Pergamon

Transpn Res.-C, Vol. 5, No. 2, pp. 109–122, 1997
© 1997 Elsevier Science Ltd
All rights reserved. Printed in Great Britain
0968-090X/97 $17.00+0.00

PII: S0968-090X(97)00005-3

A PARALLEL TABU SEARCH HEURISTIC FOR THE VEHICLE ROUTING PROBLEM WITH TIME WINDOWS

PHILIPPE BADEAU
Centre Universitaire des Sciences et Techniques, Université Blaise Pascal, Clermont Ferrand II, Campus Universitaire des Cézeaux, 63174 Aubièere Cedex, France

FRANÇOIS GUERTIN
Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succ. Centre-ville, Montréal, Québec, Canada H3C 3J7

MICHEL GENDREAU, JEAN-YVES POTVIN
Centre de recherche sur les transports and Département d'informatique et de recherche opérationnelle, Université de Montréal, C.P. 6128, succ. Centre-ville, Montréal, Québec, Canada H3C 3J7

and

ERIC TAILLARD
Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA), Corso Elvezia 36, CH-6900 Lugano, Switzerland

Abstract—The vehicle routing problem with time windows models many realistic applications in the context of distribution systems. In this paper, a parallel tabu search heuristic for solving this problem is developed and implemented on a network of workstations. Empirically, it is shown that parallelization of the original sequential algorithm does not reduce solution quality, for the same amount of computations, while providing substantial speed-ups in practice. Such speed-ups could be exploited to quickly produce high quality solutions when the time available for computing a solution is reduced, or to increase service quality by allowing the acceptance of new requests much later, as in transportation on demand systems. © 1997 Elsevier Science Ltd

1. INTRODUCTION

The vehicle routing problem with time windows (VRPTW) is a useful abstraction of many real-world problems like bank deliveries or postal deliveries or school bus routing. In these problems, a time window is often associated with each customer location to constrain the time of service.

Formally, the VRPTW can be stated as follows. Given a fleet of identical vehicles housed at a central depot and a set of customers requiring service (e.g. a quantity of goods to be delivered), a set of feasible routes starting from and terminating at the depot must be constructed so that each customer is visited exactly once, and the total distance traveled by the vehicles is minimized.

In order to be feasible, each route must also satisfy three types of constraints:

(a) The total load on a route cannot exceed the capacity of the vehicle servicing the route.
(b) The time of beginning of service at each customer location must occur before its time window's upper bound. However, a vehicle can arrive before the lower bound. In this case, the vehicle must wait, thus introducing a waiting time on the route.
(c) The route of each vehicle must be serviced within the bounds of the time window associated with the depot.

Two variants of this problem may be defined by introducing either hard or soft time windows at the customer locations. In the latter case, the time of beginning of service at any given customer location can occur after the time window's upper bound (although the time window at the central depot must still be strictly satisfied). The variant with soft time windows reduces to the hard one by adding a large penalty to the objective value when the time window constraints are not satisfied. Our problem-solving methodology addresses problems with soft time windows (and, thus, problems with hard time windows, too).

Because of its wide applicability in practical settings, the VRPTW has been an area of intense research during the last ten years. Excellent surveys may be found in Desrochers *et al.* (1988), Desrosiers *et al.* (1992) and Solomon and Desrosiers (1988). Generally speaking, the methodologies for solving this problem can be classified as:

(a) *exact algorithms*—Kolen *et al.* (1987), Desrochers *et al.* (1992);
(b) *route construction heuristics*—Solomon (1987), Potvin and Rousseau (1993), Russell (1995);
(c) *route improvement heuristics*—Baker and Schaffer (1988), Potvin and Rousseau (1995), Solomon *et al.* (1988), Thompson and Psaraftis (1993);
(d) *composite heuristics* that include both route construction and route improvement procedures—Kontoravdis and Bard (1995), Russell (1995);
(e) *metaheuristics*—e.g. tabu search (Rochat and Taillard, 1995; Taillard *et al.*, 1995; Potvin *et al.*, 1996), simulated annealing (Chiang and Russell, 1993), genetic algorithms (Blanton and Wainwright, 1993; Thangiah, 1993; Potvin and Bengio, 1996) and hybrids (Thangiah *et al.*, 1994).

This paper describes a parallel tabu search for the VRPTW. Parallelization provides benefits in practical settings where routes must be produced within a short time interval. In the local pick-up portion of less-than-truckload transportation, for example, new routes must sometimes be quickly produced in response to a batch of new pick-up requests. Parallelization can also increase service quality by allowing either a larger number of requests to be routed within reasonable computation time, or by allowing the acceptance of new service requests much later, as in transportation on demand systems.

Parallel computers can be exploited to accelerate the computationally intensive phases of the tabu search heuristic to maintain different search paths in parallel or to tackle smaller subproblems through decomposition of the original problem. In the first case, one processor typically runs the tabu search and asks other processors to perform computationally intensive subtasks, like generating and evaluating the neighborhood of the current solution. A high degree of synchronization is required to implement this type of parallelism. Illustrative examples may be found in Crainic *et al.* (1995), Garcia *et al.* (1994) and Taillard (1990, 1991, 1994). An alternative approach, exploiting the fine-grained parallelism of a Connection Machine, is reported in Chakrapani and Skorin-Kapov (1993).

In the second case, multiple processors maintain many search threads (paths) in parallel. Each tabu search process starts from a different initial solution or uses a distinctive set of parameter values to explore a particular region of the search space. Here, the processors are much less tightly coupled. A certain degree of synchronization is required when information is periodically shared through a central processor that reinitializes the search processes (Malek *et al.*, 1989; Rego and Roucairol, 1995; Taillard, 1990, 1991, 1994). The implementation can also be mostly asynchronous when the processors communicate through a common repository that contains updated information about the search. The processors get information from or put information in this repository at any time along their respective search path (Crainic *et al.*, 1993a).

In the third case, the main problem is decomposed into a number of smaller subproblems, each subproblem being solved by a different tabu search process. This approach is proposed in Taillard (1993) in the context of vehicle routing. Note that a classification scheme for parallel tabu search approaches may be found in Crainic *et al.* (1993b).

Our implementation runs on a network of workstations and follows a master-slave scheme. Each slave performs a tabu search and the master co-ordinates the work and feeds the slaves with new starting solutions. This implementation borrows from the parallelization approaches presented above since (1) many different search threads run in parallel, and (2) a decomposition method is applied to the main problem within each search thread to produce a number of smaller subproblems.

Section 2 first introduces our tabu search heuristic for the VRPTW (Taillard *et al.*, 1995). Section 3 describes the parallel implementation. Section 4 illustrates the benefits of this parallel implementation over a sequential one using standard test problems. Finally, Section 5 provides concluding remarks.

2. THE PROBLEM-SOLVING METHODOLOGY

Our algorithm is based on tabu search and follows the general guidelines provided in Glover (1989, 1990). It is also characterized by the exploitation of an adaptive memory (Rochat and Taillard, 1995). In the following, the tabu search heuristic is first introduced. Then, the overall problem-solving methodology is presented.

2.1 Tabu search

The tabu search heuristic was first introduced in Glover (1986). Starting from some initial solution, a neighborhood of solutions is generated through a particular class of transformations. In vehicle routing applications, for example, a 2-exchange (Lin, 1965) can be used to create a new solution through the replacement of two links in the current solution by two new links. For a problem of size n, there are $O(n^2)$ different ways to remove two links in a given solution. Thus, the size of the neighborhood is $O(n^2)$. The best solution in this neighborhood is then selected as the new current solution, and the procedure is repeated. As opposed to classical local search heuristics, the tabu search does not stop at the first local optimum (i.e. when no improving solution is found). The best solution in the neighborhood is always selected as the new current solution, even if it is not improving. This approach allows the method to escape from poor local optima to explore other regions of the search space. To avoid cycling, transformations leading to recently visited solutions are forbidden (tabu). Thus, a memory (tabu list) is used to remember the recent search trajectory.

Our tabu search heuristic for the VRPTW can be summarized as follows:

```
Set the current solution to an initial set of routes.
While the stopping criterion is not met do:
    Generate the neighborhood of the current solution.
    Select the best (non-tabu) solution in this neighborhood and set it as the new current
    solution.
    If the current solution is better than the best known solution then:
        set the best known solution to this solution;
        if the current solution is a local optimum, reorder the customers within each route
        using Solomon's I1 insertion heuristic for possible improvement (Solomon, 1987).
    Update the tabu list.
Return the best known solution.
```

The main components of this tabu search are now briefly introduced:

2.1.1. Objective. An important characteristic of our tabu search is its ability to explore solutions that violate the upper bound of the time windows. In such a case, a penalty for lateness is incurred. Accordingly, the objective to be minimized is:

$$\textit{total distance} + \alpha \times \sum_{i=1}^{n} \textit{lateness}_i,$$

where n is the number of customers and $\textit{lateness}_i$ is the lateness at customer i. The weighting parameter α associated with the lateness penalty is fixed. In the experiments of Section 4, where the benchmark problems are of the hard time window type, this parameter was set to 100 to drive the search to feasible regions.

2.1.2. Neighborhood. The tabu search exploits a neighborhood structure specifically designed for problems with time windows. The method for generating the neighborhood, called the CROSS exchange, is illustrated in Fig. 1. In this figure, the black square stands for the depot and the white circles are customers along the routes (note that the depot is duplicated at the start and at the end of a route). First, the two edges (X_1, X'_1) and (Y_1, Y'_1) are removed from the first route while the edges (X_2, X'_2) and (Y_2, Y'_2) are removed from the second route. Then, the two segments X'_1-Y_1 and X'_2-Y_2, which may contain an arbitrary number of customers, are swapped. by introducing the new edges (X_1, X'_2), (Y_2, Y'_1), (X_2, X'_1) and (Y_1, Y'_2),

The 2-opt* (Potvin and Rousseau, 1995) and the Or-opt (Or, 1976) are special cases of this operator. The 2-opt* only exchanges two edges taken from two different routes, and is obtained when Y_1 and Y_2 are directly connected to the depot. An Or-opt exchange moves a sequence of three consecutive customers or less from one route to another. It can be obtained, for example, by setting $X_2 = Y_2$ and $X'_2 = Y'_2$ so that an empty segment is removed from the second route. Since the sequence removed from the first route must contain three customers at most, Y_1 is either X'_1 or the first successor of X'_1 or the second successor of X'_1. In a few cases, a CROSS exchange can lead to the removal of an entire route. For example, if X'_1 is the first customer and Y_1 is the last customer on the first route, while $X_2 = Y_2$ and $X'_2 = Y'_2$, the first route is entirely inserted between X_2 and X'_2.

Note that the orientation of the routes (implicitly defined by the time windows) is preserved by a CROSS exchange. Furthermore, by selecting the exchange that leads to the largest improvement to the current solution, the swapping of segments of routes that are close from a spatial and temporal point of view is favored. One drawback is the complexity of this method. Assuming that n customers are evenly distributed among m routes, the complexity of the method is $O(\frac{n^4}{m^2})$. This neighborhood being quite large, simplification and approximation procedures are proposed in (Taillard *et al.*, 1995) to reduce its size and speed up its evaluation without compromising solution quality. In particular, exchanges are limited to route segments whose length is at most L, with L typically set to 7.

Exchanges that apply to a single route are also considered and are similar to the ones defined on a pair of routes. Namely, two edges are removed from a given route, and the segment between the two edges is moved at another location *within the same route*. This approach generalizes the Or-opt heuristic (Or, 1976) by allowing the relocation of segments of any arbitrary length.

2.1.3.Tabu list. The tabu list has length T and its positions are indexed from 0 to $T-1$. Each solution is associated with a position in the list. This position is the objective value of the solution modulo T and the value stored at this place is the iteration number at which the solution will lose its tabu status. When a neighboring solution is produced through a CROSS exchange, its objective value modulo T provides its position in the tabu list. If the value found at this position is larger than the current iteration number then the move is tabu, otherwise it is accepted. Clearly, this approach can filter out legitimate solutions. For example, two solutions will collide at the same position within the list if their objective values differ by a multiple of T. However, T is set to a large value so that such an occurrence is unlikely. The tabu tenure is equal to the number of iterations divided by 2.

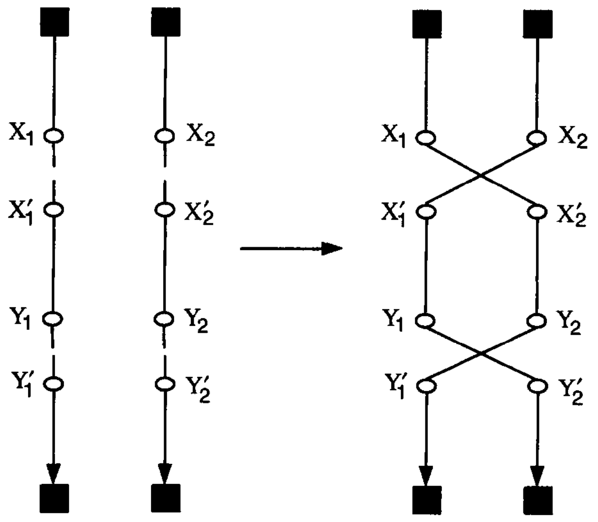

Fig. 1. The CROSS exchange.

2.1.4. Diversification. Diversification is incorporated into the tabu search by penalizing frequently performed CROSS exchanges (Taillard *et al.*, 1995). Due to the penalty, these exchanges are ignored when the neighborhood of the current solution is explored, thus driving the search in new regions of the search space.

2.1.5. Intensification. The customers within each individual route are reordered when a new best local optimum is found. This reordering is based on Solomon's I1 insertion heuristic (Solomon, 1987). In our implementation, Solomon's heuristic is applied 20 times with different parameter settings, and the best route is kept at the end. This strategy can be seen as a form of intensification in the neighborhood of an elite solution. Note that the original route is restored if the new route does not contain all customers (due to the hard time window at the depot) or if the new route is worse than the original route.

2.1.6. Stopping criterion. The tabu search stops after a prespecified number of iterations (see Decomposition/reconstruction in Section 2.2 for details).

2.2 The algorithm

The tabu search heuristic presented in Section 2.1 is embedded within the following algorithm:

```
Generate $I$ different solutions with a stochastic insertion heuristic.
Apply the tabu search heuristic to each solution and store the resulting routes in the adaptive
memory.
While the stopping criterion is not met do:
    Construct an initial solution from the routes found in the adaptive memory, and set this
    solution as the current solution.
    For $C$ cycles of decomposition/reconstruction do:
        Decompose the current solution into a number of disjoint subsets of routes (subprob-
        lems)
        Apply the tabu search heuristic on each subproblem
        Reconstruct a new current solution by merging the subsets of routes produced
        through tabu search.
    Store the routes of the current solution in the adaptive memory.
Apply a post-optimization procedure to each individual route of the best solution found.
```

Thus, a simple insertion heuristic provides starting solutions. The routes produced by the tabu search from these initial solutions are then stored in the adaptive memory. As the algorithm proceeds, the routes of the best solutions produced are kept in this memory to provide a means to construct new starting solutions of high quality for the tabu search through selection and combination of routes stored in the memory. The algorithm stops when a fixed number of solutions have been constructed from the adaptive memory.

This problem-solving scheme lends itself naturally to parallel implementations. First, many tabu search processes can run concurrently using different initial solutions constructed from the adaptive memory. Second, parallelism can be exploited through the decomposition/reconstruction mechanism that partitions a problem into a number of subproblems.

In the following, the main components of this algorithm are briefly introduced (note that a complete description may be found in Taillard *et al.*, 1995). Section 3 will then focus on the parallel issues.

2.2.1. Initialization. A randomized insertion heuristic constructs the first I starting solutions. This heuristic works as follows. First, seed customers are selected to initialize a set of routes (with one seed customer associated with each route). Then, the remaining unrouted customers are inserted one by one into these routes in a random order. The insertion location for a given customer is chosen by minimizing a weighted sum of detour and service delay (Solomon, 1987).

2.2.2. The adaptive memory. The adaptive memory contains routes associated with the best solutions produced during the search and is used to provide new starting solutions for the tabu search (Rochat and Taillard, 1995). This is done through selection and combination of routes found in the memory. This way of combining components of different solutions to create a new solution is reminiscent of the crossover operators of genetic algorithms (Holland, 1975). However, the number of 'parent' solutions is generally greater than two in our case.

The route selection process for creating a new solution is probabilistically biased in favor of the best routes in the memory. That is, routes associated with better solutions have a higher probability of being included in the new solution. Once the first route is selected, the route with one or more customers in common with this route are excluded from the selection process. Then, a second route is selected among the remaining routes. This procedure is repeated until the set of selected routes services all customers or until no admissible routes are found in the memory. In the latter case, Solomon's I1 heuristic (Solomon, 1987) is applied to insert the remaining customers. If this insertion procedure cannot accommodate all customers, an additional route is created for the remaining unrouted customers.

The routes of any new solution returned by the tabu search are stored if the adaptive memory is not filled yet. Otherwise, the new solution must be better than the worst solution found in the memory. In this case, the routes associated with the worst solution are replaced by the new routes.

2.2.3. Decomposition/reconstruction (D&R). Each starting solution is partitioned into D disjoint subsets of routes, with each subset or subproblem being processed by a different tabu search (Taillard, 1993). When every subproblem is solved, the new routes are simply merged together to form the new current solution. The decomposition is based on the polar angle associated with the center of gravity of each route. Using these polar angles, the domain is partitioned into sectors that approximately contain the same number of routes.

A total of C cycles of decomposition/reconstruction take place before the final solution is sent to the adaptive memory. At each cycle, the decomposition (i.e. the subset of routes in each subproblem) changes by choosing a different starting angle to construct the sectors. Given that the solution produced by the tabu search at a given cycle is used to start the next cycle, the solution is continuously improving.

Note that the number of iterations performed by the tabu search heuristic increases from one D&R cycle to the next according to the following formula:

$$a \times \left(1 + \frac{c-1}{b}\right)$$

In this formula, a and b are parameters and c is the current D&R cycle, $c = 1, \ldots, C$. In the experiments of Section 4, a and b are set to 30 and 3, respectively, and there are $C = 6$ cycles. Hence, 30 tabu search iterations are performed during the first D&R cycle. Then, this number increases to 40, 50, 60, 70 and 80 iterations during the second, third, fourth, fifth and sixth D&R cycle, respectively (for a total of 330 iterations).

2.2.4. Post-optimization of each individual route. Each individual route in the final solution produced by our algorithm is optimized with a specialized heuristic developed for the TSP with time windows (Gendreau *et al.*, 1995). This method is an adaptation of the GENIUS heuristic, originally devised for the classical TSP (Gendreau *et al.*, 1992). The post-optimization procedure only slightly improves the total distance of the final solution (the improvement is typically much less than 1% on average), but it is not computationally expensive as it runs for only a few seconds.

3. PARALLELIZATION

When one is faced with the task of selecting an overall parallelization approach for any algorithm, the nature of the parallel computing platform on which the algorithm will run is a critical factor to account for. In our case, the computing environment is a network of workstations connected with medium-speed links (a few megabits per second). It is thus a coarse grain, loosely connected architecture and the parallel version of our algorithm was designed accordingly. In an environment with a finer granularity, a more standard parallelization technique based on loops decomposition would almost certainly be more effective. In particular, it would speed up the neighborhood evaluation which is by far the most time-consuming portion of the algorithm.

A natural way to parallelize adaptive memory algorithms is a master-slave scheme in which the master process manages the adaptive memory and generates solutions from it; these solutions are then transmitted to slave processes that improve them by performing tabu search and return the best solutions found to the master. Apart from its low communication overhead, this scheme has one main advantage: it induces a multi-thread search strategy in which several distinct search paths are followed in parallel. Computational experiments in other settings (see, for instance, Crainic *et al.*, 1993a, 1995) have shown that a multi-thread search often produces better solutions than single-thread approaches, due to a higher level of search diversification.

The decomposition/reconstruction feature of our VRPTW algorithm allows for another level of parallelization since the subproblems created by each decomposition can be allocated to different processors. Our parallelization approach thus combines the master-slave scheme described above with this decomposition/reconstruction procedure to yield a two-level parallel organization. This allows for a higher degree of parallelization, as well as for a better control of the processor load.

The implementation involves the following processes:

- 1 *Manager* process that manages the adaptive memory and creates starting solutions for the decomposition procedure.
- *S Decomposition* processes, each of which corresponds to a distinct search thread; these processes decompose the problem into *D* subproblems.
- 1 *Dispatcher* process that dispatches the work equally among processors.
- *P Tabu* processes that apply tabu search to subproblems.
- *I Initialization* processes that generate the first initial solutions using the modified Solomon's procedure described in Subsection 2.2.

The *Manager*, *Decomposition* and *Dispatcher* processes require a computing time that is negligible with regard to other processes. They may thus be located on the same processor. In our implementations, a dedicated machine is often used to run these processes.

The most computationally expensive processes are the tabu searches and, in the first phase of the algorithm, the processes that create the initial solutions. It is thus natural to create as many

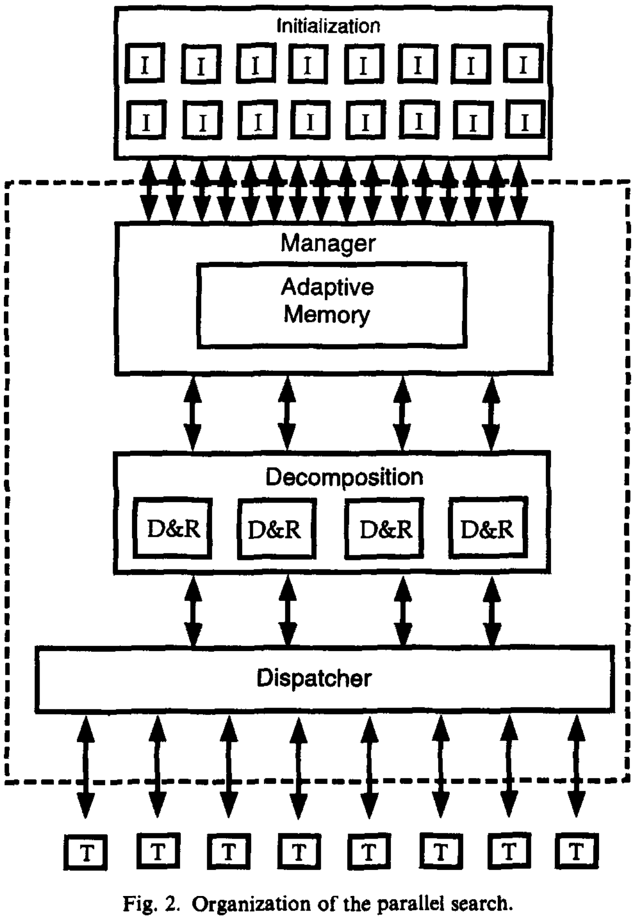

Fig. 2. Organization of the parallel search.

Tabu processes as there are available processors (after allocating the first processor to the other processes). The *Initialization* processes can run on the processors allocated to the tabu search processes. Hence, our implementation uses a total of $P + 1$ processors.

In order to balance the processor workload during the initialization phase of the algorithm, I was set to a multiple of P. Similarly, during the tabu search phase, a full decomposition/reconstruction cycle for all search threads requires solving $S \times D$ subproblems; $S \times D$ is thus also a multiple of P. In our tests, we have used $P = 4, 8, 16$, $I = 16$, and $S \times D = P$.

Figure 2 illustrates our parallelization approach for $I = 16$, $S = 4$, $D = 2$ and $P = 8$; it shows all the processes and their dependencies: in this figure, the links represent the exchange of messages between processes. We now describe more precisely each of the processes.

```
Process Manager

Read problem data (from keyboard, file, etc.).
Send these data to the $I$ Initialization processes.
Repeat $I$ times:
    Wait for an initial solution.
    Store this solution in the adaptive memory.
    Send the initial solution to a Decomposition process.
While a stopping criterion is not met, repeat:
    Wait for a solution from a Decomposition process.
    Insert this solution in the adaptive memory.
    Create a solution from routes found in the adaptive memory.
    Send this starting solution to a Decomposition process (the one that has sent the last solution).
Post-optimize the routes of the best solution found (the routes at the top of the adaptive memory).
Write the post-optimized solution.
```

```
Process Initialization

Receive problem data from the Manager process.
Generate an initial solution using the modified Solomon's procedure (see Section 2.2).
Send this solution to the Manager process.
```

```
Process Decomposition

Wait for a starting solution from the Manager process.
Repeat $C$ times:
    Decompose the current solution into $D$ subproblems.
    Send $D$ resolution requests to the Dispatcher process.
    Wait for the $D$ solutions to the subproblems.
    Construct a complete solution by merging the $D$ solutions of the subproblems.
Send the last complete solution constructed to the Manager process.
```

```
Process Dispatcher

Set to free the label associated with each Tabu process.
Initialize the queue of waiting work to the empty set.

Loop:
    Wait for a message.
    If the message is a resolution request from a Decomposition process, insert the request in the queue.
    If the message is a subproblem solution, forward it to the appropriate Decomposition process and label the Tabu process that has sent the message to free.
    While there is a free *Tabu* process and the queue is not empty, repeat:
        Remove a resolution request from the queue.
        Send this request to a free *Tabu* process.
        Label this process as *working*.

Process *Tabu*

Loop:
    Wait for a resolution request from the *Dispatcher* process.
    Solve this subproblem using tabu search.
    Send the solution to the *Dispatcher* process.
```

To simplify the description of the processes, we have not included the creation, identification, destruction, etc., of processes. Moreover, in our current implementation, all messages go through the *Manager* process which is the only process directly connected to the screen and keyboard. This was done for convenience reasons, in order to allow for an easy monitoring of the algorithm during its development. As the total number of messages exchanged is low, this creates no difficulties, but should the flow of information increase (e.g. when the number of processes is large), it would probably be advisable to explore other communation schemes.

4. COMPUTATIONAL RESULTS

The proposed parallel heuristic was implemented on a network of 17 SUN Sparc 5 workstations. Each process was programmed in C+ + and process communications were handled by the Parallel Virtual Machine (PVM) software.

The heuristic was tested on the well-known VRPTWs problems of Solomon (1987). This benchmark is made up of 56 100-customer Euclidean problems, where customers' locations are distributed within a $[0, 100]^2$ square and travel times between customers are equivalent to the corresponding Euclidean distances. Six problem sets are defined, namely R1, R2, C1, C2, RC1 and RC2. The customers are uniformly distributed in the problems of type R, clustered in groups in those of type C, and mixed in problems of type RC. Furthermore, the time window is narrow at the central depot for problems of type 1 so that many routes are required to service all customers. Conversely, this time window is large for problems of type 2, so that only a few routes are required to service all customers. Finally, a fixed service time, set at 90 time units for problems of type C and 10 time units for problems of types R and RC, is imposed at each customer location. To avoid precision problems during computations, the real Euclidean distances were multiplied by 10^3 and rounded to the nearest integer (the feasibility of solutions produced by the algorithm was later checked using real, double-precision distances).

Because the solutions of type 2 problems are made up of only a few routes, it is not possible to decompose them into subproblems. We thus decided to restrict our tests to the 29 type 1 problems which allow the two-level parallelization scheme described in Section 3. Our tests were aimed at determining the effectiveness of our parallelization approach rather than at obtaining the best possible solutions. For this reason, we kept the parameter settings suggested in Taillard *et al.* (1995) for the sequential tabu search heuristic (see Table 1), while the number of vehicles was fixed to the value reported in their paper for each problem. Using this approach, meaningful comparisons could be performed on the basis of the overall distance travelled. As in Taillard *et al.* (1995), a feasible solution servicing every customer location before its time window's upper bound was obtained on each problem.

In our tests, the first goal was to assess the quality of the solutions produced by the parallel scheme for a fixed amount of computing effort. To do this, we used the number of calls to the adaptive memory as a measure of the computing effort and we solved all problems with 1 (sequential procedure), 2, 4 and 8 search threads. In each case, problems were decomposed into two subproblems and we therefore used 4, 8 and 16 processors to run the *Tabu* processes. The average objective value over the 29 problems is plotted against the number of calls to the adaptive memory in Fig. 3 for all four cases. As can be seen from this plot, the quality of the solutions obtained is, for all practical purposes, insensitive to the number of search threads. Considering the

Table 1. Parameter settings

Objective function	
lateness penalty coefficient: $\alpha = 100$	
Initialization	
number of initial solutions: $I = 20$	
Adaptive memory	
size: $M = 30$ solutions	
Decomposition/reconstruction (D&R)	
number of D&R cycles: $C = 6$	
Tabu search	
number of iterations: 30, 40, 50, 60, 70 and 80 iterations for D&R cycle 1, 2, 3, 4, 5 and 6, respectively, for a total of 330 iterations.	
Neighborhood	
maximum length of route segments: $L = 7$	
length of tabu list: $T = 100{,}000$	
tabu tenure: number of iterations/2.	

fact that Taillard *et al.*'s (1995) sequential procedure is one of the best heuristics currently available for solving the VRPTW, this is an encouraging result. Among other things, it implies that, by resorting to parallelism, one can greatly shorten the computing times required to solve these difficult problems without incurring a degradation in the quality of the solutions produced. An interesting side effect of this insensitivity of the results to the number of search threads is that one can use the number of calls to the adaptive memory as a proxy to solution quality when assessing the efficiency of our parallel implementation. We now turn our attention to this important topic.

Wallclock (or elapsed) times for the four groups of runs were recorded after each call to the adaptative memory. To obtain meaningful times in this multi-user environment, where heavy usage of some workstations by other users could have distorted the results, the network was entirely allocated to our research team during the experiments. The wallclock times (in seconds) are reported in Table 2 for 20, 40 and 80 calls to the memory.

One of the main difficulties in coming up with a correct assessment of the efficiency of our parallel implementation is that the processor on which the *Manager*, the *Dispatcher* and the *Decomposition* processes are running (the 'controller') does not have as much work to do as the others; in fact, it is idle most of the time. Because of this, efficiency measures based on the total

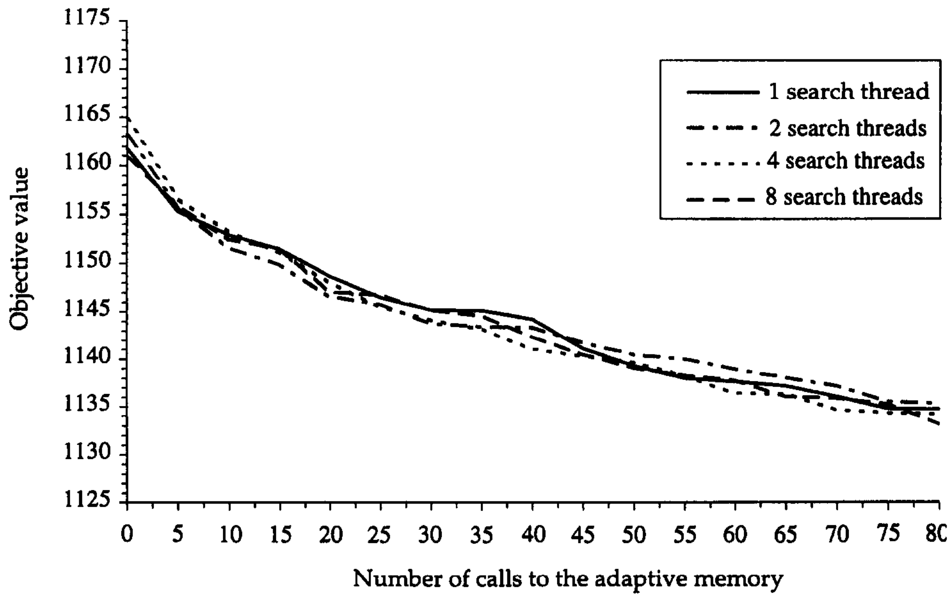

Fig. 3. Objective function value evolution for different numbers of search threads.

Table 2. Wallclock times in seconds for various numbers of processors

Number of processors	Number of search threads	Number of calls to the adaptive memory		
		20	40	80
1	1	2645	4031	6796
5	2	877	1324	2223
9	4	459	691	1146
17	8	239	363	587

number of processors are deceptive and provide very little insight on the performance of the implementation. This led us to use the number of 'slave' processors (the processors running the *Initialization* and *Tabu* processes) as the basis for efficiency measures (i.e. *sequential time/(parallel time×number of processors)*). These are plotted in Fig. 4. As observed in this figure, the efficiency of the implementation degrades slightly with the number of processors. This is due to the fact that parts of the algorithm do not run in parallel and that slave processors waste more time waiting for the controller when the number of slaves increases. An intriguing feature of the efficiency curves is the presence of 'see-saw' patterns displaying peaks located at multiples of the number of search threads. This is easily explained by the fact that the threads are somewhat synchronized: the *S Decomposition* processes reconstruct their solutions and send them to the *Manager* almost simultaneously. As time passes, the threads become less and less synchronized; this reduces bottlenecks at the level of the controller and slave processors waste less time waiting for a new subproblem to solve. This explains why a decrease in the amplitude of the patterns and a slight increase in efficiency with time is observed.

Efficiency curves with respect to the total number of processors (see Fig. 5) show that efficiency is better for a larger number of processors. This is a clear indication of the under-utilization of the controller. This also suggests that one may consider moving the processes resident on the controller to one of the slaves. When this is done, one must be careful to give high priority to the processes that were previously on the controller, for otherwise a significant degradation in performance can take place. In such a situation, the efficiency may even decrease when the processes are moved. This is illustrated in Fig. 6. where the efficiency for various configurations with 16 and 17 processors is plotted. In this figure, the upper bound curve corresponds to the efficiency of the

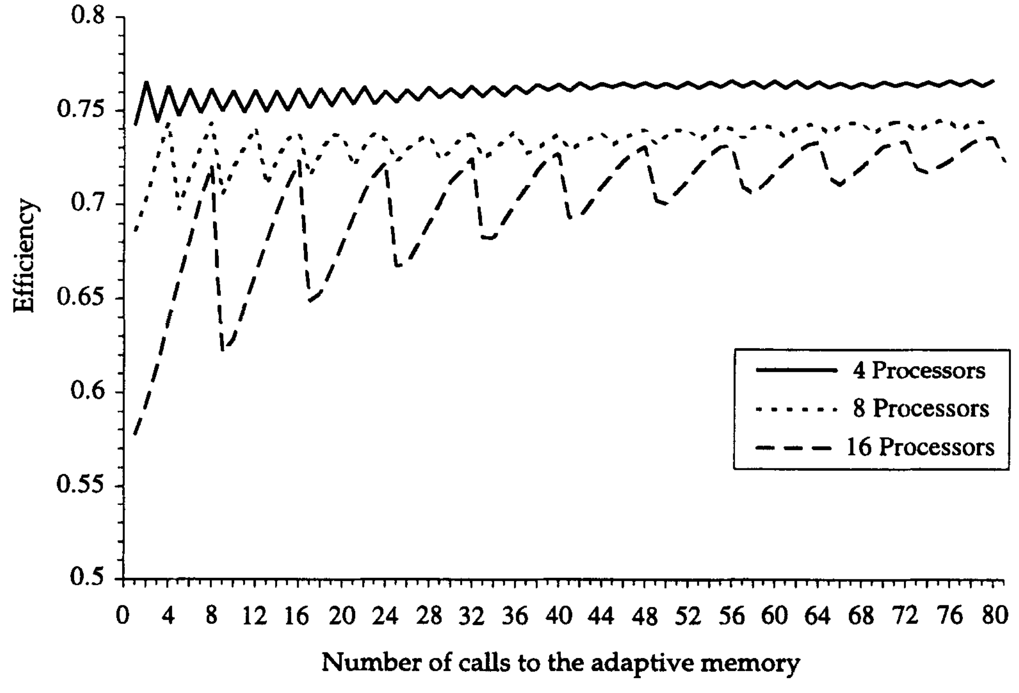

Fig. 4. Efficiency with respect to the number of slave processors.

17-processor implementation measured as if 16 processors were used: this is indeed the best we can possibly do since it corresponds to a situation where the moved processes require no CPU time at all. From this figure, it can be observed that when high priority is given to the controlling processes, it is possible to achieve a quite satisfactory performance. Figure 7 illustrates the effects of doing away with the controller for smaller numbers of slave processors.

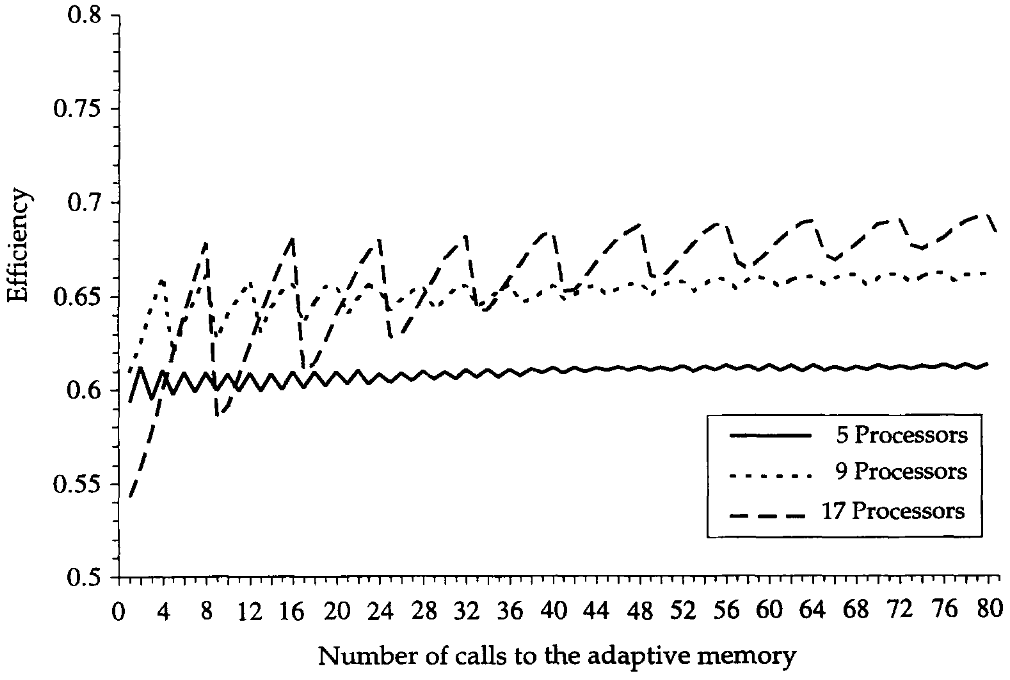

Fig. 5. Efficiency with respect to the total number of processors.

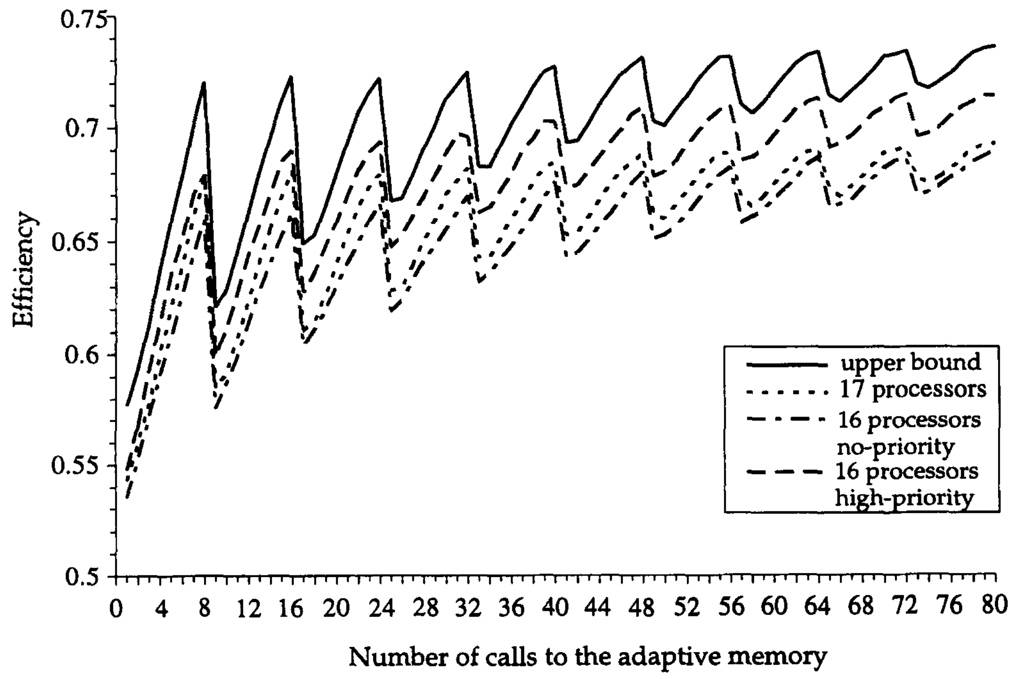

Fig. 6. Efficiency measures for configurations with 16 and 17 processors.

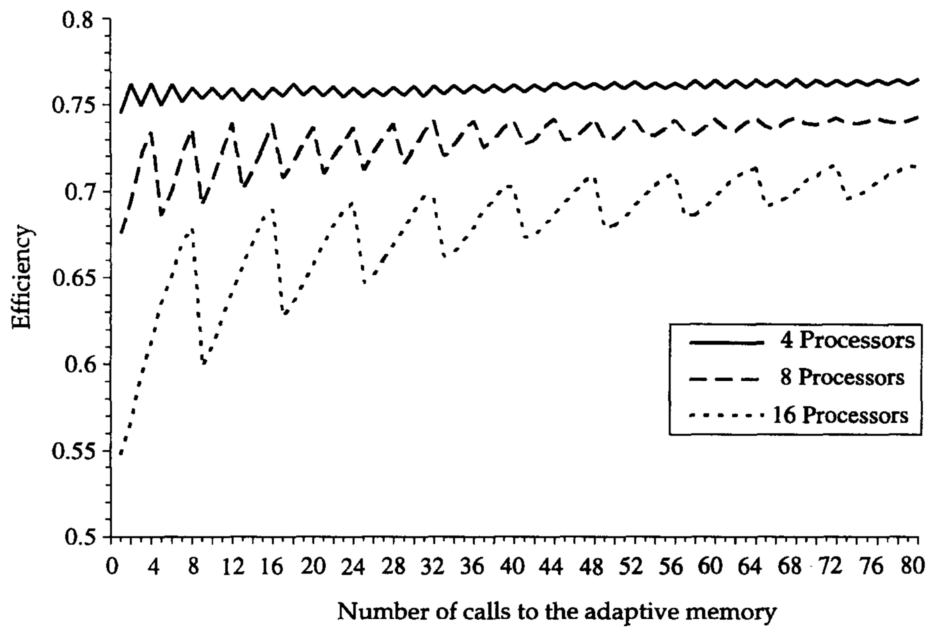

Fig. 7. Efficiency for high priority configurations with no controller.

5. CONCLUSION

A tabu search heuristic for the VRPTW has been developed and implemented on a network of Sparc workstations. Results on benchmark problems show that parallelization of the original sequential algorithm did not reduce solution quality, for the same amount of computations, while providing substantial speed-ups.

Parallel implementations, such as the one described in this paper, could be useful in practice when routes must be produced within a short time interval. Another important class of applications includes dynamic settings, like vehicle dispatching, where new service requests occur in real-time and must be inserted within the planned routes of vehicles in movement. The time available to dispatch a request to a vehicle is typically limited in these applications. Hence, the parallel tabu search could be used to improve the planned routes of the vehicles between meaningful events, like the arrival of a vehicle at a customer location or the occurrence of a new request. In the first case, the best solution in the adaptive memory would indicate the vehicle's next destination. In the second case, the new request could be quickly inserted within each solution found in the adaptive memory.

Clearly, management of the solutions in such a dynamic setting (e.g. discarding customers that have already been serviced, modifying the routes to account for the occurrence of a new request, etc.) asks for a certain number of adjustments to our current problem-solving methodology. This is the line of research that we are currently investigating.

Acknowledgements—This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and by the Fonds pour la Formation de Chercheurs et l'Aide à la Recherche (FCAR).

REFERENCES

Baker, E. K. and Schaffer, J. R. (1988) Solution improvement heuristics for the vehicle routing and scheduling problem with time window constraints. *American Journal of Mathematical and Management Sciences* **6**, 261–300.

Blanton, J. L. and Wainwright, R. L. (1993) Multiple vehicle routing with time and capacity constraints using genetic algorithms. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, ed. S. Forrest, pp. 452–459. Morgan Kaufmann, San Mateo, CA, U.S.A.

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

Chiang, W. C. and Russell, R. A. (1993) Hybrid heuristics for the vehicle routing problem with time windows. Working paper, Department of Quantitative Methods, University of Tulsa, Tulsa, OK, U.S.A. (Forthcoming in *Annals of Operations Research* **63**.)

Crainic, T. G., Toulouse, M. and Gendreau, M. (1993a) An appraisal of asynchronous parallelization approaches for tabu search algorithms. Technical report CRT-935. Centre de recherche sur les transports, Université de Montréal, Montréal, Canada. (Forthcoming in *Annals of Operations Research.*)

Crainic, T. G., Toulouse, M. and Gendreau, M. (1993b) Towards a taxonomy of parallel tabu search algorithms. Technical report CRT-933, Centre de recherche sur les transports, Université de Montreal, Montréal, Canada.

Crainic, T. G., Toulouse, M. and Gendreau, M. (1995) Synchronous tabu search parallelization strategies for multicommodity location-allocation with balancing requirements. *OR Spektrum* **17**, 113–123.

Desrochers, M., Desrosiers, J. and Solomon, M. M. (1992) A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research* **40**, 342–354.

Desrochers, M., Lenstra, J. K., Savelsbergh, M. W. P. and Soumis, F. (1988) Vehicle routing with time windows: optimization and approximation. In *Vehicle Routing: Methods and Studies*, eds B. L. Golden and A. A. Assad, pp. 64–84. North-Holland, Amsterdam.

Desrosiers, J., Dumas, Y., Solomon, M. M. and Soumis, F. (1992) Time constrained routing and scheduling. Technical Report G-92-42. Groupe d'études et de recherche en analyse des décisions, École des Hautes Études Commerciales, Montréal, Canada. (Forthcoming in *Handbooks in Operations Research and Management Science*. North-Holland, Amsterdam.)

Garcia, B. L., Potvin, J. Y. and Rousseau, J. M. (1994) A parallel implementation of the tabu search heuristic for vehicle routing problems with time window constraints. *Computers & Operations Research* **21**, 1025–1033.

Gendreau, M., Hertz, A. and Laporte, G. (1992) New insertion and postoptimization procedures for the traveling salesman problem. *Operations Research* **40**, 1086–1094.

Gendreau, M., Hertz, A., Laporte, G. and Stan, M. (1995) A generalized insertion heuristic for the traveling salesman problem with time windows. Technical report CRT-95-07. Centre de recherche sur les transports, Université de Montréal, Montréal, Canada.

Glover, F. (1986) Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research* **5**, 533–549.

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

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

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

Kolen, A., Rinnooy Kan, A. H. G. and Trienekens, H. (1987) Vehicle routing with time windows. *Operations Research* **35**, 266–273.

Kontoravdis, G. and Bard, J. (1995) A GRASP for the vehicle routing problem with time windows. *ORSA Journal on Computing* **7**, 10–23.

Lin, S. (1965) Computer solution of the traveling salesman problem. *Bell System Technical Journal* **44**, 2245–2269.

Malek, M., Guruswamy, M., Pandya, M. and Owens, H. (1989) Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem. *Annals of Operations Research* **21**, 59–84.

Or, I. (1976) Traveling salesman-type combinatorial problems and their relation to the logistics of blood banking. Ph.D. Dissertation, Department of Industrial Engineering and Management Science, Northwestern University, Evanston, IL.

Potvin, J. Y. and Bengio, S. (1996) The vehicle routing problem with time windows, Part II: Genetic search. *INFORMS Journal on Computing* **8**, 165–172.

Potvin, J. Y. and Rousseau, J. M. (1993) A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research* **66**, 331–340.

Potvin, J. Y. and Rousseau, J. M. (1995) An exchange heuristic for routing problems with time windows. *Journal of the Operational Research Society* **46**, 1433–1446.

Potvin, J. Y., Kervahut, T., Garcia, B. L. and Rousseau, J. M. (1996) The vehicle routing problem with time windows, Part I: Tabu search. *INFORMS Journal on Computing* **8**, 158–164.

Rego, C. and Roucairol, C. (1995) A parallel tabu search algorithm using ejection chains for the VRP. In *Proceedings of the Metaheuristics International Conference, Breckenridge, CO.* July, pp. 253–259.

Rochat, Y. and Taillard, E. (1995) Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics* **1**, 147–167.

Russell, R. A. (1995) Hybrid heuristics for the vehicle routing problem with time windows. *Transportation Science* **29**, 156–166.

Solomon, M. M. (1987) Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research* **35**, 254–265.

Solomon, M. M. and Desrosiers, J. (1988) Time window constrained routing and scheduling problems. *Transportation Science* **22**, 1–13.

Solomon, M. M., Baker, E. K. and Schaffer, J. R. (1988) Vehicle routing and scheduling problems with time window constraints: Efficient implementations of solution improvement procedures. In *Vehicle Routing: Methods and Studies*, eds B. L. Golden and A. A. Assad, pp. 85–105. North-Holland, Amsterdam.

Taillard, E. (1990) Some efficient heuristic methods for the flowshop scheduling problem. *European Journal of Operational Research* **47**, 65–79.

Taillard, E. (1993a) Robust tabu search for the quadratic assignment problem. *Parallel Computing* **17**, 443–455.

Taillard, E. (1993b) Parallel iterative search methods for vehicle routing problems. *Networks* **23**, 661–673.

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

Taillard, E., Badeau, P., Gendreau, M., Guertin, F. and Potvin, J. Y. (1995) A new neighborhood structure for the vehicle routing problem with time windows. Technical report CRT-95-66. Centre de recherche sur les transports, Université de Montréal, Montréal, Canada.

Thangiah, S. R. (1993) Vehicle routing with time windows using genetic algorithms. Technical Report SRU-CpSc-TR-93-23, Computer Science Department, Slippery Rock University, Slippery Rock, PA.

Thangiah, S. R., Osman, I. H. and Sun, T. (1994) Hybrid genetic algorithms, simulated annealing and tabu search methods for vehicle routing problems with time windows. Technical Report UKC/OR94/4. Institute of Mathematics & Statistics, University of Kent, Canterbury, U.K.

Thompson, P. and Psaraftis, H. (1993) Cyclic transfer algorithms for multi-vehicle routing and scheduling problems. *Operations Research* **41**, 935–946.