STRATEGIES FOR THE PARALLEL IMPLEMENTATION OF METAHEURISTICS*

VAN-DAT CUNG†, SIMONE L. MARTINS‡, CELSO C. RIBEIRO§, AND CATHERINE ROUCAIROL¶

Abstract. Parallel implementations of metaheuristics appear quite naturally as an effective alternative to speed up the search for approximate solutions of combinatorial optimization problems. They not only allow solving larger problems or finding improved solutions with respect to their sequential counterparts, but they also lead to more robust algorithms. We review some trends in parallel computing and report recent results about linear speedups that can be obtained with parallel implementations using multiple independent processors. Parallel implementations of tabu search, GRASP, genetic algorithms, simulated annealing, and ant colonies are reviewed and discussed to illustrate the main strategies used in the parallelization of different metaheuristics and their hybrids.

1. Introduction. Although metaheuristics provide quite effective strategies for finding approximate solutions to combinatorial optimization problems, the computation times associated with the exploration of the solution space may be very large. With the proliferation of parallel computers, powerful workstations, and fast communication networks, parallel implementations of metaheuristics appear quite naturally as an alternative to speedup the search for approximate solutions. Several strategies have been proposed and applied to different problems. Moreover, parallel implementations also allow solving larger problems or finding improved solutions, with respect to their sequential counterparts, due to the partitioning of the search space and to more possibilities for search intensification and diversification.

Therefore, parallelism is a possible way not only to reduce the running time of local search algorithms and metaheuristics, but also to improve their effectiveness and robustness. The latter are likely to be the most important contribution of parallelism to metaheuristics. Robust implementations can be obtained by the use of different combinations of strategies and parameter settings at each processor, leading to high quality solutions for different classes of instances of the same problem, without too much effort in parameter tuning. We present in the next section some current trends in parallel computing, concerning in particular architectural developments, parallel programming environments, and types of parallelism and parallel programming models. In Section 3, we draw some considerations about speedup and efficiency in parallel systems. We discuss some recent results showing that it is possible to achieve linear speedups in the time to find a solution within some target value, using parallel implementations of different metaheuristics based on multiple independent processors, such

*May 3, 2001

†Laboratoire PRiSM-CNRS, Université de Versailles, 45 avenue de Etats Unis, 78035 Versailles Cedex, France. E-mail: Van-Dat.Cung@prism.uvsq.fr

‡Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, RJ 22453-900, Brazil. Work of this author was sponsored by FAPERJ grant 151921/99. E-mail: simone@inf.puc-rio.br

§Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, RJ 22453-900, Brazil. Work of this author was sponsored by FAPERJ grant 150966/99 and CNPq grants 302281/85-1, 202005/89-5, and 910062/99-4. E-mail: celso@inf.puc-rio.br

¶Laboratoire PRiSM-CNRS, Université de Versailles, 45 avenue de Etats Unis, 78035 Versailles Cedex, France. E-mail: Catherine.Roucairol@prism.uvsq.fr

as simulated annealing, iterated local search, tabu search (under some conditions), and GRASP. Next, we present in Section 4 a classification of parallel implementations of metaheuristics, considering the underlying parallel local search strategy and using the notion of search threads. Different parallelization strategies are discussed and compared. Parallel implementations of tabu search, GRASP, genetic algorithms, simulated annealing, and ant colonies are reviewed in Section 5. Some concluding remarks are drawn in the last section.

2. Current trends in parallel computing. In this section, we review some trends and fundamental issues in parallel computing, which interfere in the design and in the efficiency of parallel algorithms.

2.1. Architectures. The architectures of parallel machines have considerably diversified in the nineties [18, 34, 64]. In *shared-memory* machines, all processors are able to address the whole memory space and communication between tasks is achieved through read and write operations on the shared memory. *Distributed-memory machines* have their memory physically distributed among the processors. Each processor can only address its own memory and communication among processes executing on different processors is performed by messages passed through the communication network. The MIMD family (*Multiple Instruction Stream, Multiple Data Stream* machines, according with Flynn's taxonomy [84]) evolved from shared-memory machines with a few processors (Sequent Balance, Encore Multimax) to distributed-memory machines with hundreds or even thousands of processors interconnected by networks with different topologies, such as the hierarchical ring for the KSR, the two-dimensional grid for the Intel Paragon, the three-dimensional tore for the CRAY T3D/E, and multi-stage switches for the IBM-SP systems, among many others. Synchronous SIMD (*Single Instruction Stream, Multiple Data Stream*) massively parallel machines with up to 65536 4- or 8-bit processors such as MasPar MP-1 and MP-2, and the popular Connection Machines CM-1 and CM-2 have also been developed. Their processors had a small memory capacity (e.g. 16 Kbytes on the MasPar MP-1) and were interconnected by specific networks such as the two-dimensional toric grid for the MasPar machines and the hypercube for CM-1 and CM-2.

Since development costs and commercial prices turned out to be extremely high, only large and well established companies such as IBM, SGI-Cray, and Intel still propose parallel machines with the same fundamental characteristics of the above. Several companies have even disappeared, as it was the case for Thinking Machines Corporation which developed and commercialized CM-1 and CM-2.

The end of the nineties has witnessed the resurgence of shared memory multiprocessor machines (Symmetric MP or, simply, SMP), ranging from two to a few hundred processors (SGI Origin). However, the most important innovation in recent years is the connection of SMP machines through fast local networks (Myrinet, SCI, ATM, GigaEthernet) and the connection of general purpose parallel machines through national or international very high speed networks. Besides allowing for scalability and fault-tolerance, networks of computers present good cost/performance ratios when compared to other parallel machines. This has led to an explosion in the use of clusters of computers, which account for one of the leading tendencies among the current trends in parallel processing [34, 158]. Basically, a cluster is a collection of PCs or workstations connected via a network and using off-the-shelf components. The sig-

nificant improvements in performance and reliability of workstations and high-speed networks strongly contributed to turn the use of clusters into an appealing, low-cost alternative for parallel applications. Clusters can be put together very easily, using operating systems such as public domain Linux, networks such as Ethernet, SCI, or Myrinet, and software environments such as MPI or PVM. Some cluster systems are currently among those with the best cost/performance ratios, ranking among the fastest and most powerful machines.

2.2. Parallel programming environments. The architecture of a parallel machine has a strong influence in the degree of parallelism or *granularity* of the applications it can run, i.e., the ratio between computation time and communication time. The granularity may also be seen as the amount of computation performed between two communication steps. If the architecture of the target machine allows for fast communication (high speed network or shared memory), then fine-grained applications are more suitable to be parallelized; otherwise medium- or coarse-grained applications should be envisioned for parallel implementation.

Parallelism leads to the need for new algorithms designed to exploit concurrency, because many times the best parallel strategy cannot be achieved by just adapting a sequential algorithm [85, 121]. In contrast to sequential programs, parallel algorithms are strongly dependent on the computer architecture for which they are designed. Programs implementing these algorithms are developed with parallel programming tools, which typically provide support for developing programs composed by several processes that exchange information during their execution. Many programming tools are available for the implementation of parallel programs, each of them being more suitable for some specific problem type or machine architecture. The choice of the programming tool to be used depends on the characteristics of the problem to be solved and should match the programming model underlying the parallel algorithm to be implemented. For instance, some tools might be more suitable for numerical algorithms based on regular domain decomposition, while others are more appropriate for applications that need dynamic spawning of tasks or that make use of irregular data structures.

There are basically three strategies for the development of parallel programs. The first one consists in using *parallel programming languages*, which are essentially sequential languages augmented by a set of special system calls. These calls provide low-level primitives for message passing, process synchronization, process creation, mutual exclusion, and other functions. Among them, we may cite HPF (High Performance Fortran) [105, 117, 118, 139, 140] and OpenMP [66], which explore inherent data parallelism (the same instruction set is applied to multiple items of a data structure) appearing for instance in do-loops. FORTRAN 90 provides constructs for specifying concurrent execution based on data parallelism. HPF extends FORTRAN 90 with additional parallel constructs and data placement directives. A data parallel program developed using both languages is a sequence of operations that are applied to some or all elements of an array. Besides inserting parallelisation directives, the programmer does not explicitely handle communication primitives. Parallelisation directives are activate along compilation. Such parallel programming languages are well adapted to fine-grained synchronous numerical applications.

In the second and currently most used strategy, tasks communicate by exchanging messages using *communication libraries* such as PVM (Parallel Virtual Machine) [104]

or MPI (Message-Passing Interface) [85, 102, 103, 145, 146, 147, 206]. To increase efficiency and avoid context swapping, each processor usually runs one single process (from now on, we do not make any difference between a process and the processor where it runs) in charge of computations and communication. This programming mode is particularly suited to coarse-grained applications running on clusters or networks of workstations. PVM is a pioneer, widely used message passing library, created to support the development of distributed and parallel programs executed on a set of interconnected heterogeneous machines. A PVM program consists of a set of tasks that communicate within a parallel virtual machine by exchanging messages. A configuration file created by the user defines the physical machines that comprise the virtual machine. The application programmer writes a parallel program by embedding these routines into a C, C++, or FORTRAN code. Several parallel implementations of metaheuristics have been developped in PVM, see e.g. [6, 9, 15, 20, 36, 58, 68, 81, 89, 151, 161, 162, 163, 164, 196, 197, 199, 200]. MPI is a proposal for the standardization of a message passing interface for distributed memory parallel machines, with the aim of enabling program portability among different parallel machines. It just defines a message passing programming interface, not a complete parallel programming environment. It does not provide any definition for fault tolerance support and assumes that the computing environment is reliable. Many implementations of the MPI standard are available, all based on a parallel virtual machine composed by several connected heterogeneous computers. Each of these computers executes a process used to control the exchange of messages among them. Again, several parallel implementations of metaheuristics have been developed using MPI, see e.g. [7, 23, 41, 76, 116, 132, 134, 136, 178, 183].

Finally, the increasing use of SMP clusters is leading to the generalized use of *programming with lightweight processes* such as POSIX threads [33, 110] or Java threads [150]. Threads are a general operating systems concept, not specifically related to parallel programming. However, due to their importance in providing support for concurrent programming, and to their extensive use in many of the tools discussed in other sections, their understanding is essential to a parallel programmer. Threads communicate using the global memory allocated to the associated processes. Programming with threads is specially useful in shared-memory machines, since in this case the global variables model the shared memory to which all processors have access. Smaller creation times and context swapping times lead to an improved superposition of communication and computation tasks along the execution time. The use of threads is particularly suited to medium- and coarse-grained parallel applications. Recently, the concept of threads has been extended to distributed-memory machines with programming tools such as Cilk [167, 190], Charm++/Converse [30, 111, 112, 113], Athapascan [49], PM2 [159], and Java threads [128], among others.

2.3. Types of parallelism and parallel programing models. There are two basic types of parallelism that can be explored in the development of parallel programs: data parallelism and functional parallelism [85, 121, 135, 144]. In the case of *data parallelism*, the same instruction set is applied to multiple items of a data structure. This type of parallelism can be easily implemented in shared memory machines. Data locality is a major issue to be considered, regarding the efficiency of implementations in distributed memory machines. In the case of *functional parallelism*, the program is partitioned into cooperative tasks. Each task can execute a different code and all tasks can run asynchronously. Data locality and the amount of processing within each

task are important concerns for efficient implementations.

Independently from machine architecture and programming environment considerations, we may distinguish two main parallel programming models: centralized and distributed. In the *centralized model*, also called master-slave or client-server, data are either handled by a specialized processor (the master) or stored in a shared memory. In the first case, the slaves communicate with the master to get work to be done and to send results. The master is in charge of load balancing and may also eventually perform computations. In the second case, the processors obtain the work to be done from the shared memory, where they also store the results they obtain. The *distributed model* is characterized by the absence of global data, either shared or centralized. All data is local to each processor. Information is shared or made global by the exchange of messages among the processors. SPMD (*Single Program, Multiple Data*) is a model used for parallel programming on MIMD machines, based on the distributed programming model, where the same code is executed on different processors over distinct data. It has been widely used due to the ease of designing a program that consists of a single code running on different processors [137, 160, 166].

The choice of the appropriate programming model strongly depends on the architecture of the target parallel machine where the implementation will run. With the advent of clusters of SMP machines, we see an increase in the number of implementations based on a hybrid model, i.e., implementations using a centralized model within each SMP machine, running under a distributed model with respect to the machines in the cluster. We will see in the forthcoming sections that parallel implementations of metaheuristics are usually based on either one of these models.

3. Speedup and efficiency. Given some problem $\mathcal{P}$, a parallel algorithm $\mathcal{A}_p$, and a parallel machine $\mathcal{M}$ with q identical processors, we denote by $T_{\mathcal{A}_p,\mathcal{M}}(p)$ the elapsed time taken by algorithm $\mathcal{A}_p$ to solve problem $\mathcal{P}$ on machine $\mathcal{M}$ using $p \leq q$ processors. We also denote by $T_{\mathcal{A}_s}$ the time taken by the best (i.e., the fastest) known sequential algorithm $\mathcal{A}_s$ to solve the same problem $\mathcal{P}$ on a sequential machine whose processor is equivalent to those in the parallel machine $\mathcal{M}$. Then, we define the *speedup* $s_{\mathcal{A}_p,\mathcal{M}}(p)$ of the parallel system defined by algorithm $\mathcal{A}_p$ and machine $\mathcal{M}$ when p processors are used as the ratio

$$s_{\mathcal{A}_p,\mathcal{M}}(p) = \frac{T_{\mathcal{A}_s}}{T_{\mathcal{A}_p,\mathcal{M}}(p)}.$$

We note that if the above definition is to be precise, then both the sequential algorithm $\mathcal{A}_s$ and the parallel algorithm $\mathcal{A}_p$ are supposed to always find exactly the same solution to problem $\mathcal{P}$. Although this will not necessarily be the case for most parallel implementations of metaheuristics, as discussed in Section 4, this definition will also be used in this context on a less formal basis. We also notice that due to the hardness of establishing the best sequential algorithm for general instances of some problem, the value of $T_{\mathcal{A}_s}$ in the above formula is often replaced and approximated by $T_{\mathcal{A}_p,\mathcal{M}}(1)$, i.e., the time taken by the parallel algorithm $\mathcal{A}_p$ using only one processor of the parallel machine $\mathcal{M}$.

The speedup measures the acceleration observed for the parallel algorithm running on p processors, with respect to the best sequential algorithm. The *efficiency* $\eta_{\mathcal{A}_p,\mathcal{M}}(p)$

of the above parallel system is given by

$$\eta_{\mathcal{A}_p,\mathcal{M}}(p) = \frac{s_{\mathcal{A}_p,\mathcal{M}}(p)}{p}.$$

It measures the average fraction of the time along which each processor is effectively used. Ideal efficiency values are as much close to one as possible, i.e., each processor should be used for as much time as possible in effective computations. However, some anomalies characterized by efficiency values larger than one are observed in some problem instances, usually due to wrong decisions taken by the sequential algorithm (see Porto and Ribeiro [161] for some illustrative results in the case of a single-walk parallel tabu search implementation, as well as e.g. [123, 124, 130, 131] for results and discussions in the context of parallel branch-and-bound algorithms).

Several authors have recently addressed the efficiency of parallel implementations of metaheuristics based on running multiple copies of the same sequential algorithm, classified as independent multi-thread strategies in Section 4.2.1. A given target value τ for the objective function is broadcasted to all processors which independently execute the sequential algorithm. All processors halt immediately after one of them finds a solution with value at least as good as τ. The speedup is given by the ratio between the times needed to find a solution with value at least as good as τ, using respectively the sequential algorithm and the parallel implementation with p processors. Some care is needed to ensure that no two iterations start with identical random number generator seeds. These speedups are linear for many parallel implementations of metaheuristics reported in the literature, i.e., they are proportional to the number of processors. A typical example is described in [155], for a PVM implementation of a parallel GRASP for the MAX-SAT problem. This observation can be explained if the random variable *time to find a solution within some target value* is exponentially distributed, as indicated by the following proposition [203]:

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

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

Proposition 2: Let $P_\rho(t)$ be the probability of not having found a given target solution value in t time units with ρ independent processors. If $P_1(t) = e^{-(t-\mu)/\lambda}$ with $\lambda \in \mathbb{R}^+$ and $\mu \in \mathbb{R}^+$, corresponding to a two parameter exponential distribution, then $P_\rho(t) = e^{-\rho(t-\mu)/\lambda}$.

Analogously, this proposition follows from the definition of the two-parameter exponential distribution. It implies that the probability of finding a solution within a given target value in time ρt with a sequential algorithm is equal to $1 - e^{-(\rho t-\mu)/\lambda}$, while the probability of finding a solution at least as good as that in time t using ρ independent parallel processors is $1 - e^{-\rho(t-\mu)/\lambda}$. If $\mu = 0$, then both probabilities

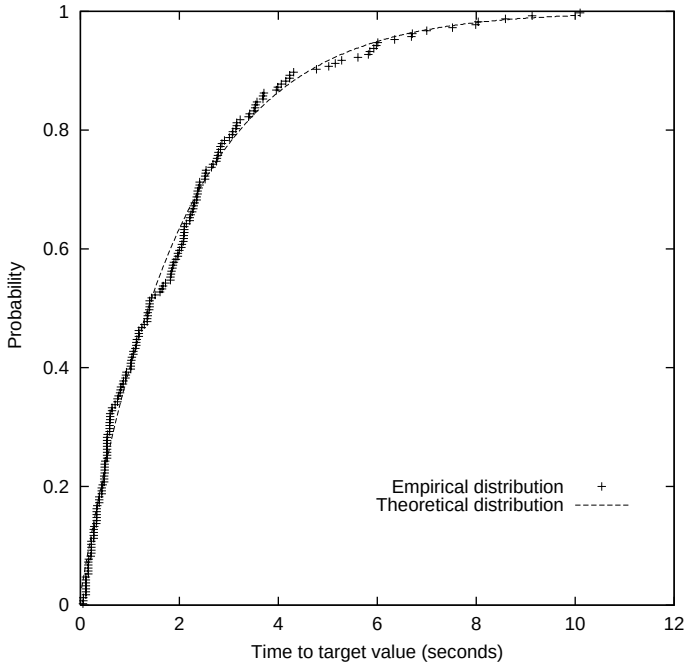

FIG. 3.1. *Superimposed empirical and theoretical distributions (times to target values measured in seconds on a SGI Challenge computer with 28 processors)*

are equal and correspond to the non-shifted exponential distribution. Furthermore, if $\rho\mu \ll \lambda$, then the two probabilities are approximately equal and it is possible to approximately achieve linear speedups in the time to find a solution within a target value using multiple independent processors [8].

This behavior has been noticed in a number of metaheuristics. These include simulated annealing [70, 152]; iterated local search algorithms for the traveling salesman problem [75], where it is shown that the probability of finding a sub-optimal solution is given by a shifted exponential distribution, allowing for the time to find the first local optimum; tabu search, provided that the search starts from a local optimum [24, 191]; and WalkSAT [186] on hard random 3-SAT problems [108]. Recently, Aiex et al. [8] have shown experimentally that the solution times for GRASP also have this property, showing that they fit a two-parameter exponential distribution. Figure 3.1 illustrates this result, depicting the superimposed empirical and theoretical distributions observed for one of the cases studied along the computational experiments reported by the authors, which involved GRASP procedures applied to 2400 instances of five different problems: maximum independent set [80, 171], quadratic assignment [129, 172], graph planarization [175, 177], maximum weighted satisfiability [173, 174], and maximum covering [170]. The same result still holds when GRASP is implemented in conjunction with a post-optimization path-relinking procedure [7].

4. 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. Although much of the initial efforts have been con-

centrated on the parallelization of simulated annealing [2, 19, 100], they were quickly followed by parallel implementations of genetic algorithms and tabu search. The first classification of parallel tabu search algorithms was proposed by Voss [205], based on the use of different search strategies and initial solutions. It was later generalized by Crainic et al. [60, 63], by taking into account additional aspects such as communication organization and information handling. We describe an architecture-independent classification considering the underlying parallel local search strategy, inspired from that proposed in [203]. We also discuss in details some applications of these strategies. Due to the high number of applications, we have chosen a few examples for each strategy, among those we are more familiar with.

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 towards 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). 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).

4.1. Single-walk parallelizations. 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. This approach may also be useful whenever one wants to explore extended neighborhoods obtained by move composition techniques such as ejection chains [93, 98].

The idea of *distributed neighborhoods* may be used to formalize parallelization strategies based on domain decomposition. A strategy based on distributed neighborhoods consists of two parts. Each iteration of the local search starts by broadcasting the current solution to all processors, together with a domain decomposition which defines the local neighborhoods (i.e., the portions of the complete neighborhood which will be investigated by each processor). This decomposition is temporary and may change from an iteration to another. In the second part, each processor finds and proposes a move within its local neighborhood. These moves are combined or the best move found within the local neighborhoods is selected, so that a new feasible solution is generated. This process is repeated until the current solution cannot be further improved.

Single-walk parallelizations have small or medium granularity and need frequent synchronizations. They are extremely dependent on the application, in terms of neighborhood structure and cost function definition. Their first applications appeared in the context of simulated annealing and genetic algorithms. They preserve convergence properties and are used in most parallel implementations of simulated annealing [2, 19]. Although domain decomposition strategies do not always provide significant reductions in computation times, they are often used due to the need to investigate large neighborhoods.

A single-walk parallelization based on neighborhood decomposition of a tabu search algorithm for solving a task scheduling problem on heterogeneous processors was proposed and discussed by Porto and Ribeiro [161, 162, 163]. The objective function to be minimized is the makespan of the parallel application, i.e., the time needed for the last processor to halt. Each solution has $O(n^2)$ neighbors, which can be obtained by reassigning each of its tasks to any other processor. Due to the precedence constraints, the evaluation of the makespan of each solution has time complexity $O(n^2)$, where n is the number of tasks. As mentioned by Fiechter [83], synchronous parallelizations usually require extensive communication and therefore are only worth applying to problems in which the computations performed at each iteration are complex and time consuming. This is exactly the case for this problem, in which the search for the best move at each iteration is a computationally intensive task running in time $O(n^4)$. Thus, from this point of view, this sequential tabu search algorithm is suitable for parallelization. Supposing p processors are available, the neighborhood is partitioned by distributing to each processor $k = 1, \ldots, p-1$ the moves defined by the transfer of the tasks numbered from $(k-1)\lfloor n/p \rfloor + 1$ to $(k-1)\lfloor n/p \rfloor + \lfloor n/p \rfloor$, and to processor p the remaining ones. Some computational results obtained on an IBM SP-1 system for problems with up to 400 tasks are reported in [161]. Different PVM implementations, using either the master-slave or the SPMD parallel programming models, showed that efficiencies close to the peak unit value (i.e., speedups of the same order of the number of processors used in the parallel implementation) can be obtained for large problems with up to $p = 16$ processors. Additional results reported in [164] for larger task graphs modeling typical program structures (such as diamonds,

divide-and-conquer, fast Fourier transform, Gauss decomposition, and partial differential equations) further illustrated the robustness of the best parallel strategy, which uses the master-slave model with a dynamic load balancing strategy.

Parallelizations of tabu search based on neighborhood decomposition and exhaustively searching for the best improving neighbor always find the same solution obtained by the sequential algorithm. They allow for efficient load balancing strategies, since the work to be done at each iteration is known beforehand and may be easily and evenly distributed among the processors, leading to efficient parallel implementations as that described in the above paragraph. However, if the parallelization involves a local search algorithm in which we seek the first improving move at each iteration, then load imbalance is more likely to deteriorate the implementation (because in this case the overall computational effort associated with the investigation of the neighborhood is not known beforehand). On the other hand, better solutions than those obtained by the sequential algorithm can be found, since in this case the application of several local searches based on finding the best solution over subsets of the neighborhood may lead to a better solution than a single local search applied to the whole neighborhood.

In the case of vehicle routing problems, parallel implementations of heuristics based on domain decomposition, such as that proposed by Fiechter [83] for the traveling salesman problem, start by partitioning the clients into subsets or clusters (according with some criteria such as proximity measures or polar coordinates), under the rationale that they should be part of the same route. Different processors acting in parallel may construct sets of different routes appropriate for each subset of clients, using constructive algorithms or metaheuristics. After synchronization, these routes are combined to yield a full solution to the routing problem. Since the combination requirements strongly limit the search for partial solutions and the quality of the final solution, the latter can be further ameliorated by a diversification procedure running at a higher level.

Aarts and Verhoeven [4, 203] make the distinction between single-step and multiple-step parallelism within this class. In the case of *single-step* implementations, neighbors are searched and evaluated in parallel after neighborhood partitioning. The algorithm subsequently selects and performs one single move, as for the above described parallelization of tabu search for task scheduling. In *multiple-step* parallelizations, a sequence of consecutive moves in the neighborhood graph is made simultaneously.

4.2. Multiple-walk parallelizations. Most parallel implementations of metaheuristics other than simulated annealing follow multiple-walk strategies. Tasks executed in parallel have a larger grain. Besides the search for speedups, improvements in solution quality are also sought. The search threads can be independent or cooperative.

4.2.1. 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 (or with a different population, in the case of population methods). 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.

The first parallel implementations of tabu search based on this type of strategy seem to concern the quadratic assignment problem and job shop scheduling [191, 192]. A parallel implementation of reactive tabu search [25] applied to the Steiner problem in graphs appears in this book [23]. Each slave processor independently performs the reactive tabu search algorithm, with a different parameter setting and from a different initial solution generated by a randomized version of the shortest-path heuristic of Takahashi and Matsuyama [195]. The master processor also handles a pool of elite solutions, which is used by a post-optimization intensification procedure based on path-relinking [94, 96] once all processors have finished their searches. Computational experiments have been performed on a cluster of 32 Pentium II-400 processors, running under Linux and using the LAM implementation of MPI. The parallel implementation found optimal solutions for all OR-Library instances [26] in series C, and for respectively 18 and 14 out of the 20 instances in each of series D and E. The maximum and the average relative errors with respect to the optimum among all instances of this type were respectively 0.14% and 0.06%. For the incidence problems [72], this strategy found optimal solutions for all instances with 80 nodes, and for respectively 96 and 82 out of each set of 100 instances with 160 and 320 nodes. The maximum and the average relative errors with respect to the optimum among all instances of this type were respectively 3.91% and 0.07%. The comparison of the results obtained by the parallel implementation with those reported for the original sequential algorithm shows that the former found improved results for all problem series and sizes, in particular for the largest instances. The main advantage of the parallel implementation is due not only to solution improvements, but mainly to its robustness, since high quality solutions are obtained without almost any effort in parameter tuning.

Rego and Roucairol [168] developed a parallel implementation of an ejection chain procedure for the vehicle routing problem with bounds on vehicle capacities and route lengths. To build an ejection chain with k levels, we first choose k clients not consecutively visited in the current solution. The first client takes the place of the second, the second that of the third, and so on. The k-th client takes either the place of the first (circular permutation) or is inserted into the best possible position between two clients. The chain is initialized by the choice of the client in the first level. Next, one computes iteratively for k ranging from one to a maximum level $k_{\max}$ the cost variation associated with the k-th level ejection chain and the best one is selected. This move is performed in the context of a tabu search procedure, which visits infeasible solutions using different strategic oscillation [95, 96] schemes. The authors implemented an independent-thread strategy within a centralized algorithm on a network of four Sun SPARC IPC workstations. Starting from the same initial solution communicated by the master, each slave runs a tabu search algorithm with different parameters. This strategy is combined with two single-walk parallelizations. The first one is based on neighborhood decomposition. All possible closing moves of the ejection chain are

evaluated in parallel by different processors, which search for the best position where the last client ejected from the chain should be inserted. The second one is a natural problem decomposition strategy based on a post-optimization procedure, which is activated whenever tabu search cannot improve the current solution after some number of iterations. Both single-walk parallelizations of the neighborhood search are quite effective and allow for approximately halving the computation time of the multiple independent-thread parallel algorithm. The latter, combining these different parallelization strategies, improved the best known solutions by 0.22% in the average, over 15 classical examples with up to 199 clients.

Typical parallelizations of GRASP make use of multiple independent threads. Basically, they consist in distributing over the p processors acting in parallel the total number of iterations to be performed. Since the GRASP iterations are completely independent and very few information is exchanged, linear speedups are easily obtained provided that no major load imbalance problems occur. The iterations may be evenly distributed over the processors or according with their demands, to improve load balancing. This strategy was applied first to the quadratic assignment problem [154] on a shared virtual memory KSR-1 machine. Parallel GRASP implementations of independent-thread strategies for the Steiner problem in graphs [134, 136] and for a matrix decomposition problem arising in the context of traffic assignment in satellite systems [13, 14, 165, 176] are reported farther in Section 5.2.

Parallelizations of genetic algorithms using multiple independent search threads correspond to the so called island model, without exchanges among the subpopulations in each island [31, 119]. This approach was also applied in the parallelization of a scatter search procedure for the quadratic assignment problem [65] on a small cluster of PCs. Different scatter search algorithms using different parameter settings are launched on the processors. Although some speedup has been observed, there was no improvement in the solutions obtained by the sequential algorithm. This is explained by the fact that small subpopulations within each processor freeze very soon, due to the absence of communication among the processors.

Multiple-walk strategies based on independent search threads can be very easily implemented. They lead to good speedups, provided the problem to be solved satisfies the conditions of Propositions 1 and 2 in Section 3. Very robust implementations can be obtained by using different parameter settings at each processor. Moreover, an appropriate partitioning of the search space allows for its good coverage and avoids redundant work. On the other hand, this model is quite poor and can be very easily simulated in sequential mode, by several successive executions with different initializations, as a multistart algorithm. The lack of cooperation between the search threads does not allow for the use of the information collected by different processors along their trajectories. Since the trajectories may be quite long, load imbalance problems are very likely to occur. Redundant work may be done if the search space is not appropriately partitioned.

4.2.2. Cooperative search threads. This is the most general and also the most promising type of strategy, 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, tabu lists, and population sizes, among others. This data can give a broader view of the search space and may be used to drive diversification and intensification procedures.

The multicommodity location-allocation problem with balancing requirements [61] 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 [59]. Improved solutions and smaller computation times are obtained for both applications, with respect to the sequential algorithms.

The same approach was used by Aiex et al. [6] in the parallelization of a tabu search algorithm for circuit partitioning, in the context of the pseudo-exhaustive logical test of VLSI combinational circuits. Each search thread runs a variant of the original sequential tabu search algorithm developed for this problem [16], using different initial solutions and move attributes. The search threads communicate through a central pool of elite solutions, implemented on a dedicated master processor which handles pool management. An elite solution is defined as a local optimum which improves the best solution already visited by the search trajectory in the same processor. Each processor performing the search sends to the pool the elite solutions it has found. A newly received solution is inserted into the pool or not, according with some criteria based on the quality of the solutions already in the pool. A processor requests a solution from the pool to restart its search whenever it is not able to improve its best solution after some number of iterations. Numerical results using nine search processors on an IBM SP-2 machine are reported for benchmark circuits with up to 207 inputs, 3405 gates, 140 outputs, and 7552 links. Significant reductions of up to 30% in the number of circuits in the partition illustrate the effectiveness of the parallel tabu search procedure. Comparative results illustrating the efficiency of implementations in PVM and Linda [43, 50, 185] are also discussed.

Parallelizations of GRASP using multiple cooperative search threads may be implemented with a similar use of a pool of elite solutions and are described in Section 5.2. The search threads cooperate through a path-relinking procedure [94, 96, 97], combining elite solutions from the pool with the local optima found at the end of each iteration (see also [7, 41, 42] for details).

Software packages such as EBSA [86] and ASA [109] with implementations of simulated anealing may run on parallel machines. ParSA is a library developped in C++ [116], containing multiple-walk strategies that allow the exchange of some statistics. Some numerical results obtained for a scheduling problem in aerial transportation with 4,000 flight segments and 130 aircrafts of ten different types lead to improved solutions and speedups of 5.4 on eight nodes of an ATM cluster and of 17.5

on a GC/PowerPlus machine with 32 nodes.

Cooperative strategies are also the most successful ones for the parallel implementation of genetic algorithms, in terms of both the number of applications and the improvements in solution quality. Several libraries are also available, such as POOGAL [31] and PGAPack [17]. This type of parallelization is defined by an additional operator of *migration*, which establishes the policy for solution exchanges: processors involved in the exchanges, frequency of exchanges, and solutions exchanged, among other information. Most cooperative parallel implementations of genetic algorithms are based on the island model with communication. Each processor has its own subpopulation and solutions are exchanged with neighbor processors according with its migration policy [31, 60]. Another variant consists for each solution to select a mate in another processor within a specified neighborhood to perform a crossover. The selection strategy may even involve another heuristic in the determination of the best possible mate. Bubak and Sowa [31] used the island model in the implementation of a parallel procedure for the traveling salesman problem on an HP/Convex Exemplar SPP1600 machine with 16 processors and on a heterogeneous cluster formed by Hewlett Packard (D-370/2 and 712/60) and IBM (RS6000/520 and RS6000/320) machines.

Since ant system optimization can be viewed as the application of an iterated greedy randomized algorithm, parallelization strategies are similar to those of GRASP. Cooperation is guided by the pheromone trails. Each thread handles one ant and the ants communicate through the pheromone trails. Typically, the ants are evenly distributed over the processors. Since each ant acts independently of the others, linear speedups can be obtained. In practice, however, the communication incurred by the management of the pheromone trails as global information is an important overhead. Since all ants use and update the pheromone trails, access to the latter is clearly the key point for efficient parallel implementations. Taillard [193] dedicated the management of the pheromone trails to a *queen* process, which is a very convenient way to implement a parallel ant system following a master-slave scheme [199, 200]. The bottleneck at the master can be dealt with by a hierarchical approach (i.e., a hierarchy of masters) or by a colony approach similar to the island model in genetic algorithms. Each colony is handled by a master processor, who manages its pheromone trails. The information exchanged between the colonies can be either ants finding good solutions or parts of the local pheromone trails to influence the searches performed by the other colonies.

5. Applications. We review in this section parallel implementations of algorithms derived from tabu search, GRASP, genetic algorithms, simulated annealing, and ant colonies. This review is not intended to be either exhaustive, or comprehensive, due to the enormous amount of work and publications in the area in recent times. Instead, we try to give a broad view of applications and implementation studies of parallel metaheuristics, referring the reader to the original references for additional details and publications.

5.1. 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. [91, 92, 96]). Tabu search goes beyond local search by employing a memory-based strat-

egy 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 towards solutions which deteriorate the cost function are permitted under certain conditions.

Single-walk parallelizations of tabu search based on neighborhood decomposition are reported in [51, 52, 62, 87, 161, 162, 163, 164, 191]. An interesting application in chemistry to determine the best energy minimum for oligopeptides is reported in [143]. Different systems such as the Connection Machine CM-2, a network of transputers, an IBM SP-1 machine, a SGI Origin 2000, a Meiko computer with 16 T-800 transputers, and a network of Sun SPARC workstations have been used for these implementations. Computation times are smaller than those of their sequential counterparts and almost-linear speedups are often obtained, but only minor improvements in solution quality are observed for some instances. A parallel tabu search algorithm for the traveling salesman problem based on domain decomposition [83] was already discussed in Section 4.1.

Badeau et al. [20] implemented a parallel algorithm for vehicle routing with time windows, combining single- and multiple-walk strategies. First, the 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 (i.e., the routes) it received, so as to generate a unique initial solution (formed by a set of routes) which is sent to slaves running decomposition processes. Each of the latter decomposes its current solution into several subproblems, which are then solved by variants of the same sequential tabu search algorithm [194]. The solutions to the subproblems are merged into a complete solution and sent to the master, which combines the new routes to obtain an improved solution. A new iteration of this procedure resumes, until some stopping criterion is attained. Computational results for an implementation on a network of Sun SPARC 5 workstations are reported. Gendreau et al. [89] implemented a parallel tabu search algorithm for real-time vehicle routing and dispatching following a similar master-slave scheme. The master processor manages the adaptive memory, decomposes the original problem into subproblems, and creates new starting solutions for the slaves. The latter apply tabu search to improve their initial solutions. Although the threads run independently, they communicate implicitly through the adaptive memory, which stores good solutions and distributes them as new starting solutions. Computational experiments have been performed on a network of 17 Sun UltraSPARC workstations, with communications between the processes being handled by PVM. As for the first application described in this paragraph, the authors report that the parallel tabu search implementation is competitive with the original sequential algorithm, in terms of solution quality for the same amount of computational work, however with much smaller elapsed times. A multiple-walk tabu search implementation for job-shop scheduling using independent search threads is reported in [192]. Talbi et al. [196] described an independent-thread multiple-walk parallel implementation of tabu search applied to the solution of the quadratic assignment problem on a network of heterogeneous workstations, based on the idea of adaptive parallelism to take into account and explore variations in machine load and availability.

A multiple-walk parallel implementation of an ejection chain procedure [168] for a vehicle routing problem with bounds on vehicle capacities and route lengths was

already described in Section 4.2.1. Another parallel implementation of a tabu search algorithm for vehicle routing is reported in [184]. The neighborhood structure is based on simple customer shifts. All routes generated by independent search threads running a tabu search heuristic are collected in a distributed pool. New initial solutions are periodically obtained by a fast set covering heuristic applied to the routes in the pool. Computational results on a Parsytec CC system with eight Motorola PowerPC 604 nodes are reported. We have already reported in Setion 4.2.1 the results obtained by Bastos and Ribeiro [23] with 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. In the last two cases, we notice that although the threads are independent, they cooperate either periodically or at a post-optimization phase using a pool of good solutions.

Cooperative multiple-walk implementations 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 exclusively through a central pool of elite solutions. Crainic et al. [61] implemented several parallelizations of tabu search for the multicommodity location-allocation problem with balancing requirements and discussed the impact on performance and solution quality of several parameters. Crainic and Gendreau [59] have also implemented a cooperative multiple-walk parallel tabu search for the fixed charge, capacitated, multicommodity network design problem, which is shown to outperform both the sequential algorithm and independent-thread multiple-walk strategies. A similar approach was used by Aiex et al. [6] in the parallelization of a tabu search algorithm for circuit partitioning, as described in Section 4.2.2. The same strategy was recently applied in the solution of a labor constrained scheduling problem on a Sun SPARC 1000 workstation with eight processors [46], which significantly improved not only the solutions found by different sequential algorithms [47, 48], but also those obtained by a parallel asynchronous team [201] for most of the benchmark instances. Another possibility for cooperation could also consist in applying path-relinking to solutions newly found by the search threads, using those in the pool of elite solutions as targets, as in some parallel implementations of GRASP [7, 41, 42] described in Section 5.2.

5.2. GRASP. A greedy randomized adaptive search procedure (GRASP) [78, 79, 82] 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.

A multiple-step single-walk parallelization of a GRASP for the maximum independent set problem is described in [80], based on the decomposition of the solution space. The problem to be solved is divided into several subproblems, which are distributed over the processors. Each processor performs a sequential GRASP on a different subproblem, defined by conditioning the vertices appearing in the solution. Almost-linear speedups are reported.

Most parallel implementations of GRASP are independent-thread multiple-walk strategies, based on the distribution of the iterations over the processors. In gen-

eral, 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. So as to avoid that the processors find the same solutions, each of them must use a different sequence of pseudorandom numbers. 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.

The parallelization of the GRASP developed by Prais and Ribeiro [165] for the problem of traffic assignment in communication satellites is discussed in [13]. The author also described the implementation of another independent multiple-walk strategy, in which each processor uses a different parameter value. Li et al. [129] reported almost-linear speedups of approximately 62 using 64 processors for a parallel GRASP for quadratic assignment. Other parallel implementations are described e.g. in [149, 154, 155].

Martins et al. [134] implemented a parallel GRASP for the Steiner problem in graphs. The construction phase is based on a probabilistic version of the distance network heuristic [138]. A hybrid local search procedure is applied only to new, previously unexplored solutions. First, a local optimum with respect to a neighborhood defined by insertions and eliminations of Steiner nodes is sought. This solution is then submitted to another local search procedure, based on the exchange of key-paths (i.e., paths connecting terminals or Steiner nodes with degree larger than two; see also [133, 136, 204] for additional details). Parallelization is achieved by the distribution of 512 iterations over the processors, with the value of the parameter α randomly chosen in the interval $[0.0, 0.3]$ at each iteration. The algorithm was implemented in C on an IBM SP-2 machine with 32 processors, using the MPI library for communication. The 60 problems from series C, D, and E from the OR-Library [26] have been used for the computational experiments, with respectively 500, 1000, and 2500 nodes. The parallel implementation obtained 45 optimal solutions over the 60 test instances and the relative deviation with respect to the optimal value was never larger than 4%. Average elapsed times in seconds (measured in exclusive mode, i.e., with no other application simultaneously running in the same machine) for both the sequential and parallel versions are reported in Table 5.1, while the speedups obtained with 2, 4, 8, and 16 processeurs are illustrated in Figure 5.1.

	Processors (times in seconds)				
Series	1	2	4	8	16
C	32.77	17.77	10.12	6.10	4.08
D	154.12	85.19	50.36	31.68	22.08
E	1379.90	828.19	485.97	307.44	216.86

TABLE 5.1
Elapsed times in seconds for 2, 4, 8, and 16 processors (512 iterations)

Alvim and Ribeiro [13, 14] have shown that multiple-walk independent-thread approaches for parallelization may benefit a lot from load balance techniques, whenever heterogeneous processors are used or if the parallel machine is simultaneously shared by several users. In this case, almost-linear speedups may be obtained with a heterogeneous distribution of the iterations over p processors in $q \geq p$ packets. Each

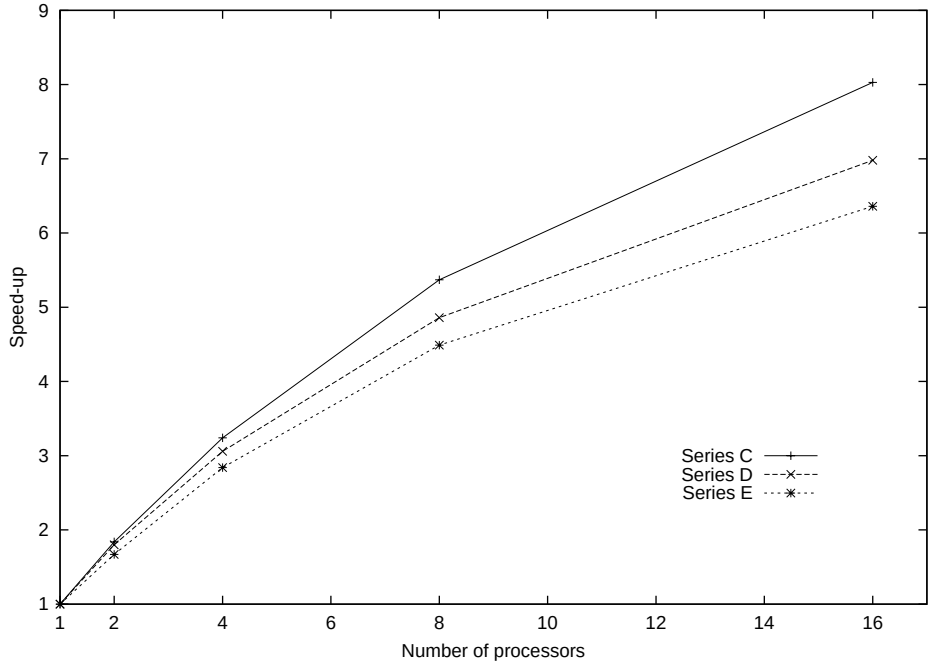

FIG. 5.1. *Average speedups on 2, 4, 8, and 16 processors*

processor starts performing one packet of $\lceil$`Max_Iterations`$/q\rceil$ iterations. Each slave processor informs the master when it finishes its packet of iterations. The master stops the execution of each slave processor when there are no more iterations to be performed and collects the best solution found. Faster or less loaded processors will perform more iterations than the others. In the case of the parallel GRASP implemented for the problem of traffic assignment, this dynamic load balancing strategy allowed reductions in elapsed times of up to 15% with respect to the times observed for the static strategy, in which the iterations were uniformly distribued over the processors. The same approach was used in [136] for a preliminar parallel implementation of the above described GRASP for the Steiner problem in graphs.

Canuto et al. [41, 42] used path-relinking as a post-optimization step to implement a multiple-walk parallel GRASP algorithm for the prize-collecting Steiner tree problem. A similar approach was recently adopted by Aiex et al. [7] for the 3-index assignment problem. In both cases, the first phase of the parallel implementation is an independent multiple-walk strategy. Each processor, upon completing its iterations, applies path-relinking to pairs of elite solutions stored in a pool. In the second case [7], the parallel implementation globaly follows an independent-thread strategy, since each processor keeps its own local pool of elite solutions. However, the strategy used in the first case [41] leads to a cooperative procedure, since pairs of elite solutions from a centralized unique central pool are distributed to the processors which perform path-relinking in parallel. Computational results obtained with implementations using MPI and running on a cluster of 32 Pentium II-400 processors [41] and on a SGI Challenge computer with 28 196-MHz MIPS R10000 processors [7] show linear speedups and illustrate the effectiveness of path-relinking procedures used in conjunction with GRASP to improve the quality of the solutions found by the latter.

5.3. Genetic Algorithms. A genetic algorithm is a basic search procedure, proposed and developed by Holland [106, 107]. It is based on the process of natural evolution, following the principles of natural selection and survival of the fittest [141, 169]. 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 [35, 37, 40, 56]: global (single-walk parallelization), diffusion model, and island model (also called migration model by some authors). Fine-grained (diffusion model) and coarse-grained (island model) parallelizations characterize cooperative multiple-walk strategies.

Global (single-walk) parallelization strategies 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. This approach was used by Chalermwat et al. [53] in the context of an application to image registration, in which parallelism is exploited for fitness evaluation. The authors reported very accurate registration results for LandSat/Thematic Mapper images and the parallel algorithm seemed to scale quite well on a 50-node Beowulf parallel cluster of Pentium Pro 200 MHz processors with a fat-tree network topology. Abramson and Abela [5] implemented a global master-slave parallelization to solve a school timetabling problem on a shared-memory Encore Max computer with 16 processors, in which the master distributes the individuals to be evaluated by each slave processor. The master also performs crossover and mutation operations to obtain a new generation. They reported limited speedups, due to critical sections of the code not being parallelized. Grefenstette [101] presented a master-slave parallel genetic algorithm for a system that learns strategies. The master maintains a population of competing strategies. At each iteration it generates a new population of strategies that are evaluated by the workers. Each worker evaluates a strategy by simulating the execution of the assigned strategy in a determined task environment. Observed speedups were closer to linear for tasks requiring larger times for evaluating the strategies. Mühlenbein [148] developed a parallel genetic algorithm where each individual performs hill-climbing and selects a partner for mating within its neighborhood. The author claimed better or comparable solutions for very large instances of both the traveling salesman problem and the graph partitioning problem, with respect to those found by other heuristics such as multi-start hill-climbing and iterated hill-climbing.

Fine-grained parallelizations 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 [56]. Talbi and Muntean [198] proposed and implemented a fine-grained algorithm on a T-800 transputer to solve a task mapping problem. Near-linear speedups have been observed and solution quality improved when the size of population was increased. Hill-climbing and a parallel simulated annealing algorithm were also implemented to be compared with the parallel genetic algorithm. The latter outperformed hill-climbing in terms of solution quality, with similar execution times. Simulated annealing obtained similar results concerning solution quality, but much greater elapsed times than the genetic algorithm. Kohlmorgern et al. [119] presented some results concerning fine-grained parallel versions of both the island and the diffusion models, applied to four problems: traveling salesman, flow shop scheduling, scheduling with resource constraints, and uncapacitated facility location. The initial population was partitioned into 1, 4, 16, 64, 256, and 1024 islands on a MasPar MP-1 parallel machine with 16 K processors organized as a two-dimensional grid, one individual per processor. In the island model implementation, the processing elements were partitioned into arrays forming the islands. Solutions are exchanged with neighbor islands in one, two, three, or four directions, after every 15, 30, or 50 generations. The authors noticed that using a high number of islands leads in general to better solutions, but with the burden of slower convergence. Good speedups in terms of the number of processors have been observed. They have also implemented another procedure, based on considering neighbors at different distances in the eight possible directions of the MasPar MP-1's X-net. The elitist strategy, where the very best neighbor is always selected, found the best solutions.

Coarse-grained or *island model* algorithms are based on dividing the population into a 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. Ocasionally, 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), see e.g. [35, 36, 157]. Some libraries to support the development of parallel genetic algorithms based on the island model were already cited in Section 4.2.2. Bubak and Sowa [31] developed an implementation for the traveling salesman problem using the island model, as described in Section 4.2.2. Belding [27] extended previous work on distributed genetic algorithms [202], focusing migration rates and intervals. A coarse-grained algorithm was developed and tested using the Royal Road class of fitness functions. For the easiest functions, the sequential algorithm performed better, while for the hardest functions the parallel algorithm was able to find better solutions. Levine [127] implemented a genetic algorithm based on the island model for solving the set partitioning problem. Tests were carried out on 128 nodes of an IBM SP-1 and showed that additional subpopulations, which increase the global population size, contributed to improve solution quality. Andre and Koza [15] described a coarse-grained parallel implementation in C and PVM of a genetic algorithm on a network of transputers. Experiments were made considering the problem of symbolic regression of the Boolean even-5-parity function and using different migration rates. Paral-

lelization delivered super-linear speedups and best results in terms of computation times have been observed with an 8% migration rate. Another coarse-grained parallel implementation of a genetic algorithm is described by Falco et al. [68], concerning transonic airfoil optimization. Solutions are exchanged at the end of each iteration. The algorithm was implemented using C and PVM. Computational experiments were performed on a Convex Meta Series machine. Innovative designs of shockless airfoils were obtained by this algorithm, with the authors reporting convergence within much lower times than the sequential algorithms.

Cantu-Paz [36] described the implementation of two different parallel genetic algorithms. The first uses a master-slave strategy, while the other uses a coarse-grained model. The code was written in C++ and the PVM library was used for communication. The computational experiments were processed on a network of workstations and the tests were performed using several fitness functions with different degrees of hardness of the evaluation function. For the master-slave implementation, the results showed that the efficiency of the parallel implementation increases with the hardness of the evaluation function (as expected, since coarse-grained applications are more suitable to overcome the communication overhead introduced by the master-slave scheme). The same set of tests used for the coarse-grained algorithm showed that there are not significant gains in terms of speedup when the subpopulations are completely isolated. They also showed that, when each process is fully connected and migrations of individuals can occur between any pair of processors, the speedup increases with the hardness of the evaluation function. The reason is that the reduction in computation time brought by the use of multiple subpopulations is not enough to overcome the increase in communication time for easy-to-compute functions. The numerical results obtained for the coarse-grained implementation are quite similar to the theoretical ones developed in [39] for predicting speedups of idealized bounding cases of parallel genetic algorithms.

Some hybrid implementations use combinations of multiple subpopulations with global or fine-grained methods. They form a hierarchy where the lower level consists of a global or fine-grained parallel genetic algorithm that manipulates the subpopulations, while a higher level implements a coarse-grained algorithm that distributes subpopulations to the lower level threads and controls solution migration among them. Bianchini and Brown [28] implemented four parallelization strategies of a genetic algorithm for solving 0-1 integer programming problems: centralized, semi-distributed, distributed, and totally distributed. These implementations were tested on a distributed memory environment consisting of a Transputer with eight 25 MHz T-805 processors, connected by 20 Mbits links. In the centralized method, the master sends some individuals to slave processors, that compute a certain number of generations and send their results back to the master, which then executes the replication algorithm for the whole population. The semi-distributed method consists of clusters of processors working with the centralized method and exchanging solutions among them. The distributed method is the traditional coarse-grained strategy, where each processor has its own subpopulation and exchanges of the best individuals occur from time to time. Finally, the totally distributed implementation consists of the distributed method without any exchange of individuals, each processor executing its own sequential algorithm over a subpopulation without communication. The authors compared these four strategies among them and with a sequential algorithm. All strategies presented similar results in terms of solution quality, with the centralized

and semi-distributed ones achieving slightly better results. The speedup achieved by the totally distributed strategy is almost linear, while the centralized and semi-distributed implementations showed smaller speedups, due to bottlenecks created by the master process.

Oussaidène et al. [153] presented a hybrid parallel implementation of a genetic algorithm, in which multiple master-slave instances are associated with the subpopulations. A subpopulation is assigned to each master. The individuals are distributed over the slaves for the computation of their fitness values. The masters can exchange individuals from their subpopulations from time to time. Computational experiments with a parallel genetic algorithm code written in Java are reported in [90] for evolving pharmaceutical drug molecules and digital circuits. This code runs under a batch system called Condor, which manages the resources of a network of workstations and assigns jobs for each workstation according to its load. The initial experiments involved 31 jobs running independently, using the same input parameters but different random seeds. The results for evolving drug molecules were fairly good, but only trivial digital circuits were evolved.

The effect on the performance of different parallelization schemes of genetic algorithms induced by variations of parameters such as the number of slaves, population size, mutation rate, and mutation interval has been addressed by several authors, see e.g. [10, 11, 37, 40, 187] among others.

5.4. Simulated annealing. Simulated annealing in the context of optimization methods was introduced by Kirkpatrick et al. [115], see e.g. [3]. As the first metaheuristic to be parallelized, several papers describing parallelization efforts appeared subsequently [1, 21, 22, 44, 45, 67, 77, 182, 208], most of them focusing cell placement problems in VLSI layout and the convergence property proved by Geman and Geman [88]. Tools such as ProperPLACE [114, 156] have been designed to solve these problems with parallel implementations of simulated annealing.

Most of the first parallelizations 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. Kravitz and Rutenbar [120, 182] described one of the first examples of this type of parallelization of the simulated annealing metaheuristic. In the case of parallel moves, each processor generates, evaluates, and accepts (or not) a different move. Synchronization could be done after only one or after several steps. However, due to the moves made by other processors, the cost function evaluation is not error-free regarding the sequential execution. To overcome the risk of inconsistencies, two approaches are proposed. The first considers only noninteracting moves (see also [21, 22, 120, 182]), for example by using a master-slave scheme in which the master monitors the cooling schedule by synchronizing the slaves at each step and choses the accepted move [180]. In the second case, interacting moves are accepted and some errors in the evaluation of the cost function are allowed [44, 45, 179]. This approach is generally used in the framework of domain decomposition. In the case of the cell placement problem, this corresponds to partitioning the cells into subsets, handling each subset by one processor, and restricting moves in the same subset. This decomposition framework is also used for the traveling salesman problem in [12, 77]. Other authors have addressed the impact of errors on the convergence properties, see

e.g. [19, 73, 74, 100]. This second approach can be viewed as a strong diversification process with respect to the sequential execution, which could explain the better quality of the results so obtained. Since the synchronization strategy can be performed more loosely between the processors, the reported speedups are better than those attained with the first, error-free synchronous approach. A variant of the single-walk strategy was proposed in [207] for the task assignment problem. The basic idea consists in doing speculative computations by evaluating in parallel by two threads both possibilities, acceptance or rejection, before the decision on the previous move has been completed. This approach does not seem to yield very promising results.

Other coarse-grained parallelizations of simulated annealing are based on the use of multiple Markov chains, which is equivalent to the multiple-walk strategy. Unlikely the previously described parallelizations, each chain is allowed to perform cell moves on the whole set of cells, and not only within subsets. The synchronous approach was first introduced by Aarts et al. [1]. Let I denotes the number of iterations (i.e., the length of the Markov chain) executed by a simulated annealing process before reaching equilibrium at temperature T. The parallel strategy consists in executing I/p iterations at temperature T by each of the p threads, followed by the synchronization of all threads to choose the best solution for the next temperature. However, this parallel process presents two main drawbacks: the communication overhead to exchange global information at each synchronization point and the lack of convergence properties, since the length of the Markov chain is arbitrarily reduced. Lee and Lee [122, 125, 126] presented some refinements to this approach, by introducing asynchronous strategies for the graph partitioning problem. The interaction between threads executing the traversal of the multiple walks is clearly the key point to efficiency in this approach.

Another implementation of the above approach, based on the execution of several cooperative threads running simulated annealing at different temperatures of the sequential cooling schedule, was proposed by Miki et al. [142]. Solutions are randomly exchanged between two consecutive temperatures of the cooling schedule. Applied to the minimization of a standard test function from a continuous optimization problem, the results reported are better than those obtained by the sequential algorithm in terms of both solution quality and computation time (speedup of approximately four in an 8-node PC cluster). However, this approach does not seem to outperform by too much the previous multiple-walks approaches. An empirical comparison of all these approaches appears in [54, 55]. The readers are also referred to [19, 60, 100] for other surveys about parallel simulated annealing and its hybrids.

5.5. Ant colonies. The ant system introduced by Colorni et al. [57] is presented as yet another evolutionary method inspired from the nature. However, an ant system applied to some combinatorial optimization problem can be viewed as an iterated greedy randomized algorithm, guided by the pheromone trails. Indeed, each greedy process correspons to an ant. The pheromone trails are used by the ants to bias the choice at each step of the greedy randomized algorithm. Ant systems differ from each other by the different strategies they use to handle the pheromone trails (updates and evaporations) [57, 71].

Bolondi and Bondanza [29] and Bullnheimer et al. [32] proposed two very similar and strongly synchronized parallel ant systems applied to the traveling salesman problem. At each parallel iteration, a set of ants is generated by the queen process and each ant explores a tour in parallel. Once every ant has finished its exploration,

they are synchronized and the pheromone trails are updated by the queen process with the best tour they have found. Since the amount of computations performed by each ant is quite small, the ants spent most of their time in synchronizations with the queen process. For this reason, the authors also proposed in [32] a less synchronized parallel ant system to reduce the synchronization time. The underlying idea consists in implementing a 2-level hierarchical approach. Each of many first-level masters handles in parallel but locally its own ant colony and pheromone trails, using a sequential algorithm. After a given number of sequential iterations, the first-level masters are synchronized by a second-level master process to globally update the pheromone trails. Since synchronizations are mainly done at the second-level master, this strategy showed less synchronization overhead than the previous one. The local/global iteration ratio is crucial for this strategy, since load imbalance is very likely to occur between the different first-level masters if this ratio is not correctly tuned.

Stützle [188] proposed a very simple parallel implementation, based on parallel independent runs of the MAX-MIN ant system. The main interest of this work is to check the property of Proposition 1, announced in Section 3. Since the execution time taken by the sequential algorithm for solving the traveling salesman problem seems to follow an exponential distribution, the relative good empirical results obtained in terms of speedups and average solution quality seem to confirm the prediction of the proposition.

The quadratic assignment problem is yet another application of parallel ant systems. Talbi et al. [199, 200] applied the master-slave approach to their ANTabu system. The master handles the pheromone matrix and the best solution found. At each iteration, the master broadcasts the pheromone matrix to all the slave ants. Each slave constructs a complete solution, improves it using tabu search, and sends the improved solution back to the master. Computational experiments on a network of ten SGI Indy workstations running PVM are reported.

In addition to these applications to the more common traveling salesman and quadratic assignment problems, we mention that ant systems have also been adapted to the distributed dynamic routing problem [69] and, more recently, to automatic programming [181].

6. Concluding remarks. The development of portable, efficient, and easy-to-use software tools, together with the availability of a wide range of faster parallel machines with increasing reliability and decreasing costs, has brought parallel processing into reality as an effective strategy for the implementation of computationally efficient techniques for the solution of large, difficult optimization problems.

Parallel implementations of metaheuristics are a quite effective alternative for the approximate solution of combinatorial optimization problems. They are usually applied in the context of complex applications, often allowing reductions in computation times. Good speedups can be obtained with parallel non-cooperative strategies. However, parallel metaheuristics not only allow solving larger problems or finding improved solutions with respect to their sequential counterparts, but parallelizations based on cooperative multiple search threads also lead to more robust implementations, which are likely to be the most important contribution of parallelism to metaheuristics. The use of multiple processors involved in a broader search of the solution space, using different srategies and parameter values, allows for larger diversity and deeper inves-

tigation of the solution space. As a consequence, improved solutions may be found, very often even faster than in sequential mode. To summarize, parallelism often leads to speedups for complex applications and to more robust algorithms, due to improved mechanisms for search intensification and diversification.

We reviewed several applications of parallel implementations of metaheuristics such as tabu search, GRASP, genetic algorithms, simulated annealing, and ant colonies to a wide variety of hard combinatorial problems. Successful implementations of metaheuristics are often based on the appropriate combination of components of different methods. In the same vein, the hybridization of different metaheuristics in the framework of a cooperative parallel implementation (such as a genetic algorithm and several tabu search threads cooperating by means of a pool of elite solutions generated by the latter and improved by the former, or the construction phase of a GRASP procedure being used to create initial subpopulations for a parallel implementation of a genetic algorithm based on an island model, among many other possibilities) is a very promising research avenue.

The judicious choice of a programming environment is essential to the implementation of a parallel metaheuristic. Implementation issues may lead to programs that perform very poorly, independently of the quality of the underlying parallel algorithm. One possible source of problems is a mismatch between the proposed algorithm and the programming model supported by a parallel machine or a development tool. It is also common for the program, once implemented, to have a different behavior from what was expected due to unforeseen communication and processing bottlenecks. Performance evaluation tools have a fundamental role in solving this kind of problem, since they can help the programmer in identifying the origin of such bottlenecks.

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 (1986), 209–238.

[2] E.H.L. Aarts and J. Korst, *Simulated annealing and Boltzmann machines*, Wiley, Chichester, 1989.

[3] E.H.L. Aarts and J. Korst, "Selected topics in simulated annealing", in *Essays and surveys in metaheuristics* (C.C. Ribeiro and P. Hansen, eds.), Kluwer, 2001 (this volume).

[4] E.H.L. Aarts and M. Verhoeven, "Local search", in *Annotated bibliographies in Combinatorial Optimization* (M. Dell'Amico, F. Maffioli, and S. Martello, eds.), 163–180, Wiley, 1997.

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

[6] 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", *Lecture Notes in Computer Science* 1457 (1998), 310–331.

[7] R.M. Aiex, M.G.C. Resende, P.M. Pardalos, and G. Toraldo, "GRASP with path-relinking for the three-index assignment problem", submitted for publication, 2000.

[8] R.M. Aiex, M.G.C. Resende, and C.C. Ribeiro, "Probability distribution of solution time in GRASP: An experimental investigation", *Journal of Heuristics*, to appear.

[9] S.M. Alaoui, O. Frieder, and T. El-Ghazawi, "A parallel genetic algorithm for task mapping on parallel machines", *Lecture Notes in Computer Science* 1586 (1999), 201–209.

[10] E. Alba and J.M. Troya, "Influence of the migration policy in parallel distributed GAs with structured and panmictic populations", *Applied Intelligence* 12 (2000), 163–181.

[11] E. Alba and J.M. Troya, "Analyzing synchronous and asynchronous parallel distributed genetic algorithms", *Future Generation Computer Systems* 17 (2001), 451–465.

[12] J.R. Allwright and D.B. Carpenter, "A distributed implementation of simulated annealing for the travelling salesman problem", *Parallel Computing* 10 (1989), 335–338.

[13] A.C. Alvim, *Parallelization stratgies of the GRASP metaheuristic* (in Portuguese), M.Sc. Dissertation, Department of Computer Science, Catholic University of Rio de Janeiro, 1998.

[14] A.C. Alvim and C.C. Ribeiro, "Load balancing for the parallelization of the GRASP metaheuristic" (in Portuguese), *Proceedings of the X Brazilian Symposium on Computer Architecture*, 279–282, Búzios, 1998.

[15] D. Andre and J.R. Koza, "Parallel genetic programming: A scalable implementation using the transputer network architecture", in *Advances in Genetic Programming 2* (P.J. Angeline and K.E. Kinnear, Jr., eds.), 317–338, MIT Press, 1996.

[16] A.A. Andreatta and C.C. Ribeiro, "A graph partitioning heuristic for the parallel pseudo-exhaustive logical test of VLSI combinational circuits", *Annals of Operations Research* 50 (1994), 1–36.

[17] Argonne National Laboratory, "PGAPack Parallel Genetic Algorithm Library", online document, `http://www-fp.mcs.anl.gov/CCST/research/reports_pre1998/comp_bio/stalk/pgapack.html`, last visited on February 11, 2001.

[18] G. Authié, A. Ferreira, J.-L. Roch, G. Villard, J. Roman, C. Roucairol, and B. Virot, *Algorithmique Parallèle: Analyse et Conception*, Hermès, 1994.

[19] R. Azencott, *Simulated annealing: Parallelization techniques*, Wiley, 1992.

[20] 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 (1997), 109–122.

[21] P. Banerjee and M. Jones, "A parallel simulated annealing algorithm for standard cell placement on a hypercube computer", *Proceedings of the 1986 International Conference on Computer-Aided Design*, 34–37, Santa Clara, 1986.

[22] 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 (1990), 91–106.

[23] M.P. Bastos and C.C. Ribeiro, "Reactive tabu search with path relinking for the Steiner problem in graphs", in *Essays and surveys in metaheuristics* (C.C. Ribeiro and P. Hansen, eds.), Kluwer, 2001 (this volume).

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

[25] R. Battiti and G. Tecchiolli, "The reactive tabu search", *ORSA Journal on Computing* 6 (1994), 126-140.

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

[27] T.C. Belding, "The distributed genetic algorithm revisited", *Proceedings of the Sixth International Conference on Genetic Algorithms* (L. Eschelman, ed.), 114–121, Morgan Kaufmann, 1995.

[28] R. Bianchini and C.M. Brown, "Parallel genetic algorithms on distributed-memory architectures", *Transputer Research and Applications* 6 (1993), 67–82.

[29] M. Bolondi and M. Bondanza, *Parallelizzazione di un algoritmo per la risoluzione del problema del commesso viaggiatore*, M.Sc. Dissertation, Dipartimento di Elettronica e Informazione, Politecnico di Milano, 1993.

[30] R.K. Brunner and L.V. Kale, "Adapting to load on workstation clusters", *Proceedings of the Seventh Symposium on the Frontiers of Massively Parallel Computation*, 106–112, IEEE Computer Society Press.

[31] M. Bubak and K. Sowa, "Object-oriented implementation of parallel genetic algorithms", in *High Performance Cluster Computing: Programming and Applications* (R. Buyya, ed.), vol. 2, 331–349, Prentice Hall, 1999.

[32] B. Bullnheimer, G. Kotsis, and C. Strauss, "Parallelization strategies for the ant system", *Applied Optimization* 24 (1998), 87–100.

[33] D. Butenhof, *Programming with POSIX Threads*, Addison Wesley, 1997.

[34] R. Buyya (editor), *High performance cluster computing: Architectures and systems* (vols. 1 and 2), Prentice-Hall, 1999.

[35] E. Cantú-Paz, "A survey of parallel genetic algorithms", *Calculateurs Parallèles, Réseaux et Systèmes Répartis* 10 (1998), 141–171.

[36] E. Cantú-Paz, "Implementing fast and flexible parallel genetic algorithms", in *Practical handbook of genetic algorithms* (L.D. Chambers, ed.), volume III, 65–84, CRC Press, 1999.

[37] E. Cantú-Paz, *Efficient and Accurate Parallel Genetic Algorithms*, Kluwer, 2000.

[38] E. Cantú-Paz, *Migration policies, selection pressure, and parallel evolutionary algorithms*, *Journal of Heuristics*, to appear.

[39] E. Cantú-Paz and D.E. Goldberg, "Predicting speedups of idealized bounding cases of parallel genetic algorithms", *Proceedings of the Seventh International Conference on Genetic Algorithms* (T. Bäck, ed.), 113–121, Morgan Kaufmann, 1997.

[40] E. Cantú-Paz and D.E. Goldberg, "Parallel genetic algorithms: Theory and Practice", *Computer Methods in Applied Mechanics and Engineering* 186 (2000), 221–238.

[41] S.A. Canuto, *Local search for the prize-collecting Steiner tree problem* (in Portuguese), M.Sc. Dissertation, Department of Computer Science, Catholic University of Rio de Janeiro, 2000.

[42] 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*, to appear.

[43] N. Carriero and D. Gelernter, "How to write parallel programs: A guide to the perplexed", *ACM Computing Surveys* 21 (1989), 323–357.

[44] 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 (1987), 727-751.

[45] A. Casotto and A. Sangiovanni-Vincentelli, "Placement of standard cells using simulated annealing on the Connection Machine", *Proceedings of the I987 International Conference on Computer-Aided Design*, 350–353, Santa Clara, 1987.

[46] C.B. Cavalcante, V.C. Cavalcante, C.C. Ribeiro, and C.C. de Souza, "Parallel cooperative approaches for the labor constrained scheduling problem", in *Essays and Surveys in Metaheuristics* (C.C. Ribeiro and P. Hansen, eds.), Kluwer, 2001 (this volume).

[47] C.B. Cavalcante, Y. Colombani, S. Heipcke, and C.C. Souza, "Scheduling under labour resource constraints", *Constraints* 5 (2000), 415–422.

[48] C.B. Cavalcante, C.C. Souza, M.W. Savelsbergh, Y.Wang, and L.A. Wolsey, "Scheduling projects with labor constraints", *Discrete Applied Mathematics*, to appear.

[49] G. Cavalheiro, F. Galilée, and J.-L. Roch, "Athapascan-1: Parallel programming with asynchronous tasks", online document, `http://www-apache.imag.fr/software/ath1/publications/files/98-ath1-yale.ps.gz`, last visited on March 14, 2001.

[50] N. Carriero and D. Gelernter, "Linda in context", *Communications of the ACM* 32 (1989), 444–458.

[51] J. Chakrapani and J. Skorin-Kapov, "Massively parallel tabu search for the quadratic assignment problem", *Annals of Operations Research* 41 (1993), 327–341.

[52] 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 (1993), 29–63.

[53] P. Chalermwat, T. El-Ghazawi, and J. LeMoigne, "2-phase GA-based image registration on parallel clusters", *Future Generation Computer Systems* 17 (2001), 467–476.

[54] J.A. Chandy and P. Banerjee, "Parallel simulated annealing strategies for VLSI cell placement", *Proceedings of the 9th International Conference on VLSI Design*, Bangalore, 1996.

[55] J.A. Chandy, S. Kim, B. Ramkumar, S. Parkes, and P. Banerjee, "An evaluation of parallel simulated annealing strategies with applications to standard cell placement", *IEEE Transactions on Computer Aided Design* 16 (1997), 398–410.

[56] A. Chipperfield and P. Fleming, "Parallel genetic algorithms", *Parallel and Distributed Cmputing Handbook* (A.Y. Zomaya, ed.), 1118–1143, McGraw-Hill, 1996.

[57] A. Colorni, M. Dorigo, and V. Maniezzo, "Distributed optimization by ant colonies", *Proceedings of the European Conference on Artificial Life*, 134–142, Elsevier, 1991.

[58] U.S. da Costa, D.B. D'eharbe, and A.M. Moreira, "Variable orderings of BDDs with parallel genetic algorithms", *Proceedings of the International Conference on Parallel and Distributes Processing Techniques and Applications*, 1181–1186, CSREA Press, 2000.

[59] T.G. Crainic and M. Gendreau, "Cooperative parallel tabu search for capacitated network design", *Journal of Heuristics*, to appear.

[60] T.G. Crainic and M. Toulouse, "Parallel metaheuristics", in *Fleet management and logistics* (T.G. Crainic and G. Laporte, eds.), 205–251, Kluwer, 1998.

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

[62] T.G. Crainic, M. Toulouse, and M. Gendreau, "Synchronous tabu search parallelization strategies for multicommodity location-allocation with balancing requirements", *OR*

Spektrum 17 (1995), 113–123.

[63] T.G. Crainic, M. Toulouse, and M. Gendreau, "Towards a taxonomy of parallel tabu search algorithms", *INFORMS Journal of Computing* 9 (1997), 61–72.

[64] D.E. Culler, P.S. Jaswinder, and A. Gupta, *Parallel computer architecture: A hardware/software aspproach*, Morgan Kaufmann, 1999.

[65] V.-D. Cung, T. Mautor, P. Michelon, and A. Tavares, "A scatter search based approach for the quadratic assignment problem", *Proceedings of the IEEE International Conference on Evolutionary Computation and Evolutionary Programming* (T. Baeck, Z. Michalewicz, and X. Yao, eds.), 165–170, IEEE, 1997.

[66] L. Dagum and R. Menon, "OpenMP: An industry-standard API for shared-memory programming", *IEEE Computational Science and Engineering* 5 (1998), 46–55.

[67] F. Darema, S. Kirkpatrick, and V. Norton, "Parallel algorithms for chip placement by simulated annealing", *IBM Journal of Research and Development*, 31 (1987), 391–402.

[68] I. De Falco, R. Del Balio, A. Della Cioppa, and E. Tarantino, "A parallel genetic algorithm for transonic airfoil", *Proceedings of the IEEE International Conference on Evolutionary Computing*, 429–434, University of Western Australia, 1995.

[69] G. Di Caro and M. Dorigo, "A mobile agents approach to adaptive routing", *Proceedings of the 31st Hawaii International Conference on System*, 74–83, IEEE Computer Society Press, 1998.

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

[71] M. Dorigo, G. Di Caro, and L.M. Gambardella, "Ant algorithms for discrete optimization", *Artificial Life* 5 (1999), 137–172.

[72] C.W. Duin and S. Voss, "Efficient path and vertex exchange in Steiner tree algorithms", *Networks* 29 (1997), 89–105.

[73] M.D. Durand, "Parallel simulated anealing: Accuracy vs. speed in placement" *IEEE Design and Test of Computers* 6 (1989), 8–34.

[74] M.D. Durand and S.R. White, "Trading accuracy for speed in parallel simulated annealing algorithms", *Parallel Computing* 26 (2000), 135–150.

[75] H.T. Eikelder, M. Verhoeven, T. Vossen, and E. Aarts, "A probabilistic analysis of local search", in *Metaheuristics: Theory & applications* (I. Osman and J. Kelly, eds.), 605–618, Kluwer, 1996.

[76] A. Fachat and K. H. Hoffman, "Implementation of ensemble-based simulated annealing with dynamic load balancing under MPI", *Computer Physics Communications* 107 (1997), 49–53.

[77] E. Felten, S.C. Karlin, and S.W. Otto, "The traveling salesman problem on a hypercubic, MIMD computer", *Proceedings of the 1985 International Conference on Parallel Processing*, 6–10, 1985.

[78] T.A. Feo and M.G.C. Resende, "A probabilistic heuristic for a computationally difficult set covering problem", *Operations Research Letters* 8 (1989), 67–71.

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

[80] T.A. Feo, M.G.C. Resende, and S.H. Smith, "A greedy randomized adaptive search procedure for maximum independent set", *Operations Research* 42 (1994), 860–878.

[81] F. Fernandez, J.M. Sanchez, M. Tomassini, and J.A. Gomez, "A parallel genetic programming tool based on PVM", *Lecture Notes in Computer Science* 1697 (1999), 241–248.

[82] P. Festa and M.G. Resende, "GRASP: An annotated bibliography", in *Essays and surveys in metaheuristics* (C.C. Ribeiro and P. Hansen, eds.), Kluwer, 2001 (this volume).

[83] C.N. Fiechter, "A parallel tabu search algorithm for large traveling salesman problems", *Discrete Applied Mathematics* 51 (1994), 243–267.

[84] M.J. Flynn, "Very high-speed computing systems", *Proceedings of the IEEE* 54 (1966), 1901–1909.

[85] I. Foster, *Designing and building parallel programs: Concepts and tools for parallel software engineering*, Addison-Wesley, 1995.

[86] R. Frost, "Ensemble Based Simulated Annealing", online document, `http://www-rohan.sdsu.edu/~frostr/Ebsa/Ebsa.html`, last visited on February 11, 2001.

[87] 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 (1994), 1025–1033.

[88] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images", *IEEE Transactions on Pattern Analysis and Mechanical Intelli-*

gence 6 (1984), 721–741.

[89] M. Gendreau, F. Guertin, J-Y. Potvin, and E. Taillard, "Parallel tabu search for real-time vehicle routing and dispatching", *Transportation Science* 33 (1999), 381–390.

[90] A. Globus, J. Lawton and T. Wipke, "Automatic molecular design using evolutionary techniques", *Nanotechnology* 10 (1999), 290–299.

[91] F. Glover, "Tabu search - Part I", *ORSA Journal on Computing* 1 (1989), 190–206.

[92] F. Glover, "Tabu search - Part II", *ORSA Journal on Computing* 2 (1990), 4–32.

[93] F. Glover, "Ejection chains, reference structures and alternating path methods for traveling salesman problems", *Discrete Applied Mathematics* 65 (1996), 223–253.

[94] F. Glover, "Tabu search and adaptive memory programing - Advances, applications and challenges", in *Interfaces in Computer Science and Operations Research* (R.S. Barr, R.V. Helgason, and J.L. Kennington, eds.), 1–75, Kluwer, 1996.

[95] F. Glover, "Multi-start and strategic oscillation methods - Principles to exploit adaptive memory", in *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research* (M. Laguna and J.L. Gonzáles-Velarde, eds.), 1–24, Kluwer, 2000.

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

[97] F. Glover, M. Laguna, and R. Martí, "Fundamentals of scatter search and path relinking", 2000.

[98] F. Glover and E. Pesch, "TSP ejection chains", *Discrete Applied Mathematics* 76 (1997), 165–181.

[99] V.S. Gordon and D. Whitley, "Serial and parallel genetic algorithms as function optimizers", *Proceedings of the Fifth International Conference on Genetic Algorithms* (S. Forrest, ed.), 177–183, Morgan Kaufmann, 1993.

[100] D.R. Greening, "Parallel simulated annealing techniques", *Physica D* 42 (1990), 293–306.

[101] J.J. Grefenstette, "Parallel adaptive algorithms for function optimizations", Technical report CS-81-19, Vanderbilt University, 1981.

[102] W. Gropp, S. Huss-Lederman, A. Lumsdaine, E. Lusk, B. Nitzberg, W. Saphir, and M. Snir, *MPI: The complete reference, Volume 2 - The MPI extensions*, MIT Press, 1998.

[103] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable parallel programming with the Message Passing-Interface*, MIT Press, 1994.

[104] A. Gueist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM: Parallel Virtual Machine - A user's guide and tutorial for networked parallel computing*, MIT Press, 1994 (http://www.netlib.org/pvm3/book/pvm-book.html).

[105] High Performance FORTRAN Forum, home page at `http://dacnet.rice.edu/Depts/CRPC/HPFF/`.

[106] J.H. Holland, *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*, University of Michigan Press, 1975.

[107] J.H. Holland, "Genetic algorithms", *Scientific American* 267 (1992), 44–50.

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

[109] L. Ingber, "Lester Ingber's Archive", online document, `http://www.ingber.com/`, last visited on February 11, 2001.

[110] Institute of Electric and Electronic Engineering, *Information Technology - Portable Operating Systems Interface (POSIX) - Part 1 - Ammendment 2: Threads Extension*, 1995.

[111] L.V. Kale, M. Bhandarkar, R. Brunner, and J. Yelon, "Multiparadigm, multilingual interoperability: Experience with Converse", *Lecture Notes in Computer Science* 1388 (1998), 111–112.

[112] L.V. Kale, R. Brunner, J. Phillips, and K. Varadarajan, "Application performance of a Linux cluster using Converse", *Lecture Notes in Computer Science* 1586 (1999), 483–495.

[113] L.V. Kale and S. Krishnan, "Charm++: Parallel programming with message-driven objects", in *Parallel Programming using C++* (G.V. Wilson and P. Lu, eds.) 175–213, MIT Press, 1996.

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

[115] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, "Optimisation by simulated annealing", *Science* 220 (1983), 671–680.

[116] G. Kliewer, K. Klohs, and S. Tschöke, "Parallel simulated annealing library (parSA): User manual", Technical report, Computer Science Department, University of Paderborn, 1999.

[117] A. Knies, M. O'Keefe, and T. MacDonald, "High Performance FORTRAN: A practical analysis", *Scientific Programming* 3 (1994), 187–199.
[118] C. Koelbel, D. Loveman, R. Schreiber, G. Steele Jr., and M. Zosel, "The High Performance FORTRAN handbook", The MIT Press, 1994.
[119] U. Kohlmorgen, H. Schmeck, and K. Haase, "Experiences with fine-grained parallel genetic algorithms", *Annals of Operations Research* 90 (1999), 203–220.
[120] S.A. Kravitz and R.A. Rutenbar, "Placement by simulated annealing on a multiprocessor", *IEEE Transactions on Computer Aided Design* 6 (1987), 534–549.
[121] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to parallel computing design and analysis of parallel algorithms*, Benjaming/Cummings, 1994.
[122] S.-Y. Lee and K.G. Lee, "Asynchronous communication of multiple Markov chains in parallel simulated annealing", *Proceedings of the International Conference on Parallel Processing*, volume III, 169–176, St. Charles, 1992.
[123] T.-H. Lai and S. Sahni, "Anomalies in parallel branch-and-bound algorithms", *Communications of the ACM* 27 (1984), 594–602.
[124] , T.-H. Lai and A. Sprague, "A note on anomalies in parallel branch-and-bound algorithms with one-to-one bounding functions", *Information Processing Letters* 23 (1986), 119–122.
[125] K.G. Lee and S.-Y. Lee, "Efficient parallelization of simulated annealing using multiple Markov chains: An application to graph partitioning", *Proceedings of the International Conference on Parallel Processing*, volume III, 177–180, St. Charles, 1992.
[126] 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 (1996), 993–1008.
[127] D. Levine, "A parallel genetic algorithm for the set partitioning problem", Technical report ANL-9423, Argonne National Laboratory, 1994.
[128] B. Lewis, *Multithreaded programming with Java technology*, Prentice Hall, 2000.
[129] Y. Li, P.M. Pardalos, and M.G.C. Resende, "A greedy randomized adaptive search procedure for quadratic assignment problem", *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 16 (1994), 237–261.
[130] G.-J. Li and B.W. Wah, "Coping with anomalies in parallel branch-and-bound algorithms", *IEEE Transactions on Computers* C-35 (1986), 568–573.
[131] B. Mans and C. Roucairol, "Performances of parallel branch and bound algorithms with best-first search", *Discrete Applied Mathematics* 66 (1996), 57–76.
[132] N. Marco and S. Lanteri, "A two level parallelization strategy for genetic algorithms applied to optimum shape design", *Parallel Computing* 26 (2000), 377–397.
[133] S.L. Martins, P.M. Pardalos, M.G.C. Resende, and C.C. Ribeiro, "GRASP procedures for the Steiner problem in graphs", *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 43 (1998), 133–146.
[134] S.L. Martins, M.G.C. Resende, C.C. Ribeiro, and P. Pardalos, "A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy", *Journal of Global Optimization* 17 (2000), 267–283.
[135] S.L. Martins, C.C. Ribeiro, and N.R. Rodriguez, "Parallel computing environments", *Handbook of Combinatorial Optimization*, Oxford, to appear.
[136] S.L. Martins, C.C. Ribeiro, and M.C. Souza, "A parallel GRASP for the Steiner problem in graphs", *Lecture Notes in Computer Science* 1457 (1998), 285–297.
[137] T.G. Mattson, "Scientific computation", in *Parallel and distributed computing handbook* (A.Y. Zomaya, ed.), 981–1002, Mc-Graw-Hill, 1996.
[138] K. Mehlhorn, "A faster approximation for the Steiner problem in graphs", *Information Processing Letters* 27 (1988), 125–128.
[139] J. Merlin, S. Baden, S. Fink, and B. Chapman, "Multiple data parallelism with HPF and KeLP", *Future Generation Computer Systems* 15 (1999), 393–405.
[140] J. Merlin and A. Hey, "An introduction to High Performance FORTRAN", *Scientific Programming* 4 (1995), 87–113.
[141] Z. Michalewicz, *Genetic algorithms + Data structures = Evolution Programs*, Springer-Verlag, 1996.
[142] M. Miki, T. Hiroyasu, and M. Kasai, "Application of the temperature parallel simulated annealing to continous optimization problems", *IPSL Transaction* 41 (2000), 1607–1616.
[143] L.B. Morales, R.Garduño-Juárez, J.M. Aguilar-Alvarado, and F.J. Riveros-Castro, "A parallel tabu search for conformational enegy optimization of oligopeptides", *Journal of Computational Chemistry* 21 (2000), 147–156.
[144] H.S. Morse, *Practical parallel computing*, AP Professional, 1994.
[145] MPI Forum, "MPI: A message-passing interface standard", *The International Journal of*

Supercomputing Applications and High Performance Computing 8 (1994), 159–416.

[146] MPI Forum, "A message-passing interface standard", online document, `http://www.mpi-forum.org/docs/docs.html`, last visited on February 19, 2001.

[147] MPI Forum, "MPI-2: Extensions to the Message-Passing Interface", online document, `http://www.mpi-forum.org/docs/docs.html`, last visited on February 19, 2001.

[148] H. Muhlenbein, "Parallel genetic algorithms in combinatorial optimization", in *Computer Science and Operations Research: New Developments in their Interfaces*(O. Balci, R. Sharda, and S. Zenios, eds.), 441–456, Pergamon Press, 1992.

[149] R.A. Murphey, P.M. Pardalos, and L.S. Pitsoulis, "A parallel GRASP for the data association multidimensional assignment problem", *IMA Volumes in Mathematics and Its Applications* 106 (1998), 159–180.

[150] S. Oak and H. Wong, *Java Threads*, O'Reilly, 1997.

[151] L.S. Ochi, L.M. Drummond, A.O. Victor, and D.S. Vianna, "A parallel evolutionary algorithm for solving the vehicle routing problem with heterogeneous fleet", *Future Generation Computer Systems Journal*, 14 (1998), 285–292.

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

[153] M. Oussaidène, B. Chopard, O.V. Pictel, and M. Tomassini, "Parallel genetic programming and its application to trading model induction", *Parallel Computing* 23 (1997), 1183–1198.

[154] P. Pardalos, L. Pitsoulis, and M.G. Resende, "A parallel GRASP implementation for the quadratic assignment problem", in *Parallel algorithms for irregular problems: State of art* (A. Ferreira and J. Rolim, eds.), 115–133, Kluwer, 1995.

[155] P. Pardalos, L. Pitsoulis, and M.G. Resende, "A parallel GRASP for MAX-SAT", *Lecture Notes in Computer Science* 1184 (1996), 575–585.

[156] S. Parkes, J.A. Chandy, and P. Banerjee, "A library-based approach to portable, parallel, object-oriented programming: Interface, implementation, and application", *Proceedings of the ACM Supercomputing'94 Conference*, 69–78, Washington, 1994.

[157] C.C. Pettey, M.R. Leuze, and J. Grefenstette, "A parallel genetic algorithm", *Proceedings of the Second International Conference on Genetic Algorithms* (J. Grefenstette, ed.), 155–161, Lawrence Erlbraum Associates, 1987.

[158] G.F. Pfister, *In search of clusters*, Prentice-Hall, 1998.

[159] B. Planquelle, J.-F. Méhaut, and N. Revol, "Multi-cluster approach with PM2", *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications*, 779–785, Las Vegas, 1999.

[160] A. Plastino, C.C. Ribeiro, and N.R. Rodriguez, "A framework for SPMD applications", *Proceedings of the XII Brazilian Symposium on Computer Architecture*, 245–252, São Pedro, 2000.

[161] S.C. Porto and C.C. Ribeiro, "Parallel tabu search message-passing synchronous strategies for task scheduling under precedence constraints", *Journal of Heuristics 1* (1995), 207–223.

[162] 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 (1995), 45–71.

[163] S.C. Porto and C.C. Ribeiro, "A case study on parallel synchronous implementations of tabu search based on neighborhood decomposition", *Investigación Operativa* 5 (1996), 233–259.

[164] S.C. Porto, J.P. Kitajima, and C.C. Ribeiro, "Performance evaluation of a parallel tabu search task scheduling algorithm", *Parallel Computing* 26 (2000), 73–90.

[165] M. Prais and C.C. Ribeiro, "Reactive GRASP: An Application to a matrix decomposition problem in traffic assignment", *INFORMS Journal on Computing* 12 (2000), 164–176.

[166] M.J. Quinn, *Parallel computing: Theory and practice*, McGraw-Hill, 1994.

[167] K.H. Randall, *Cilk: Efficient Multithreaded Computing*, Ph.D. Dissertation, MIT, Department of Electrical Engineering and Computer Science, 1998.

[168] C. Rego and C. Roucairol, "A parallel tabu search algorithm using ejection chains for the VRP", in *Metaheuristics: Theory and applications* (I.H. Osman and J.P. Kelly, eds.), 253–295, Kluwer, 1996.

[169] C.R. Reeves, "Genetic algorithms", in *Modern heuristic techniques for combinatorial problems* (C.R. Reeves, ed.), 151–196, Blackwell, 1993.

[170] M.G.C. Resende, "Computing approximate solutions of the maximum covering problem using GRASP", *Journal of Heuristics* 4 (1998), 161–171.

[171] M.G.C. Resende, T.A. Feo, and S.H. Smith, "Algorithm 787: Fortran subroutines for approximate solutions of maximum independent set problems using GRASP", *ACM Transactions on Mathematical Software* 24 (1998), 386–394.

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

[173] M.G.C. Resende, L. Pitsoulis, and P. Pardalos, "Approximate solution of weighted MAX-SAT problems using GRASP", *DIMACS Series on Discrete Mathematics and Theoretical Computer Science* 35 (1997), 393–405.

[174] M.G.C. Resende, L. Pitsoulis, and P. Pardalos, "Fortran subroutines for computing approximate solutions of MAX-SAT problems using GRASP", *Discrete Applied Mathematics* 100 (2000), 95–13.

[175] M.G.C. Resende and C.C. Ribeiro, "A GRASP for graph planarization", *Networks* 29 (1997), 173–189.

[176] C.C. Ribeiro, M. Minoux, and M.C. Penna, "An optimum column-generation-with-ranking algorithm for very large scale set partitioning problems in traffic assignment", *European Journal of Operational Research* 41 (1989), 232-239.

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

[178] W. Rivera-Gallego, "A genetic algorithm for circulant Euclidean distance matrices", *Journal of Applied Mathematics and Computation* 97 (1998), 197–208.

[179] J.S. Rose, W.M. Snelgrove, and Z.G. Vranesic, "Parallel cell placement algorithms with quality equivalent to simulated annealing", *IEEE Transactions on Computer-Aided Design*, 7(1998), 387–396.

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

[181] O. Roux and C. Fonlupt, "Ant programming: Or how to use ants for automatic programming", *Proceedings of the 2nd International Workshop on Ant Colony Optimization*, 121–129, Brussels, 2000.

[182] R.A. Rutenbar and S.A. Kravitz, "Layout by annealing in a parallel environment", *Proceedings of the IEEE International Conference on Computer Design*, 434–437, Port Chester, 1986.

[183] A. Salhi, "Parallel implementation of a genetic-programming based tool for symbolic regression", *Information Processing Letters* 66 (1998), 299–307.

[184] J. Schulze and T. Fahle, "A parallel algorithm for the vehicle routing problem with time window constraints", *Annals of Operations Research* 86 (1999), 585–607.

[185] Scientific Computing Associates, "Linda's user's guide and reference manual, version 4.0.1 - SP2/POE", 1995.

[186] B. Selman, H. Kautz, and B. Cohen, "Noise strategies for improving local search", *Proceedings of the AAAI-94*, 337–343, MIT Press, 1994.

[187] G.A. Sena, D. Megherbi, and G. Isern, "Implementation of a parallel genetic algorithm on a cluster of workstations: Traveling salesman problem, a case study", *Future Generation Computer Systems* 17 (2001), 477–488.

[188] T. Stützle, "Parallelization strategies for ant colony optimization", *Lecture Notes in Computer Science* 1498 (1998), 722–731.

[189] T. Stützle and H. Hoos, "The MAX-MIN ant system and local search for the traveling salesman problem", *Proceedings of the 1997 IEEE 4th International Conference on Evolutionary Computation*, 308–313, IEEE Press, 1997.

[190] Supercomputing Technologies Group, "CILK 5.3.1 Reference Manual", MIT Laboratory for Computer Science, online document, `http://supertech.lcs.mit.edu/cilk`, last visited on March 14, 2001.

[191] E. Taillard, "Robust taboo search for the quadratic assignment problem", *Parallel Computing* 7 (1991), 443–455.

[192] E. Taillard, "Parallel taboo search techniques for the job shop scheduling problem", *ORSA Journal on Computing* 6 (1994), 108–117.

[193] E. Taillard, "Ant systems", in *Handbook of Applied Optimization* (P. Pardalos and M.G.C. Resende, eds.), Oxford, to appear.

[194] E. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J.-Y. Potvin, "A tabu search heuristic for the vehicle routing problem with soft time windows", *Transportation Science* 31 (1997), 170–186.

[195] H. Takahashi and A. Matsuyama, "An approximate solution for the Steiner problem in graphs", *Math. Japonica* 24 (1980), 573–577.

[196] E.-G. Talbi, Z. Hafidi, and J-M. Geib, "A parallel adaptive tabu search approach", *Parallel Computing* 24 (1998), 2003–2019.

[197] E.-G. Talbi, Z. Hafidi, and J-M. Geib, "Parallel adaptive tabu search for large optimization problems", in *Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization* (S. Voss, S. Martello, I.H. Osman, and C. Roucairol, eds.), 255-266, Kluwer, 1999.

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

[199] E.-G. Talbi, O. Roux, C. Fonlupt, and D. Robillard, "Parallel ant colonies for combinatorial optimization problems", *Lecture Notes in Computer Science* 1586 (1999), 239–247.

[200] E.-G. Talbi, O. Roux, C. Fonlupt, and D. Robillard, "Parallel ant colonies for the quadratic assignment problem", *Future Generation Computer Systems* 17 (2001), 441–449.

[201] S.N. Talukdar, L. Baerentzen, and A. Gove, "Asynchronous teams: Cooperation schemes for autonomous agents", *Journal of Heuristics* 4 (1998), 295–321.

[202] R. Tanese, "Distribued genetic algorithms", *Proceedings of the Third International Conference on Genetic Algorithms* (J.D. Schaffer, ed.), 434–439, Morgan Kaufmann, 1989.

[203] M.G.A. Verhoeven and E.H.L. Aarts, "Parallel Local Search", *Journal of Heuristics* 1 (1995), 43–65.

[204] M.G.A. Verhoeven, M.E.M. Severens, and E.H.L. Aarts, "Local search for Steiner trees in graphs", in *Modern Heuristic Search Methods* (V.J. Rayward-Smith, I.H. Osman, C.R. Reeves, and G.D. Smith, eds.), 117–129, Wiley, 1996.

[205] S. Voss, "Tabu search: Applications and prospects", in *Network Optimization Problems* (D.-Z. Du and P.M. Pardalos, eds.), 333–353, World Scientific, 1993.

[206] D.W. Walker, "The design of a standard message passing interface for distributed memory concurrent computers", *Parallel Computing* 20 (1994), 657–673.

[207] E.E. Witte, R.D. Chamberlain, and M.A. Franklin, "Parallel simulated annealing using speculative computation", *IEEE Transactions on Parallel and Distributed Systems* 2 (1991), 483–494.

[208] C.P. Wong and R.D. Fiebrich, "Simulated annealing-based circuit placement algorithm on the Connection Machine system", *Proceedings of the International Conference on Computer Design*, 78–82, Rye Brook, 1987.