts from a solution S_0 and makes use of a neighborhood structure $N()$.

The effectiveness of a local search procedure depends on several aspects, such as the neighborhood structure, the neighborhood search technique, the speed of evaluation of the cost function, and the starting solution. The neighborhood

```
procedure LocalSearch(S_0)
1.  Set S ← S_0;
2.  while S is not locally optimal do
3.      Find S' ∈ N(S) with f(S') < f(S);
4.      S ← S';
5.  end;
6.  return S;
end.
```

Fig. 3. Local search algorithm for minimization.

search may be implemented using either a *best-improving* or a *first-improving* strategy. In the case of a best-improving strategy, all neighbors are investigated and the current solution is replaced by the best neighbor. In the case of a first-improving strategy, the current solution moves to the first neighbor whose cost function value is smaller than that of the current solution.

2.3 Simulated annealing

Simulated annealing was introduced by Kirkpatrick et al. [46] and its basic ideas come from an analogy with the physical annealing process. In physics, annealing is a thermal process for obtaining low energy states of a solid in a heat bath. The principle of simulated annealing lies in the following analogy between the physical process and a combinatorial optimization problem: solutions of the combinatorial problem are equivalent to states of the physical problem and their cost to the energy of a state. Neighboring solutions correspond to states generated by the perturbation method.

The pseudo-code of a simulated annealing algorithm for a minimization problem is described in Figure 4. The algorithm starts from an initial solution S_0 and the temperature is set at T_0. Iterations are performed until a certain *stopping criterion* is met. In most implementations, the latter corresponds to bringing the temperature to a very small value close to zero. For each temperature T, the inner loop is performed until *thermal equilibrium* is reached. The latter is often implemented as a fixed number of iterations which depends on the temperature. A neighbor solution S' and the variation ΔE in the objective value are computed at each iteration. The new solution S' replaces the incumbent if $\Delta E < 0$, i.e., if the new solution is better. The same happens with probability $e^{-\Delta E/(K_B T)}$ in case the new solution is worse than the current one. Once *thermal equilibrium* is reached, the temperature T is reduced according with the *annealing schedule*. In most implementations the new temperature is geometrically reduced, by the multiplication of the current temperature by a constant smaller than one.

```
procedure SimulatedAnnealing(S_0)
    Set S, S* ← S_0;
    Compute the initial temperature T_0 and set T ← T_0;
    while stopping criterion is not reached do
        while thermal equilibrium is not reached do
            Obtain a neighbor solution S' ∈ N(S) at random;
            Compute ΔE = f(S') − f(S);
            if ΔE < 0 then S ← S';
            else if e^{−ΔE/(K_B T)} > random[0, 1) then S ← S';
            end_if
            if f(S) < f(S*) then S* ← S;
        end_while
        Decrease T according with the annealing schedule;
    end_while;
    return S*;
end.
```

Fig. 4. Template of a simulated annealing heuristic for minimization.

The algorithm starts with large values for the temperature T, allowing bad solutions to replace the incumbent. As the algorithm executes, the temperature T is decreased and it becomes harder to accept bad solutions. No further deteriorations are accepted when T approaches to zero. Convergence relies on several implementation choices: (a) the initial value T_0 of the temperature, (b) the *annealing schedule* to reduce the temperature, (c) the *stopping criterion*, and (d) the conditions for reaching the *thermal equilibrium* at each temperature. Convergence and strategies for such implementation choice are discussed e.g. in [1,2,42].

The main advantage of simulated annealing is its simplicity of implementation. However, the speed of convergence and solution quality rely on several implementation choices that may not be easy to tune. Furthermore, the algorithm is memoryless and does not use cost information about complete neighborhoods.

2.4 Tabu search

The fundamental ideas of *tabu search* were independently proposed by Glover [31] and Hansen [36]. The approach was later developed by Glover [32,33]. It may be viewed as a dynamic neighborhood method that makes use of memory to drive the search by escaping from local optima and avoiding cycling [35]. Contrarily to memoryless heuristics such as simulated annealing, and to methods that use rigid memory structures such as branch-and-bound, tabu search makes use of flexible and adaptive memory designs.

For any incumbent solution S, in the case of tabu search the neighborhood $N(S)$ is not a static set and can change according to the history of the search. At each local search iteration, tabu search looks for the neighbor solution that most improves the objective function. However, some neighbors in $N(S)$ are forbidden and discarded. The set of forbidden (tabu) neighbors is stored in a short-term memory formed by the `TabuTenure` last visited solutions, so as to preclude the search from returning to previously visited solutions. Contrarily to plain local search procedures, tabu search chooses the move which least deteriorates the objective function value whenever there are no improving moves.

The basic steps of a simple tabu search heuristic for minimization are described in the pseudo-code of Figure 5. The algorithm starts from an initial solution S_0 and the short-term memory is initialized. Iterations are performed until a certain *stopping criterion* is met. In most implementations, the latter corresponds to an upper limit on the number of consecutive moves without improvement in the best solution value. Procedure `SelectBestNeighbor` returns the best non-forbidden neighbor solution S'. The incumbent solution is replaced by the latter and the best known solution is eventually updated. If the tabu list is full, then the oldest forbidden solution is removed from T. The incumbent solution solution is inserted into the tabu list.

```
procedure TabuSearch(S_0);
    Set S, S* ← S_0;
    T ← ∅;
    while stopping criterion is not reached do
        S' ← SelectBestNeighbor(N(S)\T);
        S ← S';
        if f(S) < f(S*) then S* ← S;
        if |T| = TabuTenure then remove from T the oldest solution;
        T ← T ∪ S;
    end_while;
    return S*;
end.
```

Fig. 5. Template of a short-term memory tabu search heuristic for minimization.

The tabu search scheme above described makes use of very simple ideas and can be further extended by the incorporation of more sophisticated features. Among them, we may cite the use of aspiration criteria to override the tabu status of unvisited solutions, the use of medium-term and long-term memories to implement intensification and diversification procedures, the use of candidate list strategies to speedup the search, and the use of hashing tables to speedup and filter the search.

Tabu search is certainly among the most effective approaches for solving hard combinatorial optimization problems. It has been successfully applied to a wide range of problems in many domains. However, implementations of tabu search often involve setting many parameters, that have to be appropriately tuned for achieving good performance in practice. The interested reader is referred to [35] for a thorough study of tabu search and related ideas.

2.5 GRASP

GRASP (Greedy Randomized Adaptive Search Procedure) [23,24,64,65,67] is a multistart 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 6 illustrates the main blocks of a GRASP procedure for minimization, in which `MaxIterations` iterations are performed and `Seed` is used as the initial seed for the pseudo-random number generator.

```
procedure GRASP(MaxIterations, Seed)
    Set f* ← ∞;
    for k = 1, ..., MaxIterations do
        S ← GreedyRandomizedAlgorithm(Seed);
        S ← LocalSearch(S);
        if f(S) < f* then
            S* ← S;
            f* ← f(S);
        end_if;
    end_for;
    return S*;
end.
```

Fig. 6. Template of a GRASP heuristic for minimization.

An especially appealing characteristic of GRASP is the ease with which it can be implemented. Few parameters need to be set and tuned, and therefore development can focus on implementing efficient data structures to assure quick iterations. Basic implementations of GRASP rely exclusively on two parameters: the number `MaxIterations` of iterations and the parameter used to limit the size of the restricted candidate list within the greedy randomized algorithm used by the construction phase. In spite of its simplicity and ease of implementation, GRASP is a very effective metaheuristic and produces the best known solutions for many problems, see also [26] for an extensive survey of applications.

GRASP as originally proposed is a memoryless procedure, in which each iteration does not make use of information gathered in previous iterations. Path-relinking is a major enhancement that is able to add memory to the basic GRASP procedure, leading to significant improvements in solution time and quality. Path-relinking is described in detail in Section 2.8.

In the context of a GRASP implementation, path-relinking is applied to pairs of solutions, one of them being a locally optimal solution and the other randomly chosen from a pool with a limited number `MaxElite` of elite solutions found along the search. The pool of elite solutions 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 replacing the latter. The use of path-relinking within a GRASP procedure, as an intensification strategy applied to each locally optimal solution, was first proposed by Laguna and Martí [49]. It was followed by several extensions, improvements, and successful applications [16,63–66,68,69,71].

The pseudo-code in Figure 7 illustrates the main steps of a GRASP procedure using path-relinking to implement a memory-based intensification strategy.

```
procedure GRASPwithPathRelinking(MaxIterations, Seed)
    Set f* ← ∞;
    Set Pool ← ∅;
    for k = 1, ..., MaxIterations do
        S ← GreedyRandomizedAlgorithm(Seed);
        S ← LocalSearch(S);
        if k > 1 then
            Randomly select a solution S' ∈ Pool;
            S ← PathRelinking(S', S);
        endif;
        if f(S) < f* then
            S* ← S;
            f* ← f(S);
        end_if;
        Add S to Pool if it satisfies the membership conditions;
    end_for;
    return S*;
end.
```

Fig. 7. Template of a GRASP with path-relinking heuristic for minimization.

2.6 Variable neighborhood search

VNS (Variable Neighborhood Search) is a metaheuristic proposed by Hansen and Mladenovic [37–39,54] based on the systematic change of neighborhood. It makes use of a finite set of k_{max} pre-selected neighborhood structures identified as $N_1, N_2, \ldots, N_{k_{max}}$ that may be induced from one or more metric functions introduced in the solution space X. We denote by $N_k(S)$ the set of solutions in the kth neighborhood of S.

Figure 8 summarizes the main steps of the pseudo-code of a VNS heuristic starting from an initial solution S_0. Iterations of the outer loop are performed until a certain *stopping criterion* is met. The current neighborhood is set to $k = 1$ and neighborhoods N_1 to $N_{k_{max}}$ of the current solution are progressively scanned within the inner loop. The search starts from the lowest order neighborhood $N_1(S)$ of the current solution. The initial step of each iteration is a random perturbation move applied to generate a solution S' in the current neighborhood $N_k(S)$. Next, a locally optimal solution S'' is obtained by the application of local search to S'. If solution S'' is better than the incumbent S, then the former replaces the latter and the search resumes from neighborhood $N_1(S)$. Otherwise, the current solution S is not modified, k is incremented by one, and the search resumes from a higher order neighborhood of S.

```
procedure VNS(k_max, Seed, S_0)
    Set S ← S_0;
    while stopping criterion is not reached do
        k ← 1;
        while k ≤ k_max do
            Obtain a neighbor solution S' ∈ N_k(S) at random;
            S'' ← LocalSearch(S');
            if f(S'') < f(S) then
                S ← S'';
                k ← 1;
            else
                k ← k + 1;
            endif;
        end_while;
    end_while;
    return S;
end.
```

Fig. 8. Template of a VNS heuristic for minimization.

VNS is also a memoryless heuristic, being very easy to implement and relying on very few parameters: the *stopping criterion* and the number k_{max} of neighborhoods. The former may be the maximum number of iterations, the maximum number of iterations between two improvements, or the maximum number of times the highest order neighborhood $N_{k_{max}}$ is attained.

2.7 Genetic algorithms

Genetic algorithms were first introduced and investigated by Holland [43,44]. They are population-based methods that use selection and recombination operators to generate new solutions in the search space, imitating the process of natural selection and evolution of species. Solutions are evaluated in terms of their fitness, which most often corresponds to the value of the objective function itself.

Contrarily to other metaheuristics reviewed in this chapter, genetic algorithms explore a population of solutions and not a single trajectory evolving from a unique initial solution. Given a population of solutions, the main operators involved in standard implementations of genetic algorithms are:

- `SelectMates`: a subset of solutions is selected, either randomly or using fitness information to privilege promising solutions.
- `RecombineParents`: pairs of solutions are combined and new solutions are generated (crossover operation).
- `ApplyMutation`: a few solutions are randomly selected and go through small modifications in their structure (mutation operation).
- `SelectBest`: the best solutions are selected, while the others are eliminated.

procedure `GeneticAlgorithm()`
1. Generate the initial population P of solutions;
2. `EvaluateFitness`(P);
3. **while** *stopping criterion* is not reached **do**
4. $Parents \leftarrow$ `SelectMates`(P);
5. $Children \leftarrow$ `RecombineParents`($Parents$);
6. $Children \leftarrow$ `ApplyMutation`($Children$);
7. `EvaluateFitness`($Children$);
8. $P \leftarrow$ `SelectBest`($Children \cup P$);
9. **end_while**
10. **return** the best solution in P;

end.

Fig. 9. Template of a genetic algorithm.

The pseudo-code of the basic steps of a genetic algorithm are described in Figure 9. The procedure starts by generating an initial population P of solutions, either entirely randomly or using a greedy randomized algorithm. The fitness of each solution S in the population is evaluated. The loop is performed until some *stopping criterion* is reached. Most applications use the number of iterations or the stabilization of the best solution in the population as the stopping criterion. Each iteration of this loop handles a generation of the population. The process of obtaining a new generation starts by the selection of a subset $Parents$ of solutions from the current population. Solutions are pairwise combined by crossover operations and a set $Children$ of off-springs results. A small subset of the latter go through mutations and the fitness of every newly generated solution is evaluated. Finally, the best among the old and the new solutions are selected to form a new generation of solutions and a new iteration starts.

Classical genetic algorithms do not incorporate any problem-specific knowledge and rely almost entirely on randomization for mate selection, crossover, mutation, and selection. Hybrid genetic algorithms, also called memetic algorithms, make use of specific knowledge available to solve the problem. One of such strategies consists in applying local search to some elements in the generation. Genetic algorithms are very appealing and easy to implement. However, many implementation choices for each of the above operators are possible. A good implementation requires an appropriate solution encoding as finite length data strings and well tuned strategies for the generation of the initial population, mate selection, crossover, mutation, and selection; see e.g. [60,61].

2.8 Path-relinking

Path-relinking was originally proposed by Glover [34] as an intensification strategy exploring trajectories connecting elite solutions obtained by tabu search or scatter search. Starting from one or more elite solutions, paths in the solution space leading toward other elite solutions are generated and explored 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.

The algorithm in Figure 10 illustrates the pseudo-code of the path-relinking procedure applied to a pair of solutions S_s (starting solution) and S_t (target solution). The procedure starts by computing the symmetric difference $\Delta(S_s, S_t)$ between the two solutions, i.e. the set of moves needed to reach S_t from S_s. A path of solutions is generated linking S_s to S_t. The best solution $\bar{S}$ in this path is returned by the algorithm. At each step, the procedure examines all moves $m \in \Delta(S, S_t)$ from the current solution S and selects the one which results in the least cost solution, i.e. the one which minimizes $f(S \oplus m)$, where $S \oplus m$ is the solution resulting from applying move m to solution S. The best move m^* is made, producing solution $S \oplus m^*$. The set of available

```
procedure PathRelinking(S_s, S_t)
    Compute the symmetric difference Δ(S_s, S_t);
    f̄ ← min{f(S_s), f(S_t)};
    S̄ ← argmin{f(S_s), f(S_t)};
    S ← S_s;
    while Δ(S, S_t) ≠ ∅ do
        m* ← argmin{f(S ⊕ m) : m ∈ Δ(S, S_t)};
        Δ(S ⊕ m*, S_t) ← Δ(S, S_t) \ {m*};
        S ← S ⊕ m*;
        if f(S) < f̄ then
            f̄ ← f(S);
            S̄ ← S;
        end_if;
    end_while;
    return S̄;
end.
```

Fig. 10. Path-relinking procedure for minimization.

moves is updated. If necessary, the best solution $\bar{S}$ is updated. The procedure terminates when S_t is reached, i.e. when $\Delta(S, S_t) = \emptyset$.

Several alternatives have been considered and combined in recent successful implementations of path-relinking in conjunction with tabu search, GRASP, and genetic algorithms [16,25,53,62,63,68–71]: periodical, forward, backward, back and forward, mixed, randomized, truncated, and post-optimization relinking. Resende and Ribeiro [65] reviewed these alternatives, showing that they involve trade-offs between computation time and solution quality.

3 Parallelization of metaheuristics

The computation times associated with the exploration of the solution space may be very large. With the rapid increase in the number of parallel computers, powerful workstations, and fast communication networks, parallel implementations of metaheuristics appear quite naturally as an effective approach to speedup the search for approximate solutions. Besides the accelerations obtained, the parallelization also allows solving larger problems or finding better solutions.

Cung et al. [21] reviewed parallelization strategies, implementation issues, and applications of parallel metaheuristics. Parallel implementations of metaheuristics based on local search use several processors to concurrently generate or explore the neighborhood. Two approaches can be used: single-walk and multiple-walks. In the case of a single-walk parallelization, 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 they do not exchange any information among them) or cooperative (when the information collected along each trajectory is disseminated and used by other threads to improve or to speedup the search).

Efficient parallelizations of metaheuristics are often based on multiple-walk strategies. They can be implemented using independent or cooperative search 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 very easily 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.

Cooperative strategies are the most general and promising, demanding more programming efforts and implementation skills. The search threads exchange and share information collected along the trajectories they investigate. This shared information is implemented either as global variables stored in a shared memory, or as a pool in the local memory of a dedicated central processor which can be accessed by all other processors. In this model, in which the search threads cooperate and the information collected along each trajectory is used to improve the other trajectories, one expects not only to speed up the convergence to the best solution but, also, to find better solutions than independent-thread strategies within the same computation time. The most difficult aspect to be set up is the determination of the nature of the information to be shared or exchanged to improve the search, without taking too much additional memory or time to be collected. We may cite elite solutions and their costs, best solutions found, move frequencies, and tabu lists, among others.

4 Applications in computer communications

Communication networks consist of nodes that can be computers, database repositories, instruments like tomography equipments or radio transmitters, connected by data transmission links such as copper cables, optical fibers, satellite and radio links. Their design involves making decisions on many issues like the number of nodes and their locations, routing paths, capacity installation, wavelength allocation, and frequency assignment. The main objective is often to obtain a least cost network configuration subject to constraints involving issues such as delay, throughput, reliability, link capacity, and crosstalk level.

As most optimization problems in telecommunications are NP-hard and exact methods can only solve small problems, heuristics play a very important role in their solution. Applications of metaheuristics to problems in telecommunications are plentiful in the literature. We give below an account of some of these applications, involving sequential and parallel implementations.

4.1 Applications of sequential algorithms

Kim et al. [45] worked out a simulated annealing algorithm to allocate nominal channels to the cells of a mobile radio system in such a way that the average blocking in the entire system is minimized. Experimental results show that their proposal gives the lowest overall blocking probability and the largest number of simultaneously usable channels, when compared with other three heuristics found in the literature.

Randall et al. [59] showed results obtained by a simulated annealing algorithm developed to find paths in a network which minimize the cost of transporting required origin-destination flows subject to specified link, node capacity, node degree, and chain hop-limit constraints. They showed that the heuristics found good results for very large network design and in smaller times than another genetic algorithm also developed for this problem.

Amaldi et al. [11] proposed two randomized greedy procedures and a tabu search algorithm for finding the location of universal mobile telecommunication system base stations to maximize the traffic covered and to minimize installation costs. The greedy function takes into account the fraction of traffic covered and the installation costs. The randomized greedy procedures provided good approximate solutions for medium- to large-size realistic instances, and the tabu search algorithm was able to further improve the solutions obtained by the greedy procedures.

Cox and Sanchez [18] presented a new heuristic algorithm for designing least-cost telecommunications networks to carry traffic from cell sites to wireless switches while meeting survivability, capacity, and technical compatibility constraints. A short term tabu search metaheuristic was introduced (that do not use longer-term features, such as search intensification and diversification strategies), with embedded knapsack and network flow subproblems. It was proved to be highly effective in designing such backhaul networks for carrying personal communications services traffic. It solved challenging problems for conventional branch-and-bound solvers in minutes instead of hours and found lower-cost solutions. Applied to real-world network design problems, the heuristic has successfully identified designs that save over 20% compared to the best previously known designs.

Fink et al. [27] present a more general problem formulation for ring network design problems and associated methods that apply to a broad range of problems. They implemented a simulated annealing and a tabu search algorithm using a metaheuristics framework, so that no calibrations or specializations with respect to a specific problem type were performed. Reactive tabu search presented better results than simulated annealing.

Gendron et al. [29] proposed a heuristic approach for a network loading problem that alternates construction and local search phases. Initially, a construction method provides a feasible solution, while at subsequent construction steps, a diversification approach is adopted for exploiting information gathered along previous iterations. Two local search procedures are used: a pure descent search and a tabu search heuristic. An implementation of GRASP is also proposed.

Girard et al. [30] examined the use of tabu search for the solution of an access network design problem that uses ADM equipments, which are both concentrators and multiplexers, so that demand can be sent on different SONET channels. They compared a simple tabu algorithm and a more elaborated one, using simulated and real instances. They showed that the latter found improved solutions in much less computation times. They also reported that the differences between the results obtained by tabu search and the exact values were quite small for some instances

Hao et al. [40] presented a tabu search algorithm for assigning frequencies in mobile radio networks, while minimizing electromagnetic interference. Experimental results show that the tabu search algorithm largely outperforms other algorithms based on simulated annealing, constraint programming, graph coloring, and steepest descent, also developed for this problem. Castelino and Stephens [17] developed a surrogate constraint tabu thresholding algorithm for the same problem. The aim of this work was to determine whether using surrogate constraints can improve the solutions found. The use of surrogate constraints consists in combining problem constraints to provide information for guiding the search, instead of using them in isolation. They compared their algorithm to a thresholding tabu search and showed that the surrogate thresholding tabu found better results within the same computation time.

Klincewicz [47] proposed two heuristics based on tabu search and GRASP for the p-hub location problem. The objective is to overcome the difficulty that local search algorithms encounter. The local search procedure of the GRASP algorithm is based on a 2-exchange neighborhood. The same author [48] also proposed heuristics based on GRASP and tabu search for locating hubs in a communication or transportation network. The greedy function of the GRASP heuristic takes into account the amount of originating and terminating traffic of each possible location, while the local search procedure uses a swap neighborhood.

Noronha and Ribeiro [55] developed a tabu search approach for routing and wavelength assignment in all-optical networks. Their algorithm combines the computation of alternative routes for the light-paths with the solution of a partition coloring problem. The computational experiments showed that their approach outperforms the previously best known heuristic for the problem [51].

Pamuk and Sepil [56] proposed a tabu search algorithm to perform optimal location and allocation of hubs in switched networks. They developed three strategies to generate initial solutions and another three to search for an allocation scheme. Nine versions of tabu search heuristics were developed by combining these strategies. Experimental results showed that the three initial solution strategies perform in a similar way and the performance of allocation schemes depends on the type of instance.

Xu et al. [76] employed tabu search for designing a least-cost telecommunications network where the alternate routing paths can be changed dynamically from hour to hour. They conducted computational experiments on data drawn from real and simulated problems and showed that the tabu search algorithm found better results than two other less computer intensive strategies. They concluded that the the most effective tabu search heuristic was the one which integrates tabu search memories, probabilistic rules, and a periodic solution recovery strategy.

Abello et al. [3,4] developed GRASP heuristics to approximately solve the maximum clique and maximum quasi-clique in very large graphs. Their approach is used in data mining to extract communities of interest from telephone call detail graphs.

Canuto et al. [16] developed a GRASP heuristic for the prize-collecting Steiner tree problem, with applications in the design of telecommunications access networks. They proposed a multi-start local search algorithm and path-relinking is used to improve the solutions found by local search. VNS is used as a post-optimization procedure. Their results show that the local search with perturbations approach found optimal solutions on nearly all of the instances tested, in much smaller computation than an exact branch-and-cut algorithm.

Gabrel et al. [28] presented and compared approximate algorithms for discrete cost multicommodity network optimization problems. Firstly, extensions of classical greedy heuristics, based on link-rerouting and flow-rerouting heuristics, are presented in details. Secondly, a new approximate solution algorithm, which basically consists of a heuristic implementation of the exact Benders-type cutting plane generation method, is proposed. All these algorithms are extensively compared on randomly generated graphs up to 50 nodes and 90 links.

Li et al. [50] addressed the problem of server replication and placement. In a multicast network, packets are forwarded from a source server to groups of receivers along a distribution tree, where the source is the root, the receivers are the leaves, and the multicast capable routers are the internal nodes. The problem consists in placing multiple replicated servers within the multicast-capable routers. Several heuristics are proposed, including a GRASP. The greedy function is the router cost function.

Prais and Ribeiro [58] developed a reactive GRASP heuristic for traffic assignment in TDMA communication satellites. A geostationary communication satellite has a number of spot beam antennas covering geographically distributed areas. According to the slot switching configuration and on the on-board switch, the uplink traffic received at the satellite has to be immediately sent to ground areas through a set of transponders. The slot switching configurations are determined through the solution of a time slot assignment problem, which is equivalent to the decomposition of a nonnegative traffic matrix into the sum of a family of switching mode matrices.

Resende and Ribeiro [63] proposed a family of heuristics for routing private virtual circuits. The GRASP with path-relinking variant was able to significantly improve upon some simple heuristics currently used in traffic engineering, at the expense of additional computation time.

Srinivasan et al. [72] presented an approach for efficient design of a signaling network for a network of software switches supporting Internet telephony. The optimal load balancing for given demand forecast is formulated as a quadratic assignment problem, which is solved with a GRASP.

Armony et al. [12] implemented a genetic algorithm to determine how traffic should be routed on self-healing rings, a problem known as the stacked ring design problem. The objective is to optimize the trade-off between the cost of equipments to implement the rings and the cost of exchanging traffic among rings. The initial population of the proposed GA is generated by GRASP. They showed that their GA found good quality solutions using much less time than CPLEX, and also that it always improved the initial solutions generated by GRASP.

Buriol at al. [14] presented a hybrid genetic algorithm for solving the OSPF weight setting problem, combining a genetic algorithm with a local search procedure applied to improve the solutions obtained by crossover. Experimental results showed that the hybrid heuristic found better solutions and led to a more robust implementation than the best known heuristic in the literature.

He and Mort [41] elaborated a hybrid genetic algorithm, where the objective was to minimize the number of point-to-point transmission (hops) for all source-destination pairs and also to minimize the number of congested nodes and links for the routing table. The initial population was created using an heuristic based on a shortest path algorithm with a minimum hop metric. They showed that the hybrid genetic algorithm found solutions with a small number of congested links and nodes than the shortest path with hops heuristic.

Poon et al. [57] described the GenOSys tool developed to optimize the design of secondary and distribution networks used on typical copper access network cabling to connect customers. Its objective is to determine the best locations for distribution points and to identify geographically advantageous tree-structured sub-networks to aggregate cables from customers. The tool allows the user to enter data about the network and provides information which can be used for ducting and cabling using the hybrid genetic algorithm. A practical problem on a network of 240 nodes was solved in less than 30 minutes on a Pentium 200 MHz.

Zomaya et al. [77] described a genetic algorithm to solve the location management problem in mobile computing networks. They developed a strategy to update the mobile hosts location modeled on the characteristics of a network, such as topology, host mobility patterns, and connection request rates. The authors report that experimental results obtained for these algorithms showed improvements in quality over alternative algorithms, and also saved power, processing time, and network bandwidth.

Ant colony optimization was applied by Varela and Sinclair [73] to the problem of routing and wavelength-allocation in a multi-wavelength all-optical virtual-wavelength-path routed transport network. Three variants were proposed and the best one provided results that approached those of an earlier problem-specific heuristic on small- and medium-sized networks.

Wittner et al. [75] developed a swarm algorithm to find a path of resources from a client terminal to a service provider, such that all resources in the path conform with constraints and preferences of a request profile specified by the user. Given a network composed of users, terminals, and services that have individual profiles containing quality of service parameters, the objective is to search for resource paths for each peer-to-peer communication.

Tables 1 and 2 summarize the main references and applications of sequential algorithms.

Table 1
Applications of sequential algorithms.

Year	Ref.	Application	Comments
1992	[47]	Locating hubs	TS and GRASP with several local search procedures
1997	[76]	Designing dynamic networks	TS developed and compared to simple heuristics
1998	[40]	Assigning frequencies in mobile networks	TS, SA, and pure heuristic algorithms
1999	[3]	Finding a maximum clique	GRASP to data mining on telephone call graphs
1999	[17]	Assigning frequencies in mobile networks	A surrogate constraint TS compared to a thresholding TS
1999	[27]	Designing ring networks	Algorithms implemented with a metaheuristics framework
1999	[73]	Routing on optical networks	Three ant colony algorithms developed and compared
2000	[12]	Routing on self-healing rings	GA hybridized with GRASP
2000	[18]	Routing on wireless networks	Short term TS developed
2000	[29]	Loading of a nonbifurcated network	Pure descent local search and TS algorithms
2000	[41]	Routing	GA hybridized with a heuristic to create the initial population
2000	[45]	Allocating channels on mobile radio systems	SA developed to minimize the average blocking
2000	[50]	Placing multiple replicated servers on a network	GRASP and several heuristics proposed and compared
2000	[57]	Designing secondary and distribution networks	Tool developed to use a hybrid GA
2000	[58]	Assigning traffic in TDMA communication satellites	Reactive GRASP compared to pure GRASP
2000	[72]	Designing a signaling network	QAP formulation of the problem solved by a GRASP

Table 2
Applications of sequential algorithms (cont.).

Year	Ref.	Application	Comments
2001	[16]	Designing access network	Hybridization of GRASP with path-relinking and VNS
2001	[30]	Designing access network using SONET channels	Two TS algorithms
2001	[56]	Locating hubs	Nine TS algorithms
2002	[4]	Finding a maximum quasi-clique	GRASP to data mining on telephone call graphs
2002	[48]	Locating hubs	TS and GRASP with several local search procedures
2002	[59]	Routing	SA and GA algorithms
2003	[11]	Locating UMBS	Greedy heuristics and TS algorithms
2003	[28]	Designing multicommodity networks	Extensions of classical greedy heuristics
2003	[63]	Routing private virtual circuits	GRASP with PR compared to other simple heuristics
2003	[75]	Routing	Swarm algorithm
2003	[77]	Locating hosts in mobile networks	GA algorithms
2004	[14]	Setting weights for OSPF routing	GA hybridized with a local search procedure applied to crossover
2006	[55]	Routing on all-optical networks	TS compared to best known heuristics

To give the flavor of all steps involved in the complete implementation of a parallel metaheuristic, we first report in this section a full application of a parallel GRASP heuristic in the solution of a typical network design problem [68]. This case study is followed by a review of applications of parallel metaheuristics to other optimization problems in telecommunications.

4.2.1 Parallel GRASP for the 2-path network design problem

4.2.2 Problem formulation and applications

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 $w : E \rightarrow R_+$ associated with the edges of G and a set D of pairs of origin-destination nodes, the 2-path network design problem (2PNDP) consists of finding a minimum weighted subset of edges $E' \subseteq E$ containing a 2-path between every origin-destination pair. 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 [22].

4.2.3 GRASP construction

The construction phase uses modified edge weights. They are initialized with the original weights, but are progressively modified along the iterations. 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 current value of the modified edge weights. The modified weights of the edges in this 2-path are set to zero (since they can be used at no extra cost by other 2-paths) 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.

4.2.4 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.

4.2.5 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 the determination of 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.

In the case of the independent strategy for the parallel implementation of the GRASP with path-relinking heuristic 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 parallel, 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. 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 [65] between the same pair of starting and guiding solutions, respectively.

4.2.6 Computational results

The results illustrated in this section concern an instance with 100 nodes, 4950 edges, and 1000 origin-destination pairs. The methodology proposed in [5,6] is used 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, a solution target value is fixed and each algorithm executes 200 times, recording the running time when a solution with cost at least as good as the target value is found. For each algorithm, the i-th sorted running time t_i is associated with a probability $p_i = (i - \frac{1}{2})/200$ and the points $z_i = (t_i, p_i)$, for $i = 1, \ldots, 200$ are plotted.

The 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. The plots in Figure 11 (using different scales) display the empirical probability distribution of the random variable *time to target solution value* for the independent and cooperative parallel implementations, for runs with the target value set at 683 on two, four, and eight processors. The independent strategy is naturally faster when fewer processors are used, 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, since the p-th handles the pool). This behavior changes when the number of processors increases: computation times get smaller and the plots shift to the left. The reduction in computation times is more significant for the cooperative parallelization strategy: as the number of processors increases, the gains obtained through cooperation are more important than the fact that the independent parallelization has one additional processor to perform iterations. For most runs, the cooperative implementation is already faster than the independent for eight processors.

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. A number of different strategies were investigated to further improve the performance of the cooperative implementation, by reducing the cost of the communication between the master and the slaves when the number of processors increases. In the most well succeeded strategy, 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 are very small ones with light memory requirements.

All runs were repeated using this new strategy on a faster and more powerful Linux cluster 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 for communication between the processors. 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.

4.2.7 Other parallel applications

The parallelization of the reactive GRASP heuristic developed by Prais and Ribeiro [58] for the problem of traffic assignment in TDMA communication satellites led to linear speedups in [10].

Bastos and Ribeiro [13] implemented an independent-thread multiple-walk parallel reactive tabu search strategy, which is among the most effective and robust heuristics for the Steiner problem in graphs. Although the threads are independent, they cooperate either periodically or at a post-optimization phase using a pool of good solutions.

The multicommodity location-allocation problem with balancing requirements [20] was one of the first applications addressed by cooperative-thread implementations of tabu search. The algorithm is centralized, as in most parallelizations of this type. The trajectories start from different initial solutions and communicate through a central pool of elite solutions. In a similar domain, another application to the design of capacitated multi-resource networks is described in [19]. Improved solutions and smaller computation times are obtained for both applications, with respect to the sequential algorithms.

Watanabe et al. [74] proposed a new type of parallel genetic algorithm model for multi-objective optimization problems. It was applied to solve an antenna arrangement problem in mobile communications. Their algorithm showed a very good performance when compared to other methods.

Maple et al. [52] presented a parallel genetic algorithm for finding solutions to the third generation radio network planning problem. They presented a genetic encoding of the problem and proposed an implementation of a coarse grained genetic program, but they did not implement the parallel algorithm.

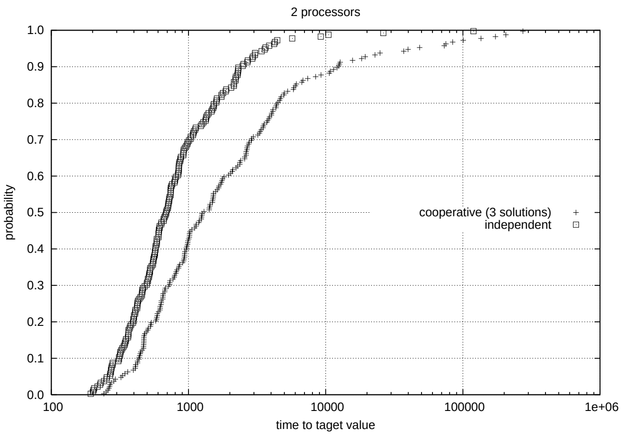
Alba et al. [9] developed some parallel evolutionary algorithms for designing error correcting codes, and finding optimal placements for antennas in radio networks. They reported several results obtained using steady-state, generational, and cellular evolutionary algorithms. From a global point of view, the results have been very satisfactory for the two problems.

Alba and Chicano [7] presented a new local search algorithm (repulsion algorithm) for the problem of finding an error coding code. They used a hybrid between a parallel genetic algorithm and this new algorithm, and compared it against a pure parallel genetic algorithm. The results showed that improvements were achieved with the repulsion algorithm and parallelism.

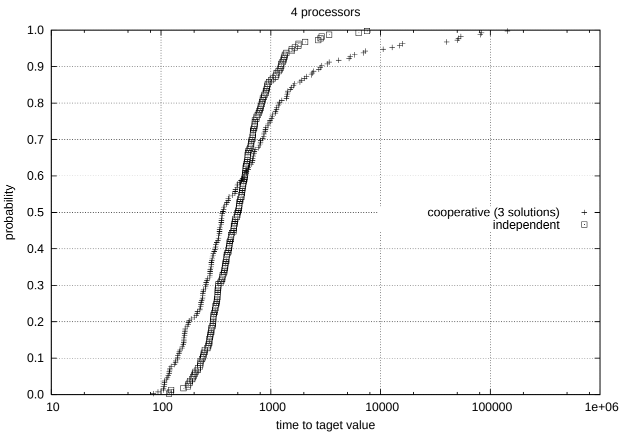
The same authors applied evolutionary algorithms [8] to solve the radio network design problem. The task is to find the best set of transmitter locations in order to cover a given geographical region at an optimal cost. They performed computational experiments on the sequential and parallel versions of the evolutionary algorithms. They report that the numerical results showed that the distributed steady state genetic algorithm is an efficient and accurate tool for solving this problem, and even outperforms existing parallel solutions.

Cahon et al. [15] developed a white-box object-oriented framework dedicated to the reusable design of parallel and distributed metaheuristics. It provides features for evolutionary algorithms, local search procedures, most common parallel and distributed models and hybridization mechanisms. They show results obtained for solving a radio network design problem. The problem consists in finding a set of sites for antennas from a set of pre-defined candidate locations, determining the type and the number of antennas, and their configuration parameters. They investigated a parallel distributed multi-objective genetic algorithm developed to solve this problem using this framework. Three parallel/distributed models are available from this framework. They implemented the three models and compared their performance.

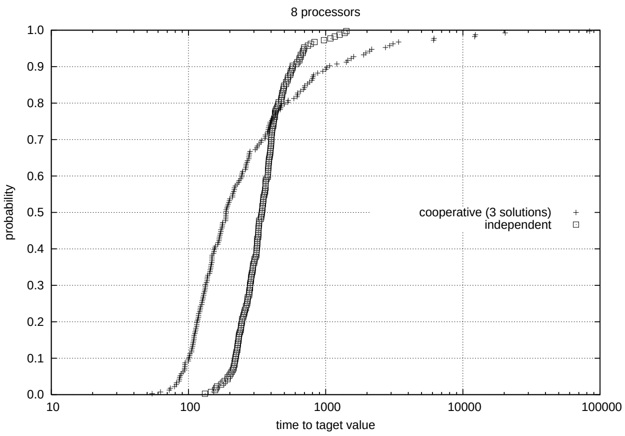
Table 3 summarizes the main references and applications of parallel algorithms.



(a)



(b)



(c)

Fig. 11. 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, (b) four processors, and (c) eight processors, with the target solution value set at 683.

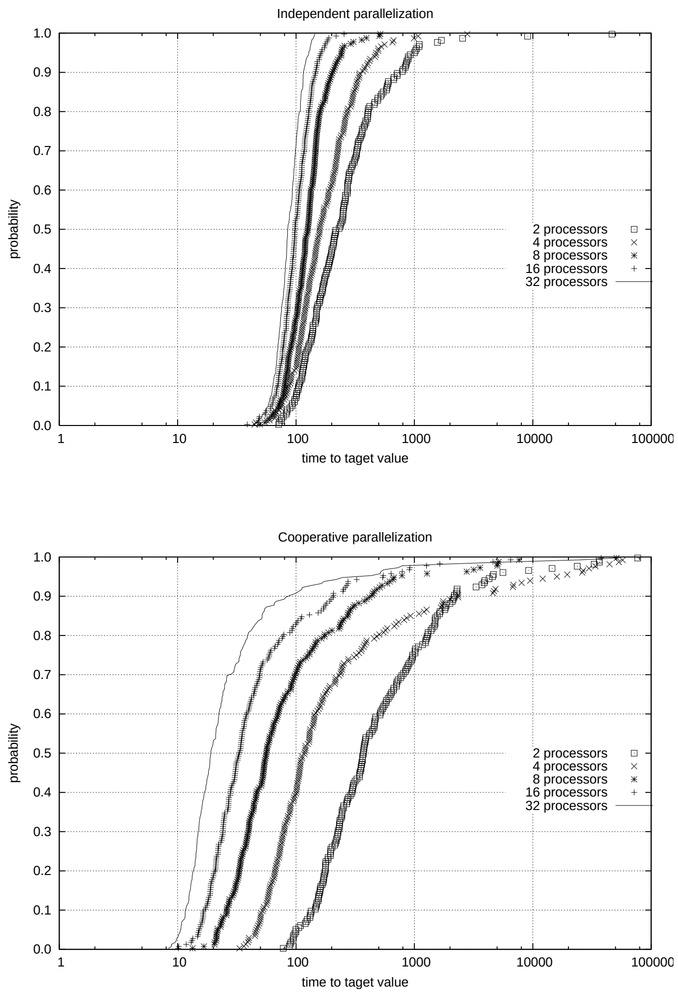


Fig. 12. Empirical distributions on the more powerful cluster of the *time to target solution value* for the independent and cooperative parallelizations.

Table 3
Applications of parallel algorithms.

Year	Ref.	Application	Comments
1995	[20]	Multicommodity location-allocation problem	Cooperative-thread implementation of TS
1998	[10]	Assigning traffic in TDMA communication satellites	Reactive GRASP implemented using independent strategy
2001	[13]	Steiner problem	Independent-thread multiple-walk reactive TS algorithm
2001	[74]	Arranging antennas in mobile networks	Parallel GA
2002	[9]	Designing error correcting codes	Parallel hybrid evolutionary algorithms
2002	[9]	Placing antennas in radio networks	Parallel hybrid evolutionary algorithms
2002	[19]	Designing of multi-resource networks	Cooperative-thread implementation of TS
2002	[68]	Designing 2-path networks	Independent and cooperative parallel GRASPs
2004	[7]	Finding an error coding code	Parallel hybrid GA with repulsion algorithm
2004	[15]	Designing radio networks	Framework to design parallel and distributed metaheuristics
2004	[52]	Designing radio networks	Specification of a coarse grained parallel GA
2005	[8]	Designing radio networks	Distributed steady state GA

References

[1] E. Aarts and J. Korst. Selected topics in simulated annealing. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 1–37. Kluwer, 2002.

[2] E.H.L. Aarts and J. Korst. *Simulated annealing and Boltzmann machines: A stochastic approach to combinatorial optimization and neural computing*. Wiley, 1989.

[3] J. Abello, P. Pardalos, and M. Resende. On maximum clique problems in very large graphs. In J. Abello and J. Vitter, editors, *External memory algorithms and visualization*, volume 50 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 119–130. American Mathematical Society, 1999.

[4] J. Abello, M. Resende, and S. Sudarsky. Massive quasi-clique detection. *Lecture Notes in Computer Science*, 2286:598–612, 2002.

[5] R.M. Aiex, M.G.C. Resende, and C.C.Ribeiro. TTTPLOTS: A perl program to create time-to-target plots, submitted for publication, 2005.

[6] 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.

[7] E. Alba and F. Chicano. Solving the error correcting code problem with parallel hybrid heuristics. In *Proceedings of the ACM Symposium on Applied Computing*, pages 985–989, Nicosia, 2004. ACM.

[8] E. Alba and F. Chicano. On the behavior of parallel genetic algorithms for optimal placement of antennae in telecommunications. *International Journal of Foundations of Computer Science*, 16:343–359, 2005.

[9] E. Alba, C. Cotta, F. Chicano, and A.J. Nebro. Parallel evolutionary algorithms in telecommunications: Two case studies. In *Congreso Argentino de Ciencias de la Computación (CACIC'2002)*, Buenos Aires, 2002.

[10] A.C. Alvim and C.C. Ribeiro. Load balancing for the parallelization of the GRASP metaheuristic. In *Proceedings of the X Brazilian Symposium on Computer Architecture*, pages 279–282, Búzios, 1998. In Portuguese.

[11] E. Amaldi, A. Capone, and F. Malucelli. Planning UMTS base station location: Optimization models with power control and algorithms. *IEEE Transactions on Wireless Communications*, 2:939–952, 2003.

[12] M. Armony, J.C. Klincewicz, H. Luss, and M.B. Rosenwein. Design of stacked self-healing rings using a genetic algorithm. *Journal of Heuristics*, 6:85–105, 2000.

[13] M.P. Bastos and C.C. Ribeiro. Reactive tabu search with path relinking for the Steiner problem in graphs. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 39–58. Kluwer, 2001.

[14] L.S. Buriol, M.G.C. Resende, C.C. Ribeiro, and M. Thorup. A hybrid genetic algorithm for the weight setting problem in OSPF/IS-IS routing. *Networks*, 46:36–56, 2005.

[15] S. Cahon, N. Melab, and E. G. Talbi. ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics*, 10:357–380, 2004.

[16] 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.

[17] D. Castelino and N. Stephens. A surrogate constraint tabu thresholding implementation for the frequency assignment problem. *Annals of Operations Research*, 86:259–270, 1999.

[18] L.A. Cox and J.R. Sanchez. Designing least-cost survivable wireless backhaul networks. *Journal of Heuristics*, 6:525–540, 2000.

[19] T.G. Crainic and M. Gendreau. Cooperative parallel tabu search for capacitated network design. *Journal of Heuristics*, 8:601–627, 2002.

[20] T.G. Crainic, M. Toulouse, and M. Gendreau. Parallel asynchronous tabu search in multicommodity location-allocation with balancing requirements. *Annals of Operations Research*, 63:277–299, 1995.

[21] V.-D. Cung, S.L. Martins, C.C. Ribeiro, and C. Roucairol. Strategies for the parallel implementation of metaheuristics. In C.C. Ribeiro and C.C. Ribeiro, editors, *Essays and Surveys in Metaheuristics*, pages 263–308. Kluwer, 2002.

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

[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] 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.

[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, 2002.

[27] A. Fink, G. Schneidereit, and S. Voss. Solving general ring network design problems by meta-heuristics. In M. Laguna and J. L. González, editors, *Computing Tools for Modeling, Optimization and Simulation*, pages 91–113. Kluwer, 1999.

[28] V. Gabrel, A. Knippel, and M. Minoux. A comparison of heuristics for the discrete cost multicommodity network optimization problem. *Journal of Heuristics*, 9:429–445, 2003.

[29] B. Gendron, J.-Y. Potvin, and P. Soriano. Diversification strategies in local search for a nonbifurcated network loading problem. *European Journal of Operational Research*, 142:231–241, 2000.

[30] A. Girard, B. Sansó, and L. Dadjo. A tabu search algorithm for access network design. *Annals of Operations Research*, 106:229–262, 2001.

[31] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533–549, 1986.

[32] F. Glover. Tabu search - Part I. *ORSA Journal on Computing*, 1:190–206, 1989.

[33] F. Glover. Tabu search - Part II. *ORSA Journal on Computing*, 2:4–32, 1990.

[34] 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.

[35] F. Glover and M. Laguna. *Tabu Search*. Kluwer, 1997.

[36] P. Hansen. The steepest ascent mildest descent heuristic for combinatorial programming. In *Congress on Numerical Methods in Combinatorial Optimization*, Capri, 1986.

[37] P. Hansen and N. Mladenović. An introduction to variable neighbourhood search. In S. Voss, S. Martello, I.H. Osman, and C. Roucairol, editors, *Metaheuristics: Advances and Trends in Local Search Procedures for Optimization*, pages 433–458. Kluwer, 1999.

[38] P. Hansen and N. Mladenović. Developments of variable neighborhood search. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 415–439. Kluwer, 2002.

[39] P. Hansen and N. Mladenović. Variable neighborhood search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 145–184. Kluwer, 2003.

[40] J. Hao, R. Dorne, and P. Galinier. Tabu search for frequency assignment in mobile radio networks. *Journal of Heuristics*, 4:47–62, 1998.

[41] L. He and N. Mort. Hybrid genetic algorithms for telecommunications network back-up routeing. *BT Technology Journal*, 18:42–50, 2000.

[42] D. Henderson, S.H. Jacobson, and A.W. Johnson. The theory and practice of simulated annealing. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 287–319. Kluwer, 2003.

[43] J.H. Holland. *Adaptation in natural and artificial systems.* University of Michigan Press, 1975.

[44] J.H. Holland. Genetic algorithms. *Scientific American*, 267:44–50, 1992.

[45] S-H. Kim, K-N Chang, and S. Kim. A channel allocation for cellular mobile radio systems using simulated annealing. *Telecommunication Systems*, 14:95–106, 2000.

[46] S. Kirkpatrick, C.D. Gelatt Jr., and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[47] J.G. Klincewicz. Avoiding local optima in the p-hub location problem using tabu search and GRASP. *Annals of Operations Research*, 40:283–302, 1992.

[48] J.G. Klincewicz. Enumeration and search procedures for a hub location problem with economies of scale. *Annals of Operations Research*, 110:107–122, 2002.

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

[50] B. Li, F. Chen, and L. Yin. Server replication and its placement for reliable multicast. In *Proceedings of the IEEE ICCCN-00*, pages 396–401, Las Vegas, 2000.

[51] P. Manohar, D. Manunath, and R.K. Shevgaonkar. Routing and wavelength assignment in optical networks from edge disjoint path algorithms. *IEEE Communication Letters*, 5:211–213, 2002.

[52] C. Maple, L. Guo, and J. Zhang. Parallel genetic algorithms for third generation mobile network planning. In *Proceedings of the 2004 International Conference on Parallel Computing in Electrical Engineering (PARELEC04)*, pages 229–236, Dresden, 2004. IEEE Computer Society.

[53] 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.

[54] N. Mladenović and P. Hansen. Variable neighbourhood search. *Computers and Operations Research*, 24:1097–1100, 1997.

[55] T.F. Noronha and C.C. Ribeiro. Routing and wavelength assignment by partition coloring. *European Journal of Operational Research*, to appear.

[56] F.S. Pamuk and C. Sepil. A solution to the hub center problem via a single-relocation algorithm with tabu search. *IIE Transactions*, 33:399–411, 2001.

[57] K.F. Poon, A. Conway, G. Wardrop, and J. Mellis. Successful application of genetic algorithms to network design and planning. *BT Technology Journal*, 18:32–41, 2000.

[58] 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.

[59] M. Randall, G. McMahon, and S. Sugden. A simulated annealing approach to communication network design. *Journal of Combinatorial Optimization*, 6:55–65, 2002.

[60] C. Reeves. Genetic algorithms. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 65–82. Kluwer, 2003.

[61] C.R. Reeves. Genetic algorithms. In C.R. Reeves, editor, *Modern heuristic techniques for combinatorial problems*, pages 151–196. Wiley, 1993.

[62] C.R. Reeves and T. Yamada. Genetic algorithms, path relinking and the flowshop sequencing problem. *Evolutionary Computation*, 6:45–60, 1998.

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

[64] 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.

[65] 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*, pages 29–63. Springer, 2005.

[66] M.G.C. Resende and R.F. Werneck. A hybrid heuristic for the p-median problem. *Journal of Heuristics*, 10:59–88, 2004.

[67] C.C. Ribeiro. GRASP: Une métaheuristique gloutone et probabiliste. In J. Teghem and M. Pirlot, editors, *Optimisation approchée en recherche opérationnelle*, pages 153–176. Hermès, 2002.

[68] C.C. Ribeiro and I. Rosseti. A parallel GRASP heuristic for the 2-path network design problem. *Lecture Notes in Computer Science*, 2400:922–926, 2002.

[69] 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.

[70] C.C. Ribeiro and D.S. Vianna. A genetic algorithm for the phylogeny problem using an optimized crossover strategy based on path-relinking. *Revista Tecnologia da Informação*, 3:67–70, 2003.

[71] 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.

[72] A. Srinivasan, K.G. Ramakrishnan, K. Kumaran, M. Aravamudam, and S. Naqvi. Optimal design of signaling networks for Internet telephony. In *IEEE INFOCOM 2000*, pages 932–941, Tel-Aviv, 2000.

[73] G.N. Varela and M.C. Sinclair. Ant colony optimisation for virtual-wavelength-path routing and wavelength allocation. In P.J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala, editors, *Proceedings of the Congress on Evolutionary Computation*, volume 3, pages 1809–1816, Washington D.C., 1999. IEEE Press.

[74] S. Watanabe, T. Hiroyasu, and M. Miki. Parallel evolutionary multi-criterion optimization for mobile telecommunication networks optimization. In *Proceedings of the EUROGEN 2001 Conference*, pages 167–172, Athens, 2001.

[75] O. Wittner, P.E. Heegaard, and B. Helvik. Scalable distributed discovery of resource paths in telecommunication networks using cooperative ant-like agents. In *Proceedings of the 2003 Congress on Evolutionary Computation*, Canberra, 2003.

[76] J. Xu, S.Y. Chiu, and F. Glover. Tabu search for dynamic routing communications network design. *Telecommunications Systems*, 8:55–77, 1997.

[77] A.Y. Zomaya, M. Haydock, and S. Olariu. Some observations on using meta-heuristics for efficient location management in mobile computing networks. *Journal of Parallel and Distributed Computing*, 63:33–44, 2003.