Chapter 1

PARALLEL STRATEGIES FOR META-HEURISTICS

Teodor Gabriel Crainic
Département de management et technologie
Université du Québec à Montréal
and
Centre de recherche sur les transports
Université de Montréal
C.P. 6128, Succursale Centre-ville
Montréal (QC) Canada H3C 3J7
Phone: (1) 514 343 7143
FAX: (1) 514 343 7121
theo@crt.umontreal.ca

Michel Toulouse
Department of computer science
University of Manitoba
Winnipeg (MB) Canada R3T 2N2
toulouse@cs.umanitoba.ca

Abstract

We present a state-of-the-art survey of parallel meta-heuristic developments and results, discuss general design and implementation principles that apply to most meta-heuristic classes, instantiate these principles for the three meta-heuristic classes currently most extensively used - genetic methods, simulated annealing, and tabu search, and identify a number of trends and promising research directions.

Key words : Parallel computation, Parallelization strategies, Meta-heuristics, Genetic methods, Simulated annealing, Tabu search, Co-operative search.

1. INTRODUCTION

Meta-heuristics are widely acknowledged as essential tools to address difficult problems in numerous and diverse fields, as this volume eloquently demonstrates. In fact, meta-heuristics often offer the only practical approach to solving complex problems of realistic scale.

Even using meta-heuristics, the limits of what may be solved in "reasonable" computing times are still reached rapidly, however, at least much too rapidly for the growing needs of research and industry alike. Heuristics do not, in general, guaranty optimality. Moreover, the performance often depends on the particular problem setting and data. Consequently, a major issue in meta-heuristic design and calibration is not only how to build them for maximum performance, but also how to make them *robust*, in the sense of offering a consistently high level of performance over a wide variety of problem settings and characteristics.

Parallel meta-heuristics aim to address both issues. Of course, the first goal is to solve larger problem instances in reasonable computing times. In appropriate settings, such as co-operative multi-thread strategies, parallel meta-heuristics also prove to be much more robust than sequential versions in dealing with differences in problem types and characteristics. They also require less extensive, and expensive, parameter calibration efforts.

The objective of this paper is to paint a general picture of the parallel meta-heuristic field. Specifically, the goals are to 1) present a state-of-the-art survey of parallel meta-heuristic developments and results, 2) discuss general design and implementation principles that apply to most meta-heuristic classes, 3) instantiate these principles for the three meta-heuristic classes currently most extensively used: genetic methods, simulated annealing, and tabu search, and 4) identify a number of trends and promising research directions.

The parallel meta-heuristic field is a very broad one, while the space available for this paper imposes hard choices and limits the presentation. In addition to the references provided in the following sections, a number of surveys, taxonomies, and syntheses have been proposed and may prove of interest: Greening (1990), Azencott (1992), Mühlenbein (1992), Shonkwiler (1993), Voß (1993), Lin, Punch, and Goodman (1994), Pardalos *et al.* (1995), Ram, Sreenivas, and Subramaniam (1995), Verhoeven and Aarts (1995), Laursen (1996), Crainic, Toulouse, and Gendreau (1997), Glover and Laguna (1997), Holmqvist, Migdalas, and Pardalos (1997), Cantù-Paz (1998), Crainic and Toulouse (1998), Crainic (2002), Cung *et al.* (2002).

The paper is organized as follows. Section 2. introduces the notation, describes a generic meta-heuristic framework, and sets genetic, simulated annealing, and tabu search methods within this framework. Section 3. is dedicated to a brief introduction to parallel computing and the presentation of three main strategies used to build parallel meta-heuristics. Sections 4., 5., and 6. are dedicated to the survey and discussion of issues related to the parallelization of genetic approaches, simulated annealing, and tabu search, respectively. Section 7. briefly treats a number of other meta-heuristic approaches, draws a number of general conclusions, and points to research directions and challenges.

2. HEURISTICS AND META-HEURISTICS

Sequential and parallel meta-heuristics are used in many disciplines - mathematics, operations research, artificial intelligence - and numerous applications: design, planning, and operation of complex systems and networks (e.g., production, transportation, telecommunication, etc.); management, allocation, scheduling, and utilization of scarce resources; speech and image recognition and enhancement; VLSI design; and so on. To simplify the presentation, and with no loss of generality, in the following we adopt the notation and vocabulary of *combinatorial optimization* formulations.

Given a set of objects, the value associated to each, and the rules specifying how objects may be joined together, the combinatorial optimization formulation aims to select a subset of objects such that the sum of their contributions is the highest/lowest among all possible combinations. Many problems of interest may be cast as combinatorial optimization formulations, including design, location, routing, and scheduling. In most cases, such formulations are extremely difficult to solve for realistically-sized problem instances, the main issue being the number of feasible solutions - combinations of objects - that grows exponentially with the number of objects in the initial set.

Combinatorial optimization problems are usually formulated as (mixed) integer optimization programs. To define notation, assume that one desires to minimize (or maximize) a function $f(x)$ subject to $x \in \mathcal{X} \subseteq \mathbb{R}^n$. The objective function $f(x)$ may be linear or not. The set $\mathcal{X}$ summarizes constraints on the *decision variables* x and defines the *feasible domain*. Decision variables are generally non-negative and all or part of the elements of x may be compelled to take discrete values. One seeks a globally *optimal solution* $x^* \in \mathcal{X}$ such that $f(x^*) \leq f(x)$ for all $x \in \mathcal{X}$.

Once various methods have been applied to re-formulate the problem and to bound the region where the optimal solution is to be found, most solution methods are based on some form of exploration of the set of feasible solutions. Explicit enumeration is normally out of the question and the search for the optimal solution proceeds by implicit enumeration. Branch-and-bound (and price, and cut, ...) methods are both typical of such approaches and one of the methods of choice used in the search for optimal solutions to combinatorial problems. Unfortunately, these methods fail for many instances, even when parallel implementations are used. Thus, heuristics have been, and continue to be, an essential component of the methodology used to address combinatorial optimization formulations.

A *heuristic* is any procedure that identifies a feasible solution $\tilde{x} \in \mathcal{X}$. Of course, one would like $\tilde{x}$ to be identical to x^* (if the latter is unique) or $f(\tilde{x})$ to be equal to $f(x^*)$. For most heuristics, however, one can only hope (and for some, prove) that $f(\tilde{x})$ is "close" to $f(x^*)$. Heuristics have a long and distinguished track record in combinatorial optimization. Often, heuristics are the only practical alternative when dealing with problem instances of realistic dimensions and characteristics.

Many heuristics are *improving* iterative procedures that *move* from a given solution to a solution in its *neighbourhood* that is better in terms of the objective function value (or some other measure based on the solution characteristics). Thus, at each iteration, such a *local search* procedure identifies and evaluates solutions in the neighbourhood of the current solution, selects the best one relative to given criteria, and implements the transformations required to establish the selected solution as the current one. The procedure iterates until no further improvement is possible.

Formally, let $\mathcal{N} \subseteq \mathcal{X}$ represent the set of neighbours of a given solution x that may be reached by a simple transformation (e.g., complement the value of an integer-valued variable) or a given sequence of operations (e.g., λ-opt modifications of routes in a vehicle routing problems). Let $m(x)$ denote the application that corresponds to these moves and that yields a solution $y \in \mathcal{N}(s)$. Then, Figure 1.1 illustrates a simple steepest descent heuristic where the objective function value is the only neighbour evaluation criterion.

```
1. Identify an initial solution $x^0 \in \mathcal{X}$; $k = 0$;
2. $k = k + 1$;
3. Find $\tilde{x} = argmin\{f(x)|x \in \mathcal{N}(x^k)\}$;
4. If $f(\tilde{x}) \geq f(x^k)$ Stop.
5. Otherwise, $x^{k+1} = m(\tilde{x})$; Goto 2.
```

Figure 1.1 Simple Local Search/Steepest Descent Heuristic

A major drawback of classical heuristic schemes is their inability to continue past the first encountered local optimum. Moreover, such procedures are unable to react and adapt to particular problem instances. Re-starting and randomization strategies, as well as combinations of simple heuristics offer only partial and largely unsatisfactory answers to these issues. The class of modern heuristics known as *meta-heuristics* aims to address these challenges.

Meta-heuristics have been defined as master strategies (heuristics) to guide and modify other heuristics to produce solutions beyond those normally identified by local search heuristics (Glover 1986; see also Glover and Laguna 1993). Compared to exact search methods, such as branch-and-bound, meta-heuristics cannot generally ensure a systematic exploration of the entire solution space. Instead, they attempt to examine only parts thereof where, according to certain criteria, one believes good solutions may be found. Well-designed meta-heuristics avoid getting trapped in local optima or sequences of visited solutions (*cycling*) and provide reasonable assurance that the search has not overlooked promising regions.

Meta-heuristics for optimization problems may be described summarily as a "walk through neighbourhoods", a search trajectory through the solution domain of the problem at hand. Similar to classical heuristics, these are iterative procedures that *move* from a given solution to another solution in its *neighbourhood*. Thus, at each iteration, one evaluates moves towards solutions in the neighbourhood of the current solution, or in a suitably selected subset. According to various criteria (objective value, feasibility, statistical measures, etc.), a number of good moves are selected and implemented. Unlike classical heuristics, the solutions implemented by meta-heuristics are not necessarily improving, however. Tabu search and simulated annealing methods usually implement one move at each iteration, while genetic methods may generate several new moves (individuals) at each iteration (generation). Moves may belong to only one type (e.g., add an element to the solution) or to several quite different categories (e.g., evaluate both add and drop moves). Moves may marginally modify the solution or drastically inflect the search trajectory. The first case is often referred to as *local search*. The diversification phase of tabu search or the application of mutation operators in an evolutionary process are typical examples of the second alternative. This last case may also be described as a change in the "active" neighbourhood.

Each meta-heuristic has its own behaviour and characteristics. All, however, share a number of fundamental components and perform operations that fall within a limited number of categories. To facilitate the comparison of parallelization strategies for various meta-heuristic classes, it is convenient to define these common elements:

1. *Initialization.* A method to create an initial solution or set of problem configurations that may be feasible or not.

2. *Neighbourhoods.* To each solution x corresponds a set of neighbourhoods and associated moves: $\{\mathcal{N}_1, \mathcal{N}_2, \dots, \mathcal{N}_q\}$, where $\mathcal{N}_i(x) = \{y = m_i(x), y \in \mathcal{X}\}, i = 1, ..., q$.

3. A *neighbourhood selection criterion* is defined when more than one neighbourhood is included. This criterion must specify not only what

neighbourhood to choose but also when to select it. Alternatives range from “each iteration” (e.g., genetic methods) to “under given conditions” (e.g., the diversification moves of tabu search).

4. *Candidate selection.* Neighbourhoods may be very large. Then, often, only a subset of moves are examined at each iteration. The corresponding *candidate list* $\mathcal{C}(x) \subseteq \mathcal{N}(x)$ may be permanent and updated from iteration to iteration (e.g., tabu search) or it may be constructed at each new iteration (e.g., genetic methods). In all cases, a selection criterion specifies how solutions are picked for inclusion in the candidate list.

5. *Acceptance criterion.* Moves are evaluated by applying a function $g(x, y)$ based on one or several attributes of the two solutions: objective function value, distance from certain constraints, penalties for violating some others, etc. External factors, such as random terms or biases from aggregated characteristics of past solutions may also be included in the evaluation. The best solution with respect to this criterion

   $$\tilde{x} = argopt\{g(x, y);\ y \in \mathcal{C}(x)\}$$

   is selected and implemented (unless forbidden by cycling-prevention mechanisms).

6. *Stopping criteria*. Meta-heuristics may be stopped on a variety of criteria: computing time, number of iterations, rate of improvement, etc. More than one criterion may be defined to control various phases of the search (usually corresponding to various neighbourhoods).

With these definitions, we introduce a generic meta-heuristic procedure illustrated in Figure 1.2 and use it to describe the three main classes of meta-heuristics: *genetic methods*, *simulated annealing*, and *tabu search*. These methodologies have been, and continue to be, most oftenly used and parallelized. They are therefore treated in more detail in the following sections. Other methods, such as *scatter search*, GRASP, *ant colony systems*, and *variable neighbourhood search* have also been proposed and we briefly discuss related parallelization issues in Section 7..

Genetic algorithms belong to the larger class of evolutionary methods and were inspired by the evolution processes of biological organisms. In biology, when natural populations are studied over many generations, they appear to *evolve* according to the principles of *natural selection* and *survival of the fittest* to produce “well adapted” individuals. Genetic algorithms mimic this process, attempting to *evolve* solutions to optimization problems (Holland 1975; Goldberg 1989; Whitley 1994; Fogel 1994; Michalewicz 1992; Michalewicz and Fogel 2000). In recent years, the genetic algorithm paradigm was considerably enriched, as it evolved to include hybridization with local improvement

```
1. Initialization: $x^0$
2. Neighbourhood selection $\mathcal{N} \in \{\mathcal{N}_1, \dots, \mathcal{N}_q\}$
3. Candidate selection $\mathcal{C}(x) \subseteq \mathcal{N}(x)$
4. Move evaluation/neighbourhood exploration $g(x, y),\ y \in \mathcal{C}(x)$
5. Move implementation $\tilde{x} = argopt\{g(x, y)\}$
6. Solution evaluation, update search parameters
7. Test stopping criteria: Stop or Goto 3 (continue local search phase)
   or Goto 2 (initiate new search phase)
```

Figure 1.2 Generic Meta-heuristic

heuristics and other meta-heuristics. Although the specialized literature is frequently replacing the term "genetic algorithms" with "evolutionary methods", we use the former in this paper to distinguish these methods from other strategies where a population of solutions evolves through an iterative process (e.g., scatter search and most co-operative parallel methods described later in this paper).

Genetic methods work on a population of solutions that evolves by generating new individuals out of combinations of existing individuals. At each iteration a *selection* operator applied to the current population identifies the parents to be used for the generation of new individuals. Thus, in a genetic context, the candidate selection and move evaluation is based solely on the value $g(x)$, the *fitness*, of the parents (this may be contrasted to the evaluations used in, for example, simulated annealing, tabu search, and scatter search). *Crossover* operators are used to generate the new individuals. *Mutation* and *hill climbing* (local search) operators modify the definition or characteristics of the new individuals to improve their fitness and the diversity of the population. In several variants, especially so for parallel genetic methods, the implementation of the move is completed by a *survival* operation that determines which of the parents and offspring advances to the next generation. Figure 1.3 displays the functions of the classical genetic operators, while Figure 1.4 summarizes the main steps of a generic genetic algorithm.

Simulated annealing methods are inspired by the *annealing* process of cooling materials in a heat bath. Here, solid materials are first heated past melting point. They are then gradually cooled back to a solid state, the *rate of cooling* directly influencing on the structural properties of the final product. The materials may be represented as systems of particles and the whole process

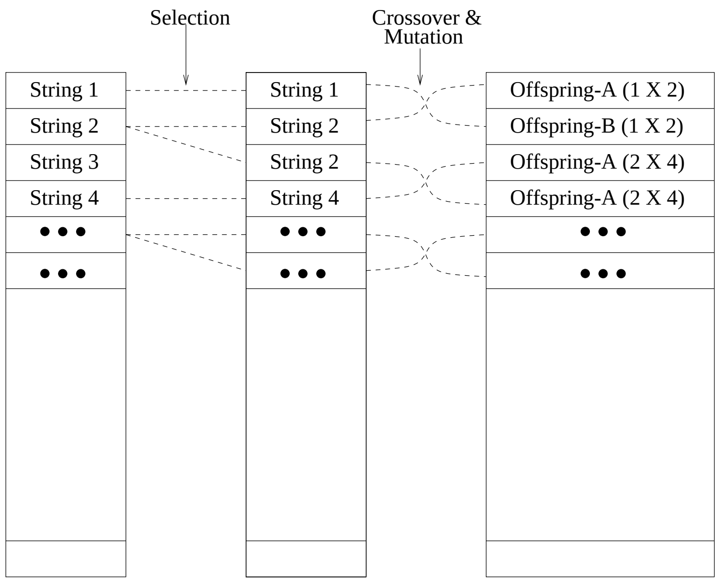

Figure 1.3 Genetic Evolutionary Algorithm Operators

1. Initialization: generation of the initial population.
2. Neighbourhood selection: selection of crossover and mutation operators.
3. Candidate - *parent* - selection: application of a selection operator to the current population
4. Move evaluation/neighbourhood exploration: none
5. Move implementation: application of crossover, mutation, hill climbing, and offspring and parent selection operators to obtain a new population
6. If stopping criteria not met, Goto 3 (continue the evolution) or Goto 1.3 (modify the evolution criteria)

Figure 1.4 A Generic Genetic Algorithm

of heating and cooling may be simulated to evaluate various cooling rates and their impact on the properties of the finished product. The *simulated anneal-*

ing metaphor attempts to use similar statistical processes to guide the search through feasible space (Metropolis *et al.* 1953; Kirkpatrick, Gelatt, and Vecchi 1983; Laarhoven and Aarts 1989; Aarts and Korst 1989, 2002; etc.). A randomized scheme, the *temperature* control, determines the probability of accepting non-improving solutions. This mechanism aims to allow escaping from local optima. The *cooling schedule* determines how this probability evolves: many non-improving solutions are accepted initially (*high temperature*) but the temperature is gradually reduced such that few (none) inferior solutions are accepted towards the end (*low temperature*). A generic simulated annealing procedure is displayed in Figure 1.5.

```
1. Initialization: Select
 ■ Initial state (solution) $x = x_0$;
 ■ Initial temperature $\tau = \tau_0$;
 ■ Temperature reduction function $\alpha$;
2. Neighbourhood and candidate selections: none (generally);
   Replaced by
3. Selection of number of iterations $L$ to approximate equilibrium
   at temperature $\tau$
4. Move evaluation/neighbourhood exploration:
   Randomly select $y \in \mathcal{X}$.
5. Move implementation
     $\Delta f := f(y) - f(x)$;
     **if** $\Delta f \leq 0$ **then** $x := y$;
     **else if** $g(x,y) = exp\,(-\Delta f/\tau) > random(0,1)$ **then** $x := y$;
6. Solution evaluation
7. Test stopping criteria
 ■ If less than $L$ iterations, Goto 4
 ■ If convergence not verified, $\tau = \alpha(\tau)$; Goto 3
```

Figure 1.5 Generic Simulated Annealing Procedure

One of the most appropriate tabu search metaphors is the capability of the human brain to store, recall, and process information to guide and enhance the efficiency of repetitive processes. *Memory* and *memory hierarchy* are major concepts in tabu search, as are the memory-based strategies used to guide the procedure into various search phases. Here, as elsewhere, a heuristic locally explores the domain by moving from one solution to the best available solution in its neighbourhood. Inferior quality solutions are accepted as a strategy to move away from local optima. Short-term *tabu status* memories record recent visited solutions or their attributes to avoid repeating or inverting recent actions. The tabu status of a move may be lifted if testing it against an *aspiration criterion* signals the discovery of a high quality solution (typically, the best one encountered so far). Medium to long-term memory structures record various informations and statistics relative to the solutions already encountered (e.g., frequency of certain attributes in the best solutions) to "learn" about the solution space and guide the search. *Intensification* of the search around a good solution and its *diversification* towards regions of the solution space not yet explored are two main ingredients in tabu search. These two types of moves are based on medium and long-term memories and are implemented using specific neighbourhoods. More details on the basic and advanced features of tabu search may be found in Glover (1986, 1989, 1990, 1996), Glover and Laguna (1993, 1997), Gendreau (2002). Figure 1.6 displays a generic tabu search procedure.

1. Initialization: x_0
2. Neighbourhood selection: local search, intensification, diversification, ...
3. Candidate selection $\mathcal{C}(x) \subseteq \mathcal{N}(x)$
4. Move evaluation/neighbourhood exploration: tabu criteria, aspiration criterion
5. Move implementation
6. Update of memories and tabu status
7. Test stopping criteria; If it fails Goto 3 (continue local search) or Goto 2 (change search phase)

Figure 1.6 A Generic Tabu Search

This very brief summary of three major meta-heuristics emphasizes the similarities of the main activities used by the various methodologies to explore the solution space of given problems. This similarity translates into "similar" requirements when strategies for parallelization are contemplated. For example, all meta-heuristic procedures encompass a rather computationally heavy stage where the neighbourhood (or the population) is explored. Fortunately, the computational burden may be reduced by performing the exploration in parallel and most implementations of the first parallelization strategy discussed in the following section address this issue. This observation explains our choice of discussing parallelization strategies not according to particular meta-heuristic characteristics but rather following a few general principles.

3. PARALLEL COMPUTATION

The central goal of parallel computing is to speed up computation by dividing the work load among several processors. From the view point of algorithm design, "pure" parallel computing strategies exploit the partial order of algorithms (i.e., the sets of operations that may be executed concurrently in time without modifying the solution method and the final solution obtained) and thus correspond to the "natural" parallelism present in the algorithm. The partial order of algorithms provides two main sources of parallelism: *data* and *functional* parallelism.

To illustrate, consider the multiplication of two matrices. To perform this operation, one must perform several identical operations executing sums of products of numbers. It is possible to overlap the execution of these identical operations on different input data. Among computer architectures with several arithmetic and logic units (ALUs), *Single Instruction stream, Multiple Data stream* (*SIMD*) computers are particularly suited to this type of parallelism as they can load the same operation on all ALUs (single flow of instructions) and execute it on different input data (multiple flows of data). The total number of computer operations required to compute the matrix product is not reduced, but given the concurrency of several operations, the total wall-clock computation time is reduced proportionally to the average number of overlapping sets of operations during the computation. This is data parallelism.

Computations may also be overlapped even when operations are different. It is usually inefficient to exploit this parallelism at the fine-grain level of a single instruction. Rather, the concurrent execution of different operations typically occurs at the coarse-grain level of procedures or functions. This is functional parallelism. For example, one process can compute the first derivative vector of a function while another computes the second derivative matrix. The two processes can overlap at least partially in time. When computations are complex and dimensions are large, this partial overlap may yield interesting speedups. Parallel computers that are well adapted to perform functional parallelism usually follow a *MIMD* (*Multiple Instructions stream, Multiple Data stream*) architecture where both data and instructions flow concurrently in the system. MIMD computers are often made up of somewhat loosely connected processors, each containing an ALU and a memory module.

Parallel computation based on data or functional parallelism is particularly efficient when algorithms manipulate data structures that are strongly regular, such as matrices in matrix multiplications. Algorithms operating on irregular data structures, such as graphs, or on data with strong dependencies among the different operations remain difficult to parallelize efficiently using only data and functional parallelism. Meta-heuristics generally belong to this category of algorithms that are difficult to parallelize. Yet, as we will see, parallelizing meta-heuristics offers opportunities to find new ways to use parallel computers and to design parallel algorithms.

3.1 PARALLELIZING META-HEURISTICS

From a computational point of view, meta-heuristics are just algorithms from which we can extract functional or data parallelism. Unfortunately, data and functional parallelism are in short supply for many meta-heuristics. For example, the local search loop (Steps 3 to 7) of the generic tabu search in Figure 1.6 displays strong data dependencies between successive iterations, particularly in the application of the tabu criterion and the update of memories and tabu status. Similarly, the passage from one generation to another in standard genetic methods is essentially a sequential process, while the replacement of the current solution of the generic simulated annealing procedure (Step 5 in Figure 1.5) cannot be done in parallel, forcing the sequential execution of the inner loop (Steps 4 to 6). As in other types of algorithms, however, operations inside one step may offer some functional or data parallelism. Moreover, the exploration of the solution space based on random restarts can be functionally parallelized since there are no dependencies between successive runs. The set of visited solutions, as well as the outcome of the search made up of random restarts are identical to those obtained by the sequential procedure provided the set of initial solutions is the same for both the sequential and parallel runs.

Meta-heuristics as algorithms may have limited data or functional parallelism but, as problem solving methods, they offer other opportunities for parallel computing. To illustrate, consider the well-known Branch-and-Bound technique. The branching heuristic is one of the main factors affecting the way Branch-and-Bound algorithms explore the search tree. Two Branch-and-Bound algorithms, each using a different branching heuristic, will most likely perform different explorations of the search tree of one problem instance. Yet, both will find the optimum solution. Thus, the utilization of different Branch-and-Bound search patterns does not prevent the technique from finding the optimal solution. This critical observation may be used to construct parallel Branch-and-bound methods. For example, the parallel exploration of the search tree based on distributing sub-trees will modify the data available to the branching heuristic and thus, for the same problem instance, the parallel and sequential search patterns will differ. Yet, the different search strategies will all find the optimal solution. Consequently, the exploration of sub-trees may be used as a source of parallelism for Branch-and-Bound algorithms. This source of parallelism is not related to data parallelism, since the data (the variables of the optimization problem) is not partitioned. It is not functional parallelism either, because the two computations, sequential and parallel, are different. Although this difference makes comparative performance analyzes more difficult to perform (since the parallel implementation does not do the same work as the sequential one), sub-tree distribution remains a valuable and widely used parallelization strategy for Branch-and-Bound algorithms.

Similar observations can be made relative to new sources of parallelism in meta-heuristics. A meta-heuristic algorithm started from different initial solutions will almost certainly explore different regions of the solution space and return different solutions. The different regions of the solution space explored can then become a source of parallelism for meta-heuristic methods. However, the analysis of parallel implementation of meta-heuristic methods becomes more complex because often the parallel implementation does not return the same solution as the sequential implementation. Evaluation criteria based on the notion of *solution quality* (i.e., does the method find a better solution?) have then to be used to qualify the more classical acceleration (speedup) measures.

We have classified the parallelization strategies applied to meta-heuristics according to the source of parallelism used:

***Type 1*:** This source of parallelism is usually found within an iteration of the heuristic method. The limited functional or data parallelism of a move evaluation is exploited or moves are evaluated in parallel. This strategy, also called *low-level* parallelism, is rather straightforward and aims solely to speed up computations, without any attempt at achieving a better exploration (except when the same total wall-clock time required by the sequential method is allowed to the parallel process) or higher quality solutions.

***Type 2*:** This approach obtains parallelism by partitioning the set of decision variables. The partitioning reduces the size of the solution space, but it needs to be repeated to allow the exploration of the complete solution space. Obviously, the set of visited solutions using this parallel implementation is different from that of the sequential implementation of the same heuristic method.

***Type 3*:** Parallelism is obtained from multiple concurrent explorations of the solution space.

Type 1 parallelism. Type 1 parallelizations may be obtained by the concurrent execution of the operations or the concurrent evaluation of several moves making up an iteration of a search method. Type 1 parallelization strategies aim directly to reduce the execution time of a given solution method. When the same number of iterations are allowed for both sequential and parallel versions of the method and the same operations are performed at each iteration (e.g., the same set of candidate moves is evaluated and the same selection criterion is used), the parallel implementation follows the same exploration path through the problem domain as the sequential implementation and yields the same solution. As a result, standard parallel performance measures apply straightforwardly. To illustrate, consider the computation of the average fitness of a population for genetic methods. Because the sequence used to compute the fitness of the individuals is irrelevant to the final average fitness of the population, it can be partitioned and the partial sums of each subpopulation can be computed in parallel. Both the parallel and sequential computations yield the same average fitness, the parallel implementation just runs faster.

Some implementations modify the sequential method to take advantage of the extra computing power available, but without altering the basic search method. For example, one may evaluate in parallel several moves in the neighbourhood of the current solution instead of only one. In Figure 1.5, one can choose several y variables in Step 4 and then perform Step 5 in parallel for each selected y. In tabu search, one may *probe* for a few moves beyond each immediate neighbour to increase the available knowledge when selecting the best move (Step 4 of Figure 1.6). The resulting search patterns of the serial and parallel implementations are different in most cases. Yet, under certain conditions, the fundamental algorithmic design is not altered, therefore these approaches still qualify as Type 1 parallelism.

Type 2 parallelism. In Type 2 strategies, parallelism comes from the decomposition of the decision variables into disjoint subsets. The particular heuristic is applied to each subset and the variables outside the subset are considered fixed. Type 2 strategies are generally implemented in some sort of master-slave framework:

- A *master* process partitions the decision variables. During the search, the master may modify the partition. Modifications may be performed at intervals that are either fixed before or determined during the execution, or, quite often, are adjusted when restarting the method.

- *Slaves* concurrently and independently explore their assigned partitions. Moves may proceed exclusively within the partition, the other variables being considered fixed and unaffected by the moves which are performed, or the slaves may have access to the entire set of variables.

- When slaves have access to the entire neighbourhood, the master must perform a more complex operation of combining the partial solutions obtained from each subset to form a complete solution to the problem.

Note that a decomposition based on partitioning the decision variables may leave large portions of the solution space unexplored. Therefore, in most applications, the partitioning is repeated to create different segments of the decision variable vector and the search is restarted.

Type 3 parallelism. The first two parallelization strategies yield a single search path. Parallelization approaches that consist of several concurrent searches in the solution space are classified as Type 3 strategies. Each concurrent thread may or may not execute the same heuristic method. They may start from the same or different initial solutions and may communicate during the search or only at the end to identify the best overall solution. The latter are known as *independent search* methods, while the former are often called *co-operative multi-thread* strategies. Communications may be performed synchronously or asynchronously and may be event-driven or executed at predetermined or dynamically decided moments. These strategies belong to the *p-control* class according to the taxonomy proposed by Crainic, Toulouse, and Gendreau (1997), and are identified as *multiple-walks* by Verhoeven and Aarts (1995).

To speed up computation by using a multi-thread strategy, one generally tries to make each thread perform a shorter search than the sequential procedure. This technique is implemented differently for each class of meta-heuristic. Let p be the number of processors. For tabu search, each thread performs $\frac{T}{p}$ iterations, where T is the number of iterations of the corresponding sequential procedure. For simulated annealing, the total number of iterations of the inner loop (Steps 4 to 6 in Figure 1.5) is reduced proportionally from L to $\frac{L}{p}$. For genetic algorithms, it is not the number of generations which is generally reduced. Rather, the size N of the sequential population is reduced to $\frac{N}{p}$ for each genetic thread.

Type 3 parallelization strategies are often used to perform a more thorough exploration of the solution space. Several studies have shown that multi-thread procedures yield better solutions than the corresponding sequential meta-heuristics, even when the exploration time permitted to each thread is significantly lower than that of the sequential computation. Studies have also shown that the combination of several threads that implement different parameter settings increases the robustness of the global search relative to variations in problem instance characteristics. We review some of these results in Sections 4. to 6..

It is noteworthy that the application of classical performance measures (e.g., Barr and Hickman 1993) to multi-thread, parallel meta-heuristics is somewhat problematic. For example, it is generally difficult to eliminate or control the overlap between the search paths (to adequately control the search overlap would involve such high levels of search synchronization and information exchanges that all benefits of parallelization would be lost). Thus, one cannot measure correctly the search efficiency in terms of the work performed.

Moreover, many Type 3 parallelizations are based on asynchronous interactions among threads. As asynchronous computations are time dependent, such computations can produce different outputs for the same input. Classical speedup measures are ill-defined to compare the performances of asynchronous parallel meta-heuristics with sequential ones. In fact, several asynchronous Type 3 parallel meta-heuristics are so different from the original sequential procedure that one can hardly consider the two implementations to belong to the same meta-heuristic class.

3.2 OTHER TAXONOMIES

The classification described above is sufficiently general to apply to almost any meta-heuristic and parallelization strategy. Moreover, the lessons learned by comparing within the same class the implementations and performances of particular meta-heuristics are of general value and may be extended to search methods not covered in depth in this paper. Most taxonomies proposed in the literature are, however, related to a specific type of meta-heuristic.

Greening (1990) divides simulated annealing parallelization strategies according to the degree of accuracy in the evaluation of the cost function associated with a move. Parallel algorithms that provide an error-free evaluation are identified as *synchronous* while the others are *asynchronous*. The synchronous category is divided further between parallel simulated annealing algorithms that maintain the convergence properties of the sequential method (*serial-like*) and those that have an accurate cost evaluation but differ from the sequential computation in the search path generation (*altered generation*). Type 1 parallelism completely covers the serial-like category. The altered generation category overlaps with the Type 1 and Type 2 strategies. Asynchronous algorithms are those that tolerate some error in the cost function in order to get better speedups and correspond to a subset of the Type 2 category. No Type 3 algorithms are considered in this work.

Cantù-Paz (1995) provides a classification of parallel genetic algorithms. The first category, called *global* parallelization is identical to the Type 1 parallelization. Two other categories classify genetic algorithms according to the size of the populations that evolve in parallel, the so-called *coarse-grained* and *fine-grained* parallelization strategies. There is also a class for *hybrid* genetic algorithm parallelizations. For example, global parallelization applied to subpopulations of a coarse-grained parallel algorithm is one instance of an hybrid algorithm. The union of these three groups forms the Type 3 category described in this paper. No Type 2 strategies are considered in Cantù-Paz's taxonomy.

Verhoeven and Aarts (1995) define local search as the class of approximation methods based on the exploration of neighbourhoods of solutions, including tabu search, simulated annealing, and genetic algorithms. Their taxonomy divides parallel methods between *single-walk* and *multiple-walk* strategies. The former corresponds to Type 1 and Type 2 parallelism. The latter includes *multiple independent walks* and *multiple interacting walks* and thus corresponds to the Type 3 parallelism of this paper. Single-walk methods are further classified as *single-step* or *multiple-step* strategies. The former corresponds to the simple parallel neighbourhood evaluation of Type 1. The latter includes probing and Type 2 strategies. The taxonomy explicitly distinguishes between synchronous and asynchronous approaches. Cung *et al.* (2002) present a classification of parallel meta-heuristic strategies based on that of Verhoeven and Aarts (1995).

Currently, the most comprehensive taxonomy of parallel tabu search methods is offered by Crainic, Toulouse, and Gendreau (1997) and Crainic (2002). The classification has three dimensions. The first dimension, *control cardinality*, explicitly examines how the global search is controlled, by a single process (as in master-slave implementations) or collegially by several processes. The four classes of the second dimension indicate the *search differentiation*: do search threads start from the same or different solutions and do they make use of the same or different search strategies? Finally, the *control type* dimension addresses the issue of information exchange and divides methods into four classes: *rigid synchronization* (e.g., simple neighbourhood evaluation in Type 1 strategies), *knowledge synchronization* (e.g., probing in Type 1 strategies and synchronous information exchanges in Type 3 methods), and, finally, *collegial* and *knowledge collegial*. The last two categories correspond to Type 3 strategies but attempt to differentiate methods according to the quantity and quality of information exchanged, created, and shared. Although introduced for tabu search, this classification applies to many other meta-heuristics and may form the basis for a comprehensive taxonomy of meta-heuristics. It refines, in particular, the classification used in this paper, which is based on the impact of parallelization on the search trajectory.

4. GENETIC ALGORITHMS

At each iteration k, genetic algorithms compute the average fitness $\bar{g}_k = \sum_{i=1}^{i=N} g(x_i^k)/N$ of the N strings in the current population (Step 3 in Figure 1.4). The time-consuming part of this operation is performing the summation $\sum_{i=1}^{i=N} g(x_i^k)$. Obviously, this computation can be distributed over several processors and, since there are no dependencies among operations, it is a good candidate for efficient data, Type 1 parallelization. This summation has indeed been the first component of genetic algorithms to be parallelized (Grefenstette 1981).

Intuitively, one expects almost linear speedups from this parallelization of the average fitness evaluation. Surprisingly, however, most experiments report significant sub-linear speedups due to the latencies of low-speed communication networks (Fogarty and Huang 1990; Hauser and Männer 1994; Chen, Nakao, and Fang 1996; Abramson and Abela 1992; Abramson, Mills, and Perkins 1993). The implementations of the selection and crossover operators are also based on simple iterative loops, but each involves relatively few computations. Therefore, considering the impact of the communication overhead on a time-consuming operation like the fitness evaluation, Type 1 parallelization of the genetic operators has received little attention.

Genetic algorithms are acknowledged to be inherently parallel. This inherent parallelism is limited to Type 1, however. We are not aware of any Type 2 parallelization strategy for genetic algorithms and, although many known parallel genetic algorithms are of Type 3, most of them are not strictly derived from the standard genetic paradigm. Standard genetic methods are based on single panmictic population, and computation is usually initiated on a new generation only after the old one has died out, thus preventing the occurrence of parallelism across generations. Parallel genetic models, on the other hand, find more opportunities for parallelism such as, concurrent computations across different generations or among different subpopulations. The literature on parallel genetic methods often identifies two categories of Type 3 approaches: *coarse-grained* and *fine-grained*. However, some Type 3 parallel genetic algorithms do not display such clear cut characterization (e.g., Moscato and Norman 1992).

Coarse-grained parallelizations usually refer to methods where the same sequential genetic algorithm is run on p subpopulations (each of size $\frac{N}{p}$), although some researchers (e.g., Schlierkamp-Voosen and Mühlenbein 1994; Herdy 1992) have pondered the possibility of using different strategies for each subpopulation. In such models, each subpopulation is relatively small in comparison with the initial population. This has an adverse impact on the diversity of the genetic material, leading to premature convergence of the genetic process associated to each subpopulation. To favor a more diversified genetic material in each subpopulation, a new genetic operator, the *migration* operator, is provided.

The migration operator defines strategies to exchange individuals among subpopulations. This operator has several parameters: the selection process that determines which individuals will migrate (e.g., best-fit, randomly, randomly among better than average individuals), the migration rate that specifies the number of strings migrated, the migration interval that determines when migration may take place (usually defined in terms of a number of generations), and the immigration policy that indicates how individuals are replaced in the receiving subpopulation. Information exchanges are further determined by the neighbourhood structure. In the *island* model, individuals may migrate towards any other subpopulation, while in the *stepping-stone* model only direct neighbours are reachable. Often the connection structure of the parallel computer determines how subpopulations are logically linked. Finally, migration may be performed either synchronously or asynchronously.

Since a genetic algorithm is associated with each subpopulation, coarse-grained parallel strategies can exploit parallelism across generations, provided each generation is related to a different subpopulation. And, it is always possible to combine Type 1 and Type 3 parallelism by computing the average fitness within each subpopulation using the Type 1 strategy as described above. MIMD computers are well adapted to coarse-grained parallel genetic methods. Migration rate and frequency are such that, in general, the quantity of data exchanged is small and can be handled efficiently even by low-speed interconnection networks. Furthermore, since the workload of each processor is significant (running a sequential genetic algorithm), latencies due to communications (if any) can be hidden by computations in asynchronous implementations. Therefore, linear speedups could be expected. Yet, few reports detail the speedups of coarse-grained parallel genetic algorithms. To some extent, this is explained by the fact that speedups do not tell the whole story regarding the performance of coarse-grained parallelizations, since one also needs to consider the associated convergence behavior. Therefore, to this day, most efforts regarding coarse-grained parallelizations have been focused on studying the best migration parameter settings. Most studies conclude that migration is better than no migration, but that the degree of migration needs to be controlled.

Schnecke and Vornberger (1996) analyze the convergence behaviour of coarse-grained parallel genetic algorithms using a Type 3 strategy where a different genetic algorithm is assigned to each subpopulation and search strategies, rather than solutions, are migrated between subpopulations. At fixed intervals, the different genetic methods are ranked (using the *response to selection* method of Mühlenbein and Schlierkamp-Voosen 1994) and the search strategies are adjusted according to the “best” one by importing some of the “best” one’s characteristics (mutation rate, crossover rate, etc). The paper contains references to several other works where self-adapting parallel genetic evolution strategies are analyzed. Lis (1996), in particular, applies self-adaptation to the mutation rate. The author implements a farming model where a master processor manages the overall population and sends the same set of best individuals to slave processors, each of which has a different mutation probability. Periodically, according to the mutation rate of the process that obtained the best results, the mutation rates of all slave processors are shifted one level up or down and populations are recreated by the master processor using the best individuals of the slave processors. Starkweather, Whitley, and Mathias (1991; see also Whitley and Starkweather 1990a,b) also suggest that an adaptive mutation rate might help achieve better convergence for coarse-grained parallel genetic algorithms.

A more conceptually focused approach to improve the convergence of coarse-grained parallel genetic strategies may be derived from co-evolutionary genetic algorithm ideas. Schlierkamp-Voosen and Mühlenbein (1994), for example, use competing subpopulations as a means to adapt the parameters controlling the genetic algorithm associated with each subpopulation. In the general setting of co-evolutionary genetic methods, each subpopulation may have a different optimization function that either competes with other subpopulations (as in a prey-predator or host-parasite relationship, Hillis 1992) or may co-operate with the other subpopulations by specializing on subproblems that are later combined to yield the full solution (Potter and De Jong 1994). In the competitive scheme proposed by Hillis, a population of solutions competes with a population of evolving problems (test cases). Fitness and selection pressure favors individuals that make life difficult in the other population. For example, fit individuals in the population of solutions are those that can solve many test cases, while fit test cases are those that only few individuals in the solution population can solve correctly. In the co-operative scheme, the selection pressure favors individuals that co-operate well to solve the global problem. The co-evolutionary setting provides the concept of complementary sub-problems as a way to improve the convergence of coarse-grained parallel genetic algorithms. To this day, however, this avenue has not been widely explored.

Fine-grained strategies for parallel genetic algorithms divide the population into a large number of small subsets. Ideally, subsets are of cardinality one, each individual being assigned to a processor. Each subset is connected to several others in its neighbourhood. Together, a subset and its neighbouring subsets form a subpopulation (or *deme*). Genetic operators are applied using asynchronous exchanges between individuals in the same deme only. Demes may be defined according to a fixed topology (consisting of individuals residing in particular processors), obtained by a random walk (applying a given Hamming distance among individuals), etc. Neighbourhoods overlap to allow propagation of individuals or individual characteristics and mutations across subpopulations. This overlapping plays a similar role to that of the migration operator for coarse-grained parallel genetic algorithms. Fine-grained parallel genetic algorithms are sometimes identified as *cellular algorithms*, because a fine-grained method with fixed topology deme and relative fitness policy may be shown to be equivalent to finite cellular automata with probabilistic rewrite rules and an alphabet equal to the set of strings in the search space (see Whitley 1993).

Fine-grained parallel genetic algorithms evolve a single population that spawns over several generations. This enables parallelism across generations. A "generation gap" (signaled by different iteration counts) tends to emerge in the population because the selection and crossover operators in one deme are not synchronized with the other demes. In effect, it is still a single population, due to the overlap among demes. Yet, the global dynamics of fine-grained parallel genetic algorithms are quite different from those of general genetic methods.

In a single panmictic population, individuals are selected based on a global average fitness value and the selected individuals have the same probability of interacting with each other through the crossover operator. In fine-grained parallel strategies, average fitness values (or whatever stand for them) are local to demes. Consequently, individuals in the population do not have the same probability to mate and the genetic information can only propagate by diffusion through overlapping demes.

Diffusion is channeled by the overlapping structure of the demes, which is often modeled on the interconnection network of the parallel computer. Consequently, network topology is an important issue for fine-grained parallelization strategies, because the diameter of the network (the maximum shortest path between two nodes) determines how long it takes for good solutions to propagate over all of the demes. Long diameters isolate individuals, giving them little chance of combining with other good individuals. Short diameters prevent genotypes (solution vectors) from evolving, since good solutions rapidly dominate, which leads to premature convergence. Individual fitness values are relative to their deme and thus individuals on processing units not directly connected may have no chance to be involved together in the same crossover operator. Schwehm (1992) implemented a fine-grained parallel genetic algorithm on a massively parallel computer to investigate which network topology is best-suited to fine-grained parallel genetic algorithms. Compared with a ring and three cubes of various dimensions, a torus yielded the best results. Baluja (1993) conducted studies regarding the capability of different topologies to prevent demes of fine-grained parallel genetic algorithms to be dominated by the genotype of strong individuals. Three different topologies were studied and numerical results suggest that 2D arrays are best suited to fine-grained parallel genetic algorithms. See also the recent work of Kohlmorgen, Schmeck, and Haase (1999).

Fine-grained parallel genetic methods have been hybridized with hill-climbing strategies. Mühlenbein, Gorges-Schleuter, and Krämer (1987, 1988), among others, have designed hybrid strategies for several optimization problems and obtained good performance. Memetic algorithms (e.g., Moscato 1989, Moscato and Norman 1992) belong to the same category. Hybrid schemes construct selection and crossover operators in a similar manner to regular fine-grained parallelizations but a hill-climbing heuristic is applied to each individual. When the computational cost of the hill-climbing heuristic (or any other heuristic) is substantial (in Memetic algorithms, for example), the population size has to be small and the computing units powerful. Such hybrids appear to be closer to coarse-grained parallelism, except that the selection and crossover operators are those usually associated with fine-grained parallelization mechanisms. Interesting comments about fine-grained parallel genetic strategies, their design and application as well as the role of hill-climbing heuristics, can be found in Mühlenbein (1991, 1992, 1992a).

Research on parallel genetic algorithms is still active and prolific. Unlike other meta-heuristics, parallelizations of Type 1, that exploit the inherent parallelism of standard genetic methods, are still quite competitive in terms of performance, degree of parallelism, adaptation to current parallel computer architectures, and ease of implementation. In terms of research directions, innovation in algorithmic design, and capacity for hybridization with other search methods, Type 3 parallelism is currently the most active area. However, good models to compare the performance of different Type 3 parallel strategies for genetic algorithms are still missing.

5. SIMULATED ANNEALING

A simulated annealing iteration consists of four main steps (Steps 4 to 6 in Figure 1.5): select a move, evaluate the cost function, accept or reject the move, update (replace) the current solution if the move is accepted. Two main approaches are used to obtain Type 1 parallel simulated annealing algorithms: *single-trial* parallelism where only one move is computed in parallel, and *multiple-trial* strategies where several moves are evaluated simultaneously.

The evaluation of the cost function for certain applications may be quite computationally intensive, thus suggesting the possible exploitation of functional parallelism. Single-trial strategies exploit functional parallelism by decomposing the evaluation of the cost function into smaller problems that are assigned to different processors. Single-trial strategies do not alter the algorithmic design nor the convergence properties of the sequential simulated annealing method. The resulting degree of parallelism is very limited, however, and thus single-trial parallelization strategies do not speedup computation significantly.

Multiple-trial parallelizations distribute the iterations making up the search among different processors. Each processor fully performs the four steps of each iteration mentioned above. This distribution does not raise particular issues relative to the first three steps since these tasks are essentially independent with respect to different potential moves. Solution replacement is, however, a fundamentally sequential operation. Consequently, the concurrent execution of several replacement steps may yield erroneous evaluations of the cost function because these evaluations could be based on outdated data.

Type 1 multiple-trial strategies for simulated annealing enforce the condition that parallel trials always result in an *error-free* cost function evaluation. This may be achieved when solution updating is restricted to a single accepted move or to moves that do not interact with each other. The latter approach, referred to as the *serializable subset* method, accepts only a subset of moves that always produces the same result when applied to the current state of the system, independent of the order of implementation (a trivial serializable subset contains only rejected moves). To implement the former approach, one processor is designated to be the holder of the current solution. Once a processor accepts a move, it sends the new solution to the holder, which then broadcasts it to all processors. Any new move accepted during the update of the current solution is rejected. Performance varies with the temperature parameter. At high temperatures, when many potential moves are accepted, communications, synchronization, and rejection of moves generate substantial overheads. At low temperatures, fewer moves are accepted and speedups improve. Yet, performance is not very satisfactory in most cases.

Most multiple-trial parallelization strategies for simulated annealing follow a Type 2 approach and partition the variables into subsets. A master-slave approach is generally used, as illustrated in Figure 1.7. To initiate the search, the master processor partitions the decision variables into p initial sets. The appropriate set is sent to each processor together with the initial values for the temperature τ_0 and the number of iterations L_0 to be executed. In Step 2, each slave processor i executes the simulated annealing search at temperature τ_k on its set of variables $\mathcal{N}_l^k$ and sends its partial configuration of the entire solution to the master processor. Once the information from all slaves has been received, the master processor merges the partial solutions into a complete solution and verifies the stopping criterion. If the search continues, it generates a new partition of the variables such that $\mathcal{N}_l^k \neq \mathcal{N}_l^{k-1}$ and sends it to the slave processors together with new values for L_k and τ_k.

```
1. Initialization (Master: Processor 0):
   (a) $x^0; \mathcal{N}_0^0, \mathcal{N}_0^1, \dots, \mathcal{N}_0^{p-1}; L_0 = L_0/p; \tau_0; k = 0;$
   (b) Send $\mathcal{N}_0^0, \mathcal{N}_0^1, \dots, \mathcal{N}_0^{p-1}$, $\tau_0$, $L_0$, $k$ to the slave processors
2. Slave processor $i$ (concurrent processing):
   (a) Perform search: Steps 4 to 6 of Figure 1.5 for $L_k$ iterations
   (b) Send partial configuration $x_i^k$ to processor 0
3. Master processor 0:
   (a) Solution update: Combine partial solutions
   (b) $L_k = L_k/p; \tau_k; \mathcal{N}_k^0, \mathcal{N}_k^1, \dots, \mathcal{N}_k^{p-1}$ such that
       $\mathcal{N}_k^l \neq \mathcal{N}_{k-1}^l,\ l = 0, 1, \dots, p-1;$
   (c) If stopping criterion not true, send $\mathcal{N}_k^0, \mathcal{N}_k^1, \dots, \mathcal{N}_k^{p-1}$,
       $\tau_k$, and $L_k$, $k$ to the slave processors
```

Figure 1.7 Type 2 Parallel Simulated Annealing

Felten, Karlin, and Otto (1985) applied this strategy to a 64-city *travelling salesman problem (TSP)* using up to 64 processors of a hypercube computer. An initial tour was randomly generated and partitioned into p subsets of adjacent cities, which were assigned to p processors. Each processor performed local swaps on adjacent cities for a given number of iterations, followed by a synchronization phase where cities were rotated among processors. Parallel moves did not interact due to the spatial decomposition of the decision variables. Moreover, each synchronization ensured the integrity of the global state. Hence, there was no error and almost linear speedups were observed.

In most cases, however, error-free strategies cannot be efficiently implemented. It is often difficult, for example, to partition variables such that parallel moves do not interact. Therefore, the two main issues in Type 2 parallel simulated annealing are “how important is the error?” and “how can errors be controlled?”.

Algorithms executed on shared-memory systems can regularly and quite efficiently update the global state of the current solution so that errors do not accumulate during the computation. However, the issue is significantly more difficult for distributed memory systems because each processor has its own copy of the data, including the “current” solution, and global updates are costly in communication time. Trade-offs, therefore, must be made between the frequency of the global state updates and the level of error one is ready to tolerate during the parallel simulated annealing computation, while acknowledging the possibility that significant errors might accumulate locally. It has been observed, however, that errors tend to decrease as temperatures decrease, because solution updates occur less frequently at low temperatures. Jayaraman and Darema (1988) and Durand (1989) specifically address the issue of error tolerance for parallel simulated annealing. As expected, they conclude that the error increases as the frequency of synchronizations decreases and the number of processors increases. In their studies, the combined error due to synchronization and parallelism had a significant impact on the convergence of the simulated annealing algorithm. Of the two factors, parallelism emerged as the most important.

One of the reasons for partitioning variables among processors is to prevent the same variable from being simultaneously involved in more than one move. This goal can also be achieved by *locking* the variables involved in a move. A locking mechanism permits only the processor that owns the lock to update a given variable. Any other processor that attempts to execute a move involving a locked variable must either wait for the variable to become available or attempt a different move. However, the use of locks results in a communication overhead which increases with the number of processors (e.g., Darema, Kirkpatrick, and Norton 1987).

Rather than having several processors execute moves from the same current solution or subset of decision variables, processors could work independently using different initial solutions and cooling schedules, or simply using the probabilistic nature of simulated annealing to obtain different search paths. This is Type 3 parallelism. Numerous efforts to develop Type 3 parallel simulated annealing strategies using independent or co-operative search threads are reported in the literature. An interesting recent development involves the systematic inclusion of principles and operators from genetic algorithms in simulated annealing multi-thread procedures. This hybrid approach tends to perform very well.

The first Type 3 parallelization strategy to emerge was the *division strategy* proposed by Aarts *et al.* (1986). Let L be the number of iterations executed by a sequential simulated annealing program before reaching equilibrium at temperature τ. The division strategy executes L/p iterations on p processors at temperature τ. Here, a single initial solution and cooling strategy is used and it is assumed that the search paths will not be the same due to the different probabilistic choices made by each processor. At the L/p-th iteration, processors can either synchronize and choose one of the solutions as the initial configuration for the next temperature, or continue from their last configurations at the preceding temperature level. When synchronization is used, the procedure corresponds to a synchronous co-operative scheme with global exchange of information (otherwise, it is equivalent to an independent search approach). Unfortunately, the length of the chain can not be reduced arbitrarily without significantly affecting the convergence properties of the method. This is particularly true at low temperatures, where many steps are required to reach equilibrium. To address this problem, the authors clustered the processors at low temperatures and applied multi-trial parallelism of Type 1 within each cluster. Kliewer and Tschöke (2000) have addressed practical issues such as the proper length of parallel chains and the best time to cluster processors.

An alternative to the division strategy is to run each processor with its own cooling schedule in an independent search framework. The Multiple Independent Runs (MIR, Lee 1995) and the Multiple Markov Chains (MMC, Lee and Lee 1996) schemes are Type 3 parallelizations based on this approach. When there are no interactions among processors, performance is negatively affected by idle processors which are waiting for the longest search path to terminate. The MIR strategy addresses this problem by calculating estimates of the total run length, and then using these estimates to end computation on all processing units. The MMC scheme addresses the same issue by allowing processes to interact synchronously and asynchronously at fixed or dynamic intervals. The authors of this co-operating multi-thread strategy observe that communication overheads from co-operation are largely compensated for by the reduction of processor idle time.

A different Type 3 initiative to increase the degree of parallelism of simulated annealing algorithms consists of moving the methodology closer to genetic algorithms by considering a population of simulating annealing threads. Laursen (1994) proposed such a population scheme based on the selection and migration operators of parallel genetic algorithms. Each processor concurrently runs k simulated annealing procedures for a given number of iterations. Processors are then paired and each processor migrates (copies) its solutions to its paired processor. Thus, after the migration phase, each processor has $2k$ initial solutions and this number is reduced to k by selection. These new k solutions become the initial configurations of the k concurrent simulated annealing threads, and the search restarts on each processor. Pairing is dynamic and depends on the topology of the parallel machine. For example, in a grid topology, processors can pair with any of their corner neighbours. Because processors are dynamically paired and neighbourhoods overlap, information propagates in the network of processors similar to the stepping-stone coarse-grained model for parallel genetic methods. Mahfoud and Goldberg (1995) also propose to evaluate concurrently a population of n Markov chains. The general idea proceeds as follows: after $n/2$ iterations, two parents are selected from the population of the n current solutions. Two children are generated using a genetic crossover operator, followed by a mutation operator. Probabilistic trial competitions are held between children and parents and the replacement step is performed according to the outcome of the competition. The temperature is lowered when the population reaches equilibrium. There are different ways to parallelize this algorithm. The asynchronous parallelization described in Mahfoud and Goldberg (1995) follows the Type 3 coarse-grained parallel genetic algorithm approach. The population of n Markov chains is divided into p subpopulations of n/p Markov chains. Crossover, mutation, and probability trials are applied to individuals of each local subpopulation. Asynchronous migration enables sharing of individuals among subpopulations.

The literature on parallel simulated annealing methods has continued to flourish in recent years. Recent research focuses on applying parallel simulated annealing to new problems and developing software packages, rather than on discovering new parallelization strategies. The relatively poor performance of Type 1 and Type 2 approaches have been noticed and, consequently, there have been few applications of these strategies. The most actively applied parallelization strategies are of Type 3: hybridation with hill-climbing (e.g., Du *et al.* 1999) or with genetic methods (e.g., Kurbel, Schneider, and Singh 1995); co-operative multi-threads (e.g., Chu, Deng, and Reinitz 1999); and massive parallelism (e.g., Mahfoud and Goldberg 1995, Bhandarkar *et al.* 1996). We believe methods of Type 3 will continue to offer the best performance for parallel simulated annealing.

6. TABU SEARCH

Tabu search has proved a fertile ground for innovation and experimentation in the area of parallel meta-heuristics. Most parallel strategies introduced in Section 3. have been applied to tabu search for a variety of applications, and a number of interesting parallelization concepts have been introduced while developing parallel tabu search methods.

Similar to most other meta-heuristics, low-level, Type 1 parallelism has been the first strategy to be applied to tabu search methods. The usual target in this case is the acceleration of the neighbourhood exploration (Step 4 of Figure 1.6). Following the ideas summarized in Section 3., most Type 1 implementations correspond to a master process that executes a sequential tabu procedure and dispatches, at each iteration, the possible moves in the neighbourhood of the current solution to be evaluated in parallel by slave processes. Slaves may either evaluate only the moves in the set they receive from the master process, or may probe beyond each move in the set. The master receives and processes the information resulting from the slave operations and then selects and implements the next move. The master also gathers all the information generated during the tabu exploration, updates the memories, and decides whether to activate different search strategies or stop the search.

The success of Type 1 strategies for tabu search appears more significant than for genetic or simulated annealing methods. Indeed, very interesting results have been obtained when neighbourhoods are very large and the time to evaluate and perform a given move is relatively small, such as in *quadratic assignment* (*QAP*: Chakrapani and Skorin-Kapov 1992, 1993, 1995; Taillard (1991, 1993a), *travelling salesman* (Chakrapani and Skorin-Kapov 1993a) and *vehicle routing* (*VRP*: Garcia, Potvin, and Rousseau 1994) applications. For the same quality of solution, near-linear speedups are reported using a relatively small number of processors. Moreover, historically (the first half of the 90's), Type 1 parallel tabu search strategies permitted improvements to the best-known solutions to several problem instances proposed in the literature.

Similarly to the other meta-heuristics, Type 1 tabu search implementations depend heavily upon the problem characteristics. Thus, performance results are less interesting when the time required by one serial iteration is relatively important compared to the total solution time, resulting in executions with only a few hundred moves compared to the tens of thousands required by a typical VRP tabu search procedure. This was illustrated by the comparative study of several synchronous tabu search parallelization strategies performed by Crainic, Toulouse, and Gendreau (1995a) for the location-allocation problem with balancing requirements. With respect to Type 1 parallelization approaches, two variants were implemented: 1) slaves evaluate candidate moves only; 2) *probing*: slaves also perform a few local search iterations. The second variant performed marginally better. However, both variants were outperformed by co-operative multi-thread (Type 3) implementations, which attempt a more thorough exploration of the solution space.

Typical tabu search implementations of Type 2 parallelization strategies partition the vector of decision variables and perform a search on each subset. This approach was part of the preliminary experimentation in the study by Crainic, Toulouse, and Gendreau (1995a). It performed poorly, mainly because of the nature of the class of problems considered; multicommodity location with balancing requirements requires a significant computation effort to evaluate and implement moves, resulting in a limited number of moves that may be performed during the search.

As with Type 1 implementations, Type 2 parallel methods were more successful for problems for which numerous iterations may be performed in a relatively short time and restarting the method with several different partitions does not require unreasonable computational efforts. TSP and VRP formulations belong to this class of applications. Fiechter (1994) proposed a method for the TSP that includes an intensification phase during which each process optimizes a specific slice of the tour. At the end of the intensification phase, processes synchronize to recombine the tour and modify (shift part of the tour to a predetermined neighbouring process) the partition. To diversify, each process determines from among its subset of cities a candidate list of most promising moves. The processes then synchronize to exchange these lists, so that all build the same final candidate list and apply the same moves. Fiechter reports near-optimal solutions to large problems (500, 3000 and 10000 vertices) and almost linear speedups (less so for the 10000 vertex problems). Porto and Ribeiro (1995, 1996) studied the task scheduling problem for heterogeneous systems and proposed several synchronous parallel tabu search procedures where a master process determines and modifies partitions, synchronizes slaves, and communicates best solutions. Interesting results were reported, even for strategies involving a high level of communications. Almost linear speedups were observed, better performances being observed for larger problem instances.

Taillard (1993) studied parallel tabu search methods for vehicle routing problems. In Taillard's approach, the domain is decomposed into polar regions, to which vehicles are allocated, and each subproblem is solved by an independent tabu search. All processors synchronize after a certain number of iterations (according to the total number of iterations already performed) and the partition is modified: tours, undelivered cities, and empty vehicles are exchanged between adjacent processors. Taillard reports very good results for the epoch. However, enjoying the benefit of hindsight, the main contribution of this paper is to mark the evolution towards one of the most successful sequential meta-heuristics for the VRP: a tabu search method called *adaptive memory* (Rochat and Taillard 1995; Glover 1996).

According to an adaptive memory approach, cities are initially separated into several subsets, and routes are built using a construction heuristic. Initial routes are then stored in a structure called an *adaptive memory*. Then, a combination procedure builds a complete solution using the routes in the memory, and the solution is further improved using a tabu search method. The routes of "good" solutions are then deposited into the same memory, which thus adapts to reflect the current state of knowledge of the search. The process then restarts with a new solution built from the routes stored in the adaptive memory. The method stops when a pre-specified number of calls to the adaptive memory have been performed. This approach clearly implements the principles of Type 2 decomposition using a serial procedure; See also the interesting developments in vocabulary building strategies for tabu search proposed by Glover (1996). Adaptive memory principles have now been successfully applied to other problem classes and are opening interesting research avenues (Glover, 1996). However, interestingly, most parallel applications of this approach are now found in co-operative multi-thread strategies (Type 3).

Type 3 parallelizations for tabu search methods follow the same basic pattern described in Section 3.: p threads search through the same solution space, starting from possibly different initial solutions and using possibly different tabu (or other) search strategies. Historically, independent and synchronous co-operative multi-thread methods were proposed first. However, currently, asynchronous procedures are being increasingly developed. Consequently, one observes an increased awareness of the issues related to the definition and modelling of co-operation.

Battiti and Tecchiolli (1992, for the QAP) and Taillard (the main study is found in his 1994 paper on parallel tabu methods for job shop scheduling problems) studied independent multi-thread parallelization schemes, where the independent search processes start the exploration from different, randomly generated, initial configurations. Both studies empirically established the efficiency of independent multi-thread procedures when compared to the best heuristics proposed at the time for their respective problems. Both studies also attempted to establish some theoretical justifications for the efficiency of independent search. Battiti and Tecchiolli derived probability formulas that tended to show that the probability of "success" increases, while the corresponding average time to "success" decreases, with the number of processors (provided the tabu procedure does not cycle). On the other hand, Taillard showed that the conditions required for the parallel method to be "better" than the sequential one are rather strong, where "better" was defined as "the probability the parallel algorithm achieves success with respect to some condition (in terms of optimality or near-optimality) by time t is higher than the corresponding probability of the sequential algorithm by time pt". However, the author also mentions that, in many cases, the empirical probability function of iterative algorithms is not very different from an exponential one, implying that independent multi-thread parallelization is an efficient strategy. The results for the job shop problem seemed to justify this claim. Similar results may also be found in Eikelder *et al.* (1999).

Malek *et al.* (1989, for the TSP), De Falco *et al.* (1994, for the QAP), and De Falco, Del Balio, and Tarantino (1995, for the mapping problem) proposed co-operative parallel strategies where the individual search threads are rather tightly synchronized. The implementation proposed by Malek *et al.* (1989) proceeds with one main process that controls the co-operation, and four child processes that run serial tabu search algorithms with different tabu conditions and parameters. The child processes are stopped after a specified time interval, solutions are compared, bad areas of solution space are eliminated, and the searches are restarted with a good solution and an empty tabu list. This implementation was part of a comparative study of serial and parallel simulated annealing and tabu search algorithms for the TSP. The authors report that the parallel tabu search implementation outperformed the serial one and consistently produced comparable or better results than sequential or parallel simulated annealing. De Falco and colleagues implemented a multi-thread strategy, where each process performs a local search from its best solution. Then, processes synchronize and best solutions are exchanged between processes that run on neighbouring processors. Local best solutions are replaced with imported ones only if the latter are better. The authors indicate that better solutions were obtained when co-operation was included compared to an independent thread strategy. Super-linear speedups are reported.

Rego and Roucairol (1996) proposed a tabu search method for the VRP based on ejection chains and implemented an independent multi-thread parallel version, each thread using a different set of parameter settings but starting from the same solution. The method is implemented in a master-slave setting, where each slave executes a complete sequential tabu search. The master gathers the solutions found by the threads, selects the overall best, and reinitializes the threads for a new search. Low-level (Type 1) parallelism accelerates the move evaluations of the individual searches, as well as the post-optimization phase. Experiments show the method to be competitive on the standard VRP problem set (Christofides, Mingozzi, and Toth 1979).

Asynchronous co-operative multi-thread search methods are being proposed in continuously increasing numbers. All such developments we have identified use some form of *central memory* for inter-thread communications. Each individual search thread starts from a different initial solution and generally follows a different search strategy. Exchanges are performed asynchronously and are done through the central memory. O ne may classify co-operative multi-thread search methods according to the type of information stored in the central memory: complete or partial solutions. In the latter case, one often refers to *adaptive memory* strategies, while *central memory*, *pool of solutions*, or *solution warehouse* methods are used for the former.

Very successful Type 3 co-operative multi-thread parallel tabu search methods are based on *adaptive memory* concepts. This strategy has been particularly used for real-time routing and vehicle dispatching problems (Gendreau *et al.* 1999), as well as for VRP with time window restrictions (Taillard *et al.* 1997; Badeau *et al.* 1997). A general implementation framework of adaptive memory strategies begins with each thread constructing an initial solution and improving it through a tabu search or any other procedure. Each thread deposits the routes of its improved solution into the adaptive memory. Each thread then constructs a new initial solution out of the routes in the adaptive memory, improves it, communicates its routes to the adaptive memory, and repeats the process. A "central" process manages the adaptive memory and oversees communication among the independent threads. It also stops the procedure based on the number of calls to the adaptive memory, the number of successive calls which show no improvement in the best solution, or a time limit. In an interesting development, Gendreau *et al.* (1999) also exploited parallelism within each search thread by decomposing the set of routes along the same principles proposed in Taillard's work (1993). Good results have been obtained by using this approach on a network of workstations, especially when the number of processors is increased. Another interesting variant on the adaptive memory idea may be found in the work of Schulze and Fahle (1999). Here, the pool of partial solutions is distributed among processes to eliminate the need for a "master". The elements of the best solutions found by each thread are broadcast to ensure that each search has still access to all the information when building new solutions. Implemented on a limited number of processors, the method performed well (it is doubtful, however, that it would perform equally well for a larger number of processors).

As far as we can tell, Crainic, Toulouse, and Gendreau (1997) proposed the first central memory strategy for tabu search as part of their taxonomy. The authors also presented a thorough comparison of various parallelization strategies based on this taxonomy (Crainic, Toulouse, and Gendreau 1995a and 1995b). The authors implemented several Type 1 and 2 strategies, one independent multi-thread approach, and a number of synchronous and asynchronous co-operative multi-thread methods. They used the multicommodity location problem with balancing requirements for experimentation. The authors report that the parallel versions achieved better quality solutions than the sequential ones and that, in general, asynchronous methods outperformed synchronous strategies. The independent threads and the asynchronous co-operative approaches offered the best performance.

Crainic and Gendreau (2001) proposed a co-operative multi-thread parallel tabu search for the fixed cost, capacitated, multicommodity network design problem. In their study, the individual tabu search threads differed in their initial solution and parameter settings. Communications were performed asynchronously through a central memory device. The authors compared five strategies of retrieving a solution from the pool when requested by an individual thread. The strategy that always returns the overall best solution displayed the best performance when few (4) processors were used. When the number of processors was increased, a probabilistic procedure, based on the rank of the solution in the pool, appears to offer the best performance. The parallel procedure improves the quality of the solution and also requires less (wall clock) computing time compared to the sequential version, particularly for large problems with many commodities (results for problems with up to 700 design arcs and 400 commodities are reported). The experimental results also emphasize the need for the individual threads to proceed unhindered for some time (e.g., until the first diversification move) before initiating exchanges of solutions. This ensures that local search histories can be established and good solutions can be found to establish the central memory as an *elite candidate* set. By contrast, early and frequent communications yielded a totally random search that was ineffective. The authors finally report that the co-operative multi-thread procedure also outperformed an independent search strategy that used the same search parameters and started from the same initial points. Other implementations of asynchronous co-operative multi-thread parallel tabu search methods are presented by Andreatta and Ribeiro (1994; see also Aiex *et al.* 1996 and Martins, Ribeiro, and Rodriguez, 1996) for the problem of partitioning integrated circuits for logical testing as well as by Cavalcante *et al.* (2002) for labor scheduling problems.

Crainic and Gendreau (1999) report the development of a hybrid search strategy combining their co-operative multi-thread parallel tabu search method with a genetic engine. The genetic algorithm initiates its population with the first elements from the central memory of the parallel tabu search. Asynchronous migration (migration rate = 1) subsequently transfers the best solution of the genetic pool to the parallel tabu search central memory, as well as solutions of the central memory towards the genetic population. The hybrid appears to perform well, especially on larger problems where the best known solutions are improved. It is noteworthy that the genetic algorithm alone was not performing well and that it was the parallel tabu search procedure that identified the best results once the genetic method contributed to the quality of the central memory.

Recently, Le Bouthiller and Crainic (2001) took this approach one step further and proposed a central memory parallel meta-heuristic for the VRP with time windows where several tabu search and genetic algorithm threads co-operate. In this model, the central memory constitutes the population common to all genetic threads. Each genetic algorithm has its own parent selection and crossover operators. The offspring are returned to the pool to be enhanced by a tabu search procedure. The tabu search threads follow the same rules as in the work of Crainic and Gendreau (2001). Only preliminary results are currently available, but they are extremely encouraging. Without any particular calibration, the parallel meta-heuristic obtains solutions whose quality is comparable to the best meta-heuristics available, and demonstrates almost linear speedups.

To conclude, Type 1 (and in some cases Type 2) parallelization strategies may still prove of value, especially for the evaluation of large neighbourhoods, or when used in hierarchical implementations to speedup computations of meta-heuristics involved in co-operative explorations. As illustrated above, Type 3, co-operative multi-thread strategies offer the most interesting perspectives. They do require, however, some care when they are designed and set up as will be discussed in the next section.

7. PERSPECTIVES AND RESEARCH DIRECTIONS

We have presented the main meta-heuristic parallelization strategies and their instantiation in the context of three major classes of methods: genetic algorithms, simulated annealing, and tabu search. Beyond the peculiarities specific to each methodology class and application domain, a number of general principles emerge:

- Meta-heuristics often have strong data dependencies. Therefore, straightforward data or functional parallelization techniques can identify only limited parallelism.
- Nevertheless, parallelization is very often beneficial. The evaluation of neighbouring solutions is a prime example of meta-heuristic algorithmic components that permit significant computational gains. Moreover, the concurrent exploration of the solution space by co-operating meta-heuristics often yields gains in solution quality, computational efficiency, and robustness of the search.
- Type 1 parallelization techniques are particularly useful for computation-intensive tasks, such as the evaluation of potential moves in the neighbourhood of a given solution. Moreover, such strategies may be advantageously incorporated into hierarchical parallel schemes where the higher level either explores partitions of the solution domain (Type 2 parallelism) or implements a co-operating multi-thread search (Type 3 parallelism).
- Hybridization, the incorporation of principles and strategies proper to one class of meta-heuristics into the algorithmic design of another, may improve performance of sequential and parallel meta-heuristics alike.
- When implemented properly, co-operating multi-thread parallel meta-heuristics appear to be the most promising strategy.

We have focussed on genetic methods, simulated annealing, and tabu search to reflect their wide-spread utilization in both sequential and parallel settings. Several other meta-heuristics have also been proposed in the literature, and some have proven quite successful for certain problem types. They include *scatter search* (Glover 1994; Glover and Laguna 1997), GRASP (Feo and Resende 1995; Festa and Resende 2002), *variable neighbourhood search* (Hansen and Mladenovic 1997, 1999, 2002), *ant colony systems* (Colorni, Dorigo, and Maniezzo 1991; Dorigo, Maniezzo, and Colorni 1996; Maniezzo and Carbonaro 2002), as well as a host of ad-hoc methods based on neighbourhood exploration, which, in some surveys, are lumped together under the "local search" heading. Several of these methods also implement some sort of hybridization scheme where, typically, the current incumbent solution is enhanced by a local improvement procedure. In most cases, the basic working principle of the method may be cast in the generic meta-heuristic framework illustrated in Figure 1.2. Thus, the main principles and strategies described in this paper apply to the parallelization of these meta-heuristics as well, as illustrated by the parallelization efforts that have been reported for these methods (e.g., Kindervater, Lenstra, and Savelsberg 1993; Pardalos, Pitsoulis, and Resende 1995; Sondergeld and Voß 1999; Verhoeven and Severens 1999). Of course, the most efficient applications of these principles and strategies to each of these meta-heuristics have yet to be established, which constitutes a most interesting research subject.

Co-operation and multi-thread parallelization appear to offer the most interesting perspectives for meta-heuristics. However, several issues and challenges remain to be addressed.

Synchronous implementations, where information is exchanged at regular intervals, have been reported for the three classes of meta-heuristics examined in this paper. In general, these implementations outperform the serial methods in solution quality. For tabu search (Crainic, Toulouse, and Gendreau 1995) and simulated annealing (Graffigne 1992), synchronous co-operative methods appear to be outperformed, however, by independent search procedures. Yet, the study by Lee and Lee (1992) contradicts this trend. Their results show the independent thread approach to be outperformed by two strategies of synchronous co-operating parallel threads. Similar finds have been reported for genetic algorithms: Cohoon *et al.* (1987) and Cohoon, Martin, and Richards (1991a, 1991b) report that parallel search with migration operators applied at regular intervals outperforms the same method without migration. These results point to interesting research issues that should be further investigated, especially since Lee and Lee used a dynamically adjusted synchronization interval that modified the traditional synchronous parallelism paradigm.

Asynchronous co-operative multi-thread search strategies appear to have been less studied but are being increasingly proposed. In fact, this strategy is probably the strongest current trend in parallel meta-heuristics, as illustrated by the development of memetic algorithms, of methods that evolve populations of simulated annealing threads, of the adaptive and central memory concepts initiated within the tabu search community but displaying general applicability characteristics. The results reported in the literature seem to indicate that asynchronous co-operative multi-thread parallelization strategies offer better results than synchronous and independent searches. More theoretical and empirical work is still required in this field, however.

An important issue for parallel meta-heuristic development and success concerns the definition of co-operation schemes and their impact on search behaviour and performance. A number of basic communication issues in designing multi-thread parallel meta-heuristics are discussed by Toulouse, Crainic, and Gendreau (1996). More thorough analysis (Toulouse, Crainic, and Sansó 1999, 1997; Toulouse *et al.* 1998; Toulouse, Crainic, and Thulasiraman 2000) shows that co-operative parallel meta-heuristics form dynamic systems and that the evolution of these systems may be more strongly determined by the co-operation scheme than by the optimization process. The selection of the information exchanged, and the part of it that is propagated past the initial exchange, significantly impacts the performance of co-operating procedures. The determination for each search thread of both when to import external information and how to use it (e.g., restart from the imported solution and clean up all history contrasted to the use of imported global information to bias local selection mechanisms) are equally important ingredients in the design of co-operating multi-thread parallel schemes. The application of these principles forms the basis of a new co-operation scheme, called *multi-level co-operative search* (Toulouse, Thulasiraman, and Glover 1999; see also Toulouse, Glover, and Thulasiraman 1998 and Ouyang *et al.* 2000, 2000a), which has proven very successful for graph partitioning problems.

The solutions exchanged by co-operating search threads form populations that do not evolve according to genetic principles, but rather follow the information exchange mechanisms that define the co-operation. Of course, genetic operators may be used to control this evolution, as seen in some co-operative simulated annealing schemes. Scatter search and ant colony systems offer alternate co-operation mechanisms. In this respect, it is noteworthy that ant systems and, more generally, swarm-based methods (Bonabeau, Dorigo, and Theraulaz 1999) appear as one of the first nature-inspired co-operation mechanisms. Yet, for now, despite the interest of its fundamental idea of trend enforcement/dilution, the co-operation principle underlying these methods appears much too rigid to offer a general purpose method. Scatter search, on the other hand, offers context-related combination mechanisms and memories for medium and long term steering of the evolution of the population. New mechanisms need to be developed, however, to relate this information to the search threads that make up the co-operating parallel algorithm. In fact, we believe that no single mechanism may adequately cover all possibilities and hybrid mechanisms will have to be defined.

Parallel co-operating methods do not have to include strategies belonging to only one meta-heuristic class. In fact, a number of recent studies (e.g., Le Bouthillier and Crainic 2001) tend to demonstrate that combining different meta-heuristics yields superior results. At the parallel computing level, this approach generalizes the trend towards hybrid development observed in meta-heuristic communities. It also opens up an exciting field of enquiry. What meta-heuristics to combine? What role can each type of meta-heuristic play? What information is exchanged and how it is used in this context? These are only a few of the questions that need to be answered.

Last but not least, recall an earlier remark that co-operating parallel mechanisms bear little, if any, resemblance to the initial meta-heuristic one attempts to parallelize. This remark is even more true when different meta-heuristics combine their efforts. In fact, more and more authors argue that co-operating multi-thread parallel methods should form a “new”, very broadly defined, class of meta-heuristics. If true, which we believe it is, we are left with the challenge of properly defining this meta-heuristic class. For example, memory mechanisms appear appropriate to record statistics on both the attributes of the solutions exchanged (or present in the solution population) and the performance of individual searches. How can this information be used to globally direct the search? What interactions are most appropriate between global and local (for each thread) information (memories)? These are among the most interesting challenges we face in this respect.

To conclude, parallel meta-heuristics offer the possibility to address problems more efficiently, both in terms of computing efficiency and solution quality. A rather limited number of strategies exist and this paper aims to both put these strategies into perspective and to briefly describe them. The study of parallel meta-heuristics design and performance still constitutes an exciting and challenging research domain with much opportunity for experimentation and development of important applications.

Acknowledgments

Funding for this project has been provided by the Natural Sciences and Engineering Council of Canada, and by the Fonds F.C.A.R. of the Province of Québec.

References

Aarts, E. and Korst, J. (2002). Selected Topics in Simulated Annealing. In Ribeiro, C. and Hansen, P., editors, *Essays and Surveys in Metaheuristics*, pages 1–37. Kluwer Academic Publishers, Norwell, MA.

Aarts, E.H.L, de Bont, F.M.J., Habers, J.H.A., and van Laarhoven, P.J.M. (1986). Parallel Implementations of Statistical Cooling Algorithms. *Integration, The VLSI Journal*, 3:209–238.

Aarts, E.H.L. and Korst, J.H.M. (1989). *Simulated Annealing and Boltzmann Machines*. John Wiley & Sons, New York, NY.

Abramson, D. and Abela, J. (1992). A Parallel Genetic Algorithm for Solving the School Timetabling Problem. In Gupta, G. and Keen, C., editors, *15th Australian Computer Science Conference*, pages 1–11. Department of Computer Science, University of Tasmania.

Abramson, D., Mills, G., and Perkins, S. (1993). Parallelization of a Genetic Algorithm for the Computation of Efficient Train Schedules. In Arnold, D., Christie, R., Day, J., and Roe, P., editors, *Proceedings of the 1993 Parallel Computing and Transputers Conference*, pages 139–149. IOS Press.

Aiex, R.M., Martins, S.L., Ribeiro, C.C., and Rodriguez, N.R. (1996). Asynchronous Parallel Strategies for Tabu Search Applied to the Partitioning of VLSI Circuits. Monografias em ciência da computação, Pontifícia Universidade Católica de Rio de Janeiro.

Andreatta, A.A. and Ribeiro C.C. (1994). A Graph Partitioning Heuristic for the Parallel Pseudo-Exhaustive Logical Test of VLSI Combinational Circuits. *Annals of Operations Research*, 50:1–36.

Azencott, R. (1992). *Simulated Annealing Parallelization Techniques*. John Wiley & Sons, New York, NY.

Badeau, P., Guertin, F., Gendreau, M., Potvin, J.-Y., and Taillard, É.D. (1997). A Parallel Tabu Search Heuristic for the Vehicle Routing Problem with Time Windows. *Transportation Research C: Emerging Technologies*, 5(2):109–122.

Baluja, S. (1993). Structure and Performance of Fine-Grain Parallelism in Genetic Algorithms. In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 155–162. Morgan Kaufmann, San Mateo, CA.

Barr, R.S. and Hickman, B.L. (1993). Reporting Computational Experiments with Parallel Algorithms: Issues, Measures, and Experts Opinions. *ORSA Journal on Computing*, 5(1):2–18.

Battiti, R. and Tecchiolli, G. (1992). Parallel Based Search for Combinatorial Optimization: Genetic Algorithms and TABU. *Microprocessors and Microsystems*, 16(7):351–367.

Bhandarkar, S.M. and Chirravuri, S. (1996). A Study of Massively Parallel Simulated Annealing Algorithms for Chromosome Reconstruction via Clone Ordering. *Parallel Algorithms and Applications*, 9:67–89.

Bonabeau, E., Dorigo, M., and Theraulaz, G., editors (1999). *Swarm Intelligence - From Natural to Artificial Systems*. Oxford University Press, New York, NY.

Cantú-Paz, E. (1995). A Summary of Research on Parallel Genetic Algorithms. Report 95007, University of Illinois at Urbana-Champain.

Cantú-Paz, E. (1998). A Survey of Parallel Genetic Algorithms. *Calculateurs Parallèles, Réseaux et Systèmes répartis*, 10(2):141–170.

Cavalcante, C.B.C., Cavalcante, V.F., Ribeiro, C.C., and de Souza, C.C. (2002). Parallel Cooperative Approaches for the Labor Constrained Scheduling Problem. In Ribeiro, C. and Hansen, P., editors, *Essays and Surveys in Metaheuristics*, pages 201–225. Kluwer Academic Publishers, Norwell, MA.

Chakrapani, J. and Skorin-Kapov, J. (1992). A Connectionist Approach to the Quadratic Assignment Problem. *Computers & Operations Research*, 19(3/4):287–295.

Chakrapani, J. and Skorin-Kapov, J. (1993). Massively Parallel Tabu Search for the Quadratic Assignment Problem. *Annals of Operations Research*, 41:327–341.

Chakrapani, J. and Skorin-Kapov, J. (1993a). Connection Machine Implementation of a Tabu Search Algorithm for the Traveling Salesman Problem. *Journal of Computing and Information Technology*, 1(1):29–36.

Chakrapani, J. and Skorin-Kapov, J. (1995). Mapping Tasks to Processors to Minimize Communication Time in a Multiprocessor System. In *The Impact of Emerging Technologies of Computer Science and Operations Research*, pages 45–64. Kluwer Academic Publishers, Norwell, MA.

Chen, Y.-W., Nakao, Z., and Fang, X. (1996). Parallelization of a Genetic Algorithm for Image Restoration and its Performance Analysis. In *IEEE International Conference on Evolutionary Computation*, pages 463–468.

Christofides, N., Mingozzi A., and Toth, P. (1979). The Vehicle Routing Problem. In Christofides, N., A., M., Toth, P., and Sandi, C., editors, *Combinatorial Optimization*, pages 315–338. John Wiley, New York.

Chu, K., Deng, Y., and Reinitz, J. (1999). Parallel Simulated Annealing Algorithms by Mixing States. *Journal of Computational Physics*, 148:646–662.

Cohoon, J., Hedge, S., Martin, W., and Richards, D. (1987). Punctuated Equilibria: A Parallel Genetic Algorithm. In Grefenstette, J., editor, *Proceedings of the Second International Conference on Genetic Algorithms and their Applications*, pages 148–154. Lawrence Erlbaum Associates, Hillsdale, NJ.

Cohoon, J., Martin, W., and Richards, D. (1991a). Genetic Algorithm and Punctuated Equilibria in VLSI. In Schwefel, H.-P. and Männer, R., editors, *Parallel Problem Solving from Nature, Lecture Notes in Computer Science 496*, pages 134–144. Springer-Verlag, Berlin.

Cohoon, J., Martin, W., and Richards, D. (1991b). A Multi-Population Genetic Algorithm for Solving the k-Partition Problem on Hyper-Cubes. In Belew, R. and Booker, L., editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 134–144. Morgan Kaufmann, San Mateo, CA.

Colorni, A., Dorigo, M., and Maniezzo, V. (1991). Distributed Optimization by Ant Colonies. In *Proceedings of the 1991 European Conference on Artificial Life*, pages 134–142. North-Holland, Amsterdam.

Crainic, T.G. (2002). Parallel Computation, Co-operation, Tabu Search. In Rego, C. and Alidaee, B., editors, *Adaptive Memory and Evolution: Tabu Search and Scatter Search*. Kluwer Academic Publishers, Norwell, MA. forthcoming.

Crainic, T.G. and Gendreau, M. (1999). Towards an Evolutionary Method - Cooperating Multi-Thread Parallel Tabu Search Hybrid. In Voß, S., Martello, S., Roucairol, C., and Osman, I.H., editors, *Meta-Heuristics 98: Theory & Applications*, pages 331–344. Kluwer Academic Publishers, Norwell, MA.

Crainic, T.G. and Gendreau, M. (2001). Cooperative Parallel Tabu Search for Capacitated Network Design. *Journal of Heuristics*. forthcoming.

Crainic, T.G. and Toulouse, M. (1998). Parallel Metaheuristics. In T.G. Crainic and G. Laporte, editors, *Fleet Management and Logistics*, pages 205–251. Kluwer Academic Publishers, Norwell, MA.

Crainic, T.G., Toulouse, M., and Gendreau, M. (1995a). Parallel Asynchronous Tabu Search for Multicommodity Location-Allocation with Balancing Requirements. *Annals of Operations Research*, 63:277–299.

Crainic, T.G., Toulouse, M., and Gendreau, M. (1995b). Synchronous Tabu Search Parallelization Strategies for Multicommodity Location-Allocation with Balancing Requirements. *OR Spektrum*, 17(2/3):113–123.

Crainic, T.G., Toulouse, M., and Gendreau, M. (1997). Towards a Taxonomy of Parallel Tabu Search Algorithms. *INFORMS Journal on Computing*, 9(1):61–72.

Cung, V.-D., Martins, S.L., Ribeiro, C.C., and Roucairol, C. (2002). Strategies for the Parallel Implementations of Metaheuristics. In Ribeiro, C. and Hansen, P., editors, *Essays and Surveys in Metaheuristics*, pages 263–308. Kluwer Academic Publishers, Norwell, MA.

Darema, F., Kirkpatrick, S., and Norton, V.A. (1987). Parallel Algorithms for Chip Placement by Simulated Annealing. *IBM Journal of Research and Development*, 31:391–402.

De Falco, I., Del Balio, R., and Tarantino, E. (1995). Solving the Mapping Problem by Parallel Tabu Search. Report, Istituto per la Ricerca sui Sistemi Informatici Paralleli-CNR.

De Falco, I., Del Balio, R., Tarantino, E., and Vaccaro, R. (1994). Improving Search by Incorporating Evolution Principles in Parallel Tabu Search. In *Proceedings International Confonference on Machine Learning*, pages 823–828.

Dorigo, M., Maniezzo, V., and Colorni, A. (1996). The Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, 26(1):29–41.

Du, Z., Li, S., Li, S., Wu, M., and Zhu, J. (1999). Massively Parallel Simulated Annealing Embedded with Downhill - A SPMD Algorithm for Cluster Computing. In *Proceedings of the 1st IEEE Computer Society International Workshop on Cluster Computing*. IEEE Computer Society Press, Washington, DC.

Durand, M.D. (1989). Parallel Simulated Annealing: Accuracy vs. Speed in Placement. *IEEE Design & Test of Computers*, 6(3):8–34.

Durand, M.D. (1989a). Cost Function Error in Asynchronous Parallel Simulated Annealing Algorithms. Technical Report CUCS-423-89, University of Columbia.

Felten, E. and Karlin, S. and Otto, S.W. (1985). The Traveling Salesman Problem on a Hypercube, MIMD Computer. In *Proc. 1985 of the Int. Conf. on Parallel Processing*, pages 6–10.

Feo, T.A. and Resende, M.G.C. (1995). Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, 6(2):109–133.

Festa, P. and Resende, M.G.C. (2002). GRASP: An Annotated Bibliography. In Ribeiro, C. and Hansen, P., editors, *Essays and Surveys in Metaheuristics*, pages 325–367. Kluwer Academic Publishers, Norwell, MA.

Fiechter, C.-N. (1994). A parallel tabu search algorithm for large travelling salesman problems. *Discrete Applied Mathematics*, 51(3):243–267.

Fogarty, T.C. and Huang, R. (1990). Implementing the Genetic Algorithm on Transputer Based Parallel Systems. In Schwefel, H.-P. and Männer, R., editors, *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, pages 145–149. Springer-Verlag, Berlin.

Fogel, D.B. (1994). Evolutionary Programming: An Introduction and Some Current Directions. *Statistics and Computing*, 4:113–130.

Garcia, B.L., Potvin, J.-Y., and Rousseau, J.M. (1994). A Parallel Implementation of the Tabu Search Heuristic for Vehicle Routing Problems with Time Window Constraints. *Computers & Operations Research*, 21(9):1025–1033.

Gendreau, M. (2002). Recent Advances in Tabu Search. In Ribeiro, C. and Hansen, P., editors, *Essays and Surveys in Metaheuristics*, pages 369–377. Kluwer Academic Publishers, Norwell, MA.

Gendreau, M., Guertin, F., Potvin, J.-Y., and Taillard, É.D. (1999). Tabu Search for Real-Time Vehicle Routing and Dispatching. *Transportation Science*, 33(4):381–390.

Glover, F. (1986). Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers & Operations Research*, 1(3):533–549.

Glover, F. (1989). Tabu Search – Part I. *ORSA Journal on Computing*, 1(3):190–206.

Glover, F. (1990). Tabu Search – Part II. *ORSA Journal on Computing*, 2(1):4–32.

Glover, F. (1994). Genetic Algorithms and Scatter Search: Unsuspected Potentials. *Statistics and Computing*, 4:131–140.

Glover, F. (1996). Tabu Search and Adaptive Memory Programming – Advances, Applications and Challenges. In Barr, R., Helgason, R., and Kennington, J., editors, *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer Academic Publishers, Norwell, MA.

Glover, F. and Laguna, M. (1993). Tabu search. In Reeves, C., editor, *Modern Heuristic Techniques for Combinatorial Problems*, pages 70–150. Blackwell Scientific Publications, Oxford.

Glover, F. and Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publishers, Norwell, MA.

Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA.

Graffigne, C. (1992). Parallel Annealing by Periodically Interacting Multiple Searches: an Experimental Study. In Azencott, R., editor, *Simulated Annealing Parallelization Techniques*, pages 47–79. John Wiley & Sons, New York, NY.

Greening, D.R. (1990). Parallel Simulated Annealing Techniques. *Physica D*, 42:293–306.

Grefenstette, J. (1981). Parallel Adaptive Algorithms for Function Optimization. Technical Report CS-81-19, Vanderbilt University, Nashville.

Hansen, P. and Mladenovic, N. (1997). Variable Neighborhood Search. *Computers & Operations Research*, 24:1097–1100.

Hansen, P. and Mladenovic, N. (1999). An Introduction to Variable Neighborhood Search. In Voß, S., Martello, S., Roucairol, C., and Osman, I.H., editors, *Meta-Heuristics 98: Theory & Applications*, pages 433–458. Kluwer, Norwell, MA.

Hansen, P. and Mladenovic, N. (2002). Developments of Variable Neighborhood Search. In Ribeiro, C. and Hansen, P., editors, *Essays and Surveys in Metaheuristics*, pages 415–439. Kluwer Academic Publishers, Norwell, MA.

Hauser, R. and Männer, R. (1994). Implementation of Standard Genetic Algorithm on MIMD Machines. In Davidor, Y., Schwefel, H.-P., and Männer, R., editors, *Parallel Problem Solving from Nature III, Lecture Notes in Computer Science 866*, pages 504–514. Springer-Verlag, Berlin.

Herdy, M. (1992). Reproductive Isolation as Strategy Parameter in Hierarchical Organized Evolution Strategies. In Männer, R. and Manderick, B., editors, *Parallel Problem Solving from Nature, 2*, pages 207–217. North-Holland, Amsterdam.

Hillis, D.W. (1992). Co-evolving Parasites Improve Simulated Evolution as an Optimization Procedure. In Langton, C. e. a., editor, *Artificial Life II*, pages 313–324. Addison-Wesley.

Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI.

Holmqvist, K. and Migdalas, A. and Pardalos, P.M. (1997). Parallelized Heuristics for Combinatorial Search. In Migdalas, A., Pardalos, P., and Storoy, S., editors, *Parallel Computing in Optimization*, pages 269–294. Kluwer Academic Publishers, Norwell, MA.

Jayaraman, R. and Darema, F. (1988). Error Tolerance in Parallel Simulated Techniques. In *Proceedings of the IEEE International Conference on Computer-Aided Design: ICCAD-88*, pages 545–548. IEEE Computer Society Press, Washington, DC.

Kindervater, G.A.P. and Lenstra, J.K. and Savelsberg, M.W.P. (1993). Sequential and Parallel Local Search for the Time Constrained Traveling Salesman Problem. *Discrete Applied Mathematics*, 42:211–225.

Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P. (1983). Optimization by Simulated Annealing. *Science*, 220:671–680.

Kliewer, G. and Tschöke, S. (2000). A General Parallel Simulated Annealing Library and Its Application in Airline Industry. In *Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS 2000)*, pages 55–61, Cancun, Mexico.

Kohlmorgen, U., Schmeck, H., and Haase, K. (1999). Experiences with Fine-grained parallel Genetic Algorithms. *Annals of Operations Research*, 90:203–219.

Kurbel, K. and Schneider, B. and Singh, K. (1995). VLSI Standard Cell Placement by Parallel Hybrid Simulated Annealing and Genetic Algorithm. In Pearson, D. W., Steele, N. C., and Albrecht, R. F., editors, *Proceedings of the*

Second International Conference on Artificial Neural Networks and Genetic Algorithms, pages 491–494. Springer-Verlag, Berlin.

Laarhoven, P. and Aarts, E.H.L. (1987). *Simulated Annealing: Theory and Applications*. Reidel, Dordrecht.

Laursen, P.S. (1994). Problem-Independent Parallel Simulated Annealing Using Selection and Migration. In Davidor, Y., Schwefel, H.-P., and Männer, R., editors, *Parallel Problem Solving from Nature III, Lecture Notes in Computer Science 866*, pages 408–417. Springer-Verlag, Berlin.

Laursen, P.S. (1996). Parallel Heuristic Search – Introductions and a New Approach. In Ferreira, A. and Pardalos, P., editors, *Solving Combinatorial Optimization Problems in Parallel, Lecture Notes in Computer Science 1054*, pages 248–274. Springer-Verlag, Berlin.

Le Bouthillier, A. and Crainic, T.G. (2001). Parallel Co-operative Multi-thread Meta-heuristic for the Vehicle Routing Problem with Time Window Constraints. Publication, Centre de recherche sur les transports, Université de Montréal, Montréal, QC, Canada.

Lee, F.-H.A. (1995). *Parallel Simulated Annealing on a Message-Passing Multi-Computer*. PhD thesis, Utah State University.

Lee, K-G. and Lee, S-Y. (1992a). Efficient Parallelization of Simulated Annealing Using Multiple Markov Chains: An Application to Graph Partitioning. In Mudge, T., editor, *Proceedings of the International Conference on Parallel Processing*, volume III: Algorithms and Applications, pages 177–180. CRC Press.

Lee, K-G. and Lee, S-Y. (1995). Synchronous and Asynchronous Parallel Simulated Annealing with Multiple Markov Chains. *Lecture Notes in Computer Science 1027*, pages 396–408.

Lin, S.-C., Punch, W., and Goodman, E. (1994). Coarse-Grain Parallel Genetic Algorithms: Categorization and New Approach. In *Sixth IEEE Symposium on Parallel and Distributed Processing*, pages 28–37. IEEE Computer Society Press.

Lis, J. (1996). Parallel Genetic Algorithm with the Dynamic Control Parameter. In *IEEE 1996 International Conference on Evolutionary Computation*, pages 324–328.

Mahfoud, S.W. and Goldberg, D.E. (1995). Parallel Recombinative Simulated Annealing: A Genetic Algorithm. *Parallel Computing*, 21:1–28.

Malek, M., Guruswamy, M., Pandya, M., and Owens, H. (1989). Serial and Parallel Simulated Annealing and Tabu Search Algorithms for the Traveling Salesman Problem. *Annals of Operations Research*, 21:59–84.

Maniezzo, V. and Carbonaro, A. (2002). Ant COlony Optimization: An Overview. In Ribeiro, C. and Hansen, P., editors, *Essays and Surveys in Metaheuristics*, pages 469–492. Kluwer Academic Publishers, Norwell, MA.

Martins, S.L., Ribeiro, C.C., and Rodriguez, N.R. (1996). Parallel Programming Tools for Distributed Memory Environments. Monografias em Ciência da Computação 01/96, Pontifícia Universidade Católica de Rio de Janeiro.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E. (1953). Equation of State Calculation by Fast Computing Machines. *Journal of Chemical Physics*, 21:1087–1092.

Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, Berlin.

Michalewicz, Z. and Fogel. D.B. (2000). *How to Solve It: Modern Heuristics*. Springer-Verlag, Berlin.

Moscato, P. (1989). On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Publication Report 790, Caltech Concurrent Computation Program.

Moscato, P. and Norman, M.G. (1992). A 'Memetic' Approach for the Traveling Salesman Problem. Implementation of a Computational Ecology for Combinatorial Optimization on Message-Passing Systems. In Valero, M., Onate, E., Jane, M., Larriba, J., and Suarez, B., editors, *Parallel Computing and Transputer Applications*, pages 187–194. IOS Press, Amsterdam.

Mühlenbein, H. (1991). Evolution in Time and Space - The Parallel Genetic Algorithm. In Rawlins, G., editor, *Foundations of Genetic Algorithm & Classifier Systems*, pages 316–338. Morgan Kaufman, San Mateo, CA.

Mühlenbein, H. (1992). Parallel Genetic Algorithms in Combinatorial Optimization. In Balci, O., Sharda, R., and Zenios, S., editors, *Computer Science and Operations Research*, pages 441–456. Pergamon Press, New York, NY.

Mühlenbein, H. (1992a). How Genetic Algorithms Really Work: Mutation and Hill-Climbing. In Männer, R. and Manderick, B., editors, *Parallel Problem Solving from Nature, 2*, pages 15–26. North-Holland, Amsterdam.

Mühlenbein, H., Gorges-Schleuter, M., and Krämer, O. (1987). New Solutions to the Mapping Problem of Parallel Systems - the Evolution Approach. *Parallel Computing*, 6:269–279.

Mühlenbein, H., Gorges-Schleuter, M., and Krämer, O. (1988). Evolution Algorithms in Combinatorial Optimization. *Parallel Computing*, 7(1):65–85.

Mühlenbein, H. and Schlierkamp-Voosen, D. (1994). The Science of Breeding and its Application to the Breeder Genetic Algorithm BGA. *Evolutionary Computation*, 1(4):335–360.

Ouyang, M., Toulouse, M., Thulasiraman, K., Glover, F., and Deogun, J.S. (2000a). Multi-Level Cooperative Search: Application to the Netlist/Hypergraph Partitioning Problem. In *Proceedings of International Symposium on Physical Design*, pages 192–198. ACM Press.

Ouyang, M., Toulouse, M., Thulasiraman, K., Glover, F., and Deogun, J.S. (2000b). Multilevel Cooperative Search for the Circuit/Hypergraph Partitioning Problem. *IEEE Transactions on Computer-Aided Design*. To appear.

Pardalos, P.M. and Pitsoulis, L. and Mavridou, T. and Resende, M.G.C. (1995). Parallel Search for Combinatorial Optimization: Genetic Algorithms, Simulated Annealing, Tabu Search and GRASP. In Ferreira, A. and Rolim, J., editors, *Proceedings of Workshop on Parallel Algorithms for Irregularly Structured Problems, Lecture Notes in Computer Science 980*, pages 317–331. Springer-Verlag, Berlin.

Pardalos, P.M. and Pitsoulis, L. and Resende, M.G.C. (1995). A Parallel GRASP Implementation for the Quadratic Assignment Problem. In Ferreira, A. and Rolim, J., editors, *Solving Irregular Problems in Parallel: State of the Art*. Kluwer Academic Publishers, Norwell, MA.

Porto, S.C.S. and Ribeiro, C.C. (1995). A Tabu Search Approach to Task Scheduling on Heteregenous Processors Under Precedence Constraints. *International Journal of High-Speed Computing*, 7:45–71.

Porto, S.C.S. and Ribeiro, C.C. (1996). Parallel Tabu Search Message-Passing Synchronous Strategies for Task Scheduling Under Precedence Constraints. *Journal of Heuristics*, 1(2):207–223.

Potter, M. and De Jong, K. (1994). A Cooperative Coevolutionary Approach to Function Optimization . In Davidor, Y., Schwefel, H.-P., and Männer, R., editors, *Parallel Problem Solving from Nature III, Lecture Notes in Computer Science 866*, pages 249–257. Springer-Verlag, Berlin.

Ram, D.J. and Sreenivas, T.H. and Subramaniam, K.G. (1996). Parallel Simulated Annealing Algorithms. *Journal of Parallel and Distributed Computing*, 37:207–212.

Rego, C. and Roucairol, C. (1996). A Parallel Tabu Search Algorithm Using Ejection Chains for the VRP. In Osman, I. and Kelly, J., editors, *Meta-Heuristics: Theory & Applications*, pages 253–295. Kluwer Academic Publishers, Norwell, MA.

Rochat, Y. and Taillard, É.D. (1995). Probabilistic Diversification and Intensification in Local Search for Vehicle Routing. *Journal of Heuristics*, 1(1):147–167.

Schlierkamp-Voosen, D. and Mühlenbein, H. (1994). Strategy Adaptation by Competing Subpopulations. In Davidor, Y., Schwefel, H.-P., and Männer, R., editors, *Parallel Problem Solving from Nature III, Lecture Notes in Computer Science 866*, pages 199–208. Springer-Verlag, Berlin.

Schnecke, V. and Vornberger, O. (1996). An Adaptive Parallel Genetic Algorithm for VLSI-Layout Optimization. In Davidor, Y., Schwefel, H.-P., and Männer, R., editors, *Parallel Problem Solving from Nature III, Lecture Notes in Computer Science 866*, pages 859–868. Springer-Verlag, Berlin.

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

Schwehm, M. (1992). Implementation of Genetic Algorithms on Various Interconnection Networks. In Valero, M., Onate, E., Jane, M., Larriba, J., and Suarez, B., editors, *Parallel Computing and Transputers Applications*, pages 195–203. Amsterdam: IOS Press.

Shonkwiler, R. (1993). Parallel Genetic Algorithms. In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 199–205. Morgan Kaufmann, San Mateo, CA.

Sondergeld, L. and Voß, S. (1999). Cooperative Intelligent Search Using Adaptive Memory Techniques. In Voß, S., Martello, S., Roucairol, C., and Osman, I.H., editors, *Meta-Heuristics 98: Theory & Applications*, pages 297–312. Kluwer, Norwell, MA.

Starkweather, T., Whitley, D., and Mathias, K. (1991). Optimization Using Distributed Genetic Algorithms. In Schwefel, H.-P. and Männer, R., editors, *Parallel Problem Solving from Nature, Lecture Notes in Computer Science 496*, pages 176–185. Springer-Verlag, Berlin.

Taillard, É.D. (1991). Robust Taboo Search for the Quadratic Assignment Problem. *Parallel Computing*, 17:443–455.

Taillard, É.D. (1993a). Parallel Iterative Search Methods for Vehicle Routing Problems. *Networks*, 23:661–673.

Taillard, É.D. (1993b). *Recherches itératives dirigées parallèles*. PhD thesis, École Polytechnique Fédérale de Lausanne.

Taillard, É.D. (1994). Parallel Taboo Search Techniques for the Job Shop Scheduling Problem. *ORSA Journal on Computing*, 6(2):108–117.

Taillard, É.D., Badeau, P., Gendreau, M., Guertin, F., and Potvin, J.-Y. (1997). A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows. *Transportation Science*, 31:170–186.

ten Eikelder, H.M.M., Aarts, B.J.M., Verhoeven, M.G.A., and Aarts, E.H.L. (1999). Sequential and Parallel Local Search for Job Shop Scheduling. In Voß, S., Martello, S., Roucairol, C., and Osman, I.H., editors, *Meta-Heuristics 98: Theory & Applications*, pages 359–371. Kluwer, Norwell, MA, Montréal, QC, Canada.

Toulouse, M., Crainic, T.G., and Gendreau, M. (1996). Communication Issues in Designing Cooperative Multi Thread Parallel Searches. In I.H. Osman and J.P. Kelly, editors, *Meta-Heuristics: Theory & Applications*, pages 501–522. Kluwer Academic Publishers, Norwell, MA.

Toulouse, M., Crainic, T.G., and Sansó, B. (1997). Systemic Behavior of Cooperative Search Algorithms. Publication CRT-97-55, Centre de recherche sur les transports, Université de Montréal, Montréal, QC, Canada.

Toulouse, M., Crainic, T.G., and Sansó, B. (1999a). An Experimental Study of Systemic Behavior of Cooperative Search Algorithms. In Voß, S., Martello, S., Roucairol, C., and Osman, I.H., editors, *Meta-Heuristics 98: Theory & Applications*, pages 373–392. Kluwer Academic Publishers, Norwell, MA.

Toulouse, M., Crainic, T.G., Sansó, B., and Thulasiraman, K. (1998a). Self-Organization in Cooperative Search Algorithms. In *Proceedings of the 1998 IEEE International Conference on Systems, Man, and Cybernetics*, pages 2379–2385. Omnipress.

Toulouse, M., Crainic, T.G., and Thulasiraman, K. (2000). Global Optimization Properties of Parallel Cooperative Search Algorithms: A Simulation Study. *Parallel Computing*, 26(1):91–112.

Toulouse, M., Glover, F., and Thulasiraman, K. (1998b). A Multi-Scale Cooperative Search with an Application to Graph Partitioning. Report, School of Computer Science, University of Oklahoma, Norman, OK.

Toulouse, M., Thulasiraman, K., and Glover, F. (1999b). Multi-Level Cooperative Search. In Amestoy, P., Berger, P., Daydé, M., Duff, I., Frayssé, V., Giraud, L., and Ruiz, D., editors, *5th International Euro-Par Parallel Processing Conference*, volume 1685 of *Lecture Notes in Computer Science*, pages 533–542. Springer-Verlag, Berlin.

Verhoeven, M.G.A. and Severens, M.M.M. (1999). Parallel Local Search for Steiner Trees in Graphs. *Annals of Operations Research*, 90:185–202.

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

Voß, S. (1993). Tabu Search: Applications and Prospects. In Du, D.-Z. and Pardalos, P., editors, *Network Optimization Problems*, pages 333–353. World Scientific Publishing Co., Singapore.

Whitley, D. (1993). Cellular Genetic Algorithms. In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 658–658. Morgan Kaufmann, San Mateo, CA.

Whitley, D. and Starkweather, T. (1990a). Optimizing Small Neural Networks Using a Distributed Genetic Algorithm. In *Proceedings of the International Conference on Neural Networks*, pages 206–209. IEEE Press.

Whitley, D. and Starkweather, T. (1990b). GENITOR II: A Distributed Genetic Algorithm. *Journal of Experimental and Theoretical Artificial Intelligence*, 2(3):189–214.

Whitley, L.D. (1994). A Genetic Algorithm Tutorial. *Statistics and Computing*, 4:65–85.