

OR Spektrum 17 (2/3)

Synchronous Tabu Search Parallelization
Strategies for Multicommodity
Location-Allocation
with Balancing Requirements

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
Université de Montréal

and

Département de génie électrique
École Polytechnique

Michel Gendreau

Centre de recherche sur les transports
and

Département d'informatique et de recherche opérationnelle
Université de Montréal

March 1995

Abstract

We study and compare synchronous parallelization strategies for tabu search. We identify the most promising parallelization approaches, and evaluate the impact on performance and solution quality of some important algorithmic design parameters: length of the synchronization steps, number of processors, handling of exchanged information, etc. Parallelization approaches are implemented and compared by using a tabu search algorithm for multicommodity location-allocation problems with balancing requirements.

Key words: Tabu search methods, Parallel algorithms, Synchronous strategies, Multicommodity location-allocation with balancing requirements

Résumé

Nous étudions et comparons plusieurs méthodes parallèles synchrones de recherche avec tabous. Nous identifions les approches de parallélisation les plus prometteuses et nous évaluons l'impact sur le comportement des méthodes et sur la qualité des solutions de plusieurs aspects importants du design algorithmique: fréquence de synchronisation, nombre de processeurs, traitement de l'information échangée, etc. Les approches de parallélisation sont examinées, implantées et comparées, à partir d'un algorithme de recherche avec tabous pour le problème de localisation-allocation multiproduits avec demandes d'équilibrage.

Mots clés: Méthodes de recherche avec tabous, parallélisation synchrones, localisation-allocation multiproduits avec demandes d'équilibrage

1 Introduction

Tabu search [9, 10, 11, 12, 13, 14] 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.

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 parts of the algorithm, or by performing several searches simultaneously. In the context of branch-and bound algorithms, Trienekens and Bruin [19] refer to the first two 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 this sense, the third option belongs to the same class with the first, and also qualifies as low level parallelism.

This distinction may become, however, significantly more blurred in the context of tabu search. Indeed, as will be seen in the following, the interactions among several search threads often yield a search pattern different from the one obtained by the simple sequential execution of the various searches, and may change the relative order of execution of the various parts of a tabu search algorithm, or alter the contents of its tabu lists and memories. Hence, high level parallelism may be achieved by a careful implementation of parallel search threads.

A recent review [7] of the published literature relative to parallel tabu search algorithms, reveals that few parallelization paradigms have yet been called for in the reported experiments. Indeed, while synchronization seems to be the adopted norm, parallel computation are mostly used to evaluate moves or to accelerate restarting strategies. The goal of the study reported in this paper is to explore a wider gamut of synchronous parallelization strategies than previously reported in the literature, and thus contribute to increase the collective insight into the behaviour of tabu search in a parallel environment, and to identify elements that may be advantageously used to develop efficient parallel tabu search procedures.

The paper is organized as follows. The study encompasses the performance evaluation and comparison of several synchronous parallel implementations of a tabu search procedure developed to solve multicommodity location-allocation problems with balancing requirements. Hence, we first briefly review the formulation and the sequential tabu search procedure. The description of the parallel procedures comes next, followed by the presentation of the experimental results. We conclude by identifying interesting parallelization strategies, and promising research directions.

2 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 non-negativity restrictions, two sets of constraints determine the feasible region for this problem: (i) a set of linking constraints 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 [4]. It is, however, worthwhile to recall that the formulation displays an interesting network structure. In particular, for fixed values of the 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. [6]. In the following we only summarize its main characteristics, illustrated in Figure 1, 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* move 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. Each add,drop or swap move that is implemented corresponds to

a single *iteration* of the search.

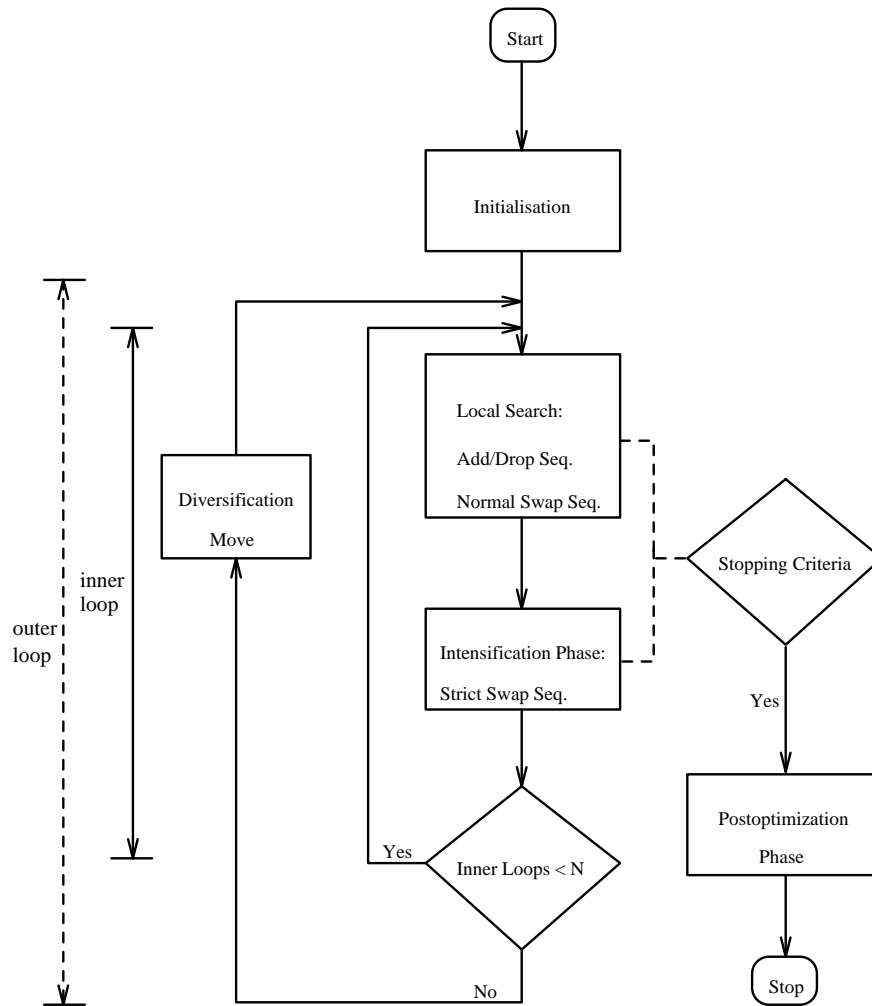


Figure 1: Sequential Tabu Search

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 solution examined by this process (we call it the "best local solution") 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-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. [6] study these issues and show, in particular, that several combinations of parameters may be efficiently used for different types of problem characteristics.

3 Synchronous Tabu Search Procedures

We have designed and tested several synchronous parallel tabu search variants of the sequential tabu search procedure presented in the previous section, best described in terms of the criteria specified by the classification scheme for parallel tabu search procedures introduced by Crainic, Toulouse and Gendreau [7]. This three dimensions taxonomy is based on the number of processes that control the search, on the communication strategies used in the design of the parallel tabu search, and on how the search space is partitioned.

The first dimension of this taxonomy thus distinguishes between 1 and p process control strategies. The second dimension of the taxonomy reflects the type and flexibility of the control, and takes into account the communication organization, synchronization and hierarchy, as well as the way information is processed and shared among processors. It allows, in particular, to distinguish between synchronous and asynchronous (collegial) strategies. In this paper, we focus on synchronous implementations, that may be further classified as either *rigid (RS)* or *knowledge (KS)* synchronous. We find under the first label the so-called master-slave approaches (1-RS), and implementations where several searches are separately conducted with no communications among the various processes (p-RS). On the other hand, procedures that assign part of the search to the slave processes (1-KS), or where the search trajectory of each individual search thread may be modified following an interprocess communication phase (p-KS), are described as knowledge synchronous.

To summarize, in a synchronous mode, a 1-control strategy implies vertical, master-slave, communication channels exclusively, while horizontal, processor to processor, communications exist in a p-control strategy. The difference between rigid and knowledge synchronization in a 1 process control context is based on how much work the master assigns to each slave, while for p-control strategies it corresponds to the absence or presence of inter-process communications and knowledge exchanges.

The third taxonomy criteria acknowledges that implementations may differ in their use of different or identical initial solutions, and of different or similar search strategies for the individual tabu threads (see [20] for a different presentation of the same concept): *Single (Initial) Point Single Strategy (SPSS)*, *Single Point Different Strategies (SPDS)*, *Multiple Points Single Strategy (MPSS)*, and *Multiple Points Different Strategies (MPDS)*. Various elements may define a tabu search strategy: the parameter settings, the type of tabu lists and memories and their updating mechanisms, the existence and design of search intensification and diversification phases, etc. In order to facilitate comparisons between the sequential procedure and the parallel versions, we have decided to use for all parallel implementations the same basic

search mechanism defined for the sequential version, and only modify the values of a number of key parameters. For the SPSS and MPSS approaches, the best parameter settings for the sequential tabu search are used. When different search strategies are implemented in SPDS and MPDS approaches, we vary 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. An initial solution is obtained by arbitrarily closing a number of depots; different initial solutions may thus be obtained by varying this number. The calibration results for the sequential tabu search procedure [6] are used to define the required initial points and tabu search variants.

3.1 The Master – Slave Approach

The first synchronous parallel tabu search procedure is built according to a 1-control, rigidly synchronized SPSS strategy. This is the classical master-slave case, where the master executes what amounts to a sequential tabu search by using the other processors to perform computing intensive tasks. There is no communication among slave processes; subordinate processes exchange information only with the master process, which initiates communications to distribute work and gather results. Thus, information concerning the state of the search is kept and handled exclusively by the master. Consequently, only an SPSS implementation may be devised in this context, and the strategy is identified as 1-RS SPSS. The methods developed by Taillard [17], and by Chakrapani and Skorin-Kapov [1, 3, 2] (albeit on a much larger scale) are examples of this type of strategy.

In our implementation, the evaluation of the n elements of the candidate list used in the local search phase is divided between p processes. Each process evaluates n/p moves, by using the surrogate functions, according to the tabu lists handed out 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 (see Section 4).

3.2 The Probing Strategy

We also implement a variant of the 1-control knowledge synchronous approach, identified as 1-KS SPSS, which is related to the *probing* strategy proposed by Glover

(see also [12]). When operating within this framework, the master process continues to be the keeper of the information concerning the state and trajectory of the search, to synchronize processes, and to dispatch work to slave processes (that do not communicate among themselves), but it delegates a larger part of the work.

In the present implementation, not only the exploration of the neighbourhood is divided among the p processes, but each process also performs a few local search iterations. The master then selects the sequence of moves that has resulted in the best improvement, and implements it by appropriately updating the various tabu lists and memories. We examine, in Section 4, the influence of the number of local iterations the slaves are allowed to perform on the efficiency of the overall search.

The 1-KS strategy is thus a generalization of the master-slave approach — only one tabu search is still conducted — where an increased amount of information is made available to the master. Therefore, even if a SPDS approach may be defined (each slave process could be given different parameters for the few iterations of its local search), we implemented only the SPSS strategy that conforms to the master-slave paradigm. Note that in both master-slave implementations, the neighbourhood is completely explored. Indeed, since several processors are available, the neighbour sampling feature is disabled, and each slave process evaluates all possible moves.

3.3 Independent Searches

No special implementation is required for p-control rigid synchronous (identified as p-RS) parallelization approaches: one simply runs totally independent tabu searches, with different initial solutions and parameter settings according to the chosen strategy. Note that because a SPSS approach reduces to p repetitions of the same search, we do not implement it.

3.4 Coordinated Searches

Three implementations of the p-control knowledge synchronous (identified as p-KS in the following) parallelization strategy have been realized, corresponding to the SPDS (e.g. the algorithm proposed by Malek et al. [15]), MPSS (Taillard’s algorithm [18] belongs to this class), and MPDS categories of the third dimension of the taxonomy. 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. (We did not implement the SPSS strategy since it reduces to p repetitions of the same search.)

In the particular implementations analyzed in this study, each process performs a given number of iterations, and then broadcasts its set of best solutions found since the previous synchronization point. Following this communication and synchronization phase, the best of all solutions becomes the initial solution for the next parallel phase of an SPDS implementation. For MPSS and MPDS approaches, the p overall best solutions are identified and distributed among the p processes.

Other than in their use of different or similar initial solutions and of different or similar search strategies for the individual tabu threads, the implementations may further differ in the treatment of the *candidate* solution received during the synchronization phase. The first approach consists in single-mindedly replacing the local best solution (i.e., the current best for a given process) with the candidate. This is equivalent to a forced diversification that is not based on the local history of the search. In the second option, one compares the candidate to the best solution found by the process during the current phase of its search. If the candidate is better, the replacement takes place and the algorithm continues. Otherwise, the imported solution is rejected, and the next phase (diversification, usually) is performed in a normal way.

Note that, when a particular process accepts an external solution at synchronization time, we do not modify the lists and memories that reflect its search knowledge and trajectory. The search then proceeds, and eventually diversifies, by using a history that may bear little relation to its current (imported) solution. Yet, one may also remark that the search history recorded into the various long-term memories of a given process, reflects its particular behaviour. This is especially true when processes use different strategies (e.g., SPDS and MPDS implementations), and is exacerbated when processors with different characteristics and speeds are linked together. Hence, this approach allows both to perform “normal” diversification steps (especially when relatively large synchronization steps are used) and to confer a “local” colour to an exploration that possibly starts from the same point as several other search threads. A larger area of the solution domain should thus be covered.

We prefer, and implement, this approach to the alternative strategy that replaces not only the (global best) solution, but also the corresponding context (this is the approach of Malek et al. [15] where the tabu lists are reinitialized). This last is equivalent to arbitrarily restarting the search, and thus, the parallel algorithm reduces to a juxtaposition of several, rather short lived, tabu searches. Furthermore, if one does not pay close attention to the implementation, and if synchronization points are separated by only a few iterations, some algorithmic components might never be executed: most certainly diversification from the process’ own solutions (diversification is disabled in [15]), and even intensification.

4 Experimental Results

Although the parallelization approaches described in the previous section can be implemented on different computing platforms, they are particularly suited to coarse grain, message passing parallel architectures. Hence, all tests have been conducted on a network of SUN Sparc5 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 [16].

We pursue several objectives with our experimental plan: to explore the impact of parallelization on the efficiency of the tabu search procedure, to identify the most promising parallelization strategies, to assess the role of some important design parameters, such as the number of processors, the number of probing iterations, etc. Note that, our objective is to compare parallelization strategies, not to fine-tune a given procedure on a given set of problems. Hence, in order to facilitate the comparisons, we did not calibrate each individual procedure for best performance over the problem set. Instead, we use the best parameter settings observed for the sequential tabu search [6] for all the experiments reported in the remainder of this section.

All procedures are stopped on a number of iterations. Hence, the “wall clock” time of the parallel procedures is similar to the total time of the sequential version for the p-RS (no communications among processes) strategies, while a price in time has to be paid for the extra work involved in the master-slave approaches or for the synchronization operations. Furthermore, the total computing effort of the parallel procedures is larger than that of the sequential version by a factor related to p , the number of processors used. Hence, it is interesting to verify if the increase in computing power is rewarded by an improvement in the behaviour of the algorithm and the quality of the final solution.

Sixteen problems are used for testing: twelve (P1 to P12) randomly generated, and four (P13 to P16) based on an actual application [5]. Problems P7 to P12 are similar to problems P1 to P6 except that fixed costs have been multiplied by 10. Similarly, problems P14 to P16 have been obtained by multiplying the fixed costs in problem P13 by 10, 100 and 1000, respectively. The problem dimensions are indicated in Table 1.

In order not to overload the exposition, we have condensed the results into four tables. The first one, Table 2, summarizes the results obtained by the two master-slave parallelization strategies (1-RS SPSS and 1-KS SPSS), while the three last,

Problem	1/7	2/8	3/9	4/10	5/11	6/12	13-16
Depots	44	44	43	44	44	44	130
Customers	219	219	220	219	219	220	289
Products	1	2	1	2	1	2	12
Customer Links	5260	10520	5282	10516	5262	10588	45936
Location Links	1892	3784	1892	3784	1892	3612	10680

Table 1: Problem Dimensions

Tables 3, 4, and 5, present side-by-side the results obtained by the methods that use several search threads, with (p-KS) and without (p-RS) interprocess communications, by using the SPDS, MPSS, and MPDS search differentiation strategies, respectively. Results are reported for $p = 4, 8$ and 16 processors, and are also graphically illustrated in Figure 2. All procedures are stopped after 300 iterations (plus the postoptimization phase).

The tables display the following information:

- The *Gap*, in percentage, between the best solution determined by each procedure and the optimal solution. Optimal solutions have been computed by a branch-and-bound algorithm [8].
- When the best solution is found during the postoptimization phase, we indicate, in between parentheses, the corresponding gap of the solution from which the postoptimization phase has been initiated.
- The iteration (*Iter*) at which the best solution was found. When the best solution is found during the postoptimization phase, we indicate the iteration corresponding to the solution from which this phase has been initiated.
- The gap information (*Seq*) corresponding to the solution determined by using the sequential procedure.

Note that the results of the sequential procedure used in this paper are generally better than those reported in [6]. This improvement results from an observation made during the development of the parallel implementations: it appears that, due to the limited number of iterations that we use, it is more efficient not to allow the search to explore unfeasible solutions. Note also that the sequential procedure has been

calibrated by using eight out of the twelve problems P1 to P12.

Several general conclusions emerge from these figures:

- All parallel versions perform better, and most of them significantly so, than the sequential algorithm. In fact, over all experiments, the parallel implementations yield an average gap of 0.26% (in an interval ranging from 0.04% to 0.56%) as compared to the sequential average of 0.63%. Hence, for a given allocation of computation time, parallelization allows one to reach better solutions.
- Parallel tabu search procedures help reach high quality solutions. Indeed, with few exceptions, either the optimal solution is found or a very low gap exists between the optimum and the proposed solution.
- Increasing the number of available processors seems to improve the performance.

Before addressing the issue of how the various parallelization strategies compare, and further looking into the relationships between the number of processors and the efficiency of the various procedures, let us examine in more detail the behaviour of each class of procedures.

The implementation of the classical master-slave approach, procedure 1-RS SPSS, makes use of the best strategy and starting point identified for the sequential version. Furthermore, the availability of several processors, permits at each iteration to evaluate all moves (no sampling). This is reflected into the general gain in solution quality apparent in Table 2 and Figure 2. Of course, this also implies that the search trajectory is different from the sequential one and it is thus to be expected that in a few cases one cannot improve on it.

An increase in the number of processors appears beneficial for the master-slave strategy, but the results indicate that a more impressive gain may be expected from augmenting the level of knowledge acquisition. Indeed, the probing (1-KS SPSS) strategy significantly outperforms the classical master-slave approach, and it even appears as an overall competing method. Two issues are relevant to the design of the 1-KS SPSS strategy: the depth of the probing (number of iterations), and the definition of a master, or probing iteration.

In a first variant of the procedure, a master iteration equals the total number of probing iterations. Hence, irrespective of how many probing iterations a slave process performs, the master counts only one, corresponding to its selection of the best solution found and the associated memory updates. This implementation requires, however, that several critical parameters, such as the total number of iterations (this

Prob	p = 4		p = 8		p = 16		Seq
	Gap	Iter	Gap	Iter	Gap	Iter	
1-RS SPSS							
P1	0	27	0	27	0	29	0
P2	0.28 (0.75)	164	0.61 (0.74)	220	0.23 (0.25)	68	0.42 (1.48)
P3	0.03	214	0	212	0	293	0.01 (0.8)
P4	0.42 (0.79)	27	1.03 (1.05)	175	0.15 (0.47)	23	0.90 (1.1)
P5	1.49 (1.74)	223	0	130	0	140	0.77 (1.45)
P6	0 (0.08)	186	0.48 (0.73)	192	0 (0.18)	18	0.42 (3.67)
P7	0.07	245	0.07	179	0	185	0.07
P8	0.79	58	0.79	58	1.32	115	2.15
P9	0	285	0	46	0.82	27	0
P10	0.49	164	0.52	183	0.3	180	1.49
P11	2.22	205	1.95	229	2.55	23	1.52
P12	0	165	0	86	0.11	25	0.11
P13	0.01 (0.06)	42	0 (0.04)	74	0.02 (0.06)	13	0.002 (0.07)
P14	0.06 (1.26)	290	0.02 (0.32)	117	0.02 (0.77)	162	0.02 (0.59)
P15	1.38	95	1.22	145	0	170	1.23 (1.54)
P16	0.89	129	41.26	90	0.14	229	0.89 (9.21)
1-KS SPSS							
P1	0	28	0	30	0	30	0
P2	0.21 (0.62)	20	0.21 (0.62)	16	0.14 (0.55)	18	0.42 (1.48)
P3	0.01	28	0	222	0	50	0.01 (0.8)
P4	0.14 (0.44)	30	0.12 (0.19)	108	0.15	199	0.90 (1.1)
P5	0.65 (0.93)	104	0.65 (0.68)	40	0	267	0.77 (1.45)
P6	0.05 (0.07)	204	0.05 (0.07)	197	0.09	164	0.42 (3.67)
P7	0	124	0.07	65	0	95	0.07
P8	0	162	0	32	0	71	2.15
P9	0	129	0	258	0	271	0
P10	1.42	282	0.3	270	0	98	1.49
P11	1.38	113	0	131	0	104	1.52
P12	0.23	39	0.18	109	0	61	0.11
P13	0.01 (0.03)	21	0.002 (0.02)	51	0.002 (2.93)	23	0.002 (0.07)
P14	0.06 (0.26)	45	0.02 (0.12)	47	0.08 (0.12)	283	0.02 (0.59)
P15	1.22	90	0.78 (0.59)	294	0.59	182	0.23 (1.54)
P16	0.89	91	0.44	182	0.89	46	0.89 (9.21)

Table 2: Master-Slave Implementations

controls the total search time), the total number of consecutive add/drop moves without improving the solution while performing local search, and the number of iterations the status of a depot modified by a diversification move stays tabu, be adjusted. Indeed, the input values of all these parameters are multiplied by the length of the probing phase, and are therefore especially critical when long probing sequences are used. For example, too large a figure for the second parameter and the algorithm essentially becomes an add/drop procedure, without the “time” to perform many (if any) swaps, intensification or diversification phases. Similarly, too long a duration of the diversification related tabu status, and the efficiency of the local search might become seriously impaired. This extensive parameter adjustment phase is not required, however, by an alternative definition of the probing procedure. Here, one counts a master iteration for each slave iteration. Therefore, the master updates not only the current solution and the memories, but its iteration counter as well.

The results of our experiments show that, as expected, the total time increases with the depth of the probing. (We do not include these results in the paper in order not to overload it.) This extra computation effort is not, however, generally compensated by an improvement in solution quality. Quite to the contrary, actually. One may explain this (counter intuitive) phenomenon by noting that, when probing, one searches the neighbourhood in a somewhat less systematic fashion than a traditional neighbour to neighbour search; hence, some solutions might never be encountered, and the search might be directed into less interesting directions. Of course, this may be alleviated by increasing the search length, but our results seem to indicate that it is not worth the corresponding additional computational effort. A short probing phase, however, seems to be beneficial. On the other hand, the second variant appears to outperform the first one. Consequently, we use the second implementation (one master iteration equals one slave iteration) with a probing depth of two iterations for the results presented in this paper.

We also performed a series of experiments related to the design of the three p-control knowledge synchronous procedures described in Section 3. Two issues have been looked into: the synchronization frequency, and the treatment of the exchanged solutions. To avoid overloading the paper, the corresponding results tables are not included, but our experiments indicate that the performance of the p-KS strategies tends to improve when processes are synchronized more often. Of course, the total time (the “wall clock”) required also increases, due to the synchronization overhead. On the other hand, synchronizing too often might change the nature of the algorithm.

Prob	p = 4		p = 8		p = 16		Seq
	Gap	Iter	Gap	Iter	Gap	Iter	
	p-RS SPDS						
P1	0	98	0	17	0	17	0
P2	0.07 (0.16)	86	0.07 (0.16)	86	0.07 (0.16)	86	0.42 (1.48)
P3	0.01	158	0.01	158	0 (0.32)	271	0.01 (0.8)
P4	0.74 (0.81)	225	0.27 (0.67)	105	0.05 (0.73)	185	0.90 (1.1)
P5	0.77 (1.45)	147	0.002 (0.12)	296	0.002 (0.12)	296	0.77 (1.45)
P6	0.28 (1.89)	118	0 (0.63)	71	0 (0.63)	71	0.42 (3.67)
P7	0	138	0	138	0	88	0.07
P8	1.68	50	0	255	0	255	2.15
P9	0	82	0	82	0	82	0
P10	1.23 (1.27)	25	1.02	264	0.60	208	1.49
P11	0.61	55	0.61	38	0	203	1.52
P12	0.11	17	0.11	17	0.11	17	0.11
P13	0 (0.12)	161	0 (0.12)	161	0 (0.12)	161	0.002 (0.07)
P14	0.02 (0.59)	299	0.02 (0.56)	299	0.02 (0.56)	299	0.02 (0.59)
P15	0.26 (0.29)	274	0.26 (0.29)	274	0.26 (0.29)	274	1.23 (1.54)
P16	0.52	278	0.52	278	0.23	291	0.89 (9.21)
	p-KS SPDS						
P1	0	273	0	116	0	116	0
P2	0.14 (0.66)	299	0.14 (0.54)	273	0.14	249	0.42 (1.48)
P3	0.57	221	0.57	221	0.01	220	0.01 (0.8)
P4	0.21 (0.55)	249	0.21 (0.79)	244	0.12 (0.41)	294	0.90 (1.1)
P5	0.04 (1.25)	204	0.997 (1.00)	227	0.34 (0.49)	254	0.77 (1.45)
P6	0.05 (0.66)	228	0	190	0	115	0.42 (3.67)
P7	0.07	240	0.07	240	0.07	240	0.07
P8	0	226	0	165	0	165	2.15
P9	0	82	0	82	0	82	0
P10	2.27	28	2.27	28	1.49	161	1.49
P11	2.22	185	2.22	185	2.22	185	1.52
P12	0	220	0	169	0	88	0.11
P13	0 (0.02)	249	0 (0.02)	249	0 (0.02)	249	0.002 (0.07)
P14	0.02 (0.13)	224	0.02 (0.10)	282	0.02 (0.05)	175	0.02 (0.59)
P15	1.23	278	0	126	0.29	286	1.23 (1.54)
P16	0.02	275	0.02 (0.10)	282	0.02	140	0.89 (9.21)

Table 3: p Search Threads – SPDS Results

Prob	p = 4		p = 8		p = 16		Seq
	Gap	Iter	Gap	Iter	Gap	Iter	
p-RS MPSS							
P1	0	18	0	18	0	18	0
P2	0.14 (1.18)	105	0.14 (1.18)	105	0.14 (0.42)	78	0.42 (1.48)
P3	0	103	0	103	0	103	0.01 (0.8)
P4	0.08 (1.07)	192	0.08 (1.07)	192	0.08 (1.07)	192	0.90 (1.1)
P5	0.34 (0.98)	217	0.04 (1.54)	146	0.03 (1.91)	222	0.77 (1.45)
P6	0 (2.20)	156	0 (2.20)	156	0 (2.20)	156	0.42 (3.67)
P7	0.07	294	0	263	0	263	0.07
P8	0	215	0	215	0	98	2.15
P9	0	218	0	123	0	123	0
P10	0.37 (1.98)	180	0.37 (1.98)	180	0	118	1.49
P11	0	246	0	141	0	79	1.52
P12	0	264	0	264	0	17	0.11
P13	0.002 (0.10)	299	0.002 (0.10)	299	0 (0.03)	299	0.002 (0.07)
P14	0 (3.42)	204	0 (3.42)	204	0 (3.42)	204	0.02 (0.59)
P15	1.23 (1.35)	245	1.23 (1.35)	245	1.23 (1.35)	245	1.23 (1.54)
P16	0.89 (3.62)	268	0.89 (3.62)	268	0.89 (3.62)	268	0.89 (9.21)
p-KS MPSS							
P1	0	65	0	65	0	65	0
P2	0.79 (1.05)	223	0.5 (1.49)	293	0.64 (1.76)	273	0.42 (1.48)
P3	0.01	275	0.01	182	0.01	112	0.01 (0.8)
P4	0.33 (0.59)	146	0.42 (1.97)	297	0.14 (1.57)	299	0.90 (1.1)
P5	0.38 (0.58)	127	0.38 (0.59)	233	0.38	232	0.77 (1.45)
P6	0 (0.43)	158	0 (0.56)	291	0 (1.09)	173	0.42 (3.67)
P7	0	52	0	139	0	274	0.07
P8	0	290	0	231	0.85 (4.15)	272	2.15
P9	0	289	0	296	0	174	0
P10	2.04	289	1.13 (46.09)	70	1.42 (5.51)	299	1.49
P11	0.61	271	0.61	205	0.61 (17.62)	36	1.52
P12	0	220	0.67 (12.05)	268	0	199	0.11
P13	0.007 (0.04)	275	0.01 (0.04)	273	0.007 (0.02)	271	0.002 (0.07)
P14	0.04 (0.24)	295	0 (0.26)	239	0 (0.02)	299	0.02 (0.59)
P15	0	299	0	233	0	226	1.23 (1.54)
P16	0.44	298	0.25	277	0.02	244	0.89 (9.21)

Table 4: p Search Threads – MPSS Results

For example, if the synchronization step length is shorter than the number of consecutive add/drop moves without improvements (this parameter signals an intensification phase - Section 2), the procedure will never perform intensification or diversification phases. Hence, for the type and dimension of problem we solve, a synchronization step of 25 iterations seemed appropriate, and has been used for the experiments reported in Tables 3, 4, and 5.

Concerning the second issue, we tested two ways of handling an imported solution at synchronization time. In both variants the improving solutions found by all processes are sorted, and the best ones are distributed among processes. (Note that each process keeps a list of all locally improving solutions found between synchronization phases, and that these lists are then merged and sorted. Note also, that each process is identified by an index, from 1 to p , and receives the corresponding solution. This implementation is used for the both variants.) The first variant then implements a simple import scheme: each process accepts the imported solution as a new starting point, initializes all tabu lists and memories, and initiates a new search. We identify this variant (similar to the one used by [15]) as *Simple Import*. In the second variant, called *Conditional Import*, each process first tests the imported solution against the best solution of its current outer loop (see Section 2), and accepts the new one only if it is better. Note that, in this case, the short term tabu lists are initialized, while the long term memories are not. The results of these experiments were conclusive: one notes that the conditional import strategy consistently yielded similar or better results than the simple import variant.

The best performances overall have been obtained by the group of p-RS strategies, where several independent search threads are run simultaneously, without any communication among processes. We observe here that a greater diversification of the search yields greater dividends: an SPDS approach outperforms a MPSS one, while being significantly outperformed by the MPDS one. An observation of the fundamental nature of this group of methods may help explain these performances.

It is noteworthy that the strategies of the p-RS group are the only ones among those considered in this paper that do not require any particular calibration. Indeed, they all make use at least once of the best strategy, and initial solution (number of depots with high fixed costs initially closed), calibrated for the sequential case. When several search strategies may be used, the SPDS and MPDS cases, all strategies that were “almost as good” in the sequential case may also be used, which improves the performance of the algorithm. In fact, more different search threads imply a wider exploration of the domain and a better performance, which explains why the performance of MPDS and SPDS strategies improves with the number of processors, while this factor does not significantly impact the performance of the MPSS method. On the other hand, no communication among processes means that the domain exploration

Prob	p = 4		p = 8		p = 16		Seq
	Gap	Iter	Gap	Iter	Gap	Iter	
	p-RS MPDS						
P1	0	21	0	21	0	18	0
P2	0.21 (1.87)	165	0.21 (0.54)	179	0.14 (1.57)	120	0.42 (1.48)
P3	0 (0.83)	89	0	236	0	178	0.01 (0.8)
P4	0.35 (1.04)	250	0.32 (1.28)	213	0.12 (1.69)	235	0.90 (1.1)
P5	0.38 (0.58)	37	0 (1.04)	54	0 (1.04)	54	0.77 (1.45)
P6	0 (1.57)	121	0 (1.15)	290	0 (1.15)	290	0.42 (3.67)
P7	0.07	182	0	264	0	264	0.07
P8	0	232	0	232	0	127	2.15
P9	0	259	0	76	0	76	0
P10	0.37	91	0.37	91	0.37	91	1.49
P11	0	121	0	121	0	121	1.52
P12	0.11 (2.27)	202	0 (0.30)	170	0 (0.30)	170	0.11
P13	0.002 (0.02)	22	0 (0.12)	71	0 (0.12)	71	0.002 (0.07)
P14	0 (0.48)	117	0 (0.48)	117	0 (0.48)	117	0.02 (0.59)
P15	0 (0.34)	133	0 (0.34)	133	0 (0.34)	133	1.23 (1.54)
P16	0.89 (3.62)	102	0.02 (11.22)	187	0.02	150	0.89 (9.21)
	p-KS MPDS						
P1	0	80	0	265	0	113	0
P2	0.23 (1.52)	284	0.33 (1.21)	197	0.21 (1.02)	287	0.42 (1.48)
P3	0	177	0.01	252	0 (0.32)	229	0.01 (0.8)
P4	0.38 (1.23)	279	0.38 (1.50)	299	0.38 (1.60)	269	0.90 (1.1)
P5	0.87 (1.00)	184	0 (.04)	255	0 (0.78)	226	0.77 (1.45)
P6	0.05 (0.48)	228	0 (.60)	95	0 (0.62)	140	0.42 (3.67)
P7	0.07	126	0	240	0.66	7	0.07
P8	1.32	79	2.15	236	1.32 (6.47)	177	2.15
P9	0.82	201	0.82	77	0.82	140	0
P10	2.22 (3.37)	151	1.08 (2.37)	151	1.45	160	1.49
P11	0	286	0 (3.14)	290	0.62	128	1.52
P12	1.25	299	0	254	0 (0.30)	240	0.11
P13	0.001 (0.01)	299	0 (0.04)	205	0 (0.08)	263	0.002 (0.07)
P14	0.06 (0.3)	225	0.05 (0.26)	299	0.02 (0.18)	284	0.02 (0.59)
P15	1.23 (1.35)	276	0	234	0	254	1.23 (1.54)
P16	0.44	263	0.44	227	0	239	0.89 (9.21)

Table 5: p Search Threads – MPDS Results

logic defined by the sequential tabu search procedure is not disturbed. Consequently, we are guaranteed a performance at least as good as that of the sequential algorithm, and, in practice, significantly more so.

This is certainly not the case with the group of p-KS strategies. Or, it appears that, even if an exchange of information concerning the best solutions found so far is beneficial as compared to the sequential performance, the injection at any moment of a “foreign” solution into a tabu search trajectory may adversely affect its performance. In fact, if the tabu search logic is to be observed, imported solutions should be accepted only at particular moments, diversification instants appearing as the most appropriate. This implies that either the acceptance of an imported solution should be delayed until precise conditions are met, or the synchronization criterion should be modified (one could, for example, synchronize processes when all reach a diversification phase). And the tabu search procedure should be calibrated anew to fully exploit the parallel environment. Such a study is, however, clearly outside the scope of this paper which aims at evaluating procedures in an “objective” basic parameter setting. In this case, and if a synchronous strategy is contemplated, it appears that the simple strategy of running several search threads from various initial points offers good guarantees for success.

We have run all procedures on 4, 8 and 16 processors for 300 iterations, and the evolution of the average gaps is illustrated in Figure 2. First, it is clear that a parallel implementation definitely improves the performance of the algorithm. On the other hand, we observe that, for a constant number of iterations, increasing the number of processors improves the performances of the algorithms, but up to a point. Indeed, while improvements are significant as one passes from 4 to 8 processors, the improvement curve tends to flatten out when 16 processors are brought into play. In this respect, synchronous tabu search appears to behave as many other optimization algorithms in a parallel environment. Furthermore, additional testing on a subset of problems suggest that beyond a certain threshold (certainly related to problem size; $p = 8$, in the case of the problem instances considered in this paper) it is more profitable to increase the length of the search than the number of processors. One may explain this phenomenon by a combination of two factors. First, when the number of processors increases, many of them will explore uninteresting regions due either to bad starting points, or to less-than-efficient search parameter settings, or a combination of the two (there are, after all, only so many interesting parameter settings). Second, increasing the number of iterations permits to the long term memory functions of tabu search to come into play, and to adequately guide the search.

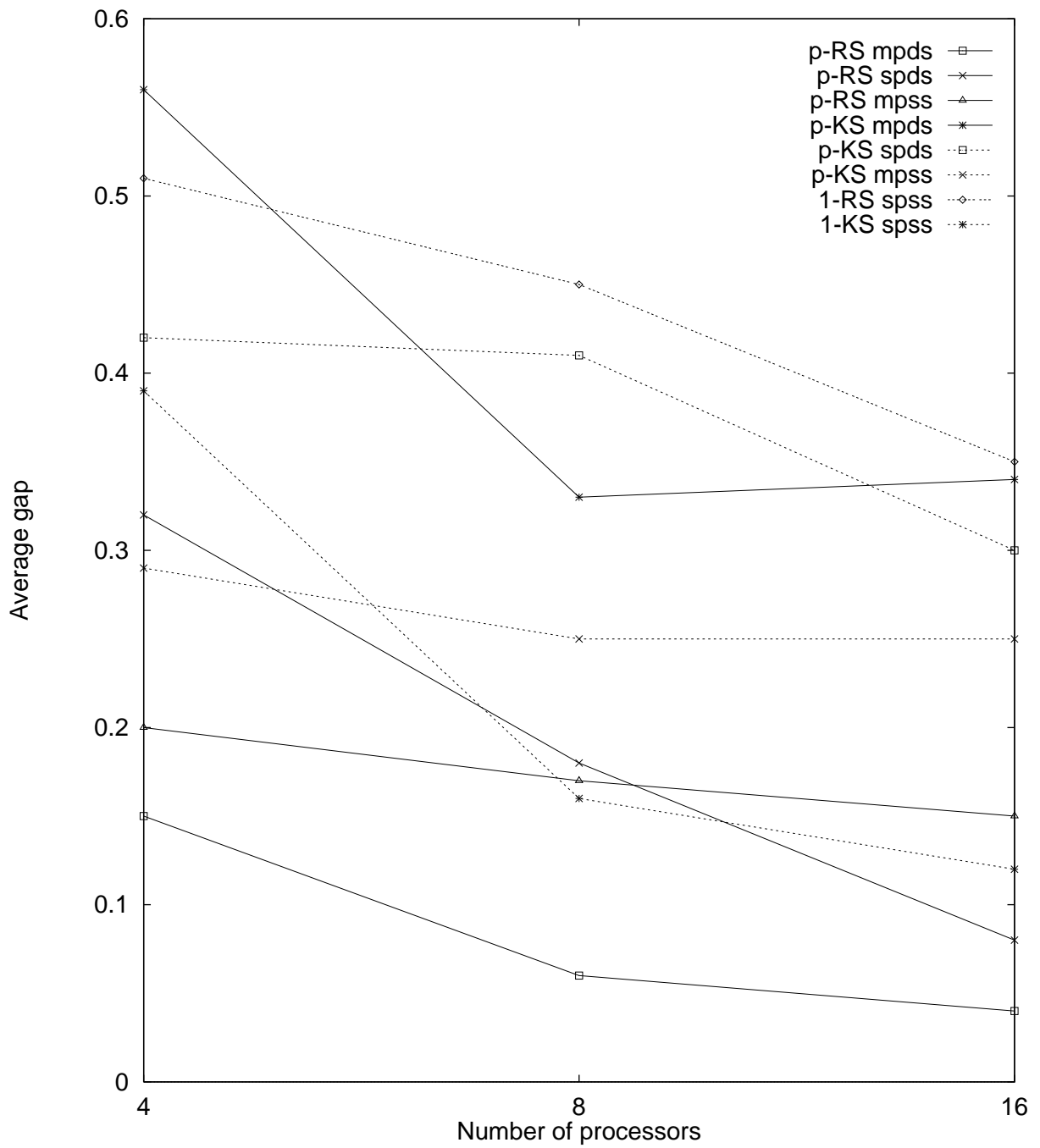


Figure 2: Gap Comparisons

With respect to computing times, the most relevant performance measure for synchronous strategies is certainly the "wallclock" time required to complete a run. In a distributed computing environment, such as ours, where processors communicate through a network shared with other concurrent applications, wallclock times can fluctuate significantly with the network traffic load, and it is therefore difficult to devise meaningful comparisons on the basis of individual problems. The following discussion thus focuses on the average times observed for the various strategies over the complete set of benchmarks.

It must first be noted that all strategies require a substantial amount of computing time with an average over all strategies and problems of slightly more than 10 minutes by problem. As could be expected beforehand, times increase significantly with the amount of communication required by the strategies: the p-RS approaches, where communication only occurs once at the end of the solution procedure, run faster than the p-KS ones with a synchronization step every 25 iterations, while these require about half the time used by the communication intensive 1-RS approaches. Computing times also increase in general with the number of processors, since the time between synchronization steps is that of the slowest processor and since more messages must be exchanged when more processors are involved. This is particularly significant in the case of the p-KS strategies where computing times for $p = 16$ are about 20 % longer than for $p = 4$.

Finally, it is noteworthy that the procedures behave in a consistent way over the entire problem set. This is despite the fact that the sequential procedure has been calibrated over a subset of the first twelve problems, and that problems P13 to P16 are significantly different, both in size and in the design of the underlying network. This emphasizes the robustness of the parallel approaches.

5 Conclusions and Perspectives

We have presented a study of synchronous parallelization strategies for tabu search. Several approaches have been compared, and the influence of several important parameters, such as the length of the synchronization step, the number of processors, etc., on the algorithmic efficiency and the solution quality have been analyzed. We have shown, in particular, that parallelization may be very profitable for tabu search, and significantly improve the algorithmic performance and the solution quality.

Our experimental results demonstrate that how often information is exchanged among processes, and how this information is treated may significantly affect the performance of a synchronous parallel tabu search method. Results also show that search strategies based on parameter settings that do not perform well for the sequential case do not contribute much to the improvement of the global solution; consequently, several search threads work without a global benefit. Therefore, care has to be taken in designing synchronous parallelization strategies.

As the taxonomy proposed in [7] has shown, several avenues exist for parallel tabu search. A particularly intriguing one consists in combining several such strategies. For instance, low-level parallelization may be applied to various parts of a given tabu search thread (such as, the exploration of the neighbourhood or the evaluation of a move by an SPSS master-slave approach), while all threads are interacting in a larger synchronous or asynchronous parallel scheme. Such a heterogeneous parallelization strategy opens up a large and fascinating research domain that we plan to explore in the near future.

Acknowledgements

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. We would like to thank Mr. Pierre Bélanger for his assistance with the execution of the multiple runs required by this study. The authors also wish to thank the referees for their valuable comments on an earlier version of this paper.

References

- [1] J. Chakrapani and J. Skorin-Kapov. A Connectionist Approach to the Quadratic Assignment Problem. *Computers & Operations Research*, 19(3/4):287–295, 1992.
- [2] 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.
- [3] J. Chakrapani and J. Skorin-Kapov. Massively Parallel Tabu Search for the Quadratic Assignment Problem. *Annals of Operations Research*, 41:327–341, 1993.
- [4] 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.
- [5] 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.
- [6] 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.
- [7] T.G. Crainic, M. Toulouse, and M. Gendreau. Towards a Taxonomy of Parallel Tabu Search Algorithms. Publication 933, Centre de recherche sur les transports, Université de Montréal, 1993.
- [8] B. Gendron and T.G. Crainic. A Branch-and-Bound Algorithm for Depot Location and Container Fleet Management. *Location Science*, 1995.
- [9] F. Glover. Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research*, 1(3):533–549, 1986.
- [10] F. Glover. Tabu Search - Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.
- [11] F. Glover. Tabu Search - Part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.
- [12] F. Glover. Tabu Search: A Tutorial. *Interfaces*, 20(4):74–94, 1990.
- [13] 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.

- [14] F. Glover, E. Taillard, and D. de Werra. A user's guide to tabu search. *Annals of Operations Research*, 41:3–28, 1993.
- [15] 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.
- [16] Grigoriadis M.D. and Hsu T. RNET – The Rutgers Minimum Cost Network Flow Subroutines. Technical report, Rutgers University, New Brunswick, New Jersey, 1979.
- [17] E. Taillard. Robust Taboo Search for the Quadratic Assignment Problem. *Parallel Computing*, 17:443–455, 1991.
- [18] E. Taillard. Parallel Iterative Search Methods for Vehicle Routing Problems. *NETWORKS*, 23:661–673, 1993.
- [19] 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.
- [20] 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.