A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows

Eric Taillard[1], Philippe Badeau[3], Michel Gendreau[1,2], François Guertin[1] and Jean-Yves Potvin[1,2]

[1] Centre de recherche sur les transports, Université de Montréal, C.P. 6128, Succ. Centre-Ville, Montréal, Québec, Canada H3C 3J7.

[2] Département d'informatique et de recherche opérationnelle, Université de Montréal, C.P. 6128, Succ. Centre-Ville, Montréal, Québec, Canada H3C 3J7.

[3] Centre Universitaire des Sciences et Techniques, Université Blaise Pascal, Clermont Ferrand II, Campus Universitaire des Cézeaux, 63174 Aubière Cedex, France.

Abstract. This paper describes a tabu search heuristic for the vehicle routing problem with soft time windows. This problem allows lateness at customer locations although a penalty is then incurred and added to the objective value. By adding large penalty values, the vehicle routing problem with hard time windows can be addressed as well. In the tabu search, a neighborhood of the current solution is created through an exchange procedure that swaps sequences of consecutive customers (or segments) between two routes. The tabu search also exploits an adaptive memory that contains the routes of the best previously visited solutions. New starting points for the tabu search are produced through a combination of routes taken from different solutions found in this memory. Many best know solutions are reported on classical test problems.

Keywords. Vehicle routing, time windows, tabu search, adaptive memory, neighborhood.

1. Introduction

An important component of many distribution systems is the routing of vehicles to service customers. For various applications, like bank deliveries, postal deliveries or school bus routing, a time interval or time window is also associated with each customer to constrain the time of service. In practice, these time windows are often relaxed to allow for early or late arrivals at customer locations. In the literature, such problems are referred to as vehicle routing problems with soft time windows or VRPSTWs.

From a graph theoretical perspective, the VRPSTW can be stated as follows. Let $G=(V, E)$ be a complete undirected graph with vertex set $V=\{v_0, v_1, v_2,..., v_n\}$ and edge set $E=\{(v_i,v_j): v_i, v_j \in V, i<j\}$. Each vertex $v_i \in V$ is associated with:

(i) a fixed quantity q_i of goods to be delivered (with $q_0=0$ at vertex v_0)

(ii) a time window $[e_i, l_i]$, where e_i and l_i are the lower and upper bound of the time window, respectively (with e_0 the earliest start time and l_0 the latest end time of each vehicle route, respectively).

(iii) a service time s_i for unloading the goods (with $s_0=0$ at vertex v_0).

Finally, a symmetric distance matrix $D=(d_{ij})$ that satisfies the triangle inequality is defined on E (with travel times t_{ij} proportional to the distances).

Given a fixed size fleet of m identical vehicles, each with capacity Q, the goal is to find a set of minimum cost vehicle routes, originating from and terminating at the depot, such that:

- each vehicle services one route;
- each vertex v_i, $i=1,\dots,n$ is visited exactly once;
- the quantity of goods to be delivered on a route never exceeds the vehicle capacity Q;
- the start time of each vehicle route is greater than or equal to e_0;
- the end time of each vehicle route is less than or equal to l_0;
- the time of beginning of service b_i at each vertex v_i, $i=1,\dots,n$ is greater than or equal to the time window's lower bound e_i; if the arrival time t_i is less than e_i, a waiting time $w_i = (e_i - t_i)$ is incurred.

The objective function f to be minimized over the set of feasible solutions S is:

$$f(s) = \sum_{k=1}^{m} d_k \; + \; \sum_{i=1}^{n} \alpha_i \times \max\{0, t_i - l_i\}, \; s \in S, \qquad (1.1)$$

where d_k is the total distance traveled on route k, k=1,...,m, and α_i is a lateness penalty coefficient associated with vertex v_i, i=1,...,n.

This definition implies that each route must satisfy the following hard constraints:

(a) the total load on a route cannot exceed the capacity of the vehicle servicing the route.

(b) Each vehicle must start and terminate its route within the time window associated with the depot.

Furthermore, a soft time window constraint is found at each customer location. The time window is "soft" because the vehicle can arrive before the lower bound or after the upper bound. If the vehicle arrives too early, it must wait up to the lower bound to begin its service. If the vehicle is too late, a penalty for lateness is incurred. That is, the upper bound of the time window is relaxed into the objective function in a Lagrangean relaxation fashion. According to (1.1), the penalty coefficients can be adjusted to each customer. For example, high coefficients can be associated with customers with rather strict time requirements and low coefficients with customers with some flexibility.

Another closely related problem, known as the vehicle routing problem with hard time windows or VRPHTW does not allow late services. That is, a vehicle must arrive before the time window's upper bound at each customer location. Furthermore, the fleet size is typically a decision variable. Quite often, a hierarchical objective function is associated with such problems: first, minimize the number of vehicles and, for the same number of vehicles, minimize the total distance traveled. Most algorithms reported in the literature address the VRPHTW (with the exception of [Koskosidis et al. 92]). However, there are many good reasons for solving the VRPSTW:

(a) The VRPSTW model is more general and includes the VRPHTW as well. The latter problem can be solved by appropriately raising the lateness penalty coefficients.

(b) The VRPSTW more closely models situations found in practice and can be used to find a good trade-off between fleet size and service quality to customers.

(c) Since there are fewer hard constraints, feasible solutions are easier to find. In cases where the set of feasible solutions for the "hard" version of the problem is empty (assuming a fixed size fleet), a solution with a few time window violations can still be produced.

However, the generality of the VRPSTW does not come for free. For example, hard time windows can be exploited within the VRPHTW model to quickly filter out infeasible solutions. Different techniques are reported in the literature to check solution feasibility in constant time after the insertion of a new customer or after a local modification to the solution [Savelsbergh 86, Savelsbergh 90, Savelsbergh 92, Duhamel et al. 95]. Furthermore, the total distance of the new solution can also be easily evaluated in constant time. Obviously, the filtering mechanism cannot be applied to soft time windows, while the penalty component in (1.1) does not lend itself to a constant time evaluation. Rather, approximations must be used to achieve this result (see Section 3).

The paper is organized as follows. Section 2 first presents a brief literature review on vehicle routing problems with time windows. Then, Section 3 introduces our neighborhood structure. Section 4 describes the problem-solving approach and provides a detailed explanation of its different components. Finally, Section 5 reports computational results on standard test problems.

2. Literature review

Due to their wide applicability in practical settings, vehicle routing problems with time windows have been intensively studied during the last ten years. These problems being NP-hard, a large spectrum of heuristics are reported in the literature (mostly for the VRPHTW). The interested reader will find excellent surveys in [Desrochers et al. 88, Solomon and Desrosiers 88, Desrosiers et al. 95].

Generally speaking, these problem-solving approaches can be classified as follows:

(a) *exact algorithms based on branch-and-bound techniques* [Kolen et al. 87, Desrochers et al. 92]. Recently, Desrochers et al. found the optimum on a few problems with 100 customers. In this case, a column generation scheme was used to solve the linear programming relaxation of a set partitioning problem. The columns were generated by solving a shortest path problem with time windows.

(b) *route construction heuristics*. Different sequential insertion heuristics are reported in [Solomon 87], while parallel route construction procedures are found in [Potvin and Rousseau 93, Russell 95]. A worst case analysis of some of these route construction heuristics is provided in [Solomon 86].

(c) *route improvement heuristics*. Edge exchange heuristics for problems with time windows may be found in [Or 76, Baker and Schaffer 88, Solomon et al. 88, Thompson and Psaraftis 93, Potvin and Rousseau 95]. Efficient implementations for speeding up the screening of infeasible solutions and the evaluation of the objective function are reported in [Savelsbergh 86, Solomon et al. 88, Savelsbergh 90, Savelsbergh 92, Duhamel et al. 95].

(d) *composite heuristics*. These heuristics mix both route construction and route improvement procedures, see [Derigs and Grabenbauer 93, Kontoravdis and Bard 95, Russell 95].

(e) *optimization-based heuristics*. In [Koskosidis et al. 92], the authors exploit a mixed integer programming model. Through heuristic means, the original problem is decomposed into an assignment/clustering subproblem and a series of routing and scheduling subproblems.

(f) *metaheuristics*. This new generation of heuristics is at the core of many recent developments. Near optimal solutions to the VRPHTW can now be produced with tabu search [Barnes and Carlton 95, Carlton 95, Rochat and Taillard 95, Potvin et al. 96], simulated annealing [Chiang and Russell 93] and genetic algorithms [Thangiah et al. 91, Blanton and Wainwright 93, Thangiah 93, Thangiah et al. 94, Potvin and Bengio 96]. An application of tabu search on a real-world vehicle routing problem with many side constraints, including time windows, may also be found in [Semet and Taillard 93].

Finally, asymptotically optimal heuristics are reported in [Bramel and Simchi-Levi 93, Bramel et al. 93].

Tabu search heuristics are of particular interest to us. [Potvin et al. 96] describes a standard tabu search heuristic based on 2-opt* [Potvin and Rousseau 95] and Or-opt [Or 76] exchanges. The search alternates between the two neighborhoods and includes intensification procedures aimed at eliminating routes with only a few customers. The tabu search in [Rochat and Taillard 95] exploits a neighborhood based on the exchange of customers between routes. An adaptive memory is defined to record the best routes produced during the search. This memory is then used to construct new starting solutions for the tabu search (see Section 3). Finally, [Carlton 95] describes a reactive tabu search that dynamically adjusts its parameter values based on the current search status. This approach is applied to many different types of problems with time windows, including the VRPHTW. Its robustness comes from a simple neighborhood structure that can be easily adapted to different problems. Namely, each customer is removed and reinserted at some other location in the current solution.

The main contribution of our work is the development of a new neighborhood structure, coupled with approximation methods for evaluating each neighboring solution in constant time (in the presence of soft time windows). This is the topic of the next section.

3. The neighborhood structure

In this section, edge exchange heuristics are introduced in general terms. Then, our exchange method is described. Finally, simplifications and approximations aimed at speeding up the examination of the neighborhood are discussed.

3.1 Edge exchange heuristics

Edge exchange heuristics, like 2-opt or 3-opt [Lin 65] are widely used to improve vehicle routing solutions. Typically, these methods are embedded within a local search framework of the following type:

1. Start with an initial solution and define this solution to be the current solution.
2. Generate all solutions in the neighborhood of the current solution by applying all modifications associated with the method under consideration.
3. Select the best solution in this neighborhood and define this solution to be the new current solution.
4. Go back to step 2.

Although the number of iterations needed to reach a local optimum can grow exponentially with the problem size (on carefully designed "pathological" problems), the neighborhood generated in step 2 is typically polynomial. This kind of iterative framework is exploited, in particular, within the tabu search heuristic [Glover 89, Glover 90].

3.2 A new exchange heuristic

In this section, a new exchange heuristic is introduced which generalizes two edge exchange heuristics previously proposed for problems with time windows. This method, called the CROSS exchange, is illustrated in Figure 1. In this figure, the black square stands for the depot and the white circles are customers along the routes (with the depot duplicated at the start and at the end of each 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 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').

Note that the time window constraints define an implicit orientation on each route. In this example, Y_1 (respectively, Y_2) is visited after X_1' (respectively, X_2'). So, the segment X_1'-Y_1 (respectively, X_2'-Y_2) has the same orientation after the move.

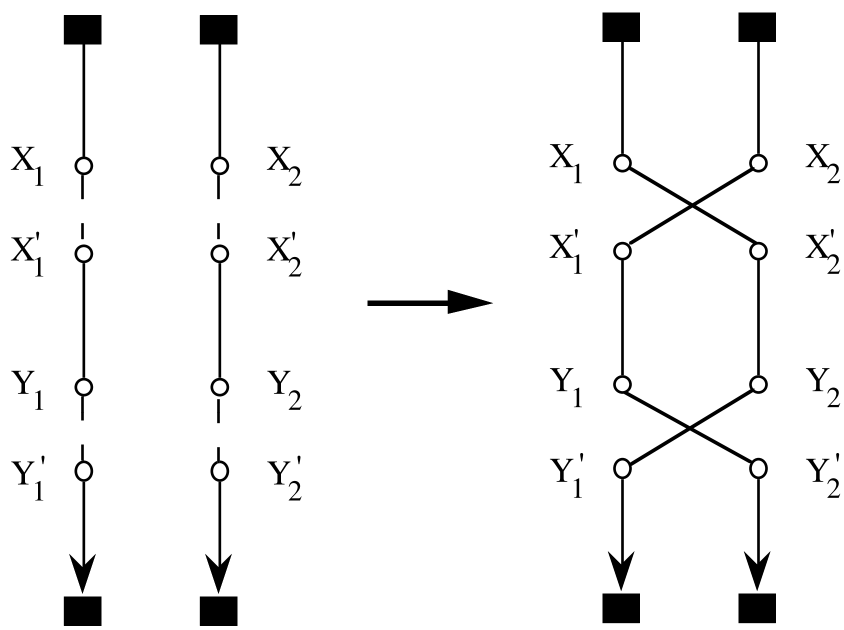

Figure 1. The CROSS exchange

As illustrated in Figures 2(a) and 2(b), the 2-opt* [Potvin and Rousseau 95] and Or-opt [Or 76] 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 is 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 create empty routes. In Figure 2(b), for example, if X_1' is the first customer on the route while Y_1 is the last customer on the same route, the entire route is inserted between X_2 and X_2'.

A CROSS exchange preserves the orientation of the routes, which is a nice feature for problems with time windows. Furthermore, by selecting the exchange that leads to the largest improvement over the current solution, the swapping of segments that are close from a spatial and temporal point of view is favored. One drawback is the complexity of this method. That is, assuming that n customers are evenly distributed among m routes, there are:

(a) $\binom{m}{2}$ ways to select a pair of routes,

(b) $\binom{\frac{n}{m}}{2}$ ways to select two edges to be removed in each route,

where $\binom{j}{i}$ is $\frac{j!}{i!\,(j-i)!}$.

Accordingly, the overall complexity of this neighborhood is:

$$\boldsymbol{O}\left(m^2\right) \times \boldsymbol{O}\left(\frac{n^2}{m^2}\right) \times \boldsymbol{O}\left(\frac{n^2}{m^2}\right) = \boldsymbol{O}\left(\frac{n^4}{m^2}\right).$$

Before closing this section, it is worth noting that CROSS exchanges, while generalizing previous exchange methods, are also a special case of λ-interchanges [Osman 93]. The latter method selects two subsets of customers (whose cardinality is less than or equal to λ) from two different routes and exchange them. However, the size of this neighborhood quickly becomes very large, even for small values of λ. A useful and manageable neighborhood can be obtained by restricting the exchanges to consecutive customers in both routes (thus, leading to CROSS exchanges). Our method is also closely related to the chain-exchange procedure in [Fahrion and Wrede 90]. Here, two chains or segments of different lengths are removed from the current solution. Then, each chain is resinserted at the location that introduces the smallest detour. Looking for the best insertion place of each chain, however, increases the complexity by a factor of $O\left(\frac{n}{m}\right)$.

In the following section, simplification and approximation procedures for reducing the number of CROSS exchanges and speeding up their evaluation will be presented.

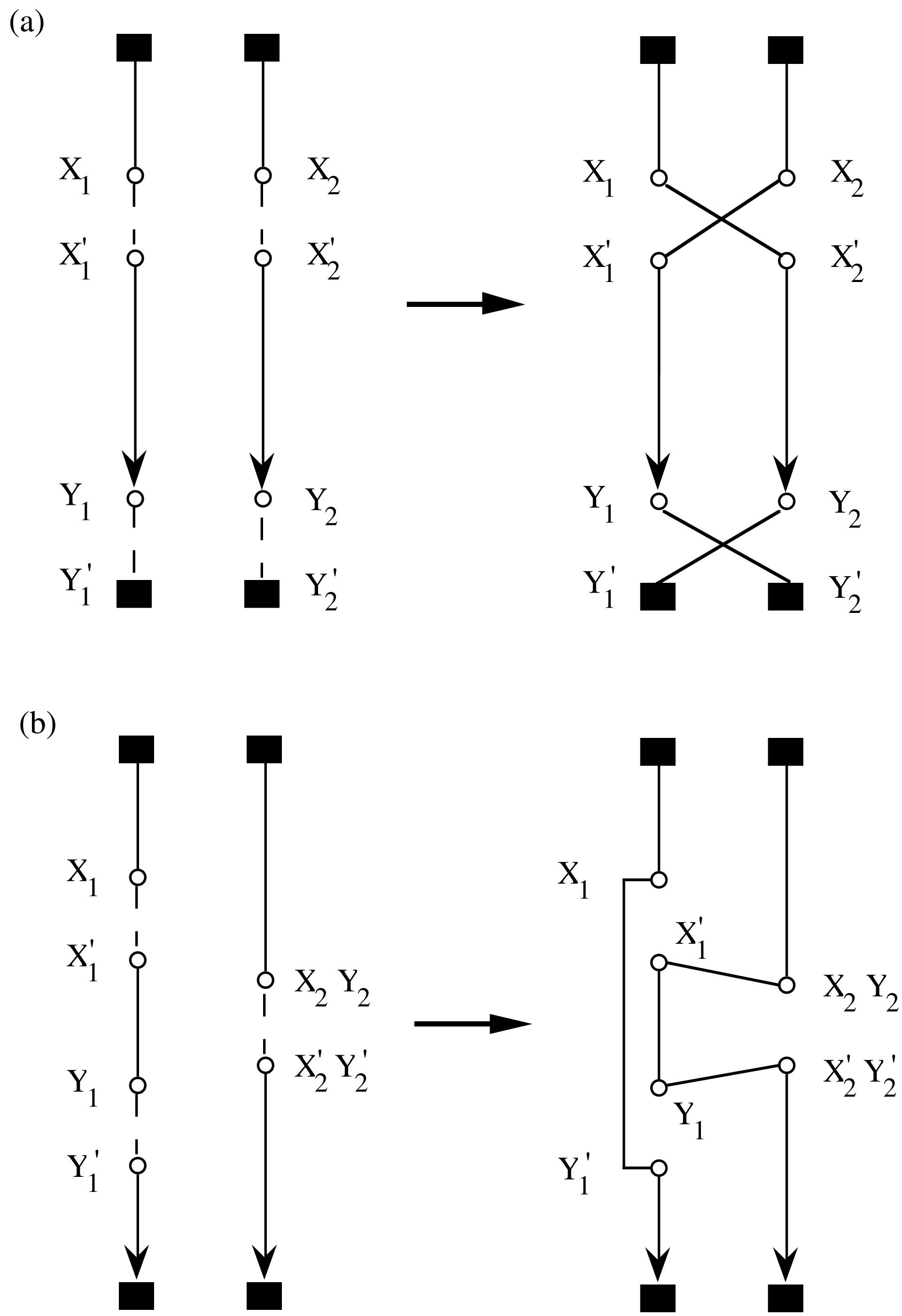

Figure 2. Special cases of the CROSS exchange: (a) 2-opt* (b) Or-opt

3.3 Generating the neighborhood

In this section, we explain how each move can be evaluated in constant time through approximations. Then, we discuss techniques aimed at discarding moves that are unlikely to yield any improvement.

3.3.1 Evaluating a move

When we evaluate a move, we are interested in the difference Δf between the value of the neighboring solution and the value of the current solution. An improvement is found when Δf is negative. It is easy to evaluate the modification Δd to the total distance of the solution after a CROSS exchange. Namely, (1) we sum up the lengths of the edges that have been introduced into the solution, (2) we sum up the lengths of the edges that have been removed from the solution and (3) we subtract (2) from (1). For the lateness penalty component of the objective, however, the impact of the CROSS exchange must be propagated to the end of the route, thus preventing a constant time evaluation. Hence, approximations must be used.

To simplify the following description, we will assume that $\alpha_i = \alpha$, i=1,...,n. In this case, the objective function (1.1) becomes:

$$f(s) = \sum_{k=1}^{m} d_k \quad + \quad \alpha \times \sum_{i=1}^{n} \max\{0, t_i - l_i\}, \, s \in S \; . \qquad (3.1)$$

The second summation in (3.1) is the total lateness of the solution (similarly, the total lateness on a route would be the summation over the subset of customers found on this route). The modification to the total lateness of a solution after a CROSS exchange is obtained by summing up the modifications to the total lateness of both routes involved in the exchange. Here, the approximate evaluation procedure will be illustrated on a single route, namely the new route servicing customers X_1, X_2', Y_2 and Y_1' in Figure 1.

The modification Δl to the total lateness of this route is estimated through to the following formula:

$$\tilde{\Delta} l = \Delta l_{X_2' - Y_2} + \tilde{\Delta} l_{Y_1' - depot} \, , \qquad (3.2)$$

where $$\tilde{\Delta} l_{Y_1' - depot} = g_{Y_1'}\left(\Delta b_{Y_1'}\right).$$

The first component of the summation is the modification to the total lateness of the route segment between X_2' and Y_2. This component can be evaluated exactly in constant time. The second component of the summation approximates the modification to the total lateness on the last route segment between Y_1' and the depot. This component cannot be evaluated exactly in constant time and is thus approximated through function $g_{Y_1'}$. We will now go further into the details of formula (3.2).

Exact evaluation of $\Delta l_{X_2'-Y_2}$

In the following, we explain how to update the time of beginning of service at each customer on route segment X_2'-Y_2 in constant time. Through this result, $\Delta l_{X_2'-Y_2}$ can be easily evaluated in constant time. The new link that now connects X_1 to X_2' (after the CROSS exchange) implies that the modification $\Delta b_{X_2'}$ to the time of beginning of service at customer X_2' is:

$$\begin{aligned} \Delta b_{X_2'} &= b_{X_2'}^{new} - b_{X_2'} \\ b_{X_2'}^{new} &= \max\left\{ e_{X_2'}, b_{X_1} + s_{X_1} + t_{X_1 X_2'} \right\} \end{aligned} \qquad (3.3)$$

Now, the impact on customer Y_2 can be evaluated by propagating $\Delta b_{X_2'}$ along the route segment X_2'-Y_2. This propagation is necessary due to possible waiting times along the segment that can absorb totally or partially $\Delta b_{X_2'}$. Fortunately, the exploration of the neighborhood is done in such a way that this propagation can be avoided. Four nested loops generate the entire neighborhood, namely:

For X_1 from *depot* to *last customer*
  Set X_1' to the immediate successor of X_1;
  ...
  For X_2 from *depot* to *last customer*
    Set X_2' to the immediate successor of X_2;
    ... (3.4)
    For Y_1 from X_1 to *depot*
      Set Y_1' to the immediate successor of Y_1;
      ...
      For Y_2 from X_2 to *depot*
        Set Y_2' to the immediate successor of Y_2;
        ...

For a fixed X_1, X_2 and Y_1, the first segment to be moved from the second route to the first route only contains customer Y_2=X_2', that is, the immediate successor of customer X_2. Then, the segment is progressively extended by setting Y_2 to the second successor of X_2, the third successor of X_2, etc. Hence, Δb_{Y_2} can be easily evaluated in constant time by exploiting the value computed at the previous neighboring solution (given that Y_2 in the previous neighboring solution is the predecessor of Y_2 in the current neighbor).

Approximation of $\Delta l_{Y_1'\text{-}depot}$

With Δb_{Y_2}, $\Delta b_{Y_1'}$ is easily evaluated in constant time via the new link (Y_2, Y_1'). The impact of $\Delta b_{Y_1'}$ on $\Delta l_{Y_1'\text{-}depot}$ is then approximated through function $g_{Y_1'}$ (see equation (3.2)). Such an approximation function is found at each customer location i and is constructed as follows. First, the exact modification Δl to the total lateness of the route is evaluated for a few Δb_i values, noted z_j, j=1,...,Z. Then, a piecewise linear function is produced by interpolation, as illustrated in Figure 3 for Z=6. This function is quite different on the positive and negative sides. Increasing Δb_i on the positive side, thus shifting the time of beginning of service later in time, always increases Δl. On the other hand, decreasing Δb_i on the negative side, thus shifting the time of beginning of service earlier in time, causes no additional effect over a certain threshold (for example, when the lateness on the route is completely eliminated). The dotted curve in Figure 3 illustrates what could be the true function.

An update of these approximation functions is performed only when the best CROSS exchange is applied to the current solution to produce a new solution (i.e., these functions remain the same during the neighborhood evaluation). Furthermore, the update is only performed at customer locations found on the two routes involved in the CROSS exchange.

In regard to the implementation, Z=6 values were used to construct the approximation function g_i at each customer location. A larger number of values would provide a better approximation of the true function, but would also require more computations. Preliminary experiments have shown that Z=6 is a good compromise, with a correlation between the approximation and the true modification varying from 0.5 to 0.8. The value z_1 was set to the maximum positive shift to the time of beginning of service observed at any customer location during the search. The values z_2 and z_3 were obtained by dividing z_1 by 50 and 2500, respectively. In our test problems, z_2 is typically around 25 time units, and z_3 around 0.5 time units. On the negative side, z_6 was set to the maximum lateness found at customer i and all its successors on the route. Note that Δl cannot be reduced by more than z_6 time units through a shift (earlier in time) at customer i. The values z_5 and z_4 were set by dividing z_6 by 50 and 2500, respectively.

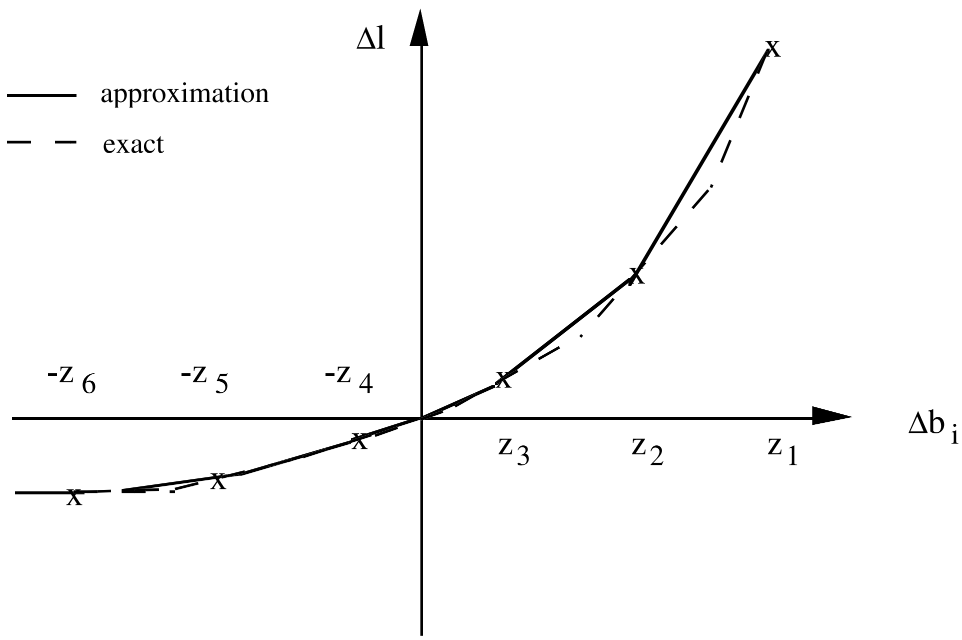

Figure 3. Approximation function g_i at customer location i

Although a good exchange according to the approximation is likely to be good according to the true objective, the E best solutions in the neighborhood (according to the approximation) are kept for further consideration. These solutions are then evaluated *exactly* and the best one (according to the exact evaluation) becomes the new current solution for the next iteration.

3.3.2 Discarding moves

The size of the neighborhood can be reduced by discarding moves that are unlikely to yield any improvement. In (3.4), four nested loops generate the entire neighborhood. At the level of Y_2, the loop is interrupted when the current segment leads to a violation of the capacity constraint when it is moved to the first route. Obviously, the situation can only be worse if the length of this segment increases. This loop is also interrupted when a monotonous degradation of the (approximate) objective value is observed over three consecutive iterations. The same approach is applied at the level of X_2 when a monotonous degradation of the objective value is observed over three consecutive iterations. In this case, the objective value associated with X_2 is the best solution found after iterating over Y_1 and Y_2. Since the X_2 loop is at the second level within the nested structure, its interruption is much more beneficial in terms of computation time than the interruption at the level of Y_2. The interruption at X_2 mostly occurs when the segments to be swapped cover different time periods. In this case, huge lateness is typically created on one route after the exchange. Hence, it makes sense to discard such "incompatible" moves.

3.3.3 Approximation matrix

The overall computation time can be reduced by exploiting previous calculations stored in a data structure known as the approximation matrix. Each entry (i,j), $i,j = 1,\ldots,m$, $i<j$, in this upper triangular matrix is associated with a pair of routes i and j. It contains information about the best CROSS exchange (according to the approximation) for this pair of routes, namely, the value of the new solution as well as the edges introduced or removed from both routes to produce this solution. When a particular exchange involving routes i' and j' is accepted, only the information stored in row i' and column j' of the approximation matrix is updated (since only these two routes are modified by the exchange). That is, the information associated with any pair or routes that does not include neither route i' nor route j' remains the same and is not recalculated.

3.4 Intra-route exchanges

The CROSS exchanges of Section 3.2 swap customers between two routes. However, intra-route optimization is equally important to find good solutions. In order to optimize individual routes, the CROSS neighborhood is enlarged by including exchanges that apply to a single route. These exchanges are similar to those 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 exchanges [Or 76], by allowing the relocation of segments of any arbitrary length.

4. The problem-solving methodology

The main ideas in this section, like the adaptive memory or the decomposition/reconstruction procedure have already been proposed in [Taillard 93, Rochat and Taillard 95]. The following presentation will thus omit many details which can be found in the above papers (but without sacrificing completeness).

The tabu search heuristic presented in this section follows the guidelines provided in [Glover 89, Glover 90]. It is embedded within the following problem-solving methodology:

Construct p different solutions using a stochastic insertion heuristic. Then, 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 define this solution to be the current solution.*
>
> *For I iterations do:*
>
> > *Decompose the current solution into C disjoint subsets of routes.*
> >
> > *Apply a tabu search on each subset of routes.*
> >
> > *Reconstruct a complete solution by merging the new routes found by the tabu search, and define this solution to be the new current solution.*
>
> *Store the routes of the current solution in the adaptive memory.*

Apply a postoptimization procedure to each individual route of the best solution.

This methodology includes many different components like an initialization procedure, a decomposition/reconstruction cycle (which will be referred to as D&R in the following), an adaptive memory, and the tabu search as such. These components will now be described.

4.1 Initialization

In order to fill the adaptive memory with different types of routes, a randomized insertion heuristic is used to construct *P* different initial solutions. First, the routes are initialized by randomly selecting *m* seed customers. Hence, each initial route only services a single customer. The remaining unrouted customers are then inserted one by one (in a random order) at the location that minimizes Solomon's insertion cost over the current set of routes (see Section 4.4 and the description of heuristic I1 in [Solomon 87]). This simple construction heuristic does not produce high quality solutions. Accordingly, the tabu search heuristic presented in Section 4.4 is applied to each solution before the resulting routes are stored in the adaptive memory. This phase introduces a high level of diversity in the adaptive memory by storing routes extracted from different types of solutions. This diversity is then exploited by the tabu search.

4.2 Decomposition/reconstruction

In order to reduce the computation time and intensify the search in specific regions of the search space, each initial solution is partitioned into *C* disjoint subsets of routes, each subset of routes or subproblem being processed by a different tabu search [Taillard 93]. The best routes found for every subproblem are simply merged together to form the new solution for the next D&R. After *I* D&Rs, the final routes are stored in the adaptive memory if it is not filled yet (see Section 4.3). Otherwise, the routes of the worst solution found in the adaptive memory are discarded and replaced by the new routes (if the new solution is better than the worst solution in memory).

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 (see Figure 4). Note that the decomposition changes from one D&R to the next by choosing a different starting angle for creating sectors, thus allowing the CROSS exchange heuristic to exploit new pairs of routes.

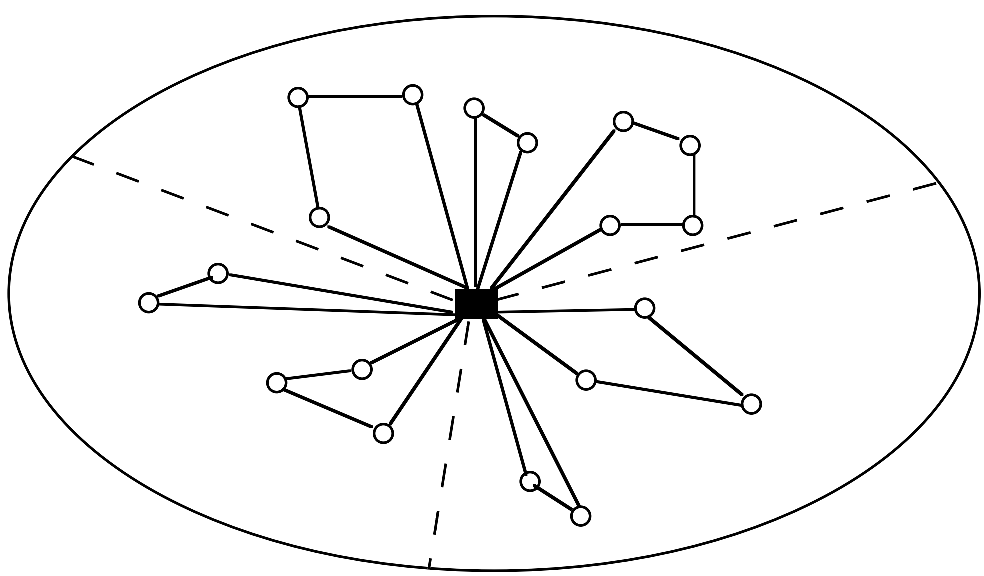

Figure 4. Decomposition of a solution into subsets of routes

4.3 The adaptive memory [Rochat and Taillard 95]

The adaptive memory is a pool of routes taken from the best solutions visited during the search. Its purpose is to provide new starting solutions for the tabu search through selection and combination of routes extracted from this memory (in a manner reminiscent of crossover operators, which are found in genetic algorithms [Holland 75]; however, the number of parent solutions in our case is generally greater than two).

First, the memory is partially filled with routes produced during the initialization procedure (see Section 4.1). All routes of a given solution are contiguously stored in memory and the solutions are sorted according to their objective value. Hence, the routes associated with the best solutions are found in the first positions of the memory. This pool is then used to construct initial solutions for the tabu search. The selection process for creating a new solution is stochastic and biased in favor of the best routes. Namely, a selection probability is associated with each route, and this probability is higher for routes associated with better solutions [Rochat and Taillard 95].

Once the first route is selected, the routes in memory with one or more customers in common with the selected route are discarded from the selection procedure. Then, a second route is selected among the remaining routes. This procedure is repeated until the set of selected routes covers all customers or until there is no admissible route in memory. In the latter case, Solomon's I1 insertion heuristic is invoked to insert the remaining customers (see [Solomon 87] and Section 4.4). If this insertion procedure cannot accommodate all customers, due to the capacity constraints or the time window at the depot, the unserviced customers are left aside. However, the insertion procedure is reinvoked at the end of each D&R to try to insert these customers into the current solution.

4.4 The tabu search

The tabu search presented in this section exploits the neighborhood structure of Section 3. It is applied to a subset of routes or subproblem through the decomposition/reconstruction procedure of Section 4.2. The tabu search is quite standard, and can be summarized as follows:

Set the current solution to some initial subset of routes.

While the stopping criterion is not met do:

> *Generate the neighborhood of the current solution by applying CROSS exchanges.*
>
> *Select the best (non tabu) solution in this neighborhood and define this solution to be the new current solution.*
>
> *If the current solution is better than the best overall solution then*
>
> > *reorder the customers within each route using Solomon's I1 insertion heuristic and define this new solution to be the new current solution and the best overall solution.*
>
> *Update the tabu list.*

Return the best overall solution

The main components of this algorithm are now briefly introduced.

(a) *Initialization*. As mentioned previously, the initial solution is produced by combining routes contained in the adaptive memory (see Section 4.3). Then, this set of routes is partitioned into subsets of routes, each subset being provided as input to a tabu search heuristic.

(b) *Stopping criterion*. The tabu search stops after a certain number of iterations. This number is calculated with the following formula:

$$A \times \left(1 + \frac{DR - 1}{B}\right)$$

where A and B are parameters and DR is the current D&R, DR=1,..., I. Hence, the number of iterations increases with the number of D&Rs. Given that the solution improves from one D&R to the next, more iterations are required to significantly improve the solutions provided to the last D&Rs.

(c) *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 loose 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 greater 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.

It is worth noting that the objective value is an integer because real distances between customers are rounded up or down at the third decimal place, and then transformed into integers by multiplying them by 10^3.

(d) *Diversification*. Dynamic diversification is incorporated into the tabu search by penalizing CROSS exchanges that are frequently performed during the search. Let fr_e be such a frequency for a given exchange e, let fr_{max} be the maximal frequency observed and let $iter$, n and m be the current iteration number, the number of customers and the number of routes, respectively. If x is a random value uniformly chosen in the interval]0.0, 0.5] and $\Delta_{iter}^{\max}$ is the maximal absolute difference observed between the objective values of two consecutive solutions up to iteration $iter$, then the exchange e is penalized by:

$$p_e = x \times \Delta_{iter}^{\max} \times \frac{fr_e}{fr_{max}}.$$

The fr_{max} value normalizes the frequencies since they tend to be lower (higher) for larger (smaller) neighborhoods. The $\Delta_{iter}^{\max}$ component adjusts the penalty to the magnitude of the moves.

(e) *Reordering of each route*. The customers within each individual route are reordered when an overall best solution is found. This reordering is based on Solomon's I1 insertion heuristic [Solomon 87]. It works as follows for a given route. First, all customers on this route are unrouted. Then, the farthest unrouted customer from the depot is selected as a seed customer (i.e., the vehicle starts from the depot, services the seed customer and comes back to the depot). The remaining unrouted customers are then inserted one by one into the new route. The next customer to be inserted is the one that maximizes a generalized savings measure. The classical savings [Clarke and Wright 64] can be obtained by setting the parameters of Solomon's heuristic to appropriate values. The selected customer is then inserted at the location that minimizes a weighted sum of detour in space and time. Solomon's heuristic is applied *R* times with different parameter settings, and the best route is selected 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 service all customers (due to the time window at the depot) or if the new route is worse than the original one.

4.5 Postprocessing of each individual route

We applied a specialized heuristic developed for the TSP with time windows [Gendreau et al. 95] to each individual route in the final solution produced by our algorithm. This method is an adaptation of the GENIUS heuristic, originally devised for the TSP [Gendreau et al. 92]. This heuristic only slightly improved the total distance of 10 solutions in Solomon's test set (the improvement was less than 1% in all cases, but one), but it runs for only a few seconds and three additional best known solutions were obtained (see Section 5.2).

5. Computational results

Although our method was designed for the VRPSTW, it has been tested on Solomon's set of VRPHTWs [Solomon 87]. This choice comes from a need to compare our method with competing approaches on standardized test problems. In the following, these test problems are first introduced. Then, the best solutions produced by our algorithm are reported. Some figures about the improvement to solution quality with increasing computation times are also provided. Finally, the impact of the approximation techniques of Section 3 on solution quality and computation time is examined.

5.1 Test problems

Our algorithm was tested on standard problems found in [Solomon 87]. In these 100-customer Euclidean problems, the travel times are equivalent to the corresponding Euclidean distances. The customer locations are distributed within a $[0,100]^2$ square. Six different sets of problems are defined, namely C1, C2, R1, R2, RC1 and RC2. The customers are uniformly distributed in the problems of type R and clustered in groups in the problems of type C. These characteristics are mixed in the problems of type RC. Furthermore, the time window is narrow at the central depot for the problems of type 1, so that only a few customers can be serviced on each route. Conversely, this time window is large for the problems of type 2, so that many customers can be serviced on the same route. Finally, a fixed service time is found at each customer location (c.f., time for unloading the goods). This service time is set at 10 time units per customer for the problems of type R and RC, and 90 time units per customer for the problems of type C.

5.2 Numerical results on Solomon's test set

The experiments reported in this section were performed on a Sun Sparc 10 workstation (50 Mhz). The real Euclidean distances between customers were rounded up or down at the third decimal place, and then transformed into integers by multiplying them by 10^3. In this way, precision problems were avoided during the calculations. However, the feasibility of the best solutions produced by our algorithm was later checked using real, double-precision distances. Our results were obtained with the parameter settings presented in Table 1.

To reduce computation time, in particular on the problems of type 2, the length of the route segments to be swapped could not exceed a threshold value L. Different values were tried during our experiments. The best solutions reported in this section were obtained with L=5 or L=7. Note also that no D&R took place on the problems of type 2, because only a few routes are needed to service all customers. In this case, each tabu search was applied to the entire set of routes.

The best solutions produced by our algorithm are reported in Tables 2 and 3 for all problems in Solomon's test set (based on real, double-precision distances), using the format: number of routes/total distance. The fleet size m was set to the number of routes of the best solution reported in the literature for each problem. A feasible solution to the VRPHTW was produced in each case, with the exception of problem R101 (where one additional route is needed). On problems RC102 and RC106, the algorithm was able to save an additional route. In the tables, our best solutions are compared with the best solutions previously reported in the literature. A single asterisk * means a tie with the best published solution and a double asterisk ** means that the best published solution has been improved. Overall, our algorithm has improved 17 and tied 20 best known solutions on the 56 test problems.

objective function

lateness penalty coefficient: α= 100.

stopping criterion

3 identical best know solutions visited during the search or 100 calls to the adaptive memory.

initialization

number of initial solutions: P=20

decomposition/reconstruction (D&R)

number of D&Rs: I=6.

cardinality of the decomposition: C=2 subsets of routes

adaptive memory

size: M=30 solutions.

tabu search

number of iterations: A=30, B=3; thus 30, 40, 50, 60, 70 and 80 iterations for DR=1, 2, 3, 4, 5 and 6, respectively, for a total of 330 iterations.

length of tabu list: T=100,000.

tabu tenure: *number of iterations/2*.

number of best approximate neighboring solutions: E=15

number of reorderings of each route: R=20

Table 1. Parameter settings for the tabu search heuristic

Problem	Best published solution	Reference	Our best solution	
R101	18/1607.7	Desrochers et al. 92	19/1650.79	
R102	17/1434.0	Desrochers et al. 92	17/1487.60	
R103	13/1207	Thangiah et al. 94	13/1294.24	
R104	10/982.01	Rochat and Taillard 95	10/982.72	
R105	14/1377.11	Rochat and Taillard 95	14/1377.11	*
R106	12/1252.03	Rochat and Taillard 95	12/1259.71	
R107	10/1159.85	Rochat and Taillard 95	10/1126.69	**
R108	09/980.95	Rochat and Taillard 95	09/968.59	**
R109	11/1235.68	Rochat and Taillard 95	11/1214.54	**
R110	11/1080.36	Rochat and Taillard 95	11/1080.36	*
R111	10/1129.88	Rochat and Taillard 95	10/1104.83	**
R112	10/953.63	Rochat and Taillard 95	10/964.01	
Problem	Best published solution	Reference	Our best solution	
C101	10/827.3	Desrochers et al. 92	10/828.94	*
C102	10/827.3	Desrochers et al. 92	10/828.94	*
C103	10/828.06	Rochat and Taillard 95	10/828.06	*
C104	10/824.78	Rochat and Taillard 95	10/824.78	*
C105	10/828.94	Potvin and Bengio 93	10/828.94	*
C106	10/827.3	Desrochers et al. 92	10/828.94	*
C107	10/827.3	Desrochers et al. 92	10/828.94	*
C108	10/827.3	Desrochers et al. 92	10/828.94	*
C109	10/828.94	Potvin and Bengio 93	10/828.94	*
Problem	Best published solution	Reference	Our best solution	
RC101	14/1669	Thangiah et al. 94	14/1696.94	
RC102	13/1477.54	Rochat and Taillard 95	12/1554.75	**
RC103	11/1110	Thangiah et al. 94	11/1264.27	
RC104	10/1135.83	Rochat and Taillard 95	10/1135.83	*
RC105	13/1733.56	Rochat and Taillard 95	13/1643.38	**
RC106	12/1384.92	Rochat and Taillard 95	11/1448.26	**
RC107	11/1230.95	Rochat and Taillard 95	11/1230.54	**
RC108	10/1170.70	Rochat and Taillard 95	10/1139.82	**

Table 2. Best results on problems of type 1

Problem	Best published solution	Reference	Our best solution	
R201	4/1281.58	Rochat and Taillard 95	4/1254.80	**
R202	3/1530.49	Potvin and Bengio 93	3/1214.28	**
R203	3/948.74	Rochat and Taillard 95	3/951.59	
R204	2/869.29	Rochat and Taillard 95	2/941.76	
R205	3/1063.24	Rochat and Taillard 95	3/1038.72	**
R206	3/833	Thangiah et al. 94	3/932.47	
R207	3/814.78	Rochat and Taillard 95	3/837.20	
R208	2/738.60	Rochat and Taillard 95	2/748.01	
R209	3/855	Thangiah et al. 94	3/959.47	
R210	3/967.50	Rochat and Taillard 95	3/980.90	
R211	2/949.50	Rochat and Taillard 95	2/923.80	**

Problem	Best published solution	Reference	Our best solution	
C201	3/591.56	Potvin and Bengio 93	3/591.56	*
C202	3/591.56	Potvin and Bengio 93	3/591.56	*
C203	3/591.17	Rochat and Taillard 95	3/591.17	*
C204	3/590.60	Potvin and Bengio 93	3/590.60	*
C205	3/588.88	Potvin and Bengio 93	3/588.88	*
C206	3/588.49	Potvin and Bengio 93	3/588.49	*
C207	3/588.29	Rochat and Taillard 95	3/588.29	*
C208	3/588.32	Rochat and Taillard 95	3/588.32	*

Problem	Best published solution	Reference	Our best solution	
RC201	4/1249	Thangiah et al. 94	4/1413.79	
RC202	4/1165.57	Rochat and Taillard 95	4/1164.25	**
RC203	3/1079.57	Rochat and Taillard 95	3/1112.55	
RC204	3/806.75	Rochat and Taillard 95	3/831.69	
RC205	4/1333.71	Rochat and Taillard 95	4/1328.21	**
RC206	3/1212.64	Rochat and Taillard 95	3/1158.81	**
RC207	3/1085.61	Rochat and Taillard 95	3/1082.32	**
RC208	3/833.97	Rochat and Taillard 95	3/847.90	

Table 3. Best results on problems of type 2

Different transformations of the distance matrix and/or final solutions were applied by the authors referred to in Tables 2 and 3. In [Desrochers et al. 92], the distances were truncated at the first decimal place; in [Thangiah et al. 94], the final solution value was rounded up or down to the nearest integer; in [Rochat and Taillard 95, Potvin and Bengio 96], real double-precision distances were used. One consequence is that slightly different objective values can be associated with the same solution (in particular, for the problems in class C1). Another consequence is that solutions found on truncated or rounded distances are not necessarily feasible when real distances are used (unless the final solutions are checked for feasibility with real, double-precision distances, as we did).

Table 4 compares the averages of the best solutions, as produced by our method and by other methods reported in the literature, for each set of problems. The headings in the Table should be interpreted as follows: CR [Chiang and Russell 93], PB [Potvin and Bengio 96], RT [Rochat and Taillard 95] and TH [Thangiah et al. 94]. The two numbers in each entry are the average number of routes and average distance, respectively. Note that the previous best averages were all produced by RT and that five additional routes were saved by our method over RT on the 56 test problems.

	CR	PB	TH	RT	Our method
R1	12.42 1289.95	12.58 1296.80	12.33 1238	12.25 1208.50	12.17 1209.35
C1	10.00 885.86	10.00 838.01	10.00 832	10.00 828.38	10.00 828.38
RC1	12.38 1455.82	12.13 1446.20	12.00 1284	11.88 1377.39	11.50 1389.22
R2	2.91 1135.14	3.00 1117.70	3.00 1005	2.91 961.72	2.82 980.27
C2	3.00 658.88	3.00 589.93	3.00 650	3.00 589.86	3.00 589.86
RC2	3.38 1361.14	3.38 1360.57	3.38 1229	3.38 1119.59	3.38 1117.44

Table 4. Best averages for each problem type

A hierarchical objective function is typically associated with the VRPHTW. That is, the number of routes is first minimized and, for the same number of routes, the total distance is minimized. An exception is found in [Barnes and Carlton 95, Carlton 95] where the objective is simply to minimize the total distance. Accordingly, the distances of the best solutions reported in the work of Barnes and Carlton are typically lower than the numbers presented in Table 4. On the other hand, the number of routes is higher.

In order to compare our method with the reactive tabu search of Barnes and Carlton, the fleet size *m* was set to the numbers reported in [Carlton 95, Appendix B] for each problem. The results are summarized in Table 5. The first column is the average number of routes. The second column *Reactive Tabu (best)* contains two numbers. The first one is the average of the best distances reported in [Carlton 95, Appendix B] when the parameter settings of the reactive tabu search are adjusted to each problem. The second number is the average computation time of the best runs on a IBM 6000 RISC workstation (in seconds). The third column *Adaptive Tabu (avg.)* is the average distance produced by our algorithm over five different runs, when each run is given the same amount of computation time as the best run of the reactive tabu search. Here, the processing power of their machine was assumed to be similar to ours, which is true only for lower end IBM RISC 6000 workstations. Finally, the last column *Adaptive Tabu (best)* is the average of the best solutions produced by our algorithm in the course of multiple experiments.

First, it should be pointed out that the comparison between *Adaptive Tabu (avg.)* and *Reactive Tabu (best)* is quite unfair. Namely, the computation time associated with the reactive tabu search should be the sum of all runs leading to the best solution, not only the computation time of the best run. In spite of this enormous handicap, the *average* solutions produced by our algorithm are competitive with the *best* solutions produced by the reactive tabu search, for the "same" amount of computation time. Furthermore, our best solutions are significantly better than those reported in [Carlton 95, Appendix B]. The reactive tabu search of Barnes and Carlton is thus able to produce good solutions in relatively low computation times. However, due to its simple neighborhood structure, it reaches its "peak" much earlier than our method.

Finally, our tabu search heuristic was applied on two larger problem instances with 417 customers. These problems are known as D417 and E417 and are reported in [Russell 95]. A feasible solution with

55 vehicles was found in both cases, with a total distance of 3439.8 for D417 and 3707.1 for E417. Note that [Rochat and Taillard 95] report total distances of 3467.8 and 3693.2, respectively, for the same number of vehicles. Hence, a better solution was found on problem D417. It is worth noting that [Rochat and Taillard 95] also report solutions with 54 vehicles, but the total distance then increases to 6264.8 for D417 and to 7211.8 for E417. Our algorithm did not find feasible solutions with m=54.

	Avg. Number of routes	Reactive Tabu (best)	Adaptive Tabu (avg.)	Adaptive Tabu (best)
R1	13.91	1221.2 568.0	1228.3	1198.8
C1	10.00	831.9 273.5	850.8	828.4
RC1	13.25	1408.3 564.8	1401.6	1363.7
R2	3.09	1009.1 988.9	1040.7	976.3
C2	3.25	616.5 408.9	605.7	592.1
RC2	3.38	1303.3 750.5	1268.9	1187.2

Table 5. Comparison with the reactive tabu search in [Carlton 95]

5.3 Solution quality versus computation time

The results reported in Tables 2 and 3 are the best solutions produced by our algorithm during the course of multiple experiments. Accordingly, no meaningful computation times can be provided. However, Table 6 illustrates the improvement to solution quality with increasing computation times, using L= 7. These numbers were obtained over five independent runs. For each problem set, we show the average computation time in seconds after 20, 50 and 100 calls to the adaptive memory, including all preprocessing that takes place before the search actually starts, the average solution value of the best run (*Minimum*), the average solution value of the worst run (*Maximum*) and the overall average (*Average*). The first number is the average number of routes and the second number is the average distance.

In these experiments, the number of routes was set to $m+1$, where m is the minimum value reported in the literature for each problem. By introducing an additional route, feasible solutions to the VRPHTW were quickly produced by our algorithm, so that all numbers in Table 6 refer to solutions that satisfy the time window constraint at each customer location. Note that the number of routes can decrease during the search if an empty route is created by a CROSS exchange. In the case of C1 and C2, the growth of the route segments was not stopped when a monotonous degradation of the objective value was observed over three consecutive iterations (c.f., Section 3.3.2). On these well-structured problems, the method quickly converges to good local optima and comes back to these same optima in the remaining of the run, thus causing an early stop (c.f., when the same best solution is found three times). By allowing a more thorough exploration of the neighborhood of the current solution, better solutions are found before the stopping criterion is met.

Problem type	CPU time	Minimum		Maximum		Average	
R1	2296	12.58	1222.24	12.67	1248.28	12.64	1233.88
	6887	12.25	1231.89	12.58	1227.50	12.39	1230.48
	13774	12.25	1216.70	12.50	1219.71	12.33	1220.35
C1	2926	10.00	829.22	10.00	834.10	10.00	830.41
	7315	10.00	828.50	10.00	828.77	10.00	828.59
	14630	10.00	828.45	10.00	828.45	10.00	828.45
RC1	1877	11.88	1385.69	12.25	1424.28	12.08	1404.59
	5632	11.88	1373.11	12.13	1400.84	12.00	1387.01
	11264	11.88	1367.51	12.00	1402.12	11.90	1381.31
R2	3372	3.00	1025.24	3.00	1067.87	3.00	1046.56
	10116	3.00	1011.04	3.00	1048.25	3.00	1029.65
	20232	3.00	995.38	3.00	1031.33	3.00	1013.35
C2	3275	3.00	590.37	3.00	596.85	3.00	592.75
	8187	3.00	590.37	3.00	593.58	3.00	591.14
	16375	3.00	590.30	3.00	592.57	3.00	590.91
RC2	1933	3.38	1204.17	3.38	1289.81	3.38	1248.34
	5798	3.38	1186.57	3.38	1263.12	3.38	1220.28
	11596	3.38	1165.62	3.38	1239.14	3.38	1198.63

Table 6. Solution quality versus CPU time for our algorithm

In Table 7, results found in [Rochat and Taillard 95] for a single run of their method on a Silicon Graphics Indigo (100 Mhz) are shown. By comparing the solution values reported in Table 5 under "Average" with those found in Table 6 for similar computation times, we observe that our method performs better on problem sets RC1, R2, and RC2. Namely, an average of 12.08 routes is found on RC1 after 1877 seconds, as compared to 12.38 routes after 2600 seconds with RT. Furthermore, RT did not find the averages of 3.00 and 3.38 routes on problem sets R2 and RC2, after 9800 and 7800 seconds, respectively. On the other hand, RT found 12.58 routes on set R1 after 2700 seconds, as compared to an average of 12.64 routes after 2296 seconds with our method. A slight improvement in regard to the distance traveled is also observed in favor of RT on problem set C1 for similar computation times.

Problem type	CPU time	RT	
R1	450	12.83	1208.43
	1300	12.58	1202.31
	2700	12.58	1197.42
C1	540	10.00	832.59
	1600	10.00	829.01
	3200	10.00	828.45
RC1	430	12.75	1381.33
	1300	12.50	1368.03
	2600	12.38	1369.48
R2	1600	3.18	999.63
	4900	3.09	969.29
	9800	3.09	954.36
C2	1200	3.00	595.38
	3600	3.00	590.32
	7200	3.00	590.32
RC2	1300	3.62	1207.37
	3900	3.62	1155.47
	7800	3.62	1139.79

Table 7. Solution quality versus CPU time for RT [Rochat and Taillard 95]

5.4 CROSS exchanges versus 2-opt* and Or-opt

In order to evaluate the benefits associated with the CROSS neighborhood, Table 8 shows results obtained on problem set RC1 when this neighborhood is replaced by the 2-opt* or the Or-opt neighborhood within the tabu search heuristic. The format of this table is the same as Table 6. It shows the average computation time in seconds after 20, 50 and 100 calls to the adaptive memory, the average solution value of the best run (over five different runs), the average solution value of the worst run and the overall average. In the last three columns, the first number is the average number of routes and the second number is the average distance.

As we can see, it is more computationally expensive to generate the CROSS neighborhood than the 2-opt* and Or-opt neighborhoods. However, by comparing results for similar computation times (e.g., the third line of 2-opt* and Or-opt with the second line of CROSS), the CROSS neighborhood clearly leads to better solutions. Note that the same trends are observed on the other problem sets.

	CPU time	Minimum		Maximum		Average	
CROSS	2296	12.58	1222.24	12.67	1248.28	12.64	1233.88
	6887	12.25	1231.89	12.58	1227.50	12.39	1230.48
	13774	12.25	1216.70	12.50	1219.71	12.33	1220.35
2-opt*	1259	12.67	1272.81	13.17	1340.13	12.88	1304.03
	3147	12.58	1268.75	13.00	1320.97	12.73	1293.97
	6294	12.58	1268.75	12.75	1309.82	12.58	1288.92
Or-opt	1857	12.67	1274.63	13.08	1326.80	12.85	1301.14
	4643	12.58	1261.30	12.83	1315.61	12.72	1288.93
	9286	12.58	1243.60	12.67	1303.40	12.62	1275.36

Table 8. Comparison between CROSS, 2-opt* and Or-opt neighborhoods on problem set RC1

5.5 Benefits of approximations

This section evaluates the usefulness of the techniques proposed in Section 3 for approximating the objective value. Table 9 shows the average solution value (number or routes, distance, computation time in seconds, in this order) at the first local minimum for a typical run on each problem type, using either an exact or approximate evaluation. Different values of parameter L were also tested. That is, the neighborhood of the current solution was progressively reduced by decreasing L from ∞ to 5. Note that when $L=\infty$, the restriction on the monotonous degradation of the objective value over three consecutive iterations was removed. Hence, the entire neighborhood of the current solution was examined. *Approx* uses the approximation to evaluate each new solution, while *Exact* evaluates each new solution exactly. The row "Initial" in Table 9 refers to the value of the starting solution.

	R1	C1	RC1	R2	C2	RC2
Initial	13.67 1523.51	10.00 1609.56	16.50 1903.89	4.27 1347.01	3.00 813.09	3.38 1648.19
Approx L=5	13.08 1343.15 7.8	10.00 1106.43 10.89	13.75 1530.31 8.25	3.36 1169.84 50.8	3.00 669.47 9.38	3.38 1513.69 34.00
Approx L=7	13.00 1317.41 12.4	10.00 1080.29 19.2	13.63 1486.89 16.1	3.36 1175.22 61.36	3.00 669.21 11.25	3.38 1490.28 41.63
Approx L=10	13.00 1295.02 18.42	10.00 971.77 32.56	13.63 1494.32 20.38	3.36 1170.76 75.90	3.00 686.82 15.63	3.38 1422.11 77.38
Approx L=20	13.00 1290.9 20.0	10.00 961.30 36.67	13.38 1488.58 20.50	3.36 1157.52 185.73	3.00 694.30 37.63	3.38 1395.66 169.25
Approx L=∞	13.00 1285.13 34.4	10.00 924.55 55.44	13.38 1444.16 36.13	3.36 1046.79 1201.64	3.00 620.33 283.63	3.38 1360.77 758.00
Exact L=7	13.08 1294.30 99.83	10.00 1019.10 102.67	13.50 1468.67 90.25	3.36 1106.50 424.909	3.00 632.72 74.63	3.38 1333.20 224.88
Exact L=∞	13.00 1280.18 223.25	10.00 959.11 248.89	13.13 1436.74 200.88	3.36 1024.79 4118.55	3.00 600.31 1002.00	3.38 1275.73 3306.62

Table 9. Approximate versus exact solution at the first local minimum

A comparison between *Approx*, $L=\infty$ and *Exact*, $L=\infty$ shows that the approximation generates solutions that are competitive with those obtained through the exact evaluation in a fraction of the computation time. The performance of the approximation degrades a little bit on the large routes found in the problems of type 2 (in particular RC2). By comparing *Exact* and *Approx* with $L=\infty$ and $L=7$, we also note that solution quality does not degrade as much when the exact evaluation is used. However, *Exact* with $L=7$ is still much more computationally expensive than *Approx* with $L=7$.

6. Conclusion

This paper has introduced a new neighborhood structure, based on CROSS exchanges, which is exploited within a tabu search heuristic. As the search progresses, good routes are stored in an adaptive memory and new starting solutions are produced by selecting and combining routes found in this memory. That is, new solutions are created from routes associated with different solutions visited during the search (in a manner reminiscent of the recombination or crossover operator of genetic algorithms). This problem-solving methodology has produced many best known solutions on Solomon's test set [Solomon 87] and seems to be fairly robust along the different classes of problems found in this test set.

The CROSS exchanges are a useful generalization of the 2-opt* [Potvin and Rousseau 1995] and Or-opt [Or 76] exchanges and may be applied to other types of vehicle routing problems as well. Note also that our problem-solving methodology is readily amenable to parallelization. For example, many different starting solutions can be constructed from the adaptive memory and assigned to different tabu search processes that run in parallel. Furthermore, each tabu search process can be parallelized through the decomposition/reconstruction procedure that partitions a problem into smaller subproblems. With this parallelization, it is possible to envision the application of our methodology in a dynamic setting where each service request is dispatched soon after it is received.

Acknowledgments. Financial support for this research was provided by the Natural Sciences and Engineering Research Council of Canada (NSERC) and by the Quebec Fonds pour la Formation de Chercheurs et l'Aide à la Recherche (FCAR).

References

1. E.K. Baker and J.R. Schaffer, 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.

2. J.W. Barnes and W.B. Carlton, 1995. Solving the Vehicle Routing Problem with Time Windows using Reactive Tabu Search, presented at the Fall 1995 INFORMS Conference, New Orleans, LA.
3. J.L. Blanton and R.L. Wainwright, 1993. Multiple Vehicle Routing with Time and Capacity Constraints using Genetic Algorithms, in S. Forrest Eds, Proceedings of the Fifth International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, pp. 452-459.
4. J. Bramel and D. Simchi-Levi, 1993. Probabilistic Analyses and Practical Algorithms for the Vehicle Routing Problem with Time Windows, Working paper, Columbia University.
5. J. Bramel, C.L. Li and D. Simchi-Levi, 1993. Probabilistic Analysis of a Vehicle Routing Problem with Time Windows, *American Journal of Mathematical and Management Sciences 13*, 267-322.
6. W.B. Carlton, 1995. A Tabu Search Approach to the General Vehicle Routing Problem. Ph.D. Dissertation, The University of Texas at Austin, Texas.
7. W.C. Chiang and R.A. Russell, 1993. Hybrid Heuristics for the Vehicle Routing Problem with Time Windows, Working paper, Department of Quantitative Methods, University of Tulsa, Tulsa, OK, Forthcoming in *Annals of Operations Research*.
8. G. Clarke and J. Wright, 1964. Scheduling of Vehicles from a Central Depot to a Number of Delivery Points, *Operations Research 12*, 568-581.
9. U. Derigs and G. Grabenbauer, 1993. INTIME - A New Heuristic Approach to the Vehicle Routing Problem with Time Windows with a Bakery Fleet Case, *American Journal of Mathematical and Management Sciences 13*, 249-266.
10. M. Desrochers, J. Desrosiers and M.M. Solomon, 1992. A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows, *Operations Research 40*, 342-354.
11. M. Desrochers, J.K. Lenstra, M.W.P. Savelsbergh and F. Soumis, 1988. Vehicle Routing with Time Windows: Optimization and Approximation, in B.L. Golden and A.A. Assad Eds, Vehicle Routing: Methods and Studies, North-Holland, Amsterdam, pp. 65-84.
12. J. Desrosiers, Y. Dumas, M.M. Solomon and F. Soumis, 1995. Time Constrained Routing and Scheduling, in Handbooks in Operations Research and Management Science, Volume 8, Network Routing, M.O. Ball, T.L. Magnanti, C.L. Monma, G.L. Nemhauser Eds, North-Holland, Amsterdam, pp. 35-139.

13. C. Duhamel, B.L. Garcia and J.Y. Potvin, 1995. Accélération du Calcul des Fonctions Objectif pour les Problèmes de Tournées de Véhicules avec Fenêtres de Temps, Technical report CRT-95-04, Centre de recherche sur les transports, Université de Montréal, Montréal, Canada.

14. R. Fahrion and M. Wrede, 1990. On a Principle of Chain-exchange for Vehicle Routing Problems (1-VRP), *Journal of the Operational Research Society 41*, 821-827.

15. M. Gendreau, A. Hertz, G. Laporte, M. Stan, 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.

16. M. Gendreau, A. Hertz, G. Laporte, 1992. New Insertion and Postoptimization Procedures for the Traveling Salesman Problem, *Operations Research 40*, 1086-1094.

17. F. Glover, 1989. Tabu Search - Part I, *ORSA Journal on Computing 1*, 190-206.

18. F. Glover, 1990. Tabu Search - Part II, *ORSA Journal on Computing 2*, 4-32.

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

20. A. Kolen, A.H.G. Rinnooy Kan and H. Trienekesn, 1987. Vehicle Routing with Time Windows, *Operations Research 35*, 266-273.

21. G. Kontoravdis and J. Bard, 1995. A GRASP for the Vehicle Routing Problem with Time Windows, *ORSA Journal on Computing 7*, 10-23.

22. Y.A. Koskosidis, W.B. Powell and M.M. Solomon, 1992. An Optimization-Based Heuristic for Vehicle Routing and Scheduling with Soft Time Window Constraints, *Transportation Science 26*, 69-85.

23. S. Lin, 1965. Computer Solutions of the Traveling Salesman Problem, *Bell System Technical Journal 44*, 2245-2269.

24. I. Or, 1976. Traveling Salesman-type Combinatorial Problems and their relation to the Logistics of Blood Banking, Ph.D. thesis, Department of Industrial Engineering and Management Science, Northwestern University, Evanston, IL.

25. I.H. Osman, 1993. Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem, *Annals of Operations Research 41*, 421-451.

26. J.Y. Potvin, T. Kervahut, B.L. Garcia and J.M. Rousseau, 1996. The Vehicle Routing Problem with Time Windows - Part I: Tabu Search, *INFORMS Journal on Computing 8(2)*.

27. J.Y. Potvin and S. Bengio, 1996. The Vehicle Routing Problem with Time Windows - Part II: Genetic Search, *INFORMS Journal on Computing 8(2)*.

28. J.Y. Potvin and J.M. Rousseau, 1995. An Exchange Heuristic for Routing Problems with Time Windows, *Journal of the Operational Research Society 46*, 1433-1446.

29. J.Y. Potvin and J.M. Rousseau, 1993. A Parallel Route Building Algorithm for the Vehicle Routing and Scheduling Problem with Time Windows, *European Journal of Operational Research 66*, 331-340.

30. Y. Rochat and E. Taillard, 1995. Probabilistic Diversification and Intensification in Local Search for Vehicle Routing, *Journal of Heuristics 1*, 147-167.

31. R.A. Russell, 1995. Hybrid Heuristics for the Vehicle Routing Problem with Time Windows, *Transportation Science 29*, 156-166.

32. M.W.P. Savelsbergh, 1992. The Vehicle Routing Problem with Time Windows: Minimizing Route Duration, *ORSA Journal on Computing 4*, 146-154.

33. M.W.P. Savelsbergh, 1990. An Efficient Implementation of Local Search Algorithms for Constrained Routing Problems, *European Journal of Operational Research 47*, 75-85.

34. M.W.P. Savelsbergh, 1986. Local Search in Routing Problems with Time Windows, *Annals of Operations Research 4*, 285-305.

35. F. Semet and E. Taillard, 1993. Solving Real-Life Vehicle Routing Problems Efficiently using Tabu Search, *Annals of Operations Research 41*, 469-488.

36. M.M. Solomon, 1986. On the Worst-Case Performance of Some Heuristics for the Vehicle Routing and Scheduling Problem with Time Window Constraints, *Networks 16*, 161-174.

37. M.M. Solomon, 1987. Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints, *Operations Research 35*, 254-265.

38. M.M. Solomon and J. Desrosiers, 1988. Time Window Constrained Routing and Scheduling Problems, *Transportation Science 22*, 1-13.

39. M.M. Solomon, E.K. Baker and J.R. Schaffer, 1988. Vehicle Routing and Scheduling Problems with Time Window Constraints: Efficient Implementations of Solution Improvement Procedures, in B.L. Golden and A.A. Assad Eds, Vehicle Routing: Methods and Studies, North-Holland, Amsterdam, pp. 85-105.

40. E. Taillard, 1993. Parallel Iterative Search Methods for Vehicle Routing Problems, *Networks 23*, 661-673.

41. S.R. Thangiah, 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.

42. S.R. Thangiah, K. Nygard and P. Juell, 1991. GIDEON: A Genetic Algorithm System for Vehicle Routing with Time Windows, in Proceedings of the 7th IEEE Conference on Artificial Intelligence Applications, IEEE Press, Miami, FL, pp. 422-425.

43. S.R. Thangiah, I.H. Osman and T. Sun, 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, UK.

44. P. Thompson and H. Psaraftis, 1993. Cyclic Transfer Algorithms for Multi-Vehicle Routing and Scheduling Problems, *Operations Research 41*, 935-946.