1 Applications of Parallel Metaheuristics to Optimization Problems in Telecommunications and Bioinformatics

S. L. MARTINS, C. C. RIBEIRO and I. ROSSETI

Universidade Federal Fluminense
Departamento de Ciência da Computação
Niterói, RJ 22410-240, Brazil
E-mails: {simone,celso,rosseti}@ic.uff.br

1.1 INTRODUCTION

Recent years have witnessed huge advances in areas such as computer technology, communication networks, biology, and genetics. Many hard optimization problems in network design and routing are related with new applications and technologies such as cellular mobile radio systems and optical fibers. The research in genomics pushed the need for solving optimization and graph problems in biology.

Combinatorial optimization problems in telecommunications and other areas 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 a metaheuristic to a given problem yields a heuristic to the latter.

Metaheuristics offer a wide range of possibilities for effective parallel algorithms running in much smaller computation times, but requiring efficient implementations. Cung et al. [41] showed that parallel implementations of metaheuristics appear quite naturally as an effective approach to speedup the search for good solutions to optimization problems. They allow solving larger problems and finding better solutions with respect to their sequential counterparts. They also lead to more robust algorithms and this is often reported as the main advantage obtained with parallel implementations of metaheuristics: they are less-dependent on parameter tuning and their success is not limited to few or small classes of problems.

Parallel implementations of metaheuristics and their applications to problems in telecommunications, network design and routing, and bioinformatics are studied in this work. In the next section, we summarize the main issues on parallelization strategies of metaheuristics. Section 1.3 reviews the main principles and ideas involved in parallelizing algorithms derived from four widely and successfully used metaheuristics: GRASP, simulated annealing, genetic algorithms and tabu search. Case studies regarding applications of parallel metaheuristics to some problems in telecommunications and bioinformatics are presented in detail in Section 1.4. Applications of metaheuristics to problems in telecommunications and bioinformatics are listed and reviewed in the last section.

1.2 PARALLELIZATION STRATEGIES

Even though parallelism is not yet systematically used to speed up or to improve the effectiveness of metaheuristics, parallel implementations abound in the literature. Parallelization strategies considerably differ from one technique to another. An architecture-independent classification of parallelization strategies considering the underlying parallel local search strategy, inspired from that proposed in [92] was proposed by Cung et al. [41].

Metaheuristics based on local search may be seen as the exploration of the *neighborhood graph* (or state space graph) associated with a problem instance, in which nodes correspond to solutions and edges connect neighbor solutions. Each iteration consists basically in the evaluation of the solutions in the neighborhood of the current solution, followed by a move toward one of them, avoiding as much as possible to prematurely stop in a local optimum and until no further improvement of the best known solution can be achieved. The solutions visited along the search define a *walk* (or a *trajectory*) in the neighborhood graph. Parallel implementations of metaheuristics use several processors to concurrently generate or explore the neighborhood graph. Since this graph is not known beforehand, parallelizations of metaheuristics are *irregular applications*, whose efficiency strongly depends on the appropriate choice of the granularity of the algorithm and on the use of load balancing techniques.

We distinguish between two approaches for the parallelization of the local search, according with the number of trajectories investigated in the neighborhood graph:

(1) Single walk: fine- to medium-grained tasks

(2) Multiple walks: medium- to coarse-grained tasks

   a. independent search threads

   b. cooperative search threads

In the case of a *single-walk parallelization*, one unique trajectory is traversed in the neighborhood graph. The search for the best neighbor at each iteration is performed in parallel, either by the parallelization of the cost function evaluation or by domain decomposition (the neighborhood evaluation or the problem instance itself are decomposed and distributed over different processors). The goal of this strategy is basically to speed up the sequential traversal of the neighborhood graph. The task whose execution is distributed among the processors may be the evaluation of the cost function at each neighbor of the current solution or the construction of the neighborhood itself. In the first case, speedups may be obtained without any modification in the trajectory followed by the sequential implementation. In the second case, the neighborhood decomposition and its distribution over several processors often allow more deeply examining a larger portion of the neighborhood than that examined by a sequential implementation, which often uses neighborhood reduction techniques. Thus, the trajectory followed in the neighborhood graph may be better guided in parallel than in sequential mode, possibly leading to improved solutions. Single-walk parallelizations have small or medium granularity and need frequent synchronizations.

A *multiple-walk parallelization* is characterized by the investigation in parallel of multiple trajectories, each of which by a different processor. We call by a *search thread* the process running in each processor traversing a walk in the neighborhood graph. These threads can be either *independent* (the search threads do not exchange any information they collect) or *cooperative* (the information collected along each trajectory is disseminated and used by the other threads).

When using independent search threads, we distinguish between two basic approaches:

- Exploration in parallel of multiple trajectories originating from different nodes of the neighborhood graph: each walk starts from a different solution. The search threads may use the same local search algorithm or different ones, with the same parameter values or not. The trajectories may intersect at one or more nodes of the neighborhood graph. If p processors are used, this strategy corresponds to the successive execution of p sequential independent searches.

- Exploration in parallel of subgraphs of the neighborhood graph, obtained by problem decomposition (e.g. by variable fixation): several subgraphs of the neighborhood graph are explored in parallel, without intersection of the corresponding trajectories. We have a total and permanent decomposition of the neighborhood graph formed by complete solutions.

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.

Toulouse et al. [91] attempted to show how this cooperation behaves by modeling cooperative procedures as discrete-time dynamical systems. They performed simulations describing different cooperative procedures with the same sequential programs, initial conditions, and parameter values, which exhibit completely different search patterns under the influence of systemic cooperations.

1.3 PARALLELIZATION OF METAHEURISTICS

In this section we review the main principles and ideas involved in parallelizing algorithms derived from four widely and successfully used metaheuristics: GRASP, simulated annealing, genetic algorithms and tabu search. The reader is referred e.g. to [41] for a more comprehensive overview of parallel algorithms based on metaheuristics.

1.3.1 GRASP

A greedy randomized adaptive search procedure (GRASP) [48, 81] is a multi-start algorithm, in which each iteration consists of two phases. In a construction phase, a feasible solution is iteratively constructed, one element at a time, by a greedy randomized procedure. Next, in a local search phase, a local optimum in the neighborhood of the constructed solution is sought. The best solution over all iterations is kept as the result.

Most parallel implementations of GRASP are independent-thread multiple-walk strategies [8, 65, 66, 70, 72, 73, 78], based on the distribution of the iterations over the processors. In general, each search thread has to perform `Max_Iterations`/p iterations, where p and `Max_Iterations` are, respectively, the number of processors and the total number of iterations. Each processor has a copy of the sequential algorithm and a copy of the problem data. One of the processors acts as the master, reading and distributing problem data, generating the seeds which will be used by the pseudorandom number generators at each processor, distributing the iterations, and collecting the best solution found by each processor.

Path-relinking is often used in the implementation of cooperative strategies [82]. Computational results illustrate the effectiveness of path-relinking procedures used in conjunction with GRASP to improve the quality of the solutions found by the latter.

1.3.2 Simulated annealing

Simulated annealing in the context of optimization methods was introduced by Kirkpatrick et al. [60]. As the first metaheuristic to be parallelized, several papers describing parallelization efforts appeared subsequently [28, 13, 29, 1, 44, 59, 14].

Most of the first parallelizations [62, 84] followed the single-walk strategy and were based on the ideas of *move acceleration* and *parallel moves*. In the first case, each move is evaluated by breaking it up into several subtasks executed in parallel: selection of a feasible move, evaluation of the associated cost change, acceptance or rejection of the move, and global information update.

In the case of parallel moves [7, 13, 95, 62, 14, 83, 84], each processor generates, evaluates, and accepts a different move. Synchronization can be done after only one or after several steps. Other coarse-grained parallelizations [1, 64] of simulated annealing are based on the use of multiple Markov chains, which is equivalent to the multiple-walk strategy. The readers are also referred to [11, 35, 56] for other surveys about parallel simulated annealing and its hybrids.

1.3.3 Genetic Algorithms

A genetic algorithm is a basic search procedure, proposed and developed by Holland [57, 58]. It is based on the process of natural evolution, following the principles of natural selection and survival of the fittest [68, 79]. First, a population of individuals (solutions) is created. Each individual is evaluated according to a fitness function. Individuals with higher evaluations (i.e., more fitted) are selected to generate a new generation of this population. Crossover is a genetic operation which combines parts from pairs of individuals to generate new ones. Mutation is a unary genetic transformation that creates a new individual by small changes applied to an existing one. New individuals created by crossover or mutation replace all or part of the initial population. The process of evaluating fitness and creating a new population generation is repeated until a termination criterion is achieved.

Many different approaches to the parallelization of genetic algorithms can be found in literature and they differ in many aspects. Basically, there are three types of parallelization strategies [22, 24, 26, 34]: global (single-walk parallelization), diffusion model, and island model. Fine-grained (diffusion model) and coarse-grained (island model) parallelizations characterize cooperative multiple-walk strategies.

Global (single-walk) parallelization strategies [2, 69, 33] manipulate a single population, but the individuals are distributed among several processors for fitness evaluation. The trajectory followed by the parallel implementation is the same followed by the sequential algorithm. Almost-linear speedups may be obtained whenever the evaluation of the objective function is significantly more computationally expensive than the other steps (i.e., the genetic operators) of the algorithm.

Fine-grained parallelizations [34, 61, 90] using the *diffusion model* are developed to take advantage of massively parallel computers. The population is divided into many small subpopulations, usually formed by a single individual. Each subpopulation is mapped onto the processing elements. Selection and mating are only possible within a small, restricted neighborhood defined by the topology of the massively parallel computer. The use of local selection and reproduction rules leads to a continuous diffusion of individuals over the population [34].

Coarse-grained or *island model* algorithms [10, 16, 19, 22, 23, 25, 47] are currently the most used strategies. They are based on dividing the population into few subpopulations. Each subpopulation is assigned to a different processor, which performs a sequential genetic algorithm starting from this subpopulation. Mating involves only individuals within the same subpopulation. Occasionally, the processors exchange individuals through a migration operator. Some important technical issues influence the implementation of this strategy, such as the size of the subpopulations, the topology of the interprocessor connection network that defines the exchange of individuals, and the migration rate (i.e, the number of individuals to be exchanged during a migration phase and the frequency in which migration takes place).

1.3.4 Tabu search

Tabu search is an adaptive procedure for solving combinatorial optimization problems, which guides an improvement heuristic to continue exploration without being confounded by an absence of improving moves (see e.g. [52, 53, 54]). Tabu search goes beyond local search by employing a memory-based strategy for modifying the neighborhood of the current solution as the search progresses. Depending on the search stage, this modified neighborhood may be an extension or a restriction of the regular one. As a consequence, and contrary to the basic local search scheme, moves toward solutions which deteriorate the cost function are permitted under certain conditions.

Single-walk parallelization has been the first strategy applied to tabu search methods [35], aiming to accelerate the search in the neighborhood. Most of these implementations use a master-worker paradigm. The master process executes a sequential tabu procedure and ,at each iteration, asks the workers to evaluate the possible moves in the neighborhood of the current solution.

Usually, workers may perform this evaluation by two variants: slaves evaluate candidate moves only or they also perform a few local search iterations. The master receives and process the information obtained from workers and then selects and implements the next move.

Some implementations of tabu search based on neighborhood decomposition are reported in [50, 31, 32, 49, 74, 76, 77, 75, 88, 39] and show very good results, specially when neighborhoods are very large and the time to evaluate and perform a move is relatively small. Almost all the time near linear speed-ups are achieved. This type of parallelization depends heavily upon the problem characteristics. The results are not so good, when the time required for one sequential iteration is relatively large compared to the total solution time [37].

Other implementations [12, 51] combine single- and multiple-walk strategies. First, a master processor distributes the problem data to slaves running initialization processes. Each of these slave processors builds a different initial solution and returns it to the master. The master combines the solutions it received, to generate a unique initial solution which is decomposed into several subproblems that are sent to the slaves. Each subproblem is solved by variants of the same sequential tabu search algorithm by the slaves and the solutions generated by them are sent to the master. These solutions are combined by the master to obtain an improved solution. A new iteration of this procedure resumes, until some stopping criterion is attained.

Multiple-walk implementations using independent search threads can be found in [15, 80, 87, 89]. Each thread starts from possibly different initial solutions and use diverse tabu search strategies. The experimental results show that this kind of implementation leads to very efficient and robust heuristics.

Cooperative multiple-walk implementations [30, 36, 3, 38] are among the most promising tabu search parallelization strategies, due to their effectiveness and robustness. In most implementations, the processors start from different initial solutions and their searches follow paths generated by different sequential tabu search algorithms. Communication is performed through a central pool of elite solutions.

1.4 CASE STUDIES IN NETWORK DESIGN AND BIOINFORMATICS PROBLEMS

In this section, we describe in detail some parallel implementations of the metaheuristics described in the previous section to optimization problems in network design and bioinformatics.

1.4.1 Parallel GRASP for 2-path network design

1.4.1.1 Problem formulation Let $G = (V, E)$ be a connected graph, where V is the set of nodes and E is the set of edges. A k-path between nodes $s, t \in V$ is a sequence of at most k edges connecting them. Given a non-negative weight function $w : E \rightarrow R_+$ 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 [42].

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

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

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

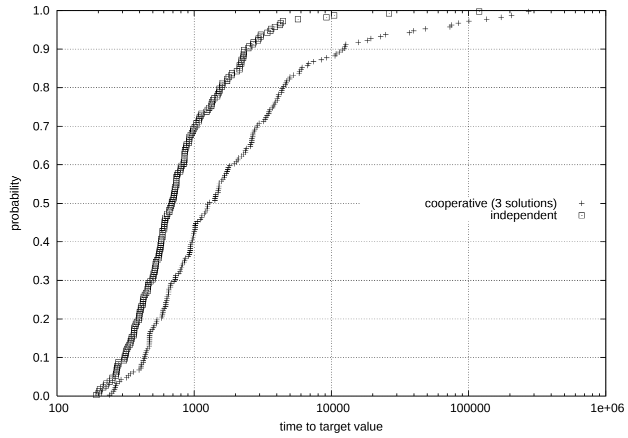
1.4.1.4 Path-relinking A solution to 2PNDP is represented as a set of 2-paths connecting each origin-destination pair. Path-relinking starts by determining all origin-destination pairs whose associated 2-paths are different in the starting and guiding solutions. These computations amount to determining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

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

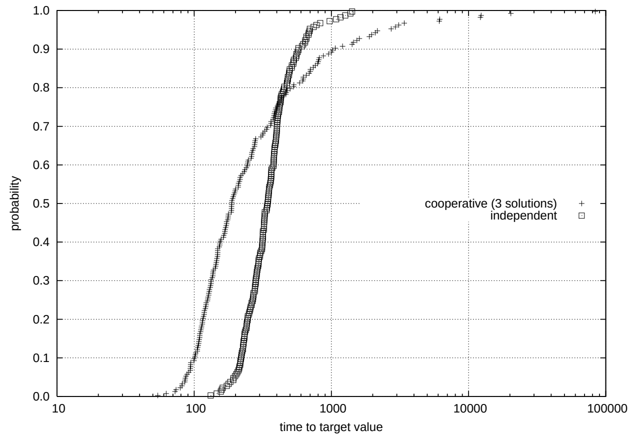
However, in the case of the cooperative-thread parallel implementation of GRASP with path-relinking for 2PNDP, the master handles a centralized pool of elite solutions, collecting and distributing them upon request. The $p-1$ slaves exchange the elite solutions found along their search trajectories. 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 [82] between the same pair of starting and guiding solutions, respectively.

1.4.1.5 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 [4] 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.

Results obtained for both the independent-thread and the cooperative-thread parallel implementations of GRASP with path-relinking on the above instance with the target value set at 683 are reported in Figure 1.1. The cooperative implementation is already faster than the independent one for eight processors. For fewer processors the independent implementation is naturally faster, since it employs all p processors in the search (while only $p-1$ slave processors take part effectively in the computations performed by the cooperative implementation).



(a)



(b)

Fig. 1.1 Running times for 200 runs of the multiple-walk independent-thread and the multiple-walk cooperative-thread implementations of GRASP with path-relinking using (a) two processors and (b) eight processors, with the target solution value set at 683.

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

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

(2) Only one solution is sent to the pool at each GRASP iteration.

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

Comparative results for these three strategies on the same problem instance are plotted in Figure 1.2. The first strategy outperformed all others.

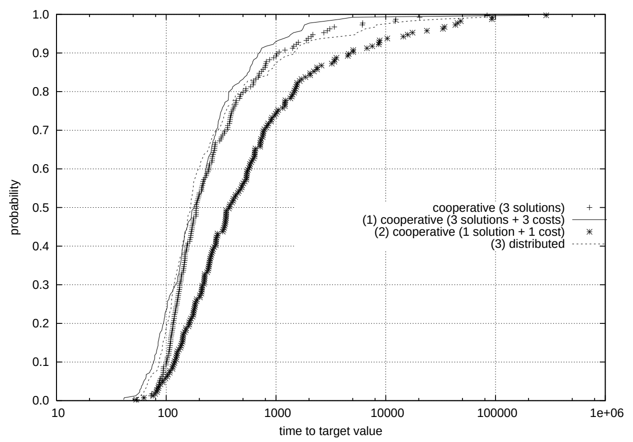


Fig. 1.2 Strategies for improving the performance of the centralized multiple-walk cooperative-thread implementation on eight processors.

Table 1.1 shows the average computation times and the best solutions found over ten runs of each strategy when the total number of GRASP iterations is set at 3200. There is a clear degradation in solution quality for the independent-thread strategy when the number of processors increases. As fewer iterations are performed by each processor, the pool of elite solutions gets poorer with the increase in the number of processors. Since the processors do not communicate, the overall solution quality is worse. In the case of the cooperative strategy, the information shared by the processors guarantees the high quality of the solutions in the pool. The cooperative implementation is more robust. Very good solutions are obtained with no degradation in quality and significant speedups.

Table 1.1 **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

1.4.2 Parallel tabu search for capacitated network design

1.4.2.1 Problem formulation Let $G = (V, E)$ be a connected graph, where V is the set of nodes and E is the set of arcs, and $\mathcal{P}$ be set of commodities to be distributed. Three measures characterize each arc $e \in E$: c_e^p is the unit cost of moving commodity $p \in \mathcal{P}$ through the arc e; f_e is the fixed cost of including the arc e in the final design of the network; and u_e is the total capacity of the arc e.

Each commodity $p \in \mathcal{P}$ has an associated demand w_p moving from its origin to its destination by using one or more paths in $\mathcal{L}^p$, where the latter denotes the set of all possible routes for the commodity p. The capacitated multicommodity network design problem (CMNDP) may be formulated as follows:

$$\min z(h,y) = \sum_{e\in E} f_e \cdot y_e + \sum_{p\in\mathcal{P}} \sum_{l\in\mathcal{L}^p} k_l^p \cdot h_l^p$$

$$\text{subject to:} \sum_{l\in\mathcal{L}^p} h_l^p = w_p, \quad \forall p \in \mathcal{P}$$

$$\sum_{p\in\mathcal{P}} \sum_{l\in\mathcal{L}^p} h_l^p \cdot \delta_{el}^p \le u_e \cdot y_e, \quad \forall e \in E$$

$$h_l^p \ge 0, \quad \forall p \in \mathcal{P}, \quad \forall l \in \mathcal{L}^p$$

$$y_e \in \{0,1\}, \quad \forall e \in E,$$

where $\delta_{el}^p = 1$ if the arc e belongs to l^{th} path for commodity p, $\delta_{el}^p = 0$ otherwise. The unit transportation cost of path $l \in \mathcal{L}^p$ is $k_l^p = \sum_{e\in E} c_e^p \cdot \delta_{el}^p$. $y_e = 1$ if the arc e is included in the final design of the network, and $y_e = 0$ otherwise. h_l^p represents the flow of commodity $p \in \mathcal{P}$ on path $l \in \mathcal{L}^p$, and H is the set of all selected h_l^p of a solution.

Applications of CMNDP can be found in transportations, logistics, telecommunications, power system design, and production planning.

1.4.2.2 Sequential tabu search The sequential implementation is based on the exploration of the space of the extreme points of the polyhedron of the linear relaxation. To determine the neighbors solutions of any given solution, Crainic and Gendreau [36] used the revised pivoting rules of the simplex method [43].

An initial solution can be generated performing simplex pivots without the use of tabu mechanisms until a first feasible solution is obtained. The neighborhood of the current solution is defined as the subset of all extreme points which can be reached from this solution by one simplex pivot. The best element (e.g., the move with the largest decrease in the objective function) of this subset is selected. Non-improving moves are accepted. Moves conducting to non-feasible solutions are also accepted if they represent the best local option. The corresponding overflows are assigned to artificial arcs with arbitrarily high costs.

Since not all variables are available to the local search procedure at each iteration, column generation is used to expand the neighborhood and to enrich the set of candidates whenever a local optimum solution appears to have been reached. This condition is detected when the best solution identified during the current series of pivot moves, Z_{local}, did not improve for max_move consecutive moves. A modified shortest path algorithm is used to generate new paths. In this method, all saturated arcs are eliminated. To capture the interplay between the fixed costs and the capacities of the arcs, a relaxation of the formulation is used. A fixed number, k_gen, of paths is generated for all commodities each time the column generation procedure is executed.

Local search terminates when no improvement to the current local best solution is obtained after max_col_gen consecutive column generation cycles. The method stops or proceeds to diversify the search. A small number of arcs is selected for diversification. Paths containing these arcs are not allowed to enter the basis during their tabu tenure. These arcs are not available to be used during the column generation phase. The tabu tenure is defined in terms of a number of column generation cycles.

1.4.2.3 Parallel implementations In the independent-thread parallel implementation of tabu search for CMNDP, each processor has a copy of the sequential tabu search algorithm and a copy of the data. One processor acts as the master, reading and distributing the problem data, and collecting the best solution found by each processor. All the p available processors perform the tabu search method.

In the case of the cooperative-thread parallel implementation of tabu search for CMNDP, the master handles a centralized pool of solutions, collecting and distributing them upon request. The $p-1$ slaves exchange the solutions found along their search trajectories. Each slave executes the sequential tabu search procedure and sends its solution to the pool whenever it improves the local best solution (Z_{local}). Solutions are sent during regular local search phases, as well as immediately after a diversification move.

A solution is requested from the pool by a slave before a diversification operation. If the imported solution is better than the current best, then the latter is replaced by the former, and the diversification proceeds from the new solution. Otherwise, the imported solution is simply discarded, and the diversification proceeds as usual.

1.4.2.4 Computational results The implementation of the sequential tabu search heuristic was done in FORTRAN 77. The cooperation mechanisms are programmed in C. The parallel experiments have been performed on a cluster of SUN Ultra SPARC 1/140 workstations with a 143 MHz clock, 512 Kbytes of cache memory, and 64 Mb of RAM memory (three out of the 16 processors had 128 Mbytes of RAM memory). No other applications were running during the parallel experimentation.

A set of ten instances was used in the experiments. These problems are identified with the letter C and a quintuple containing the number of nodes, the number of arcs, the number of commodities, an indicator showing whether the fixed costs are relatively high (F) or low (V), and another indicator showing if the problem is tightly (T) or somewhat loosely (L) capacitated.

The objective of the runs was to analyze the influence of the information exchanged among the processors in solution quality. Tables 1.2, 1.3, and 1.4 display results obtained with four, eight, and 16 processors, respectively. For each instance, these tables indicate results for the independent parallel search (column IND), and five selection strategies to extract a solution from the pool in cooperative strategies:

- $SBest$: extracts the minimum cost solution from the pool of solutions;
- $SProb$: randomly selects a solution among the best in the pool;
- $SMobil$: randomly selects a solution of the pool according to weighted probabilities;
- $SHigh$: similar to $SMobil$, except that it randomly selects a solution among those with high weighted probability values; and
- $SLow$: similar to $SMobil$, except that it randomly selects a solution among those with low weighted probability values.

The performance of each strategy is measured for each problem as the percentage improvement in solution quality, with respect to the corresponding result obtained by the sequential procedure. A negative value indicates that a worse solution was found.

According with Tables 1.2 to 1.4, the parallel strategies found better solutions. Only six out of the 180 parallel runs found a solution worse than that obtained by the sequential algorithm, while in 11 runs the same solution was obtained. The results also indicate that cooperative search performs better than independent search. This observation is true for two selection strategies when four processors were used. When eight or 16 processors were used, all cooperative strategies significantly outperformed the independent search approach.

Table 1.2 Percentage improvement in solution quality for parallel executions on four processors.

C	IND	$SBest$	$SProb$	$SMobil$	$SHigh$	$SLow$
(25,100,10,F,T)	0.46	0.46	1.87	1.87	0.61	0.46
(25,100,30,F,T)	0.00	-0.04	0.14	0.14	0.19	-0.17
(100,400,10,F,T)	1.50	0.52	1.77	1.77	1.50	1.50
(20,230,40,F,T)	0.13	0.02	0.00	0.00	0.00	0.02
(20,300,40,F,T)	0.16	0.00	0.16	0.16	0.16	0.00
(20,300,200,F,T)	5.08	8.01	7.74	7.38	5.50	5.31
(30,520,100,F,T)	4.03	2.41	3.13	3.13	3.13	3.13
(30,520,400,F,T)	1.37	1.44	1.08	2.12	1.03	1.36
(30,700,100,F,T)	1.17	1.65	1.31	1.17	1.17	1.31
(30,700,400,F,T)	2.77	2.00	1.70	1.70	0.55	0.54
avg. improvement	1.67	1.64	1.89	1.95	1.38	1.38

1.5 APPLICATIONS IN TELECOMMUNICATIONS AND BIOINFORMATICS

We present some applications of parallel implementations of metaheuristics to optimization problems in telecommunications and bioinformatics. As many problems in these areas are NP-hard and exact methods can only solve small problems, approximate algorithms based on metaheuristics play a very important role in their solution in practice.

Table 1.3 Percentage improvement in solution quality for parallel executions on eight processors.

C	IND	$SBest$	$SProb$	$SMobil$	$SHigh$	$SLow$
(25,100,10,F,T)	0.46	0.46	1.19	0.81	0.61	0.46
(25,100,30,F,T)	0.00	-0.04	0.19	0.19	0.19	-0.17
(100,400,10,F,T)	1.50	0.52	-0.84	-0.46	1.52	3.16
(20,230,40,F,T)	0.13	0.02	0.02	0.02	0.00	0.13
(20,300,40,F,T)	0.16	0.00	0.16	0.16	0.16	0.16
(20,300,200,F,T)	5.08	9.26	8.84	5.93	7.98	8.94
(30,520,100,F,T)	4.03	3.27	4.03	3.51	4.03	3.64
(30,520,400,F,T)	1.37	3.68	5.16	4.20	3.60	3.25
(30,700,100,F,T)	1.17	1.72	1.71	1.71	1.53	1.31
(30,700,400,F,T)	2.77	3.70	4.06	2.77	2.77	3.53
avg. improvement	1.67	2.26	2.45	1.88	2.24	2.24

Table 1.4 Percentage improvement in solution quality for parallel executions on 16 processors.

C	IND	$SBest$	$SProb$	$SMobil$	$SHigh$	$SLow$
(25,100,10,F,T)	0.56	0.61	0.79	0.46	0.50	0.79
(25,100,30,F,T)	0.00	0.09	0.11	0.14	0.23	0.23
(100,400,10,F,T)	1.50	1.89	1.50	1.50	1.77	1.77
(20,230,40,F,T)	0.13	0.00	0.02	0.02	0.02	0.02
(20,300,40,F,T)	0.16	0.16	0.16	0.16	0.16	0.16
(20,300,200,F,T)	5.08	7.98	7.72	7.98	7.54	8.05
(30,520,100,F,T)	4.03	3.51	4.03	4.03	3.51	3.51
(30,520,400,F,T)	1.37	1.93	1.37	2.08	3.90	1.42
(30,700,100,F,T)	1.17	1.50	1.17	1.17	1.33	1.68
(30,700,400,F,T)	2.77	3.47	2.97	2.84	2.84	2.77
avg. improvement	1.67	2.12	1.98	2.04	2.18	2.18

The parallel algorithms presented here are developed aiming the achievement of lower execution times and/or finding better quality solutions. Parallelizations of metaheuristics designed to accelerate sequential procedures are usually simple to implement and quite often lead to linear speedups. The cooperative work, performed by parallel execution threads exchanging information during their execution, show that better results can be achieved in less computational times for different types of metaheuristics.

1.5.1 Applications in telecommunications and network design

The outbreak of new technologies in telecommunications networks, together with the demand for more computer intensive applications, leads to huge developments and needs in network design and routing.

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 such as 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.

Applications of metaheuristics to problems in telecommunications are plentiful in the literature [67, 46]. Some parallel applications are surveyed below, according to the main metaheuristic used by each of them.

As shown in Section 1.4.1, GRASP with path-relinking for the 2-path network design problem has been shown to be efficiently implemented in parallel with linear speedups. As noticed before, applications can be found in the design of communications networks, in which paths with few edges are sought to enforce high reliability and small delays.

Canuto et al. [27] developed a parallel GRASP heuristic for the prize-collecting Steiner tree problem. Given an undirected graph with prizes associated with its nodes and weights associated with its edges, this problem consists of finding a subtree of this graph minimizing the sum of the weights of its edges plus the prizes of the nodes not spanned. It has important applications in the design of local access networks. The authors proposed a multi-start local search algorithm. Path-relinking is used to improve the solutions found by local search. Their results show that the local search with perturbations approach found optimal solutions on nearly all of the instances tested, in much smaller computation times than an exact branch-and-cut algorithm that is able to handle only small problems. An independent parallelization obtained important speedups.

The parallelization of the reactive GRASP heuristic developed by Prais and Ribeiro [78] for the problem of traffic assignment in TDMA communication satellites led to linear speedups in [9].

Bastos and Ribeiro [15] 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 [38] 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 [36]. Improved solutions and smaller computation times are obtained for both applications, with respect to the sequential algorithms.

Buriol at al. [20] 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. Preliminary parallelization results have shown almost linear speedups.

Watanabe et al. [94] 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. [21] 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. [6] 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 [5] 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.

Cahon et al. [85] 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.

1.5.2 Applications in bioinformatics

Bioinformatics is a multidisciplinary area, where computational and mathematical techniques are used to solve problems in biology. The cost of biological experiments is quite high and many of them are also time consuming. Therefore, many computational methods for analyzing and interpreting results of genetic experiments have been developed. Usually, solving biological problems involves enormous amount of collected data, and the interpretation of the results of biochemical experiments requires complicated combinatorial models.

This is a quite new area, with relatively few applications of metaheuristics reported in the literature to date. Parallel evolutionary algorithms are most often used in these applications.

Bhandarkar et al. [17] developed a parallel genetic algorithm for physical mapping of chromosomes in the presence of errors. The main goal was to obtain a minimal set of clones and probes that covers the entire chromosome. They implemented a two-tier parallel strategy. The upper level comprised a parallel discrete optimization using a genetic algorithm, while the lower level was performed by a parallel conjugate gradient descent procedure for continuous optimization. They obtained some results showing that for some instances this implementation was able to achieve linear and super-linear speed-ups.

Jourdan et al. [93] presented a parallel genetic algorithm to treat the linkage disequilibrium, used to discover haplotypes, which are candidates to explain multi-factorial diseases such as diabetes or obesity. Since the evaluation of the fitness of the individuals is time consuming, they developed a synchronous parallel implementation of the evaluation phase. The implementation was based on a master/slave model. During the fitness evaluation phase, the master gives to each slave an individual to be evaluated. The slave computes the fitness of this individual and sends it back to the master.

A well-known model used for the protein folding problem is the 2D-HP model [63]. Santos et al. [86] performed a theoretical analysis of some parallel genetic algorithms to solve this problem. They analyzed some configurations for the parallel implementation, including master-slave, fine-grained, coarse grained, and their variants. They used the LogP model [40] of a distributed memory multiprocessor, in which processors communicate by point-to-point messages. They also implemented the algorithms analyzed theoretically, and showed that their performance prediction was correct.

A genetic algorithm-based method was proposed by Nguyen et al. [71] for solving the sum-of-pairs multiple protein sequence alignment. First, a chromosome representation and the genetic operators were proposed. Next, two kinds of local search heuristics were combined with this algorithm: one used to generate individuals in the initial population and the other used as the mutation operator for improving individuals during the search. Then, a parallel implementation of this method was presented. Experimental results of benchmarks from the BAliBASE benchmark showed that the proposed method is superior to other well-known methods in terms of solution quality and running time.

Gras et al [55] presented a new classification of metaheuristics considering the property of cooperation between entities exploring the search space. This classification scheme involves three sub-classes: centralized, individual, and concerted cooperation, depending on how the cooperation is accomplished. They gave new definitions and described new developments of these approaches, and tried to show their individual benefits. They show that a hierarchical division of the problem into independent tasks can lead to spread and simplify optimization steps. The results of these optimizations are then associated to produce a global solution benefiting from the structures appearing in the different levels considered. They applied these techniques to two central problems of proteomics: automatic protein identification and multiple sequence alignment based on motif inference. For the first problem, they developed a parallel cooperative ant algorithm [45], where the ants find some solutions and exchange information about these solutions among them. For the second problem, a cooperative multiagent strategy was developed which takes advantage of concerted cooperation to achieve a fully automated clustering of biological sequences. The global algorithm follows an asynchronous course where each agent performs a succession of actions independently of the progress of other agents. Once the agents are initialized, they run an evolutionary algorithm to compute a preliminary solution. The agents exchange information about their solutions, so that some individuals from one agent can be sent to another agent to improve the quality of population.

Brauer et al. [18] presented a parallel genetic algorithm for phylogenetic inference under the maximum-likelihood optimality criterion. Parallelization was accomplished by assigning each individual of the population to a separate processor, so that the number of processors used was equal to the size of the evolving population. The genetic algorithm incorporated branch-length and topological mutation, recombination, selection on the maximum-likelihood score, and migration and recombination among subpopulations. They tested this parallel genetic algorithm with empirically observed and simulated DNA sequence data. For both observed and simulated data, the improvement in search time was nearly linear with respect to the number of processors. They also explored various ways of optimizing and tuning the parameters of the genetic algorithm.

References

1. E.H.L. Aarts, F.M.J. de Bont, J.H.A. Habers, and P.J.M. van Laarhoven. Parallel implementations of the statistical cooling algorithm. *Integration*, 4:209–238, 1986.

2. D. Abramson and J. Abela. A parallel genetic algorithm for solving the school timetabling problem. In *Proceedings of the 15th Australian Computer Science Conference*, pages 1–11, 1992.

3. R.M. Aiex, S.L. Martins, C.C. Ribeiro, and N.R. Rodriguez. Cooperative multi-thread parallel tabu search with an application to circuit partitioning. In *Lecture Notes in Computer Science*, volume 1457, pages 310–331, 1998.

4. 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.

5. 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. ACM, 2004.

6. E. Alba, C. Cotta, F. Chicano, and A.J. Nebro. Parallel evolutionary algorithms in telecommunications: Two case studies. In *Proceedings of the CACIC02, 2002*, 2002.

7. J.R. Allwright and D.B. Carpenter. A distributed implementation of simulated annealing for the travelling salesman problem. *Parallel Computing*, 10:335–338, 1989.

8. A.C. Alvim. Parallelization strategies for the metaheuristic GRASP. Master's thesis, Department of Computer Science, Catholic University of Rio de Janeiro, Brazil, 1998. (In Portuguese).

9. 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. SBC. (In Portuguese).

10. D. Andre and J.R. Koza. Parallel genetic programming: A scalable implementation using the transputer network architecture. In P.J. Angeline and Jr. K.E. Kinnear, editors, *Advances in Genetic Programming 2*, pages 317–338. MIT Press, 1996.

11. R. Azencott. *Simulated annealing: Parallelization techniques.* Wiley, 1992.

12. P. Badeau, F. Guertin, M. Gendreau, J.-Y. Potvin, and E. Taillard. A parallel tabu search heuristic for the vehicle routing problem with time windows. *Transportation Research-C*, 5:109–122, 1997.

13. P. Banerjee and M. Jones. A parallel simulated annealing algorithm for standard cell placement on a hypercube computer. In *Proceedings of the 1986 International Conference on Computer-Aided Design*, pages 34–37, 1986.

14. P. Banerjee, M. Jones, and J. Sargent. Parallel simulated annealing algorithms for standard cell placement on hypercube multi-processors. *IEEE Transactions on Parallel and Distributed Systems*, 1:91–106, 1990.

15. 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.* Kluwer, 2001.

16. T.C. Belding. The distributed genetic algorithm revisited. In L. Eschelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 114–121. Morgan Kaufmann, 1995.

17. S. M. Bhandarkar, J. Huang, and J. Arnold. A parallel genetic algorithm for physical mapping of chromosomes. In *Proceedings of the 2nd IEEE Computer Society Bioinformatics Conference*, pages 567–572. IEEE Computer Society, 2003.

18. M.J. Brauer, M.T. Holder, L.A. Dries, D.J. Zwickl, P.O. Lewis, and D.M. Hillis. Genetic algorithms and parallel processing in maximum-likelihood phylogeny inference. *Molecular Biology and Evolution*, 19:1717–1726, 2002.

19. M. Bubak and K. Sowa. Object-oriented implementation of parallel genetic algorithms. In R. Buyya, editor, *High performance cluster computing: Programming and applications*, volume 2, pages 331–349. Prentice Hall, 1999.

20. 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*, to appear.

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

22. E. Cantú-Paz. A survey of parallel genetic algorithms. *Calculateurs Parallèles, Réseaux et Systèmes Répartis*, 10:141–171, 1998.

23. E. Cantú-Paz. Implementing fast and flexible parallel genetic algorithms. In L.D. Chambers, editor, *Practical handbook of genetic algorithms*, volume III, pages 65–84. CRC Press, 1999.

24. E. Cantú-Paz. *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer, 2000.

25. E. Cantú-Paz and D.E. Goldberg. Predicting speedups of idealized bounding cases of parallel genetic algorithms. In T. Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 113–121. Morgan Kaufmann, 1997.

26. E. Cantú-Paz and D.E. Goldberg. Parallel genetic algorithms: Theory and practice. *Computer Methods in Applied Mechanics and Engineering*, 186:221–238, 2000.

27. 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.

28. A. Casotto, F. Romeo, and A. Sangiovanni-Vincentelli. A parallel simulated annealing algorithm for the placement of macro-cells. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-6:727–751, 1987.

29. A. Casotto and A. Sangiovanni-Vincentelli. Placement of standard cells using simulated annealing on the connection machine. In *Proceedings of the 1987 International Conference on Computer-Aided Design*, pages 350–353, 1987.

30. C.B. Cavalcante, V.C. Cavalcante, C.C. Ribeiro, and C.C. de Souza. Parallel cooperative approaches for the labor constrained scheduling problem. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*. Kluwer, 2001.

31. J. Chakrapani and J. Skorin-Kapov. Connection machine implementation of a tabu search algorihm for the traveling salesman problem. *Journal of Computing and Information Technology*, 1:29–63, 1993.

32. J. Chakrapani and J. Skorin-Kapov. Massively parallel tabu search for the quadratic assignment problem. *Annals of Operations Research*, 41:327–341, 1993.

33. P. Chalermwat, El.-G. Talbi, and J. LeMoigne. 2-phase GA-based image registration on parallel clusters. *Future Generation Computer Systems*, 17:467–476, 2001.

34. A. Chipperfield and P. Fleming. Parallel genetic algorithms. In A.Y. Zomaya, editor, *Parallel and Distributed Cmputing Handbook*, pages 1118–1143. McGraw-Hill, 1996.

35. T. Crainic and M. Toulouse. Parallel strategies for metaheuristics. In G. Kochenberger F. Glover, editor, *State-of-the-Art Handbook in Metaheuristics*. Kluwer Academic Publishers, 2002.

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

37. T.G. Crainic and M. Toulouse. Parallel metaheuristics. In T.G. Crainic and G. Laporte, editors, *Fleet management and logistics*, pages 205–251. Kluwer, 1998.

38. 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.

39. T.G. Crainic, M. Toulouse, and M. Gendreau. Synchronous tabu search parallelization strategies for multicommodity location-allocation with balancing requirements. *OR Spektrum*, 17:113–123, 1995.

40. D.E. Culler, R.M. Karp, D. Patterson, A. Sahay, E.E. Santos, K.E. Schauser, R. Subramonian, and T. von Eicken. LogP: A practical model of parallel computation. *Communications of the ACM*, 39:78–85, 1996.

41. 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.

42. G. Dahl and B. Johannessen. The 2-path network problem. *Networks*, 43(3):190–199, 2004.

43. G.B. Dantzig and M.N. Thapa. *Linear Programming: Introduction*. Springer-Verlag, 1997.

44. F. Darema, S. Kirkpatrick, and V. Norton. Parallel algorithms for chip placement by simulated annealing. *IBM Journal of Research and Development*, 31:391–402, 1987.

45. M. Dorigo and T. Stützle. The ant colony optimization metaheuristic: Algorithms, applications and advances. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 251–285. Kluwer, 2003.

46. S.L. Martins e C.C. Ribeiro. Metaheuristics and applications to optimization problems in telecommunications. In P. Pardalos and M.G.C. Resende, editors, *Handbook of Optimization in Telecommunications*. Springer-Verlag, 2005. To be published.

47. I. De Falco, R. Del Balio, A. Della Cioppa, and E. Tarantino. A parallel genetic algorithm for transonic airfoil. In *Proceedings of the IEEE International Conference on Evolutionary Computing*, pages 429–434, 1995.

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

49. C.N. Fiechter. A parallel tabu search algorithm for large traveling salesman problems. *Discrete Applied Mathematics*, 51:243–267, 1994.

50. B.-L. Garcia, J.-Y. Potvin, and J.-M. Rousseau. A parallel implementation of the tabu search heuristic for vehicle routing problems with time windows constraints. *Computers and Operations Research*, 21:1025–1033, 1994.

51. M. Gendreau, F. Guertin, J-Y. Potvin, and E. Taillard. Parallel tabu search for real-time vehicle routing and dispatching. *Transportation Science*, 33:381–390, 1999.

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

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

54. F. Glover and M. Laguna. *Tabu Search.* Kluwer, 1997.

55. R. Gras, D. Hernandez, P. Hernandez, N. Zangger, Y. Mescam, J. Frey, O.Martin, J. Nicolas, and R. D. Appel. Cooperative metaheuristics for exploring proteomic data. *Artificial Intelligence Review*, 20:95–120, 2003.

56. D.R. Greening. Parallel simulated annealing techniques. *Physica D*, 42, 1990.

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

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

59. S. Kim, J.A. Chandy, S. Parkes, B. Ramkumar, and P. Banerjee. ProperPLACE: A portable, parallel algorithm for cell placement. In *Proceedings of the 8th International Parallel Processing Symposium*, pages 932–941, 1994.

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

61. U. Kohlmorgen, H. Schmeck, and K. Haase. Experiences with fine-grained parallel genetic algorithms. *Annals of Operations Research*, 90:203–220, 1999.

62. S.A. Kravitz and R.A. Rutenbar. Placement by simulated annealing on a multiprocessor. *IEEE Transactions on Computer Aided Design*, 6:534–549, 1987.

63. K. F. Lau and K. A. Dill. A lattice statistical mechanics model of the conformational and sequence spaces of proteins. *Macromolecules*, 22:3986–3997, 1989.

64. S.-Y. Lee and K.G. Lee. Synchronous and asynchronous parallel simulated annealing with multiple markov chains. *IEEE Transactions on Parallel and Distributed Systems*, 7:993–1008, 1996.

65. Y. Li, P.M. Pardalos, and M.G.C. Resende. A greedy randomized adaptive search procedure for the quadratic assignment problem. In P.M. Pardalos and H. Wolkowicz, editors, *Quadratic assignment and related problems*, volume 16 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 237–261. American Mathematical Society, 1994.

66. S.L. Martins, M.G.C. Resende, C.C. Ribeiro, and P.M. Pardalos. A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy. *Journal of Global Optimization*, 17:267–283, 2000.

67. 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.

68. Z. Michalewicz. *Genetic algorithms + Data structures = Evolution Programs*. Springer-Verlag, 1996.

69. H. Muhlenbein. Parallel genetic algorithms in combinatorial optimization. In O. Balci, R. Sharda, and S. Zenios, editors, *Computer Science and Operations Research: New Developments in their Interfaces*, pages 441–456. Pergamon Press, 1992.

70. R.A. Murphey, P.M. Pardalos, and L.S. Pitsoulis. A greedy randomized adaptive search procedure for the multitarget multisensor tracking problem. In P.M. Pardalos and D.-Z. Du, editors, *Network design: Connectivity and facilities location*, volume 40 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 277–301. American Mathematical Society, 1998.

71. H.D. Nguyen, K. Yamamori, I. Yoshihara, and M. Yasunaga. Aligning multiple protein sequences by parallel hybrid genetic algorithm. *Genome Informatics*, 13:123–132, 2002.

72. P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A parallel GRASP implementation for the quadratic assignment problem. In A. Ferreira and J. Rolim, editors, *Parallel Algorithms for Irregularly Structured Problems – Irregular'94*, pages 115–130. Kluwer, 1995.

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

74. S. C. Porto and C.C. Ribeiro. Parallel tabu search message-passing synchronous strategies for task scheduling under precedence constraints. *Journal of Heuristics 1*, pages 207–223, 1995.

75. S.C. Porto, J.P. Kitajima, and C.C. Ribeiro. Performance evaluation of a parallel tabu search task scheduling algorithm. *Parallel Computing*, 26:73–90, 2000.

76. S.C. Porto and C.C. Ribeiro. A tabu search approach to task scheduling on heterogeneous processors under precedence constraints. *International Journal of High Speed Computing*, 7:45–71, 1995.

77. S.C. Porto and C.C. Ribeiro. A case study on parallel synchronous implementations of tabu search based on neighborhood decomposition. *Investigación Operativa*, 5:233–259, 1996.

78. 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.

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

80. C. Rego and C. Roucairol. A parallel tabu search algorithm using ejection chains for the vrp. In I.H. Osman and J.P. Kelly, editors, *Metaheuristics: Theory and applications*, pages 253–295. Kluwer, 1996.

81. 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.

82. 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.

83. P. Roussel-Ragot and G. Dreyfus. A problem-independent parallel implementation of simulated annealing: Models and experiments. *IEEE Transactions on Computer-Aided Design*, 9:827–835, 1990.

84. R.A. Rutenbar and S.A. Kravitz. Layout by annealing in a parallel environment. In *Proceedings of the IEEE International Conference on Computer Design*, pages 434–437, 1986.

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

86. E.E. Santos, L.Lu, and E. Santos Jr. Efficiency of parallel genetic algorithms for protein folding on the 2-D HP mode. In *Proceedings of the Third International Workshop on Frontiers of Evolutionary Algorithms, 2000*, pages 1094–1097, 2000.

87. J. Schulze and T. Fahle. A parallel algorithm for the vehicle routing problem with time window constraints. *Annals of Operations Research*, 86:585–607, 1999.

88. E. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 7:443–455, 1991.

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

90. E.-G. Talbi and T. Muntean. Hill-climbing, simulated annealing and genetic algorithms: A comparative study. In H. El-Rewini and T. Lewis, editors, *Proceedings of the International Conference on Systems Sciences: Task scheduling in parallel and distributed systems*, pages 565–573. IEEE Computer Society Press, 1993.

91. M. Toulouse, T. G. Crainic, and B. Sansó. Systemic behavior of cooperative search algorithms. *Parallel Computing*, 30(1):57–79, 2004.

92. M.G.A. Verhoeven and E.H.L. Aarts. Parallel local search. *Journal of Heuristics*, 1:43–65, 1995.

93. L. Vermeulen-Jourdan, C. Dhaenens, and El-G. Talbi. A parallel adaptive GA for linkage disequilibrium in genomics. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS04)*. IEEE Computer Society, 2004.

94. S. Watanabe, T. Hiroyasu, and M. Miki. Parallel evolutionary multicriterion optimization for mobile telecommunication networks optimization. In *Proceedings of the EUROGEN 2001 Conference*, pages 167–172, Athens, 2001.

95. E.E. Witte, R.D. Chamberlain, and M.A. Franklin. Parallel simulated annealing using speculative computation. *IEEE Transactions on Parallel and Distributed Systems*, 2:483–494, 1991.