1 PARALLEL TABU SEARCH

TEODOR GABRIEL CRAINIC
Centre de recherche sur les transports
Université de Montréal
and
École des sciences de la gestion
Université du Québec à Montréal
Montreal, Canada

MICHEL GENDREAU
Centre de recherche sur les transports
and
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montreal, Canada

JEAN-YVES POTVIN
Centre de recherche sur les transports
and
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montreal, Canada

1.1 INTRODUCTION

Like other metaheuristics, Tabu Search (TS) has been the object over the last fifteen years or so of several efforts aimed at taking advantage of the benefits offered by parallel computing (see the surveys of Crainic and Toulouse [23, 24], Cung *et al.* [28], Holmqvist, Migdalas, and Pardalos [54], and Pardalos *et al.* [64]). As it is the case with other metaheuristics, the main goal pursued when resorting to parallel implementations of TS is to reduce the overall ("wallclock") time required to solve a problem instance. This is a particularly important objective when tackling problems that must be solved in real time, as we shall see in a latter section of this chapter. There are, however, other possible objectives. Among these, we must mention enhancing the robustness of the algorithm at hand by performing a broader search of the solution space. In some cases, this may even lead to a more efficient search scheme, i.e., a search scheme capable of finding better solutions than the corresponding sequential search approach *for the same overall computational effort*. This may also significantly reduce the calibration effort required to achieve good results consistently over diverse sets of problem instances.

The purpose of this chapter is two-fold: first, to describe, discuss, and illustrate the main strategies that have been used over time to parallelize TS heuristics; second, to provide an updated survey of the literature in the rapidly moving field of parallel TS. The remainder of the chapter is organized as follows. In section 1.2, we briefly recall the main features and components of Tabu Search. Section 1.3 describes parallelization strategies for TS. Parallel TS implementations from the literature are then reviewed in section 1.4. This is followed in section 1.5 by a description of two applications of parallel TS schemes for the real-time management of fleets of vehicles. Section 1.6 summarizes the main conclusions of the papers and identifies some interesting research directions for parallel TS.

1.2 TABU SEARCH

Tabu Search was introduced by Glover in 1986 in a seminal paper [45] in which he also coined the term *metaheuristics* and defined these as strategies designed to guide *inner heuristics* aimed at specific problems. TS is an extension of classical *local search* methods typically used to find approximate solutions to difficult combinatorial optimization problems. As in other local search techniques, TS explores the solution or *search space* by moving at each iteration from the current solution to a neighbouring one, where the neighbourhood of the current solution is defined by the transformations of the solution allowed by the specific inner heuristic (also called the *neighbourhood operator*). Usually, the next solution is the one that improves the most the objective function ("best improvement" rule), or the first improving one that is encountered when exploring the neighbourhood ("first improvement" rule). Traditional local search techniques rely on the monotonic improvement of the objective function to guide and control the search and, therefore, they typically end up trapped in local optima of the neighbourhood operator. The main distinctive feature of TS

is that it can overcome local optima and keep the search going. When a local optimum is encountered, the search moves instead to the best deteriorating solution in the neighbourhood. To prevent cycling, a history of the search is maintained in a *short-term memory* and moves that would bring the search trajectory back toward recently visited solutions are said to be *tabu* and are disallowed, unless they meet some conditions (*aspiration criteria*). In theory, the search could continue for ever. Stopping criteria must thus be specified in actual implementations. The most commonly used are: a fixed CPU time allotment, the total number of iterations performed, the number of iterations performed without observing an improvement in the best objective value recorded, etc.

The previous description relates to what could be termed *basic Tabu Search*. In reality, Tabu Search in its full implementation goes much further than that and can be interpreted globally as the combination of local search principles with the exploitation of information stored in various types of memories. Two key concepts in TS are those of *search intensification* and *search diversification*. The idea behind search intensification is that regions of the search space that appear "promising" (usually because good solutions have been encountered close by) should be explored more thoroughly in order to make sure that the best solutions in these regions are found. Intensification is usually based on some type of *intermediate-term memory*, such as a recency memory, in which one would record the number of consecutive iterations that various "solution components" have been present without interruption in the current solution. Intensification is typically performed by periodically restarting the search from the best currently known solution and by "freezing" (fixing) in this solution the components that seem more attractive. It also often involves switching to a more powerful neighborhood operator for a short period of time. Search diversification addresses the complementary need to perform a broad exploration of the search space to make sure that the search trajectory has not been confined to regions containing only mediocre solutions. It is thus a mechanism that tries to force the search trajectory into previously unexplored regions of the search space. Diversification is usually based on some form of *long-term memory*, such as a frequency memory, in which one would record the total number of iterations (since the beginning of the search) that various "solution components" have been present in the current solution or have been involved in the selected moves. There are three major diversification techniques. The first one, called *restart diversification*, involves introducing some rarely used components in the current or the best known solution and restarting the search from this point. The second diversification technique, called *continuous diversification*, integrates diversification considerations directly into the regular search process by biasing the evaluation of possible moves to account for component frequencies. A third method for achieving diversification is through so-called *strategic oscillation*, which is a systematic technique for driving the search trajectory into infeasible space (i.e., to solutions that do not satisfy all the constraints of the problem at hand) and then back into feasible space in the hope that it will end up into a different region of the search space.

Readers wishing to learn more about Tabu Search are referred to the several introductory chapters that have been written on the topic (e.g., [38, 50, 53]) and to

the fundamental papers of Glover [46, 47, 48]. The book by Glover and Laguna [51] is the most comprehensive reference on the topic.

1.3 PARALLELIZATION STRATEGIES FOR TABU SEARCH

Because of its potentially heavy computational requirements, Tabu Search is a natural candidate for the application of parallel computing techniques. In fact, fairly early after the introduction of the method in 1986, researchers began to use such techniques in the development of TS heuristics. Most of these early efforts focused on the parallelization of the most computationally intensive step of the method, namely the neighbourhood evaluation, using rather straightforward master-slave schemes (see, e.g., [18, 60, 73]). It soon became apparent, however, that one could go much further in the parallelization of TS than suggested by these *low level parallelization* schemes, which turn out to be only faster versions of sequential TS implementations, but with the same overall behaviour (i.e., they will produce the same search trajectories as their sequential counterparts, but in lower wallclock time). In fact, one could easily envision *high level parallelization* approaches that would display a completely different algorithmic behaviour; this would be the case, in particular, of algorithmic schemes relying on several *search threads* exploring simultaneously the search space in a coordinated and purposeful fashion.

1.3.1 A taxonomy

In 1993, Crainic, Toulouse, and Gendreau introduced a taxonomy of parallel TS approaches (later published in 1997 [27]) that had several objectives: first, to provide a comprehensive picture of the then existing parallelization strategies for TS; second, to contribute to a more meaningful analysis and comparison of these methods; third, to foster a better understanding of the relationships between TS and parallel computing; and fourth, to identify new potential parallelization strategies and suggest interesting research avenues. To this date, this taxonomy remains the most comprehensive one on the topic. We now summarize this taxonomy and the parallelization strategies it sought to classify, before reviewing the literature using it in the next section.

The taxonomy is based on a three-dimensional classification of algorithmic features. The first dimension is called *Control cardinality*; it defines whether the search is controlled by a single process (as in master-slave implementations) or collegially by several processes that may collaborate or not. In the latter case, each process is in charge of its own search, as well as of establishing communications with the other processes, and the global search terminates once each individual search stops. These two alternatives are respectively identified as *1-control (1C)* and *p-control (pC)*.

The second dimension (*Control and communication type*) relates to the type and flexibility of the search control. As we shall see, it is probably the most important dimension on which to differentiate parallel TS implementations, and it deserves a detailed discussion. This control type dimension accounts for the communication organization, synchronization and hierarchy, as well as the way in which processes

handle and share information. There are four degrees or levels along this dimension; they correspond to progressively more complex and sophisticated control schemes. The two first levels cover parallelization schemes that rely on *synchronized communications*, i.e., where all processes have to stop and engage in some form of communication and information exchange at set moments (number of iterations, time intervals, specified algorithmic stages, etc.) that are exogenously determined, being either hard-coded or determined by a control (master) process. The first level (*Rigid Synchronization (RS)*) refers to situations where little, if any, information takes place between processes that are at the same level of the communication hierarchy. This would typically the case in classical *1*-control master-slave schemes in which the master process dispatches fixed computing-intensive tasks to the slaves processes, waits for all these tasks to be completed, and then proceeds with the remainder of the search (calling again upon the slave processes whenever needed). Another example of rigid synchronization, but with a *p*-control control cardinality, is that of the direct parallelization of independent search processes. In such a case, each individual process executes its own search without communicating with the others except at the very end when the best solutions found by each process are compared to determine the best overall. The second level along this control type dimension (*Knowledge Synchronization (KS)*) corresponds to more sophisticated synchronous parallelization schemes. In *1*-control settings, slaves thus perform on their own more complex tasks, such as executing a limited number of TS steps on a subset of the search space. In a *p*-control environment, the knowledge synchronization mode covers situations where processes follow independent search trajectories for some time, but stop at predetermined moments (e.g., after performing a set number of iterations) to engage into intensive communication to share information between themselves. The third level along the control type dimension (*Collegial (C)*) covers situations where *asynchronous* communications are used; it only makes sense in a *p*-control context. In these situations, each search process explores the search space (or sometimes, part of it) according to its own logic, storing and processing local information as it goes along. It also communicates with the other processes (all or just a subset of them) or with a central memory at times dictated by the results of its own search (albeit in the context of the overall global search): for instance, if a process finds a globally improving solution, it might broadcast it to all other processes or deposit it in the central memory; if it finds that its own search is stagnating, it may request a new solution from the other processes or from the central memory. The fourth and final level on the control type dimension is called *Knowledge Collegial (KC)*. It refers to more advanced asynchronous communication schemes in which the contents of communications are analyzed to infer additional information concerning the global search pattern performed so far and/or the characteristics of good solutions. This may be implemented using global memory structures that can be accessed by the processes while they conduct their own search. The main difference between the collegial and the knowledge collegial organizations is that in the former the information recovered by a process from another is identical to the information sent by that process, while in the latter case, the information received is richer, thus helping to build a picture of the overall dynamics of the asynchronous exploration of the search space.

The third dimension of the taxonomy pertains to *Search differentiation*: do search threads start from the same or from different initial solutions? Do they use of the same or different search strategies (parameter settings, memory management rules, neighbourhood operators, etc.)? These two questions lead to a four-way classification along this dimension: *Same initial Point, Same search Strategy (SPSS)*; *Same initial Point, Different search Strategies (SPDS)*; *Multiple initial Points, Same search Strategies (MPSS)*; *Multiple initial Points, Different search Strategies (MPDS)*. It should be noted that this dimension had, in fact, been introduced earlier by Voß in his own attempt to classify parallelization schemes for TS [89].

It should be pointed out that, although it was originally developed for TS, this taxonomy could apply equally well to several other classes of metaheuristics and thus possibly constitute a valuable basis for a comprehensive taxonomy of parallel metaheuristics.

1.3.2 More on cooperative search

As we shall see in the next section, the trend in parallel TS has been to move from the low-level parallelism (e.g., the 1C/RS methods) of the early implementations toward more and more complex high-level parallelism schemes. In fact, most recent parallel TS heuristics implement some form of cooperative search. While cooperation seems to offer the most promising avenue for superior performance, it also involves the greatest challenges in terms of algorithm design and it is worth discussing somewhat further before moving on to the literature review.

Cooperative multi-thread TS methods launch several independent *search threads* (each defining a trajectory in the search space) and implement information-exchange mechanisms among these threads. The key challenge in such a context is to ensure that *meaningful* information is exchanged in a *timely* manner between the threads, to allow the global parallel search to achieve a better performance than the simple concatenation of the results of the individual threads, where performance is measured in terms of computing time and solution quality [5, 24].

Toulouse, Crainic, and Gendreau [81] have proposed a list of fundamental issues to be addressed when designing cooperative parallel strategies for metaheuristics:

- What information is exchanged?
- Between what processes is it exchanged?
- When is information exchanged?
- How is it exchanged?
- How is the imported data used?

It is not our intention to address these issues at this point, but it is useful to bear them in mind as we survey the literature in the next section.

1.4 LITERATURE REVIEW

This literature review is divided into four subsections that cover respectively *1*-control heuristics, *p*-control synchronized methods, asynchronous search approaches, and hybrid metaheuristics involving parallel TS. This division also roughly corresponds to a chronological arrangement of the literature, starting with the earliest efforts and finishing with the most recent ones.

1.4.1 *1*-control parallel heuristics

As we already mentioned before in this chapter, early parallel implementations of TS were based on the classical master-slave approach and aimed solely at accelerating the search, thus lowering computing times. This allowed researchers to tackle more effectively difficult problems such as the *quadratic assignment problem* (QAP) [16, 18, 19, 73, 75], the *traveling salesman problem* (TSP) [17], *vehicle routing problems* (VRP) [33], and the *task scheduling problem on heterogeneous systems* [65, 66, 67].

As explained in the previous section, in this type of implementation, a "master" process executes a regular sequential TS procedure, but dispatches computing-intensive tasks to be executed in parallel by "slave" processes. The master receives and processes the information resulting from the slaves' computations, selects and implements moves, updates the search memories, and makes all decisions pertaining to the activation of search strategies (e.g., deciding when to perform intensification or diversification) and to the termination of the search. The search step usually parallelized and assigned to slave processes is the neighbourhood evaluation. At each iteration, the moves that make up the neighbourhood of the current solution are partitioned into as many sets as the number of available slave processors and the evaluation is carried out in parallel by slave processes.

This 1C/RS/SPSS strategy proved quite successful for problems that display large neighbourhoods and relatively low computing requirements to evaluate and perform a given move, such as the ones listed above. In implementations with a relatively small number of processors, near-linear speedups were reported or the same quality of solutions. This approach also permitted, at the time, to improve the best-known solutions for several problem instances proposed in the literature. In fact, the approach has not been totally abandoned: in 1999, Randall and Abramson [68] proposed a general framework for applying to a variety of problem and recently Blazewicz, Moret-Salvador, and Walkowiak [8] used it to tackle two-dimensional cutting problems).

In 1995, Crainic, Toulouse, and Gendreau [25] realized a comparative study of several synchronous TS parallelizations for the location-allocation problem with balancing requirements. Apart from a straightforward 1C/RS/SPSS approach and some *p*-control ones, they also implemented a 1C/KS/SPSS heuristic based on the *sequential fan candidate list* strategy, also known as the *look ahead* or *probing* approach [53, 51]. In this approach, slave processes perform a small number of (look ahead) TS iterations before synchronization, and the selection of the best neighbouring solution from which the next iteration is initiated, is based on the

value of the objective *after the look ahead iterations*. Both the 1C/KS/SPSS and the 1C/RS/SPSS heuristics yielded better solutions than sequential TS on the tested instances, with the KS one being consistently superior to the RS one. To the best of our knowledge, this paper is the only ever to report results on a parallel implementation of the sequential fan candidate list strategy.

Another major parallelization strategy that has been implemented using *1*-control schemes is *search space decomposition*. The basic idea behind this approach is to divide the search space of the problem into several (usually disjoint, but not necessarily exhaustive) sets and to run TS (or any other heuristic or metaheuristic) on each subset, thus accelerating the global search. The approach can be implemented in two fairly different fashions: in the first, all search threads consider complete solutions to the problem, while in the second, they handle partial ones, in which case a complete solution has to be reconstructed at some point. It must stressed, however, that in both cases each search process has only access to a restricted of portion of the search space. Furthermore, the decomposition of the search space is often non-exhaustive, i.e., the union of the search space subsets considered by the slave processes at a given point in time may be significantly smaller than the complete search space. Therefore, to increase the thoroughness of the search and allow all potential solutions to be examined, the decomposition is modified at regular intervals and the search is then restarted using this new decomposition. This strategy is naturally implemented using 1C/KS master-slave schemes (with either an MPSS or MPDS search differentiation strategy): the master process determines the partition, synchronizes slave processes, reconstructs solutions (if required), and determines stopping conditions, while slave processes perform the search on their assigned search space subset. This approach has proved quite successful for problems for which a large number iterations can be performed in a relatively short time and restarting the method with a new decomposition does not require an unreasonable computational effort (see, e.g., Fiechter [31] for the TSP and Laganière and Mitiche [57] for image filtering). An extension of this strategy was used by Gendreau, Laporte, and Semet [43] to solve efficiently in real time several variants of the same problem instance; their method is described in detail in subsection 1.5.1.

1.4.2 *p*-control synchronized parallel heuristics

Independent multi-searches were also among the earliest parallel TS strategies implemented. Most implementations launch several independent search processes from different, often randomly generated, initial solutions. No communications take place between the multiple search threads running in parallel, except once once all processes have stopped, when the best overall solution is identified. As mentioned in subsection 1.3.1, these approaches clearly belong to the pC/RS class of the taxonomy. Note, however, that, in most implementations, a designated processor verifies that the others have completed their search and collects the information. While these procedures, like 1C/RS ones, essentially amount to repeated sequential TS heuristics, they turn out to be effective, simply because of the sheer quantity of computing power they allow one to apply to a given problem. This was indeed established empirically by

several papers, including those of Battiti and Tecchiolli [7] for the QAP, and Taillard [76] for job shop scheduling problems, in which excellent results were obtained when compared to the best existing heuristics at the time. This parallelization of the classic sequential multi-start heuristic is also very easy to implement and remains popular for this reason [10, 79].

As was mentioned in the taxonomy, pC/KS strategies attempt to take advantage of the parallel exploration of the search space by synchronizing search processes at predetermined intervals. They are also generally implemented in a master-slave configuration, in which the designated master process collects information from the other processes at synchronization instants and usually restarts the search from the best solution found so far (see Malek *et al.* [60] for the TSP, and Rego and Roucairol [70] and Rego [69] for the VRP using ejection chains). De Falco *et al.* [30] and De Falco, Del Balio, and Tarantino [29] attempted to overcome the limitations of the master-slave setting, by allowing processes, when they terminate their local search phase, to synchronize and exchange information (best solutions) with processes running on neighbouring processors.

A more sophisticated pC/KS approach was proposed in 1997 by Niar and Fréville [61]. In this pC/KS/MPDS scheme, a master process controls p slaves processors executing synchronous parallel TS threads by dynamically adjusting their respective search strategy parameters according to the results they have obtained so far. Computational results reported for the 0-1 Multi-dimensional Knapsack Problem show that this dynamic adjustment of search parameters is indeed beneficial.

Authors generally report good performance and results, but synchronous cooperative implementations tend to show a poorer performance when compared to asynchronous and even independent searches (see Crainic, Toulouse, and Gendreau [25, 26, 27]), especially when synchronization points are predetermined (as in most existing implementations). This is mainly due to the large computation overheads that must be incurred at synchronization instants when all processes need to wait for the slowest of them. Furthermore, the predefinition of synchronization points makes these strategies less reactive to the progress of the search on any given problem instance than asynchronous approaches. This being said, synchronous cooperative implementations are quite simple to implement and they have yielded good results on several problems.

As pointed out by Crainic [20], there are two other issues that merit further discussion in connection with these pC/KS strategies. The first has to do with the way in which memories are handled in them. In the implementations reported in the literature, memories are emptied at synchronization instants before restarting the search. Considering the central role played my memories in TS, one may wonder whether any precious information is lost when doing so. It might thus be interesting to conduct an investigation to determine if any useful information could be passed from one search phase to the next, how it could be used, and how it might impact on the overall performance of these methods. The second issue concerns specifically pC/KS/SPDS strategies. It has to do with the fact that in those strategies, the new searches that are launched after synchronization usually all restart from the same best known solution, thus concentrating the search in the same region of the search. It is

well known, however, that the main weakness of TS is its tendency to explore a too limited region of the search space, i.e., the search lacks breadth, unless systematic and effective diversification schemes are used. pC/KS/SPDS strategies may therefore end up exaggerating this weakness. Because they use different solutions to restart the search, pC/KS/MPSS and pC/KS/MPDS may be less prone to this problem, but it is not obvious that they do not also suffer from it in an attenuated fashion. This issue would certainly be worth investigating too.

Search space decomposition (see subsection 1.4.1) may also be implemented in a pC/KS framework, as in Taillard's early TS heuristic for the VRP [74, 75]. In this implementation, customers are partitioned on a geographical basis and vehicles are allocated to the resulting regions to create smaller VRP instances. Each subproblem is then solved by an independent TS procedure. These independent search processes stop after a number of iterations that varies according to the total number of iterations already performed. The partition is then modified by an information exchange phase, during which tours, undelivered cities, and empty vehicles are exchanged between processes handling adjacent regions. Taillard's results at the time were excellent, but his approach did require considerable computing time (in fact, he did not even report computing times, but these are known to be quite substantial). This is not surprising considering the fact that, as the other pC/KS strategies described above, it had to incur the inherent overhead stemming from synchronization.

1.4.3 Asynchronous methods

Historically, independent and synchronous cooperative methods were the first multi-thread search approaches to be developed. However, because of the shortcomings of these methods, which we have discussed at length in the previous subsections, researchers have increasingly turned their attention to asynchronous procedures, which now largely define the "state-of-the-art" in parallel multi-thread search. These asynchronous procedures all follow the same general pattern: starting from possibly different initial solutions and using possibly different tabu (or other) search strategies, p threads explore simultaneously the search space. As indicated in subsection 1.3.1, they belong either to the pC/C or to the pC/KC class of the taxonomy, the main difference between the two being whether or not any "new" knowledge is inferred on the basis of the information exchanged between the search threads.

A key issue in the development of asynchronous procedures is the definition of effective mechanisms to allow the asynchronous exchange of information between the search threads. The simplest scheme for achieving this is simply by having threads engage in communication when some triggering events occur, such as the discovery of a globally improving solution. One may implement, for instance, a "broadcast and replace" strategy: when a search thread improves the global best solution, this solution is broadcast to all other processes that then their own exploration and restart it from this new solution (alternatively, processes might broadcast every locally improving solution, but this clearly increases significantly the communication overhead). This type of approach was applied successfully to complex vehicle routing problems by Attanasio *et al.* [2] and Caricato *et al.* [14].

Crainic [20] observes that these methods in which the "local" exploration phase of processes can be interrupted do not always yield good results. In fact, as he correctly points out, cooperative metaheuristics with unrestricted access to shared knowledge may experience serious premature "convergence", especially when the shared knowledge reduces to one solution only (the overall best or the new best from a given thread): eventually, all threads end up exploring the same restricted region of the search space, thus forfeiting one of the main potential of parallel cooperative search, that is, search breadth. For a more detailed discussion of these issues, see [84, 82, 83, 85].

Most asynchronous implementations of parallel TS do, however, handle information exchange rather differently in order to avoid the pitfall mentioned above. In these approaches, communications are *controlled*, i.e., they occur rather infrequently and only at well-specified stages of the exploration conducted by the individual threads. Exchanges of information take place through some form of *memory* (or *blackboard*) that is used to store various information on solutions and/or solution components. In most cases, the main information stored in the memory is a list of best known or *elite* solutions. The memory is then often referred to as the *central memory*, the *solution pool*, the *solution warehouse* or, even, the *reference set*. In other cases, only partial solutions are recorded and the memory is then referred to as the *adaptive memory*. This terminology was coined in 1995 by Rochat and Taillard in a seminal paper [71], in which they proposed (sequential) TS heuristics for the classical Vehicle Routing Problem and the Vehicle Routing Problem with time windows (VRPTW) that are still among the most effective ones for both problems. The main idea in the adaptive memory approach is to record in a structure the individual components (in routing problems, the vehicle routes) making up elite solutions as they are found. These components are kept sorted in the adaptive memory with respect to the objective function value of the solution they belong to. When the search stagnates and needs to be restarted from a new solution, this solution is constructed by combining randomly-selected routes from the adaptive memory. In almost all cases, the new solution will be made up of routes from different elite solutions (in what could be interpreted, in genetic algorithms terminology, as a multi-parent crossover operation), thus inducing a powerful diversification effect. For more on adaptive memory concepts, see Glover [48] and Taillard *et al.* [78]. The adaptive memory approach is eminently amenable to parallelization, since the search threads can all feed a single, common adaptive memory from which new solutions that naturally combine information from different search threads can be constructed. It has been applied very successfully to the VRPTW by Badeau *et al.* [3] and to real-time vehicle routing and dispatching by Gendreau *et al.* [40]; this latter application is described in more detail in the next section. A similar approach was used with good results by Schulze and Fahle [72] to solve the VRPTW: all the routes generated by the TS threads are collected in a pool, but they are recombined by solving a set covering heuristic whenever a new solution is needed.

Badeau *et al.* [3] also report a number of other interesting findings. First, the performance of their method with respect to the quality of the solution is almost independent of the number of search processes (as long as this number remains

within reasonable bounds) for a fixed computational effort (measured in terms of the overall number of calls to the adaptive memory by all search threads). Second, while traditional parallelization schemes rely on a one-to-one relationship between actual processors and search processes, it turned out that their method did run significantly faster when using more search processes than the number of available processors, because this allowed to overcome the bottlenecks created when several threads were trying to access simultaneously the processor on which the adaptive memory was located. Furthermore, computational evidence showed that it is not, in general, a good idea to run a search thread concurrently with the adaptive memory management procedure on the same processor. These are lessons that should be kept in my mind when developing asynchronous multi-thread procedures, whether they use adaptive memory or not.

To the best of our knowledge, Crainic, Toulouse, and Gendreau were the first, in 1995, to propose a central memory approach for asynchronous TS in their heuristics for multi commodity location with balancing requirements [26]. In their method, individual TS record their current solution into central memory whenever it improves on their *local best* solution (i.e., the best solution found up to that point by the same thread), but they only import a solution from the central memory when they are about to undertake a diversification phase. If the imported solution is better than the current local best one, it replaces it. Diversification then proceeds from the (possibly modified) local best solution. It is important to note that the memories local to search threads are never re-initialized. Five strategies for retrieving a solution from the pool when requested by an individual thread were tested. When few (4) processors were used, the strategy that returns the overall best solution produced the best results. When the number of processors was increased, the best performance was achieved by a probabilistic procedure that selects solutions on the basis of their the rank in the pool. The parallel procedure improves the quality of the solution and also requires less (wallclock) computing time compared to the sequential version, particularly for large problems with many commodities. The same approach was applied to the fixed cost, capacitated, multi commodity network design problem with similar results [22]. Over the last few years, several other authors have implemented fairly similar approaches to a variety of problems, including the partitioning of integrated circuits for logical testing [1], two-dimensional cutting [8], the loading of containers [11], and labor constrained scheduling [15].

Another attempt to overcome the problems stemming from the uncontrolled sharing of information is the so-called *multi-level cooperative search* proposed by Toulouse, Thulasiraman, and Glover [87, 86]. This approach is based on the principle of the controlled diffusion of information, which is achieved by having search processes work at different aggregation levels of the original problem and by allowing communications only between processes at immediately adjacent aggregation levels. These communications consists in asynchronous exchanges of improving solutions at various moments dynamically determined by each process according to its own logic, status, and search history. Communications are further limited by the fact that an imported solution will not be transmitted further until a number of iterations have

been performed. The approach has proved very successful for graph partitioning problems [62, 63].

1.4.4 Hybrids involving parallel Tabu Search

In his 2001 review of recent advances in TS, Gendreau [37] mentioned that hybrid metaheuristics, which combine principles from two or more families of metaheuristics, were probably one of the most exciting developments in the field. The main motivation for developing such hybrid methods is that it is hoped they will display the desirable properties of all the original "pure" methods that they draw upon. For instance, one may replace the mutation operator of a *Genetic Algorithm* (GA), thus creating a GA-TS hybrid, with the objective of achieving simultaneously search aggressiveness (the TS component) and breadth (the GA component) in the exploration of the search space. Hybrids involving parallel TS and other metaheuristics are not as recent as our earlier statement may suggest. In fact, a method combining the principles of TS and of *Hopfield-Tank neural networks* to solve simple plant location problems on massively parallel architectures was proposed by Vaithyanathan, Burke, and Magent [88] as early as 1996, and they may have been earlier parallel TS hybrids.

Broadly speaking, hybrid metaheuristics can be divided into three main classes. The first class is made up of methods that sequentially apply metaheuristics of different families to a given problem. The two-phase approach of Gehring and Homberger for the VRPTW [34, 35, 36] is a typical example of such a method: it first applies an evolution strategy to reduce the number of vehicles required by a solution and then TS to minimize total travel distance. The parallelization is a multi-thread cooperative one in which each process executes the two-phase heuristic (possibly with different parameters) and information exchanges take place according to the asynchronous central memory strategy of the previous subsection. Bastos and Ribeiro [6] describe a somewhat different two-phase hybrid for the Steiner problem: in their approach, a parallel multi-thread reactive TS phase (using again the asynchronous central memory strategy) is followed by a distributed *Path Relinking* (PR) [49, 51, 52] phase, i.e., all processes switch from TS to PR simultaneously.

In the second class of hybrid metaheuristics, the algorithmic elements of the original methods are assembled into a single (monolithic), complex algorithmic design, as in the example provided at the beginning of this subsection. An illustration of this class of methods is provided in the TS-PR hybrid of Gallego, Romero, and Monticelli [32] for transmission network expansion planning. In this heuristic, the diversification step of a parallel multi-thread TS is implemented, in some situations, by applying PR instead of rather simple modifications of solutions. Other hybrids of this class include those of Talbi *et al.* [80], which applies Simulated Annealing (SA) principles in the intensification step of a multi-start TS procedure for the QAP, of Jozefowiez, Semet, and Talbi [56], which integrates evolutionary algorithms and parallel TS to solve multi-objective vehicle routing problems, and of Baños et al. [4], which combines TS, SA, and multi-level cooperative search to tackle graph partitioning problems.

The third class of hybrids exploits differently the hybridization concept by relying on parallel multi-agent search architectures in which individual agents run "pure" methods, but exchange information among themselves. Crainic and Gendreau [21] proposed such a hybrid search strategy by adding a GA thread to their asynchronous multi-thread TS heuristic for multicommodity location-allocation with balancing requirements [26]. This distinct GA thread is launched when a certain number of elite solutions have been recorded in the central memory of the parallel TS, using these solutions as its initial population. Asynchronous migration subsequently transfers the best solution of the genetic pool to the parallel TS central memory, as well as solutions of the central memory toward the genetic population. This strategy did perform well, especially on larger instances. An interesting observation of this study was that the best overall solution was never found by the GA thread, but that its inclusion allowed the TS threads to find better solutions. It is hypothesized that this superior performance stemmed from a more effective diversification of the search. This scheme was recently further refined by Le Bouthillier and Crainic [59] who combined several different GA (i.e., with different parent selection and crossover operators) and TS threads in their hybrid metaheuristic for the VRPTW. In this implementation, there is one central memory that is common to all threads. Computational results show that, without any particular calibration, the parallel metaheuristic is competitive with the best metaheuristics available, and demonstrates almost linear speedups.

The hybridization of parallel TS approaches with population-based methods such as GA, PR, and Scatter Search [49, 51, 52, 58] appears very promising, because it addresses what is probably the greatest weakness of TS, namely its tendency to remain confined in a too small area of the search space. The third class of hybrids seems particularly attractive with respect to the benefits it has to offer for a relatively low implementation effort.

1.5 TWO PARALLEL TABU SEARCH HEURISTICS FOR REAL-TIME FLEET MANAGEMENT

Real-time fleet management problems are found in numerous transportation and logistics applications. In the following, we describe two parallel TS heuristics for dispatching fleets of vehicles in the context of emergency services and courier services, respectively. Given that fast response times are required in these cases, parallel implementations are particularly indicated since they allow for more optimization work to be performed within the allotted time.

1.5.1 Real-time ambulance relocation

In the work of Gendreau, Laporte, and Semet [43], a real-time redeployment problem for a fleet of ambulances is addressed. Basically, when a call is received, an ambulance is first assigned to it. This assignment is done through the application of relatively simple dispatching rules. Then, the remaining available ambulances can be relocated

to other waiting sites to provide a better coverage of the demand. The latter problem is tackled with a TS heuristic that moves ambulances between potential waiting sites. The objective is to maximize the proportion of demand covered by at least two vehicles within a given (time) radius, minus relocation penalties. The problem-solving approach exploits a TS heuristic previously developed for a static ambulance location problem by the same authors [42].

As life or death dispatching and relocation decisions must be taken under considerable time pressure in the real-time context, a parallel implementation is proposed to speed up the decision process. Basically, the available time between two emergency calls is exploited to precompute possible scenarios. More precisely, for each site currently occupied by an available ambulance, a relocation plan is computed with TS by assuming that an ambulance from this site will be assigned to the next incoming call. When a new call occurs, an ambulance is assigned according to a given dispatching rule and the precomputed redeployment scenario associated with the site of this ambulance is then directly applied. If there is not enough time to compute a complete solution before the next call, then no redeployment takes place after the ambulance assignment.

The parallel algorithm is based on a pure master-slave scheme. The master manages global data structures with precalculated information on each ambulance and sends the relocation problems associated with each occupied site to the slaves. The CPU time allotted to each slave for solving a problem is strictly controlled by fixing the number of iterations in TS. This number is based on the frequency of calls (e.g., low frequency implies that more CPU time can be alloted to the search). When every problem has been solved once, a new attempt to improve the solutions is performed, using a larger number of iterations.

This algorithm has been implemented on a network of SUN UltraSparc workstations, using simulated data based on real-life call distributions on the island of Montreal, Canada. The results that were obtained demonstrate the suitability of this algorithm. In the simulations, every call was serviced within the required time range of 15 minutes and 98% of urgent calls were serviced within 7 minutes, with an average of 3.5 minutes (the current practice in Montreal requires that 90% of the urgent calls should be responded to within 7 minutes). Furthermore, in 95% of the cases, the algorithm succeeded in precomputing a complete redeployment strategy before the occurrence of the next call.

1.5.2 Real-time vehicle routing and dispatching for courier services

The problem considered by Gendreau *et al.* [40] is motivated from courier services where customer requests for the transportation of small items (e.g., letters, small parcels) must be accommodated in real-time and incorporated into the current planned routes of a fleet of vehicles. A planned route here corresponds to the sequence of requests that have been assigned to a given vehicle but have not been serviced yet. Due to the presence of soft time constraints for servicing a customer, the problem is modeled as an uncapacitated Vehicle Routing Problem with Soft Time Windows (VRPSTW). The objective function to be minimized relates to the total distance

traveled (or total travel time) for servicing the customers plus penalties for lateness at customer locations.

As in subsection 1.5.1, the problem-solving approach exploits a TS heuristic previously developed for a static version of the problem, where all customer requests are known in advance [77]. Thus, a series of static problems are solved over time, based on the current planned routes. A new static problem is defined each time an input update occurs, due to the arrival of a new request, and TS is applied on this problem until the next input update.

The general problem-solving framework, within which the TS heuristic is embedded, is described below. In this description, it is assumed that a certain number of "static" requests are known at the start of the day (those have been received the previous day, but too late to be accommodated the same day). These requests are incorporated into initial planned routes that are used to start the search process.

** Initialization **

1. Generate different initial solutions with the static requests using a constructive (insertion) heuristic.
2. Apply the TS heuristic to these solutions and store the resulting routes in adaptive memory.

** Search algorithm **

3. For a number of iterations do:
   - 3.1 Construct a starting solution by combining routes in adaptive memory.
   - 3.2 For a number of iterations do:
     - Decompose the set of planned routes in the current solution into disjoint subsets of routes.
     - Apply TS to each subset of routes.
     - Merge the resulting routes to create the new current solution.
   - 3.3 Apply a post optimization procedure to each individual route.
   - 3.4 Add the resulting routes to the adaptive memory (if the solution is good enough).

As we can see, an adaptive memory stores the best solutions found during the search. The routes in these solutions are then used to feed the TS with new starting points. The optimization is performed by the latter by moving customers between routes. It should be noted that search space decomposition takes place and that a distinct TS is applied to each subset of routes in the current decomposition. These subsets are then merged together to form a complete solution. After a number of decompositions, each individual route in the final solution is further improved with a specialized heuristic for the Traveling Salesman Problem with Time Windows [41].

A two-level parallelization scheme is proposed to implement this problem-solving framework.

1. The so-called "Search algorithm" is launched in parallel on multiple processors, thus implementing a multi-thread search. At this upper level, we have asynchronous cooperative threads that communicate through a common adaptive memory, as they all feed and fetch solutions from this memory. In the taxonomy, this approach corresponds to a pC/KC/MPSS parallelization scheme.

2. Within each search thread, search space decomposition is realized in parallel through a master-slave implementation, where each slave runs a TS on a different subset of routes. After a given number of search iterations, each slave returns its best (partial) solution. The partial solutions are then merged together to obtain a complete solution. At the end, the best solution found is sent for possible inclusion in the adaptive memory. We thus have a 1C/KS/MPSS parallelization scheme at this level.

This algorithm has been run in a coarse-grained parallel environment, namely a network of SUN Sparc workstations. The results on simulated data have shown that the TS-based optimization procedure provides substantial benefits over simpler dispatching approaches. Through an appropriate adaptation of the basic neighborhood operator that moves customers between routes, it is possible to exploit this framework to solve either problems where a customer request involves a single location or both a pick-up and a delivery location [39].

In the work reported above, the current destination of each vehicle was not included in the planned routes and could not be moved by the neighborhood operator. Hence, a vehicle in movement had to reach its planned destination. In a more recent development, Ichoua, Gendreau, and Potvin [55] included the current destination of each vehicle in planned routes. Upon request arrival, it is thus possible for TS to insert the new request between the current position of a given vehicle and its planned destination, thus redirecting (or *diverting*) the vehicle to serve the new request. As the current destination of every vehicle can now be freely moved around by TS, the solution obtained can also modify the current destination of other vehicles, if it leads to a better solution. The proposed algorithm thus implements a generalized form of diversion that might involve more than one vehicle. As vehicles are moving fast, exploiting redirection opportunities when new requests occur must be realized under considerable time pressure. A parallel implementation is thus particularly appropriate in this context.

The interested reader will find in Attanasio *et al.* [2] the description of another parallel TS heuristic for a real-time vehicle routing and dispatching problem. The algorithm is developed in the context of a dial-a-ride system where people are transported. Different issues related to real-time vehicle routing and parallel computing can also be found in the recent paper of Ghiani, Guerriero, and Laporte [44].

1.6 PERSPECTIVES AND RESEARCH DIRECTIONS

Parallel Tabu Search has come a long way since its early beginnings fifteen years ago when parallel implementations of TS almost exclusively focused on the parallelization of the neighbourhood evaluation step. The main trend is now clearly toward asynchronous cooperative multi-thread methods and hybrids, which attempt to bring to bear on a given problem all the algorithmic machinery that is at hand. Throughout this evolution, our understanding of the key features that make a particular implementation successful has significantly deepened (for instance, most experimented researchers in the area are now keenly aware of the efficiency loss inherent in synchronous search models), but a lot of a research is still required in order to fully understand the subtle interactions that occur when using complex cooperative strategies, whether in a "pure" TS scheme or in a hybrid one. One step in that direction would be the definition and collection of relevant *performance measures and statistics*. Such measures are acutely needed to track down what really takes places in complex cooperative schemes. Furthermore, if used at execution time, they could help improve their efficiency. For instance, unproductive search threads could be identified and then terminated or redirected. Statistics could also be attached to the elite solutions in the pool (e.g., measures related to the quality of the solutions visited by search threads starting from these elite solutions) to better target the most promising regions of the search space.

Apart from the questions related to the design of effective search strategies, the evolution toward complex parallel TS schemes has created formidable challenges with respect to the actual implementation of the methods. To be quite honest, implementing from scratch any of the heuristics described in the latter parts of our survey requires considerable programming skills. Fortunately, over the last few years, one has witnessed the emergence of dedicated environments for the development of parallel TS algorithms and hybrids (e.g., [9, 13]). These environments provide *skeletons* (templates) or *frameworks* that one may instantiate to obtain an algorithm for tackling a specific problem with a given search strategy. In all cases, a strict separation between the generic and the problem-specific parts of algorithm is enforced. We believe that these environments are an essential step in the further development of parallel TS approaches and we expect to see more of them being proposed in the coming years.

As a final remark, we would like to recall that parallel TS has proved over time to be an extremely effective metaheuristic for tacking a large variety of very difficult combinatorial optimization problems, especially in a real-time context. With the exciting recent developments in the field, it should remain so for many years.

Acknowledgments

Funding for this project has been provided by the Natural Sciences and Engineering Council of Canada and by the Fonds FQRNT of the Province of Québec.

REFERENCES

1. R. M. Aiex, S. L. Martins, C. C. Ribeiro, and N. R. Rodriguez. Cooperative Multi-Thread Parallel Tabu Search with an Application to Circuit Partitioning. In *Proceedings of IRREGULAR'98 - 5th International Symposium on Solving Irregularly Structured Problems in Parallel, Lecture Notes in Computer Science*, volume 1457, pages 310–331, 1998. Springer-Verlag.

2. A. A. Attanasio, J.-F. Cordeau, G. Ghiani, and G. Laporte. Parallel Tabu Search Heuristics for the Dynamic Multi-Vehicle Dial-a-Ride Problem. *Parallel Computing*, 30:377–387, 2004.

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

4. R. Baños, C. Gil, J. Ortega, and F. G. Montoya. Cooperative Parallel Tabu Search for Capacitated Network Design. *Journal of Heuristics*, 10(3):315–336, 2004.

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

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

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

8. J. Blazewicz, A. Moret-Salvador, and R. Walkowiak. Parallel tabu search approaches for two-dimentional cutting. *Parallel Processing Letters*, 14(1):23–32, 2004.

9. M. J. Blesa, L. Hernàndez, and F. Xhafa. Parallel Skeletons for Tabu Search Method Based on Search Strategies and Neighborhood Partition. In R. Wyrzykowski, J. Dongarra, M. Paprzycki, and J. Waniewski, editors, *Parallel Processing and Applied Mathematics : 4th International Conference (PPAM 2001), Lecture Notes in Computer Science*, volume 2328, pages 185–193, Naleczow, Poland, 2002. Springer-Verlag.

10. S. Bock and O. Rosenberg. A New Parallel Breadth First Tabu Search Technique for Solving Production Planning Problems. *International Transactions in Operational Research*, 7(6):625–635, 2000.

11. A. Bortfeldt, H. Gehring, and D. Mack. A parallel tabu search algorithm for solving the container loading problem. *Parallel Computing*, 29:641–662, 2003.

12. W. Bozejko and M. Wodecki. Solving the flow shop problem by parallel tabu search. In *Proceedings, International Conference on Parallel Computing in Electrical Engineering, PARELEC '02*, pp. 189–194, 2002.

13. S. Cahon, N. Melab, and E.-G. Talbi. ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics. *Journal of Heuristics*, 10(3): 357–380, 2004.

14. P. Caricato, G. Ghiani, A. Grieco, and E. Guerriero. Parallel tabu search for a pickup and delivery problem under track contention. *Parallel Computing*, 29:631-639, 2003.

15. C. C. B. Cavalcante, V. C. Cavalcante, C. C. Ribeiro, and C. C. de Souza. Parallel Cooperative Approaches for the Labor Constrained Scheduling Problem. In C. C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 201–225, 2001. Kluwer Academic Publishers.

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

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

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

19. J. Chakrapani and J. Skorin-Kapov. 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, 1995. Kluwer Academic Publishers.

20. T. G. Crainic. Parallel Computation, Co-operation, Tabu Search. In C. Rego and B. Alidaee, editors, *Metaheuristic Optimization via Memory and Evolution: Tabu Search and Scatter Search*, 2004 (forthcoming). Kluwer Academic Publishers.

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

22. T. G. Crainic and M. Gendreau. Cooperative Parallel Tabu Search for Capacitated Network Design. *Journal of Heuristics*, 8(6):601–627, 2002.

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

24. T. G. Crainic and M. Toulouse. Parallel Strategies for Meta-heuristics. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 475–513, 2003. Kluwer Academic Publishers.

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

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

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

28. V.-D. Cung, S. L. Martins, C. C. Ribeiro, and C. Roucairol. Strategies for the Parallel Implementations of Metaheuristics. In C. C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 263–308, 2001. Kluwer Academic Publishers.

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

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

31. C.-N. Fiechter. A Parallel Tabu Search Algorithm for Large Travelling Salesman Problems. *Discrete Applied Mathematics*, 51(3):243–267, 1994.

32. R. A. Gallego, R. Romero, and A. J. Monticelli. Tabu Search Algorithm for Network Synthesis. *IEEE Transactions on Power Systems*, 15(15):490–495, 2000.

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

34. H. Gehring and J. Homberger. A Parallel Hybrid Evolutionary Metaheuristic for the Vehicle Routing Problem with Time Windows. In K. Miettinen, M. M. Mäkelä and J. Toivanen, editors, *Proceedings of EUROGEN99 – Short Course on Evolutionary Algorithms in Engineering and Computer Science*, pages 57–64, Jyväskylä, Finland, 2002.

35. H. Gehring and J. Homberger. Parallelization of a Two-Phase Metaheuristic for Routing Problems with Time Windows. *Asia-Pacific Journal of Operational Research*, 18:35–47, 2001.

36. H. Gehring and J. Homberger. Parallelization of a Two-Phase Metaheuristic for Routing Problems with Time Windows. *Journal of Heuristics*, 8(3):251–276, 2002.

37. M. Gendreau. Recent Advances in Tabu Search. In C. C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 369–377, 2001. Kluwer Academic Publishers.

38. M. Gendreau. An Introduction to Tabu Search. In F. Glover and G. A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 37–54, 2003. Kluwer Academic Publishers.

39. M. Gendreau, F. Guertin, J.-Y. Potvin, and R. Séguin. Neighborhood Search Heuristics for a Dynamic Vehicle Dispatching Problem with Pick-ups and Deliveries. Technical Report CRT-98-10, Centre de recherche sur les transports, Université de Montréal, 1998.

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

41. M. Gendreau, A. Hertz, G. Laporte, and M. Stan. A Generalized Insertion Heuristic for the Traveling Salesman Problem with Time Windows. *Operations Research*, 46:330–335, 1998.

42. M. Gendreau, G. Laporte, and F. Semet. Solving an Ambulance Location Model by Tabu Search. *Location Science*, 5:75–88, 1997.

43. M. Gendreau, G. Laporte, and F. Semet. A Dynamic Model and Parallel Tabu Search Heuristic for Real-Time Ambulance Relocation. *Parallel Computing*, 27:1641–1653, 2001.

44. G. Ghiani, G. Guerriero, G. Laporte, and R. Musmanno. Real-Time Vehicle Routing: Solution Concepts, Algorithms and Parallel Computing Strategies. *European Journal of Operational Research*, 151:1–11, 2003.

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

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

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

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

49. F. Glover. A Template for Scatter Search and Path Relinking. In J. K. Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, editors, *Artificial Evolution*, *Lecture Notes in Computer Science*, volume 1363, pages 13–54, 1997. Springer Verlag.

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

51. F. Glover and M. Laguna. *Tabu Search*, 1997. Kluwer Academic Publishers.

52. F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39(3):653–684, 2000.

53. F. Glover, É. D. Taillard, and D. de Werra. A User's Guide to Tabu Search. *Annals of Operations Research*, 41:3–28, 1993.

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

55. S. Ichoua, M. Gendreau, and J.-Y. Potvin. Diversion Issues in Real-Time Vehicle Dispatching. *Transportation Science*, 34:426–438, 2000.

56. N. Jozefowiez, F. Semet, and E.-G. Talbi. Parallel and Hybrid Models for Multi-objective Optimization: Application to the Vehicle Routing Problem. In J. J. Merelo Guervós, P. Adamidis, H.-G. Beyer, J.-L. Fernández-Villacañas, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VII: 7th International Conference*, *Lecture Notes in Computer Science*, volume 2439, Granada, Spain, 2002. Springer-Verlag.

57. R. Laganière and A. Mitiche. Parallel Tabu Search for Robust Image Filtering. *Proceedings of IEEE Workshop on Nonlinear Signal and Image Processing*, volume 2, pages 603–605, Greece, 1995.

58. M. Laguna and R. Martí. *Scatter Search: Methodology and Implementations in C*, 2003. Kluwer Academic Publishers.

59. A. Le Bouthillier and T. G. Crainic. A Cooperative Parallel Meta-Heuristic for the Vehicle Routing Problem with Time Windows. *Computers & Operations Research*, 2004.

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

61. S. Niar and A. Fréville. A Parallel Tabu Search Algorithm For The 0-1 Multidimensional Knapsack Problem. In *11th International Parallel Processing Symposium (IPPS '97)*, Geneva, Switzerland, 1997.

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

63. M. Ouyang, M. Toulouse, K. Thulasiraman, F. Glover, and J. S. Deogun. Multi-level Cooperative Search for the Circuit/Hypergraph Partitioning Problem. *IEEE Transactions on Computer-Aided Design*, 21(6):685–693, 2002.

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

65. S. C. S. Porto, J. P. F. W. Kitajima, and C. C. Ribeiro. Performance Evaluation of a Parallel Tabu Search Task Scheduling Algorithm. *Parallel Computing*, 26:73–90, 2000.

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

67. S. C. S. Porto and C. C. Ribeiro. A Case Study on Parallel Synchronous Implementations of Tabu Search Based on Neighborhood Decomposition. *Investigación Operativa*, 5:233–259, 1996.

68. M. Randall and D. Abramson. General Parallel Tabu Search Algorithm for Combinatorial Optimisation Problems In W. Cheng and A. Sajeev, editors, *PART99: Proceedings of the 6th Australasian Conference on Parallel and Real Time Systems*, pages 68–79, Singapore, 1999. Springer-Verlag.

69. C. Rego. Node Ejection Chains for the Vehicle Routing Problem: Sequential and Parallel Algorithms. *Parallel Computing*, 27:201–222, 2001.

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

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

72. J. Schulze and T. Fahle. A Parallel Algorithm for the Vehicle Routing Problem with Time Window Constraints. *Annals of Operations Research*, 86:585–607, 1999.

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

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

75. É. D. Taillard. *Recherches itératives dirigées parallèles*. Ph.D. dissertation, École Polytechnique Fédérale de Lausanne, 1993.

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

77. É. D. Taillard, P. Badeau, M. Gendreau, and J.-Y. Potvin. A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows. *Transportation Science*, 31(2):170–186, 1997.

78. É. D. Taillard, L. M. Gambardella, M. Gendreau, and J.-Y. Potvin. Adaptive memory programming: a unified view of metaheuristics. *European Journal of Operational Research*, 135:1–16, 1997.

79. E.-G. Talbi, Z. Hafidi, and J.-M. Geib. Parallel adaptive tabu search approach. *Parallel Computing*, 24:2003–2019, 1998.

80. E.-G. Talbi, Z. Hafidi, D. Kebbal, and J.-M. Geib. A fault-tolerant parallel heuristic for assignment problems. *Future Generation Computer Systems*, 14:425–438, 1998.

81. M. Toulouse, T. G. Crainic, and M. Gendreau. 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, 1996. Kluwer Academic Publishers.

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

83. M. Toulouse, T. G. Crainic, and B. Sansó. Systemic Behavior of Cooperative Search Algorithms. *Parallel Computing*, 21(1):57-79, 2004.

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

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

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

87. M. Toulouse, K. Thulasiraman, and F. Glover. Multi-Level Cooperative Search: A New Paradigm for Combinatorial Optimization and an Application to Graph Partitioning. In P. Amestoy, P. Berger, M. Daydé, I. Duff, V. Frayssé, L. Giraud, and D. Ruiz, editors, *5th International Euro-Par Parallel Processing Conference*, *Lecture Notes in Computer Science*, volume 1685, pages 533–542, 1999. Springer-Verlag.

88. S. Vaithyanathan, L. I. Burke, and M. A. Magent. Massively parallel analog tabu search using neural networks applied to simple plant location problems. *European Journal of Operational Research*, 93(2):317–330, 1996.

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