Towards a Taxonomy of Parallel Tabu Search Heuristics

Teodor Gabriel Crainic
Centre de recherche sur les transports
Université de Montréal
and
Département des sciences administratives
Université du Québec à Montréal

Michel Toulouse
Centre de recherche sur les transports
and
Ecole Polytechnique
Université de Montréal

Michel Gendreau
Centre de recherche sur les transports
and
Département d'informatique et de recherche opérationnelle
Université de Montréal

February 1995

Abstract

In this paper we present a classification of parallel tabu search metaheuristics based, on the one hand, on the control and communication strategies used in the design of the parallel tabu search procedures and, on the other hand, on how the search space is partitionned. These criteria are then used to review the parallel tabu search implementations described in the literature. The taxonomy is further illustrated by the results of several parallelization implementations of a tabu search procedure for multicommodity location-allocation problems with balancing requirements.

Key words: Tabu search metaheuristics, Parallelization strategies, Taxonomy

Résumé

Nous présentons un schéma de classification des algorithmes parallèles de recherche avec tabous. La taxonomie est basée, d'une part, sur les stratégies de contrôle et de communication des algorithmes parallèles de recherche avec tabous et, d'autre part, sur les règles de partitionnement du domaine. Ces critères sont ensuite utilisés lors de la revue des réalisations décrites dans la littérature et sont illustrés de façon plus détaillée par les résultats d'une étude de plusieurs approches de parallélisation d'un algorithme de recherche avec tabous pour le problème de localisation-allocation multiproduits avec des demandes d'équilibrage.

Mots clés: Méthodes de recherche avec tabous, stratégies de parallélisation, schéma de classification.

1 Introduction

Tabu search [15, 16, 17, 18] is often described as a *higher level* heuristic for solving optimization problems, designed to guide other heuristics, or parts thereof, to avoid the trap of local optimality. Thus, tabu search is an adaptive search technique that aims to intelligently explore the solution space in search of good, hopefully optimal, solutions. Broadly speaking, two mechanisms are used to direct the search trajectory. The first is intended to avoid cycling through the use of *tabu lists* that keep track of recently examined solutions. The second mechanism makes use of one or several *memories* to direct the search either into a thorough exploration of a promising neighbourhood, or towards previously unexplored regions of the solution space.

It is noteworthy that these memory mechanisms may be viewed as *learning* capabilities that gradually build up images of good or promising solutions. The existence of these learning capabilities and guidance mechanism implies, on the one hand, that the knowledge tabu search supplies about the instance of the problem to be solved is richer than, say, the one generated during execution of a branch-and-bound algorithm for the same problem. On the other hand, it also clearly differentiate tabu search from random search by introducing a purposeness into the process of domain exploration. Hence, a wider gamut of tabu search procedures may be designed for a given class of problems, and this characteristic is emphasized when parallel implementations are contemplated.

Parallel computer architectures offer the possibility to design procedures that explore more efficiently the solution space. Generally, this extra efficiency may be achieved by accelerating some particularly tedious computational phases of the algorithm, or by redesigning the algorithm. In the context of branch-and-bound algorithms, Trienekens and Bruin [27] refer to these approaches as *low* and *high level parallelization*, respectively, because a low level parallel implementation of an algorithm does not change the interactions between its various parts; hence, it is not intrinsically different from its sequential version, only faster. In the context of tabu search, this distinction may become significantly more blurred. In particular, one has to consider how the parallelization strategy affects the information relative to the global search trajectory and history, as well as how much of this knowledge is available to each process at any given moment. Hence, issues relative to inter-process information exchanges and treatment, central to the design of any parallel procedure, take on an even more preeminent position when parallel tabu search is considered.

The taxonomy that we propose explicitly addresses these considerations by incorporating classification criteria based not only on how the search space is partitioned, but also on the control and communication strategies used in the design of the parallel tabu search procedures. It thus aims to present the first comprehensive picture of parallelization strategies for tabu search, and contributes toward performing a more meaningful analysis and comparison of the various procedures proposed in the literature. The taxonomy may also help better understand the relationships between the nature of tabu search, especially its knowledge acquisition and utilization features, and parallel computation. Finally, it identifies new parallelization strategies, and suggests interesting future work.

The next section details the taxonomy and its criteria, while Section 3 is dedicated to a review of the main strategies proposed in the literature according to the parameters of the proposed classification. Finally, Section 4 further illustrates the taxonomy by using results from several parallel implementations of the same sequential tabu search algorithm for multicommodity location-allocation problems with balancing requirements, and shows that several other parallelization strategies, besides those usually found in the literature, may be advantageously used to develop efficient parallel tabu search procedures.

2 Classification of Parallel Approaches

There exists so far only a limited body of knowledge concerning the design of parallel tabu search methods, and we are aware of only one attempt (Voß[28]) to classify the different types of parallelism that may be applied in this context. Voß's classification is based on an analogy to the classical taxonomy of parallel machine models proposed by Flynn [12]. It discriminates parallel algorithms into four categories according to the choice of identical or different initial solutions, and of identical or different exploration strategies for each process. In our opinion, this classification is incomplete since it fails to account both for the differences in control and communication strategies which are so important when designing parallel algorithms, and for the various mechanisms used to exchange and process information central to tabu search metaheuristics. The taxonomy we present aims to fill these gaps.

2.1 The Tabu Search Approach

We briefly recall the main components of tabu search. For more detailed descriptions of the method, as well as for reviews of successful applications, see Glover [15, 16, 17], Glover and Laguna [18], Glover, Taillard and de Werra [19], and references therein.

A schematic tabu search procedure for solving an optimization problem

$$(P) \ \ \text{Minimize} \ \ f(x) \ \ \text{subject to} \ \ x \in X \subseteq R^n$$

may be viewed as the combination of three main steps: local search, intensification of the search in a selected subregion, moving the search to a previously unexplored region. While exploring the domain according to the rules of one of these procedures, knowledge is collected, stored and processed in order to gain an understanding of the problem and its domain, to extract an image of a good solution, to identify regions where such good solutions might be found, and to guide the search.

Typically, *local search* is performed by evaluating moves from a current solution $\bar{x} \in X$. A *move* is any procedure that allows to pass from a solution to (P) to a different solution to (P) (in some applications, either one of these solutions, or both, may be allowed to be infeasible). All solutions that may be thus reached from $\bar{x}$ form the *neighbourhood* $N(\bar{x})$ of $\bar{x}$. Local search may then be performed over the entire neighbourhood, or only on a selected subset, identified as the *candidate list* at iteration k, $C(\bar{x}, k) \subseteq N(\bar{x})$. The best move-candidate, with respect to some criterion (usually based on the objective value), in the candidate list is selected and implemented.

To avoid cycling, a record is kept of the recent search history; this short term memory is implemented as *tabu* lists that forbid the selection of certain moves. The number, size, contents and management policies of the tabu lists are as varied as the specific applications and the researchers' imagination permit, and jointly form one of the main strengths of tabu search. In this context, local search selects the best move *that is not tabu*, that is from $\bar{x}$ to $\tilde{x} \in C(\bar{x}, k)$ and $\tilde{x} \notin ST(k) \cup T$, where $ST(k)$ represents the set of short term tabu lists at iteration k, while T stands for any combination of longer term tabu memories. The combined effect of these restrictions implies that the absolute best candidate might not be selected at each move. Of course, one may always override the tabu status of a candidate by using an *aspiration criterion*. While locally exploring the solution space, one registers the best solutions found and, eventually, some of their attributes. The search is then continued until a certain stopping criterion (typically, a maximum number of iterations without improvement in the best solution found) is met.

The procedure may be enhanced by using intensification and diversification phases. *Intensification* corresponds to a more intense (even thorough) exploration of part of the solution space identified during the current local search cycle as containing good solutions, and is based on solution attributes stored in medium term memories $MT(k)$. Intensification of the search often implies identifying and fixing the desired attribute values ("fix the solution core"), and then looking out for the best corresponding solution. If an improving solution is found, local search is resumed. *Diversification*, on the other hand, is a device used to guide the search towards zones believed not yet explored. This is achieved by recording in long term memories $LT(k)$ information (attributes) concerning the (best) solutions encountered so far, and by selecting a new solution with different attribute values ("complement the solution core").

This is a rather coarse summary of tabu search, and it overlooks many of the finer aspects of its implementations. Yet, it captures the essence of the method and it is sufficient for the purposes of this paper. In particular, it allows the description of the tabu search procedures reviewed later in the paper.

2.2 Taxonomy Dimensions

As previously mentioned, tabu search makes extensive use of information concerning the regions already explored and the attributes of the solutions found during the search. We do not intend to classify parallel tabu search algorithms according to their basic exploration and knowledge acquisition design. This is clearly beyond the scope of this paper. Yet, we want to emphasize that since this is one of the fundamental building blocks of tabu search, the strategies used for its parallelization must constitute an important criterion of the taxonomy. Furthermore, any parallelization strategy implies some decomposition either of the domain, or of the basic steps and tasks of the algorithm, or of both. Consequently, not all information is necessarily available at all times during a parallel resolution of a problem instance, and, therefore, how the knowledge gathered during the parallel exploration of the domain is exchanged and combined among processes is as important as how the domain is divided among, or how the tasks are allocated to, the various processes.

Our taxonomy is built according to three dimensions meant to capture all these factors. The first two represent the parallelization schemes relative to the control of the search trajectory and the communication and information processing approach, while the third accounts for the strategies used to partition the domain and to specify the parameters for each search. The three dimensions are illustrated in Figure 1, summarized in Table 1, and detailed in the following.

Control Cardinality	1-control p-control
Control and Communication Type	Rigid Synchronization Knowledge Synchronization Collegial Knowledge Collegial
Search Differentiation	SPSS SPDS MPSS MPDS

Table 1: Taxonomy Dimensions

2.2.1 Search Control Cardinality

Control of the parallel search may either stay with one processor, usually called *master* or *main* processor, or be distributed among several processors. Two categories may be defined.

The first case, that we call **1-control**, trivially corresponds to the sequential case. In a parallel context, it represents the approach where one processor essentially executes the algorithm, but delegates some of its work to other processors. The master collects and reconciliates the information, distributes the tasks to be executed by the other processors, and determines when the search has to stop. The tasks that are delegated may consist of only time-consuming numerical computations; this corresponds to what Trienekens and de Bruin [27] call *low level parallelism* in the branch-and-bound context. It may also imply, however, the parallel exploration of the neighbourhood [3, 5], or the construction and evaluation of the candidate list. A straightforward implementation of the sequential fan candidate list strategy [19] falls under this heading.

In the second case, control of the search is shared among p, $p > 1$, processors; hence, we identify it as **p-control**. The classical *collegial* or *multithread* arrangement of processes belongs to this category. Each process is in charge of its own search, as well as of establishing communications with the other processes. The global search terminates once each individual search stops. Coordination of information exchanges and attempts to ensure that the adequate information is available when required are among the main issues in this context, and also play an important role in defining the type of control that is exerted.

2.2.2 Control and Communication Type

The second dimension of the taxonomy is based on the type and flexibility of the control: it takes into account the communication organization, synchronization and hierarchy, as well as the way information is processed and shared among processes. The control-type dimension is made up of four stages or degrees, that combine to the two levels of cardinality control to define the parallelization strategies relative to process and information handling.

The first degree corresponds to a **rigid synchronization** of the processes. A synchronous operation mode [2] usually indicates that all processes have to stop, and engage in some form of communication and information exchange, at points (number of iterations, time intervals, specified algorithmic stages, etc.) exogenously determined: either hard-coded into the procedures or determined by a control process. We qualify such an organization as "rigid" when little, if any, information exchange takes place among processes that are dedicated to executing the same level of tasks.

In particular, rigid synchronization ideally complements the 1-control approach. This is the classical master–slave case, where the master executes what amounts to a sequential tabu search by using other processors to perform computing intensive tasks. There is no communication among the slave processes, and information is kept and handled exclusively by the master, which also initiates all communication phases.

The extension to the p-control case is the straightforward parallelization strategy where independent searches are performed simultaneously. Each search may start from a different initial solution, or may be using a different set of parameters, or both. Again, there is no communication among processes during the search, and each terminates when its own stopping criteria are met. The best solution is selected once all processes have stopped.

The next stage is also characterized by a synchronous operating mode, but an increased level of communication permits to build and exchange knowledge. Hence, we identify it as **knowledge synchronization**.

When operating within the 1-control framework, the master continues to be the keeper of the information, to synchronize the processes, and to dispatch work to the slaves, but it delegates a larger part of the work. The slave processes still do not communicate among themselves. Their tasks, however, are more complex than in the rigid synchronization case, and may imply that local memory structures are present. Hence, for example, a slave process may execute a limited sequence of tabu search steps on a given subset of the neighbourhood (e.g., intensification on promising candidates). But, on request from the master (when it synchronizes, for example) the slave process returns the problem and the results, and waits for a new task. A more sophisticated implementation of the fan candidate list belongs to this category.

When a p-control strategy is adopted, the knowledge synchronization mode corresponds to several independent search trajectories which all stop at a predetermined moment (e.g., number of iterations), the same for all processes. At that moment, an intensive communication phase begins among all control processes. This may be viewed as a hybrid approach between rigid synchronization and independent collegial.

To summarize, in synchronous mode, the 1-control strategy implies vertical, master–slave, communication channels exclusively, while only horizontal, process to process, communications exist in a p-control strategy. The difference between rigid and knowledge synchronization is not always clear in the 1-control context, since it is mostly based on how much work the master assigns to each slave. This difference is much more significant for p-control strategies, since it corresponds to the absence or presence of inter-process communications and knowledge exchanges.

The third and fourth degrees of the control strategy dimension make use of asynchronous communication modes. In this context, each process stores and treats its own information, initiating communications with some or all other processes according to its own internal logic and status. We define two such degrees according to the quantity, quality and treatment of the exchanged information. Note that we do not intend to classify parallel procedures according to the precise means of communicating information and work (see, for example, the survey by Gendron and Crainic [13] or the recent work by Karp and Zhang [20]). Rather, we focus on the role that communication play in reconstructing a global search pattern when several independent search threads explore the solution domain.

In the third stage, that we call **collegial**, each process executes an eventually different tabu search on all or on part of the domain. When a process finds an improving solution (locally or globally, according to the chosen strategy), it broadcasts it (together, eventually, with its context and history) to all or to some (e.g., the neighbouring ones) of the other search processes. It may also deposit it in a central memory, and only broadcast (if at all) that a better solution has been found. In all cases, however, communications are simple, in the sense that the message sent corresponds to the message received.

This is not necessarily the case, however, in the fourth, **knowledge collegial**, stage. Here, the contents of communications are analyzed to infer additional information concerning the global search trajectory and the global characteristics of good solutions. Global memories (e.g., the status change frequency of some variables) and tabu lists that reflect the dynamics of the asynchronous parallel exploration of the domain may thus be built, while new solutions may be constructed based on the solutions and memory contents sent by the individual searches. Therefore, the message received by a process is generally richer than, and not identical to, the one initially sent by another process.

2.2.3 Search Differentiation Strategy

In Voß's classification [28], the only criteria considered refer to the number of different starting solutions, and to the number of different solution strategies (parameter settings, tabu list management policies, etc.) used by the particular implementation. This corresponds to our third dimension, that we identify as the *search differentiation strategy*.

Although the balls and mountains imagery Voß uses in naming the classes of his taxonomy has a certain appeal, we prefer to refer directly to the decision to start the exploration of the domain from the same or from different points, and to use either a unique or different search strategies for each search thread. We use the term "search strategy" in its most general sense that includes different neighbourhood definitions, parameter settings, memory management rules, diversification schemes, etc.

We identify the following four cases:

SPSS: The *Single (Initial) Point Single Strategy* is the most simple case, and it generally allows for only low level parallelism.

SPDS: The *Single Point Different Strategies* approach refers to the case when each processor runs a different tabu search but starts with the same initial solution.

MPSS: The *Multiple Points Single Strategy* label identifies the case when each processor starts from a different solution of the domain, but use the same tabu search settings and rules to explore the domain.

MPDS: Finally, the *Multiple Points Different Strategies* class is the most general and has all others as special cases.

3 Review of Parallel Tabu Search Algorithms

Although parallel tabu search is still in its infancy, a number of significant contributions have already been realized. We now examine how the taxonomy applies to some of the implementations of parallel tabu search found in the literature.

Malek et al. [21] implement and compare serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem. The parallel experiments are performed on a 10 processor Sequent Balance 8000 computer. The authors report that the parallel tabu search implementation outperforms the serial one, and consistently produces comparable or better results than sequential or parallel simulated annealing. Their implementation may be described as a 1-control, knowledge synchronization, SPDS method with one main process and four child processes. Each child process runs a serial tabu search algorithm with different tabu conditions and parameters. The child processes are stopped after a specified time interval, the solutions are compared, and bad areas of solution space are eliminated. The child processes are then restarted with a good solution and an empty tabu list. Note that, in order to strictly implement this strategy, the diversification long term memory function is disabled.

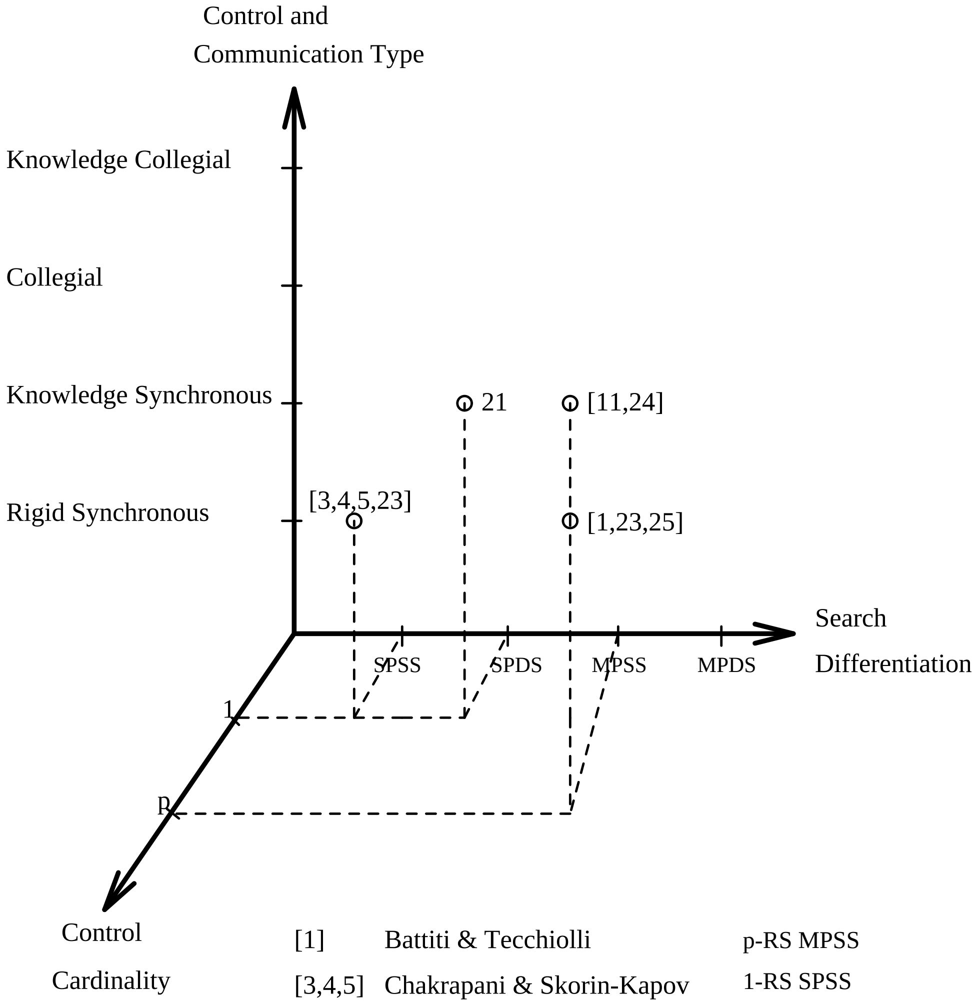
Taillard [24] studies tabu search based algorithms for vehicle routing problems. His parallelization strategies are based on partitioning the solution space, using a p-control, knowledge synchronization, MPSS approach, and are simulated for $p = 4$ on a Silicon Graphics 4D/35 workstation. The first strategy applies to Euclidean problems with uniformly distributed cities, and decomposes the domain into polar regions, to which vehicles are allocated. Once the initial partition is performed, each subproblem is solved by an independent tabu search. All processors stop after a certain number of iterations (this number varies according to the total number of iterations already performed), and the partition is modified. This is done by an information exchange phase, during which tours, undelivered cities and empty vehicles are exchanged between adjacent processors (corresponding to neighbouring regions). Load balancing problems seem to impair this approach. The second strategy is aimed at non-Euclidian problems, or at problems where cities are not uniformly distributed. The main difference between the two strategies appears in the partitioning method (the space is partitioned based on the arborescence build by the shortest paths from the depot to all cities), and in the information that is exchanged (the best solution only).

Fiechter [11] also makes use of a p-control, knowledge synchronization, MPSS strategy to parallelize his tabu search algorithm for traveling salesman problems. The exact operation that is to be executed in parallel is specific to the particular step of the tabu search procedure. For the intensification phase, each process optimizes a specific slice of the tour. At the end of the intensification phase, processes synchronize to recombine the tour and to modify (shift part of the tour to a predetermined neighbouring process) the slice of the tour each process will continue to work on. For the diversification phase, each process determines among its subset of sities a candidate list of most promising moves. The processes then synchronize to exchange these lists, so that all processes build the same final candidate list and apply the moves. The algorithm has been implemented on a network of transputers arraanged in a ring structure. The author reports near-optimal solutions to large (500, 3000 and 10000 vertices) problems, and almost linear (less so for the 10000 vertices problems) speedups.

Taillard [23] makes use of a 1-control, rigidly synchronized SPSS parallelization approach for his tabu search aimed at the quadratic assignment problem. The set of possible moves is partitioned into p sets, and each set is assigned to a different processor. Each processor then evaluates the pairwise interchange moves and identifies the best one. Intriguingly, it seems that Taillard dispenses with a specific master processor. Indeed, once each processor finds its best move, it communicates it to all other processors. Then, each processor performs all the tasks of the master: choosing the best overall move, implementing it, making the necessary adjustments and updates, partitioning the neighbourhood, etc. No implementation details are given. Load balancing through partition of the neighbourhood is acknowledged as critical, but no indication is given on how it is performed. A ring of 10 transputers (T800C-G20S) is used for the experiments.

Chakrapani and Skorin-Kapov [3, 5] also address the quadratic assignment problem by using a parallelization approach which is essentially a 1-control, rigidly synchronous, SPSS procedure, where the search is performed sequentially, while the move evaluation is performed in parallel. However, the implementation is specifically designed to take advantage of the special features of the Connection Machine CM-2, a massively parallel SIMD machine: for a size n problem, n^2 processors are used to evaluate moves and communicate information. The authors report that the best known or improved solutions were obtained for problems studied in other comparative studies and that their method required a significantly smaller number of iterations. Furthermore, they were also able to determine good suboptimal solutions to bigger problems in reasonable time.

Chakrapani and Skorin-Kapov [4] apply a similar strategy to the problem, approximated by a very large quadratic assignment problem with sparse flow matrix, of mapping tasks to processors in a multi-processor system in order to minimize the time spent in inter-processor communication. It is noteworthy that, due to the sparsity of the task graph, implementing a move (swap a single pair of tasks) does not signifi-



[1]	Battiti & Tecchiolli	p-RS MPSS
[3,4,5]	Chakrapani & Skorin-Kapov	1-RS SPSS
[11]	Fiechter	p-RS MPSS
[21]	Malek & all	1-KS SPDS
[23]	Taillard	1-RS SPSS
[23,25]	Taillard	p-RS MPSS
[24]	Taillard	p-KS MPSS

Figure 1: Taxonomy Dimensions

cantly affect the values of most other possible moves; hence, most improving moves are still improving. Two operations are therefore performed in parallel: the candidate moves are identified and evaluated and, second, multiple moves are implemented. To alleviate the evaluation error inherent in such a procedure (the total value of multiple swaps is not equal to the sum of the individual moves), an aggressive diversification phase is introduced into the procedure. Very good results are reported on a 8192 processor hypercube configuration of a CM-2 Connection Machine.

Battiti and Tecchiolli [1] also use the quadratic assignment problem to present a tabu search with hashing procedure, and to discuss a parallelization scheme based on several independent searches. The hashing feature is used to have the search react to the detection of cycles by suitably modifying the length of the tabu lists. The authors then analyze a parallelization scheme where several independent search processes start the exploration of the domain from different, randomly generated, initial configuration. This corresponds to the p-control, rigid synchronization, MPSS strategy of the taxonomy. The authors then proceed to derive probability formulas for the success of the global search that tend to show that the independent search parallelization scheme is efficient – the probability of success increases, while the average success time decreases with the number of processors – provided the tabu procedure does not cycle.

Taillard also studies the p-control, rigid synchronization, MPSS parallelization strategy that performs many independent searches, starting every one with different initial solutions. The main study is to be found in his paper on parallel tabu methods for job shop scheduling problems [25]. For this type of problems, Taillard shows that a tabu search approach (that includes a diversification phase) is very competitive: simpler to implement and generally more efficient than either the simulated annealing or the shifting bottleneck procedures (the two best heuristics proposed at the time), it helped establish new best known solutions for every problem in two sets of benchmark problems, while optimally solving random problems with m machines $\ll n$ jobs (e.g., $m = 5$, $n = 2000$) in polynomial mean time. Several parallelization ideas focusing on speeding up computations related to the neighbourhood evaluation (1-control, rigid synchronization) did not yield good results, either because the available computing platforms (a ring of transputers and a 2-processor Cray computer) were not suitable for the implementations, or because the communication times were much higher than the computation ones.

Taillard then proceeds to examine the theoretical bases of the many independent searches parallelization approach for "random" iterative algorithms (tabu search, simulated annealing, etc.). His results show that the conditions needed for the parallel approach to be "better" than the sequential one, i.e., that the probability of the parallel algorithm to achieve 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, are rather strong. However, the author also mentions that, in many cases, the empirical probability function of iterative algorithms is not very far from an exponential one and, so that the many independent searches parallelization approach is very efficient. The results for the job shop problem [25] and the quadratic assignment problem [23] seem to justify this claim.

This brief literature survey emphasizes a few points:

- The use of parallelism may improve the performance of tabu search procedures.
- The parallelization of a tabu search procedure may conflict with some of the basic tabu search mechanisms (e.g. the diversification feature in [21]).
- The taxonomy we propose is sufficiently comprehensive to account for the parallelization strategies already reported.
- Despite significant implementation differences, due to the specificity of the problems, tabu search characteristics, computer environment, etc., few parallelization paradigms have yet been called for in the reported experiments. Indeed, as illustrated in Figure 1, synchronization seems to be the adopted norm, parallel computation being mostly used to evaluate moves, or to accelerate a restarting strategy.

In the following sections, we show that other strategies, identified by our taxonomy, are available to build efficient parallel tabu search procedures.

4 Illustrating the Taxonomy

To further illustrate the taxonomy presented previously, we briefly review the study of Crainic, Toulouse and Gendreau who have designed and tested several synchronous [10] and asynchronous [9] parallel tabu search variants of a sequential tabu search procedure for the multicommodity location-allocation problem with balancing requirements.

Our main objective is to demonstrate that the proposed taxonomy does not constitute an empty shell: that each group of parallel implementation strategies it defines does indeed correspond to a particular algorithmic case with distinctive characteristics and behaviour. Hence, while the results of extensive testing are reported and analyzed in [10] and [9], we present in this section only illustrative synthetic performance measures.

4.1 Model and Sequential Tabu Search Procedure

The multicommodity location-allocation problem with balancing requirements typically arises in the context of the medium term management of a fleet of heterogeneous vehicles (containers, in our application), where vehicle depots have to be selected, the assignment of customers to depots has to be established for each type of vehicle, and the interdepot vehicle traffic has to be planned to account for differences in supplies and demands in various zones of the geographical territory served by the company. One aims to minimize the total system cost: the "fixed" cost associated to the selection of depots, plus the transportation costs between customers and depots, plus the costs of the inter-depot movements required to balance supply and demand for each type of vehicle. The problem is formulated as a linear mixed integer programming model, where integer (binary) variables represent the decision to select or not the corresponding depots, while continuous variables capture the vehicle flows on the arcs of the network. Other than the usual sign restrictions, two sets of constraints determine the feasible region for this problem: (i) a set of linking constrains that forbid the use of an unselected depot, and (ii) the usual uncapacitated multicommodity demand-flow conservation equations of a network flow problem.

The mathematical model is fully presented and analyzed in [6]. It is, however, worthwhile to recall that the formulation displays an interesting network structure. In particular, for fixed binary variables, it becomes an uncapacitated multicommodity minimum cost network flow problem, a well known model for which efficient solution methods exist. This property has been used to define a tabu search procedure, which is fully described and analyzed in Crainic et al. [8]. In the following, we only summarize its main characteristics, illustrated in Figure 2, to facilitate the presentation of the parallelization developments.

The search space is defined with respect to the binary depot decision variables that specify the *depot configuration*. For any configuration, the optimal values of the continuous flow variables and the corresponding value of the objective function, may be computed by solving an uncapacitated multicommodity network flow problem. The neighbourhood of any such solution includes all configurations that may be obtained by either opening (*add* move) or closing (*drop* move) a currently closed or open, respectively, depot, or by performing a *swap* that simultaneously opens a depot while closing another. Such a neighbourhood is usually too large, however, and sampling is used to build a candidate list. Furthermore, the evaluation of all possible moves by solving the associated network flow problem is too time consuming, and surrogate functions (based on estimates of differences in objective function values) are used in most instances; the real value is however computed once a move is selected and implemented.

The search strategy combines a local search with intensification and diversification phases, and terminates with a postoptimization phase.

Local search consists of an add/drop sequence (stopped once a predefined number of iterations are performed without improving the solution), followed by a normal swap (the best candidate move evaluated by using the surrogate functions is implemented regardless of its real impact on the objective function) sequence that is initiated from the best solution found by performing the add/drops. When the best local solution yielded by this process is feasible, search intensification is immediately performed, otherwise the local search phase is continued until a feasible local solution is encountered.

Add/drop and swap sequences use different short-term memory tabu lists. For add and drop moves, lists record the last depots added or dropped from the solution, and the reverse moves are forbidden. The swap tabu list records the most recently performed swaps as pairs of depots, and the reversal or repetition of the moves is forbidden. Note that long term (diversification) tabu lists further affect the status of candidates while performing local search.

An *intensification phase* consists of a strict swap sequence, which starts from the best solution identified during the previous local search phase, and implements only those selected moves that improve on the current solution. A *diversification move* is performed starting from the best global solution found so far in the search, and is based on a long-term memory that records the level of "activity" of each depot: the number of times its status has been modified (changed from open to closed or vice-

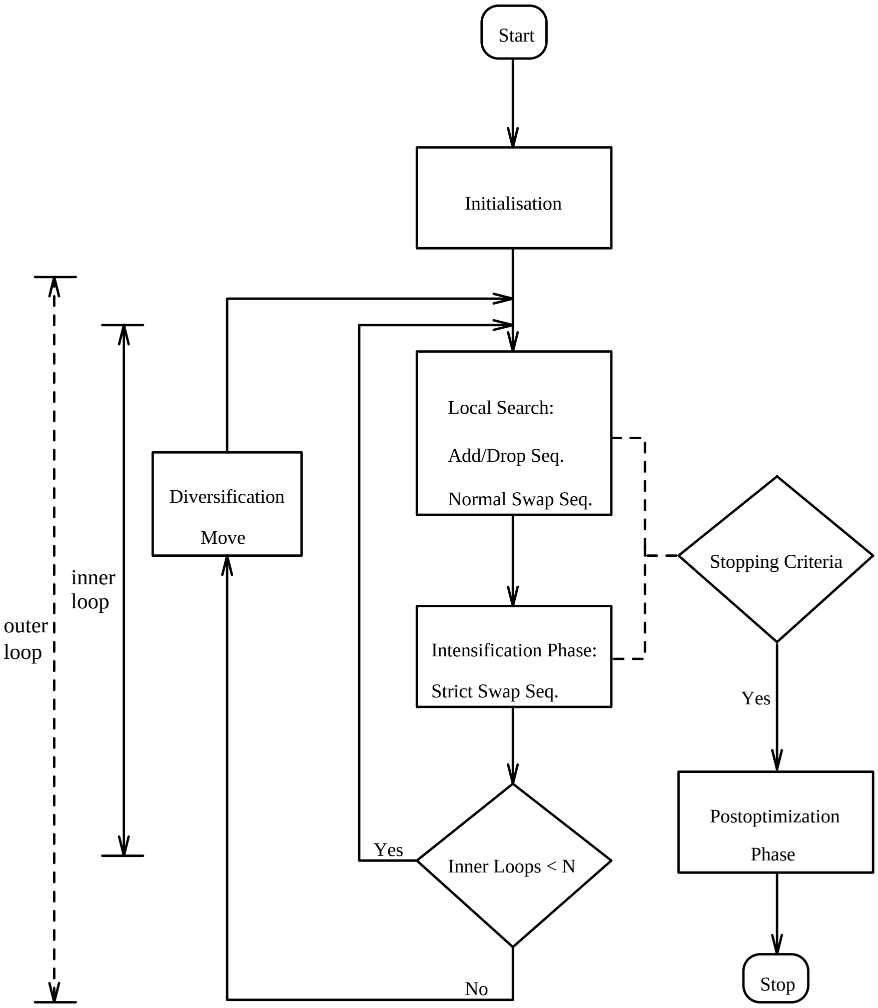


Figure 2: Sequential Tabu Search

versa). Based upon the values stored in this memory, the predefined number of depots with the lowest activity counts are selected and complemented. Considering the fact that values in the long-term memory tend to evolve rather slowly, another memory has been provided to record the last set of depots selected for diversification. This list is used both to exclude depots from being considered in the next diversification phases, and to prevent too quick a reversal of the diversification moves during the following local search steps.

A sequence of local search and intensification phases is called an *inner loop*. After executing N inner loops, the search procedure is re-directed to previously unexplored regions of the search space by performing a diversification step, which completes an *outer loop*. The overall search procedure starts from an initial solution and performs a sequence of outer loops until some termination criterion is met. In the current implementation, this termination criterion is the total number of iterations since the beginning of the search.

A postoptimization phase, which aims at ensuring that no better solution exists close to the best solution identified so far, is invoked once the prespecified number of regular iterations has been performed. This phase consists in a comprehensive neighbourhood exploration search that considers all possible simple (add, drop) moves. Surrogates are used to rank moves, while exact evaluations determine the (first) improving move to be implemented. This procedure continues for as long as strictly better solutions are found.

Several parameters influence the efficiency of the search: the lengths of the tabu lists and memories, the lengths of the add/drop and swap sequences, the selection probabilities of the add, drop and swap moves, how these probabilities vary during the search, the initial solution that is chosen, etc. Crainic et al. [8] study these issues and show, in particular, that several combinations of parameters may be efficiently used for different types of problem characteristics.

4.2 Experimentation Environment

Sixteen problems are used for testing: twelve randomly generated, and four based on an actual application [7]. The random problems have some 44 depots (integer variables), 220 customers, and either 1 or 2 products, which yields more than 7000 and 14000 continuous variables, respectively. For the last four problems, the corresponding figures are 130 integer and 56616 continuous variables.

All procedures are stopped after 300 iterations, and the solution quality is measured by computing the *gap*, in percentage, between the best solution determined by each procedure and the optimal solution computed by a branch-and-bound algorithm [14]. Note that the objective is to illustrate the taxonomy, not to fine-tune a given procedure on a given set of problems. Hence, we did not calibrate each individual procedure for best performance over the problem set. Instead, the best parameter settings observed for the sequential tabu search [8] were used for all the experiments reported in this section.

All tests have been conducted on a heterogeneous network of SUNSparc workstations. Communications are handled by our own set of procedures, written in C, that use the TLI/UDP protocol, modified to ensure that all packets reach their destination. The tabu search is programmed in FORTRAN77, while the minimum cost network flow subproblems are solved by using the RNET code [22].

4.3 Parallel Tabu Search Implementations

For each parallelization strategy defined by the taxonomy, several different implementations are generally possible. The procedures described in the following represent one such possible implementation for each of the 14 parallelization strategies. Tables 2 and 3 display the average gaps, computed over the set of test problems, obtained by each parallel implementation for 4, 8 and 16 processors, as well as (the "SEQ" column) the average gap of the sequential procedure. The evolution of the average gaps with the number of processors is also illustrated in Figures 3 and 5 for the synchronous and asynchronous procedures, respectively.

The first synchronous parallel tabu search procedure is built according to a 1-control rigidly synchronized (1-RS) SPSS strategy. In this implementation, the master process executes the tabu search algorithm, while the evaluation of the N elements of the local search candidate list is divided between p processes. Each process evaluates N/p moves, by using a surrogate function, according to the tabu lists transmitted by the master process, and returns the best move found. Note that each slave process also computes the exact value of its best move. Hence, with little additional cost, the master may choose among several exactly evaluated moves. This improvement relative to the sequential version is reflected in the quality of the solutions found.

A variant of the 1-control knowledge synchronous (1-KS) SPSS approach is related to the *probing* strategy proposed by Glover (see also [17]). Here, not only the exploration of the neighbourhood is divided among the p processes, but each process also performs a few (two, for the results reported in this section) local search iterations. The master then selects the sequence of moves that has resulted in the best

p	SEQ	1-RS	1-KS	p-RS			p-KS		
		SPSS	SPSS	SPDS	MPSS	MPDS	SPDS	MPSS	MPDS
4	0.63	0.51	0.39	0.32	0.20	0.15	0.53	0.65	0.61
8	0.63	0.45	0.16	0.18	0.17	0.06	0.49	0.50	0.32
16	0.63	0.35	0.12	0.08	0.15	0.04	0.44	0.41	0.21

Table 2: Average Gaps (%) – Synchronous Procedures

p	SEQ	p-C			p-KC		
		SPDS	MPSS	MPDS	SPDS	MPSS	MPDS
4	0.63	0.19	0.28	0.10	0.25	0.44	0.15
8	0.63	0.17	0.15	0.05	0.31	0.36	0.20
16	0.63	0.04	0.17	0.04	0.13	0.13	0.19

Table 3: Average Gaps (%) – Asynchronous Procedures

improvement, and implements it by appropriately updating the various tabu lists and memories.

Three p-control knowledge synchronous (p-KS) parallelization strategies were implemented corresponding to the SPDS, MPSS and MPDS search differentiation approaches identified by the taxonomy. Note that we did not implement the SPSS strategy since, in a synchronous environment, it reduces to p repetitions of the same search. Here, p independent tabu search threads explore the problem domain and exchange information, at predetermined synchronization points, that may modify the current trajectory of any given process. Each process performs a given number of iterations (25, in the present case), then broadcasts either its best solution (the SPDS case), or its set of p best solutions (for the MPSS and MPDS strategies). Following this communication and synchronization phase, the best of all solutions becomes the initial solution for the next parallel phase of a SPDS implementation. For MPSS and MPDS approaches, the p overall best solutions are identified and distributed among the p processes. For single initial solution strategies, the best parameter settings for the sequential tabu search are used. When different search strategies are implemented (in SPDS and MPDS strategies), we varied the lengths of the short and medium term tabu lists, the number of consecutive add/drop iterations without improvement, and the number of depots temporarily fixed by a diversification move.

No special implementation is required for the p-control rigid synchronous (p-RS) parallelization approach: one simply runs totally independent tabu searches, by varying the initial solution and the parameter settings according to the chosen strategy. Note that because a SPSS approach reduces to p repetitions of the same search, it was not implemented.

The introduction of synchronization points into parallel iterative search procedures is often motivated by a desire to ensure that parallel computations display a deterministic behavior and a search trajectory similar to that of a sequential method. Yet, in most cases, this is achieved at a price in algorithmic efficiency, since a significant number of processes are often idle waiting for other processes to complete their activities. Consequently, to improve the algorithmic performance, various levels of asynchronism may be introduced into the parallel procedure. In the context of our taxonomy, such strategies fall under the p-control collegial or knowledge collegial headings.

Let the *context* of a solution of a n-decision-variable problem be a vector of cardinality n that contains the values of the n decision variables. There could be several contexts for a given objective function value but, given a context, there is only one possible objective function value. The present implementation of the asynchronous parallel framework, illustrated in Figure 4, makes use of a *central memory* through which pass all communications, and that captures the global knowledge acquired during the search. Note that this is an implementation device which helps to keep in check communication and accounting efforts, as compared, for example, with a strategy where each process broadcasts its solution to all other processes, which, in turn, have to accept, compare, update, and store the information. It also enforces the asynchronous paradigm, since it lifts the need for an acceptance decision by each process at broadcast time: each thread decides to access the central memory information based exclusively on its own internal schedule and history.

By using this framework, the p-C parallel strategy proceeds as follows: (i) Each process sends its solution and context to the central memory each time it improves its best global solution. (ii) The central memory keeps and updates the best global solution found so far. Together with its associated context, we call this solution the *central memory best solution*. (iii) If the best global solution of a process is worst than the central memory best solution, the process retrieves the central memory best solution. (iv) After a certain number of iterations without improving its global solution, a process requests the best central memory solution. More complex versions of this framework can be defined (e.g., instead of storing the overall best solution in the central memory, build a pool of good solutions to distribute, on request, among the search threads [9]; more sophisticated approaches would use the solution pool to derive new solutions), but this is beyond the scope of this paper.

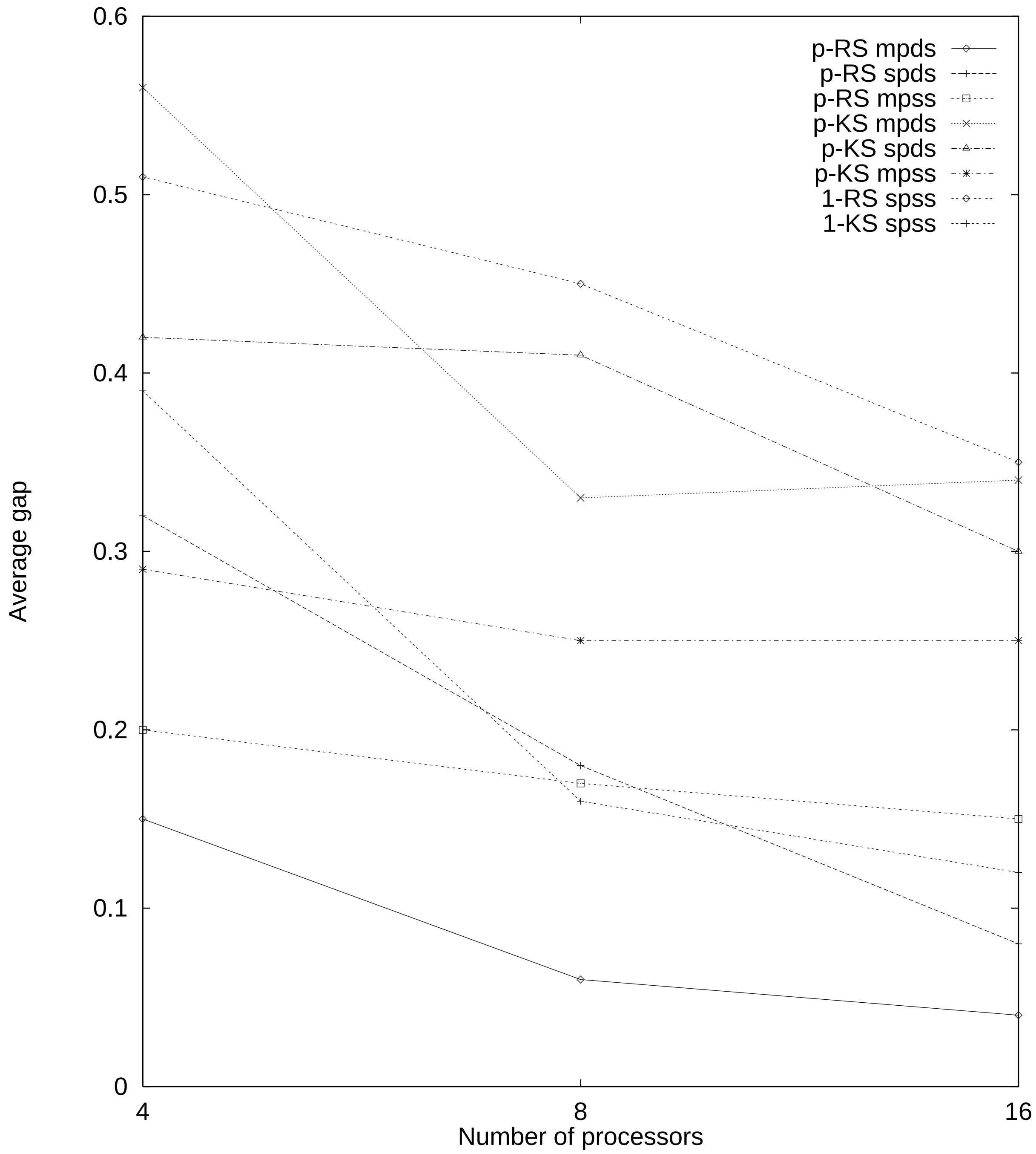
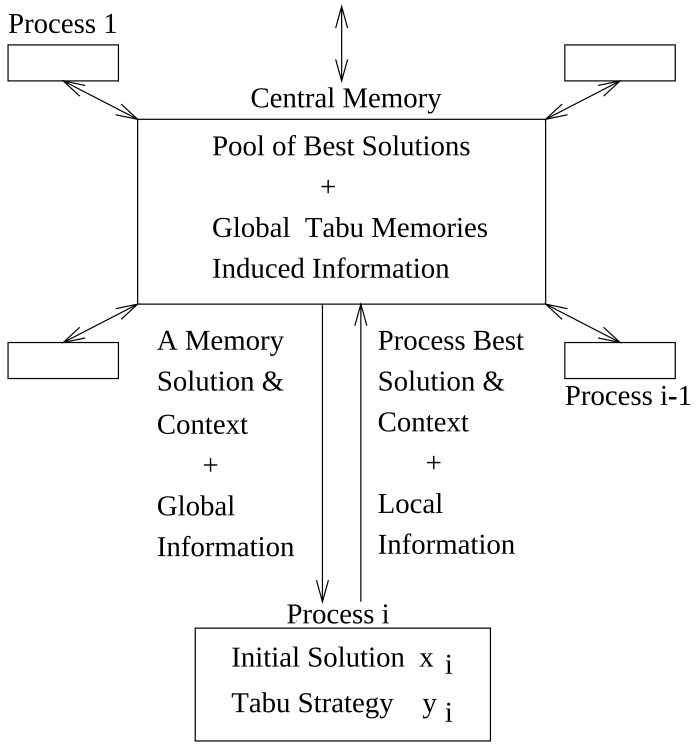


Figure 3: Gap Comparisons – Synchronous Implementations



	$x_i = x_j \forall i,j$	$x_i \neq x_j \forall i,j$
$y_i = y_j \forall i,j$	SPSS	MPSS
$y_i \neq y_j \forall i,j$	SPDS	MPDS

Figure 4: Examples of p-Control Asynchronous Parallelization

Note that a process always resumes its computations from the same state (short term tabu lists and memories, best local search solution, current solution context, etc.) it was in just before communicating with the central memory. However, before initiating a diversification phase, the process compares and, eventually, replaces its best global solution and context by the central memory best solution. Then, either the search resumes from the central memory best solution (this amounts to an externally imposed diversification), or a normal diversification step is performed. In this way, one can reconciliate the tabu search behaviour based on long term memories of each process, to the import of exogenous information. Implementation details may be found in [9].

It is noteworthy that in the classic asynchronous collegial parallel framework illustrated above, at any given moment, a process knows, at best, its own history and the value and context of the current best global solution, without any indication of the global evolution of the search. In a sense, we do not achieve the global picture of the combined effects of the search threads performed by the individual processes, and lose, at least partially, the effect of the learning and memory mechanisms central to tabu search approaches. The class of strategies the taxonomy identifies as p-control knowledge collegial (p-KC), are intended to address this issue.

Figure 4 illustrates a very simple version of the p-KC strategy. The previously defined general framework is used, but more information is exchanged among processes and a new structure is defined at the level of the central memory to record the evolution of the global search. This long term global memory is updated each time the central memory solution is improved, and records the frequency in the change of status of each depot in the sequence of best solutions reported by the various processes. Hence, in the long run, this *consistency* memory tends to build an image of a good solution: depots that seem definitively open or closed and undecided ones. The consistency memory is part of the solution context that is communicated to each process, which then uses it to inflect its own search trajectory. In the current implementation, this is accomplished via a "diversification-like" tabu mechanism: the tabu status of candidate moves is modified during local search according to the consistency values.

The experimental results, summarized in Tables 2 and 3 and illustrated in Figures 3 and 5, support the conclusion of Section 3 that the use of parallelism may improve the performance of tabu search procedures. Indeed, on average, all parallel strategies have yielded better quality solutions. Furthermore, the results of [10] and [9] show that over the 672 reported runs, parallel procedures stopped with the same solution as the sequential procedure in 25% of the cases, while improving the solution for 68% other. In particular, the optimal solution is identified 48% of the time, as compared to 12% for the sequential version.

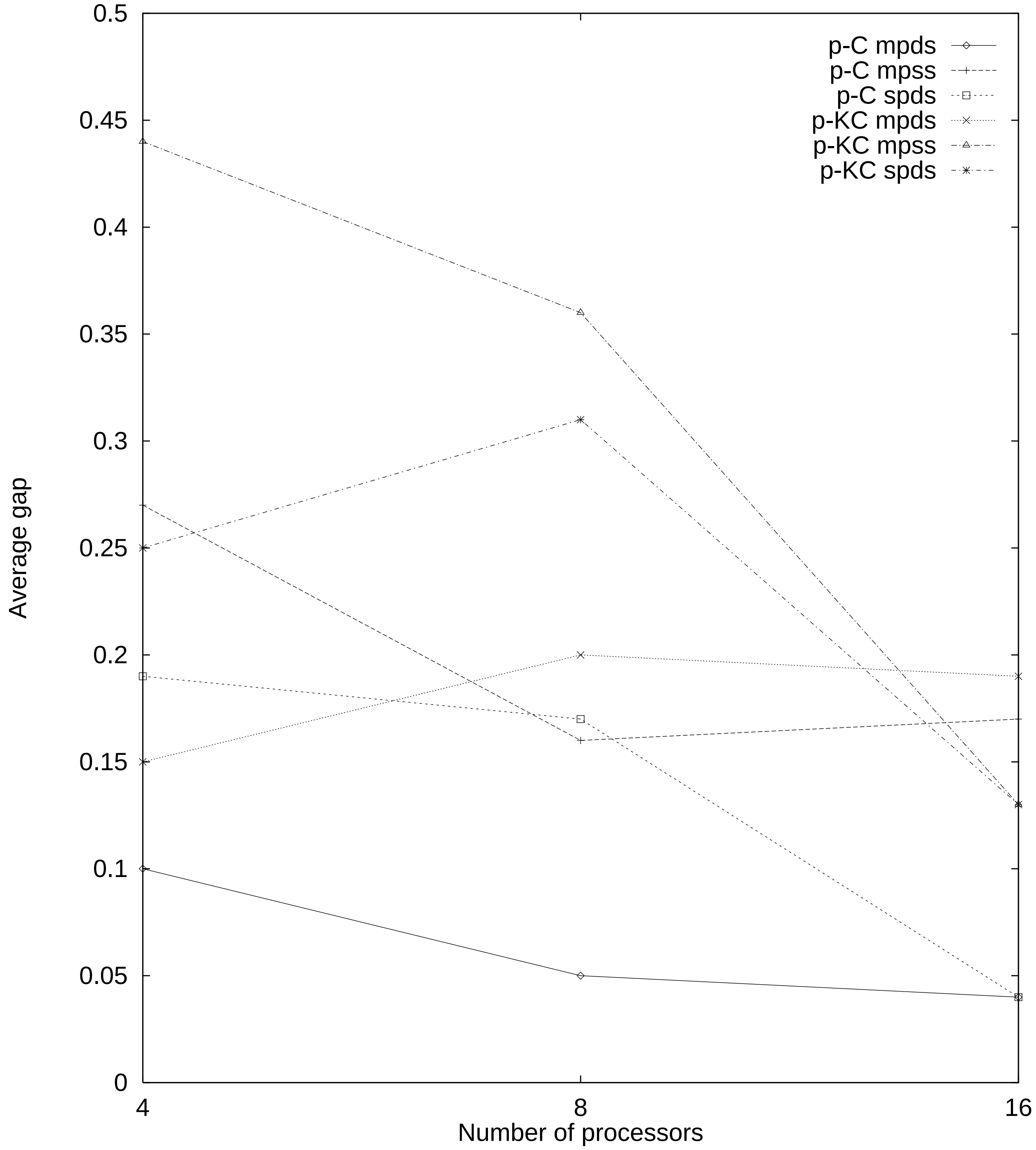


Figure 5: Gap Comparisons – Asynchronous Implementations

The taxonomy has allowed to imagine new, relative to reported implementations, parallelization strategies, and offered a comprehensive framework for comparative studies. First, it appears that, indeed, the class of multiple independent search threads strategies performs very well, and outperforms most of the synchronous strategies. However, if this strategy is chosen, it also appears that it is beneficial to vary not only the initial point, but also the search strategy. Furthermore, strategies that exploit the search for a better knowledge of the neighbourhood and the consequences of promising moves (1-KS strategies, see also [19]) seem to hold their own and offer interesting perspectives for further research. Secondly, asynchronous strategies suggested by the taxonomy appear to hold great promises. Opening up communications among processes without imposing the need to regularly synchronize the processes improves, for the class of problems studied, the performances and appears to be better than the multi threads strategy. The taxonomy also suggests that improved parallelization strategies could be obtained by extracting knowledge from the information exchanges among processes. While the implementations referred to in this paper do not quite realize these promises, they indicate that this is an interesting research direction.

5 Conclusions and Further Research

We have presented a taxonomy of parallel tabu search procedures. In our opinion, it is the most comprehensive yet to be proposed, since it accounts for the main parameters of parallelization strategies: how the control of the acquired knowledge and of the parallel processes is managed, the type and complexity of communications, the differentiation strategies for the various search threads.

The taxonomy permits to classify the reported parallel tabu search procedures. This review also reveals that a rather limited range of parallelization strategies have been implemented so far. The taxonomy also points out to different approaches that may yield more efficient procedures. In particular, the whole dimension of asynchronous parallelization and information management appears to hold great promises and constitutes an exiting area of research.

The taxonomy is independent of any particular problem class or tabu search design. It is also independent of particular computing platforms. The reported implementations have been carried out over the years by using a great number of computers, operating systems and computer languages. Of course, each individual parallel implementation of a particular tabu search method for a given class of problems may gain in performance if it takes advantage of the characteristics of the computer it is to run on. As a general indication, however, it is clear that all the strategies suggested by the taxonomy can be developed on both shared memory and distributed message-passing MIMD (Multiple Instruction Multiple Data [12]) computers. SIMD (Single Instruction Multiple Data) architectures could also be used to implement some instances (e.g., master-slave strategies as in [3, 5]), but appear less interesting for all cases where several search threads are used.

More work is required in order to apply, fine-tune, and evaluate the behaviour and efficiency of the parallelization strategies defined by the proposed taxonomy to various optimization problems. Also needed is a thorough exploration of a number of fundamental questions related to the parallelization of tabu search, such as speed up anomalies, the impact of parallelization on the long term behaviour of tabu search, what information to exchange and how it can be transformed to gain additional knowledge about the search (the recent work of Toulouse, Crainic and Gendreau [26] constitutes a first step in this direction), etc. Yet, it is already clear that tabu search may benefit significantly from a parallel environment, and that, for any given problem, it is worth the effort to explore alternate parallelization paradigms.

Acknowledgments

We wish to thank one anonymous referee whose insightful comments helped us improve the paper. This research has been supported by grants from the Fonds F.C.A.R. of the Province of Québec, and the Natural Sciences and Engineering Research Council of Canada.

References

[1] R. Battiti and G. Tecchiolli. Parallel Biased Search for Combinatorial Optimization: Genetic Algorithms and TABU. *Microprocessors and Microsystems*, 16:351–367, 1992.

[2] D.P. Bertsekas and J.N. Tsitsiklis. *Parallel and Distributed Computation, Numerical Methods*. Prentice-Hall, 1989.

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

[4] J. Chakrapani and J. Skorin-Kapov. Mapping Tasks to Processors to Minimize Communication Time in a Multiprocessor System. Technical report, Harriman School for Management and Policy State University of New York at Stony Brook, 1993.

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

[6] T.G. Crainic, P.J. Dejax, and L. Delorme. Models for Multimode Multicommodity Location Problems with Interdepot Balancing Requirements. *Annals of Operations Research*, 18:279–302, 1989.

[7] T.G. Crainic, L. Delorme, and P.J. Dejax. A Branch-and-Bound Method for Multicommodity Location with Balancing Requirements. *European Journal of Operational Research*, 65(3):368–382, 1993.

[8] T.G. Crainic, M. Gendreau, P. Soriano, and M. Toulouse. A Tabu Search Procedure for Multicommodity Location/Allocation with Balancing Requirements. *Annals of Operations Research*, 41:359–383, 1993.

[9] T.G. Crainic, M. Toulouse, and M. Gendreau. Parallel Asynchronous Tabu Search for Multicommodity Location-Allocation with Balancing Requirements. Publication 935, Centre de recherche sur les transports, Université de Montréal, 1993.

[10] 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), 1995.

[11] C.-N. Fiechter. A parallel tabu search algorithm for large travelling salesman problems. *Discete Applied Mathematics*, 51:243–267, 1994.

[12] M.J. Flynn. Very High-Speed Computing Systems. *Proceedings of the IEEE*, 54:1901–1909, 1966.

[13] B. Gendron and T.G. Crainic. Parallel Branch-and-Bound Algorithms: Survey and Synthesis. *Operations Research*, 42(6):1042–1066, 1994.

[14] B. Gendron and T.G. Crainic. A Branch-and-Bound Algorithm for Depot Location and Container Fleet Management. *Location Science*, 1995.

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

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

[17] F. Glover. Tabu Search: A Tutorial. *Interfaces*, 20(4):74–94, 1990.

[18] F. Glover and M. Laguna. Tabu search. In C.R. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, pages 70–150. Blackwell Scientific Publications, London, 1993.

[19] F. Glover, E. Taillard, and D. de Werra. A user's guide to tabu search. *Annals of Operations Research*, 41:3–28, 1993.

[20] R. Karp and Y. Zhang. Randomized Parallel Algorithms for Backtrack Search and Branch-and-Bound. *Journal of the Association for Computing Machinery*, 40(3):765–789, 1993.

[21] 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.

[22] Grigoriadis M.D. and Hsu T. RNET – The Rutgers Minimum Cost Network Flow Subroutines. Technical report, Rutgers University, New Brunswick, New Jersey, 1979.

[23] E. Taillard. Robust Taboo Search for the Quadratic Assignment Problem. *Parallel Computing*, 17:443–455, 1991.

[24] E. Taillard. Parallel Iterative Search Methods for Vehicle Routing Problems. *NETWORKS*, 23:661–673, 1993.

[25] E. Taillard. Parallel Taboo Search Techniques for the Job Shop Sheduling Problem. *ORSA Journal on Computing*, 6(2):108–117, 1994.

[26] M. Toulouse, T.G. Crainic, and M. Gendreau. Communication Issues in Designing Cooperative Multi Thread Parallel Searches. Publication, Centre de recherche sur les transports, Université de Montréal, 1995.

[27] H.W.J.M. Trienekens and A. de Bruin. Towards a taxonomy of parallel branch and bound algorithms. Report EUR-CS-92-01, Department of Computer Science, Erasmus University Rotterdam, 1992.

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