PERGAMON

Transportation Research Part B 37 (2003) 579–594

TRANSPORTATION RESEARCH PART B

www.elsevier.com/locate/trb

A tabu search heuristic for the static multi-vehicle dial-a-ride problem

Jean-François Cordeau, Gilbert Laporte *

Canada Research Chair in Distribution Management and GERAD, HEC Montréal, 3000 chemin de la Côte-Sainte-Catherine, Montreal, Canada H3T 2A7

Received 30 May 2002; received in revised form 29 August 2002; accepted 16 September 2002

Abstract

This article describes a tabu search heuristic for the *dial-a-ride problem* with the following characteristics. Users specify transportation requests between origins and destinations. They may provide a time window on their desired departure or arrival time. Transportation is supplied by a fleet of vehicles based at a common depot. The aim is to design a set of least cost vehicle routes capable of accommodating all requests. Side constraints relate to vehicle capacity, route duration and the maximum ride time of any user. Extensive computational results are reported on randomly generated and real-life data sets.

Keywords: Dial-a-ride problem; Door-to-door transportation; Tabu search heuristic

1. Introduction

In the *dial-a-ride problem* (DARP), n users specify transportation requests between given origins and destinations. Users may provide a time window on their desired departure or arrival time, or on both. Transportation is supplied by a fleet of m vehicles based at a common depot. The aim is to design a set of least cost vehicle routes capable of accommodating all requests, under a set of constraints. The most common constraints relate to vehicle capacity, route duration and maximum ride time, i.e., the time spent by a user in the vehicle. In several applications, users specify two requests per day: an *outbound* request from home to a destination, and an *inbound*

* Corresponding author. Address: Universite de Montreal, Centre de recherche transports, CP 6128, Montreal, Quebec H3C 3J7, Canada. Tel.: +1-514-343-6143; fax: +1-514-343-7121.
E-mail address: gilbert@crt.umontreal.ca (G. Laporte).

0191-2615/03/$ - see front matter
doi:10.1016/S0191-2615(02)00045-0

request for the return trip. A common DARP application arises in door-to-door transportation services for the elderly and the disabled.

Dial-a-ride services may operate according to one of two modes. In the *static mode*, all transportation requests are known in advance, which makes it possible to plan all vehicle routes ahead of time. In the *dynamic* mode, transportation requests are gradually revealed throughout the day, as users call, so that vehicle routes are constructed in real-time. Even when the dynamic mode is used, a static problem is usually solved on a set of initial requests to obtain a starting solution that will later be modified as new requests are received. This study is concerned with the static DARP.

The DARP belongs to the generic class of vehicle routing and scheduling problems which have been extensively studied over the past 40 years (see, e.g., Toth and Vigo, 2002). What makes the DARP different and somewhat more difficult than most other routing problems is that transportation cost and user inconvenience must be weighed against each other when designing a solution. At one extreme, designing vehicle routes without sufficient consideration for users may cause some people to spend undue amounts of time traveling. At the other extreme, letting users impose overly tight constraints on their pick-up and drop-off times can increase transportation costs drastically. Most of the scientific literature on the DARP assumes that users first specify desired pick-up and drop-off times and the transporter then designs routes that minimize deviations from these desired times or ensure that deviations fall within a pre-specified tolerance. This is not, in our view, the best way to model the problem, at least from the transporter's point of view. As in the work of Jaw et al. (1986), we follow a different approach in line with the current practice of several transporters.

A more realistic way to model the DARP is to let users impose a time window of a pre-specified width on the arrival time of their outbound trip and, similarly, a window on the departure time of their inbound trip. An upper bound L is imposed on the ride time of any user. Under these constraints, the transporter determines the most suitable pick-up and drop-off times for the outbound and inbound trip, respectively. Our purpose is to present a tabu search heuristic for this version of the DARP.

The remainder of this article is organized as follows. In the next section, we briefly review the operational research literature for the DARP. This is followed by a description of our algorithm, by computational results, and by the conclusion.

2. Literature review

Several versions of the DARP have been studied over the past 30 years. While none is identical to ours, it helps reviewing them to put our contribution into perspective. Since the definition of the DARP varies from one author to the next, we only consider cases where time window constraints are imposed. In the absence of time windows, the problem only contains precedence relationships on pick-up and drop-off, which do not fully capture the true nature of the DARP.

While some early consultancy studies (Wilson et al., 1971; Wilson and Weissberg, 1976; Wilson and Colvin, 1976) sought real-time solutions to the dynamic DARP, it seems that thereafter most work has concentrated on the static version. The single-vehicle problem has been widely studied. Psaraftis (1980, 1983) developed an exact dynamic programming algorithm for the case where

time windows are imposed on each pick-up and drop-off. User inconvenience is controlled through a "maximum position shift" constraint limiting the difference between the position of a user in the list of requests and its position in the vehicle route. Only very small instances ($n \leqslant 10$) can be handled through this algorithm. Sexton and Bodin (1985a,b) have later proposed a heuristic for a similar version of the problem where user inconvenience is measured this time as a weighted sum of two terms. The first measures the excess ride time, i.e., the actual travel time minus the direct travel time. The second computes, for each customer, the difference between his desired drop-off time and his actual drop-off time, under the assumption that late drop-offs never occur. The heuristic applies Benders decomposition to a mathematical formulation of the problem and attempts to improve an initial solution by adjusting the vehicle route through user reinsertions. Results are reported on several data sets from Baltimore and Gaithersburgh involving between 7 and 20 users. Finally, an exact column generation based algorithm was proposed by Desrosiers et al. (1986) for the same problem. It was applied to the solution of instances containing up to 40 users.

Fewer studies have addressed the multi-vehicle version of the DARP. Jaw et al. (1986) have analyzed a version of the problem where windows are imposed on the pick-up time of inbound requests and on the drop-off time of outbound requests. A maximum ride time expressed as a linear function of the minimum ride time is also specified. The quality of a solution is measured through a non-linear objective incorporating several types of disutility. The problem is solved by sequentially inserting users into vehicle routes so as to yield the least possible increase in the objective function value. Computational results are provided on artificial instances involving 250 users and on a real-life dataset with 2617 users and 28 vehicles. The heuristic developed by Madsen et al. (1995) applies to the dynamic DARP with multiple vehicle capacities and multiple objectives. It is based on the insertion heuristic proposed by Jaw et al. (1986) and was applied to a real-life instance containing 24 vehicles and 300 users. Bodin and Sexton (1986) have developed a cluster first, route second heuristic for the problem, employing a sequential insertion heuristic to form the clusters of users. An improvement to this two-phase approach is presented by Dumas et al. (1989) who incorporate part of the clustering phase into the routing phase. Desrosiers et al. (1991) further improve upon this methodology by performing the insertions in parallel, while Ioachim et al. (1995) use a mathematical programming technique to form the clusters. Tests were carried out on instances involving almost 3000 users. Dumas et al. (1991) have extended their single-vehicle exact algorithm to the multiple-vehicle case and applied it to instances with $n \leqslant 55$.

A real-life problem arising in Bologna was tackled by Toth and Vigo (1996). Users specify requests with a time window on their origin and destination. A limit proportional to direct distance is imposed on the time spent by a user in the vehicle. Transportation is supplied by a fleet of capacitated minibuses and by the occasional use of taxis. The objective is to minimize the total cost of service. The problem is solved by a heuristic consisting of first assigning requests to routes by means of a parallel insertion procedure, and then performing intra-route and inter-route exchanges. Results were reported on instances involving between 276 and 312 requests. Toth and Vigo (1997) have also proposed a tabu thresholding procedure to improve the initial solution obtained by the insertion algorithm. In addition, Borndörfer et al. (1997) report the results of a study related to the transportation of handicapped in Berlin. The method applies branch-and-cut to solve two set partitioning formulations of the problem: one for creating clusters and another

one for chaining these clusters into vehicle routes. Results are reported for real-life instances involving between 859 and 1765 transportation requests per day.

Finally, we mention a study by Fu (2002) relevant in the context of the DARP. Its emphasis is on determining a solution in a context where travel times throughout the day are time-dependent and stochastic. For further details on DARP algorithms and applications, the reader is referred to Cordeau and Laporte (2002) and Desaulniers et al. (2002).

3. Formulation

The DARP is defined on a complete graph $G = (V, A)$, where $V = \{v_0, v_1, \ldots, v_{2n}\}$ is the vertex set and $A = \{(v_i, v_j) : v_i, v_j \in V, i \neq j\}$ is the arc set. Vertex v_0 represents a depot at which is based a fleet of m vehicles, and the remaining $2n$ vertices represent origins and destinations for the transportation requests. Each vertex pair (v_i, v_{i+n}) represents a request for transportation from origin v_i to destination v_{i+n}. With each vertex $v_i \in V$ are associated a load q_i (with $q_0 = 0$), a non-negative service duration d_i (with $d_0 = 0$) and a time window $[e_i, l_i]$, where e_i and l_i are non-negative. The load is equal to 1 for vertices $v_1, \ldots, v_n$ and to -1 for vertices $v_{n+1}, \ldots, v_{2n}$. Service duration corresponds to the time needed to let the user get on or off the vehicle. Let T denote the end of the planning horizon. In the case of an outbound request, it is assumed that $e_i = 0$ and $l_i = T$. Similarly, $e_{i+n} = 0$ and $l_{i+n} = T$ for an inbound request. Each arc (v_i, v_j) has an associated non-negative routing cost c_{ij} and a non-negative travel time t_{ij}. Finally, let L denote the maximum ride time of a user. The DARP consists of designing m vehicle routes on G such that

(i) every route starts and ends at the depot;
(ii) for every request i, vertices v_i and v_{i+n} belong to the same route and vertex v_{i+n} is visited later than vertex v_i;
(iii) the load of vehicle k does not exceed at any time a preset bound Q_k;
(iv) the total duration of route k does not exceed a preset bound T_k;
(v) the service at vertex v_i begins in the interval $[e_i, l_i]$, and every vehicle leaves the depot and returns to the depot in the interval $[e_0, l_0]$;
(vi) the ride time of any user does not exceed L;
(vii) the total routing cost of all vehicles is minimized.

We denote by A_i the arrival time of a vehicle at vertex v_i, by $B_i \geqslant \max\{e_i, A_i\}$ the beginning of service at vertex v_i, and by $D_i = B_i + d_i$ the departure time from vertex v_i. We assume here that waiting at any vertex v_i is allowed before service starts but is not allowed after service has finished. The time window constraint at vertex v_i is violated if $B_i > l_i$. Arrival before e_i is, however, allowed and the vehicle then incurs a waiting time $W_i = B_i - A_i$. The ride time associated with request i is computed as $L_i = B_{i+n} - D_i$. Ride time thus corresponds to the elapsed time between the end of service at vertex v_i and the beginning of service at vertex v_{i+n}. If there were no ride time constraints, it would always be optimal to set $B_i = \max\{e_i, A_i\}$. However, it may sometimes be profitable to delay the beginning of service at vertex v_i so as to reduce the unnecessary waiting time at vertex v_{i+n} (or at any other vertex visited between v_i and v_{i+n}) and thus, the ride time associated with request i.

4. Solution methodology

We propose a tabu search algorithm for the DARP. Starting from an initial solution s_0, the algorithm moves at iteration t from s_t to the best solution in a neighbourhood $N(s_t)$ of s_t. To avoid cycling, solutions possessing some attributes of recently visited solutions are declared forbidden, or *tabu*, for a number of iterations, unless they constitute a new incumbent. As is common in such algorithms, a continuous diversification mechanism is put in place in order to reduce the likelihood of being trapped in a local optimum. Tabu search was proposed by Glover (1986) and has quickly become one of the most widespread heuristic methods for combinatorial optimization (see, e.g., Glover and Laguna, 1997).

Tabu search has been applied to the closely related pick-up and delivery problem with time windows by Nanry and Barnes (2000). These authors have considered three types of move. The first removes a vertex pair (v_i, v_{i+n}) from its current route and reinserts it in a different route. The second swaps two pairs of vertices between two distinct routes while the last consists of moving a vertex within its current route. As in our implementation, solutions that violate time window and vehicle capacity constraints are allowed during the search. However, the variant addressed by Nanry and Barnes does not include ride time constraints or route duration constraints. As we explain in Section 4.7, these two constraints considerably complicate the evaluation of the neighbourhood.

4.1. Relaxation mechanism

An important feature of our approach is the possibility of exploring infeasible solutions during the search. Let S denote the set of solutions that satisfy constraints (i) and (ii) defined in Section 3. Each solution $s \in S$ is represented by a set of m routes (starting and ending at the depot) such that every request is assigned to exactly one route and the two vertices associated with that request are visited in the appropriate order. This solution may, however, violate the maximum load and duration constraints associated with the vehicles, the time window constraints associated with the vertices, or the ride time constraints associated with the requests.

For a solution $s \in S$, let $c(s)$ denote the total routing cost of the vehicles, and let $q(s)$, $d(s)$, $w(s)$ and $t(s)$ denote the total violation of load, duration, time window and ride time constraints, respectively. The routing cost of a vehicle k corresponds to the sum of the costs c_{ij} associated with the arcs (v_i, v_j) traversed by this vehicle. The total violation of load and duration constraints is computed on a route basis with respect to Q_k and T_k, whereas the total violation of time window constraints is equal to $\sum_{i=0}^{2n}(B_i - l_i)^+$, where $x^+ = \max\{0, x\}$. Similarly, the total violation of ride time constraints is equal to $\sum_{i=1}^{n}(L_i - L)^+$.

Solutions are evaluated using a cost function $f(s) = c(s) + \alpha q(s) + \beta d(s) + \gamma w(s) + \tau t(s)$, where α, β, γ and τ are self-adjusting positive parameters. By dynamically adjusting the values of the four parameters during the search, this relaxation mechanism facilitates the exploration of the solution space and is particularly useful for tightly constrained instances. It also encourages the use of simple exchange operators because the complex modification of a feasible solution into another feasible solution can then be achieved by a series of simpler modifications through intermediate infeasible solutions.

4.2. Neighbourhood structure

Another important ingredient of our method is the definition of attributes to characterize the solutions of S. With each solution $s \in S$ is associated an attribute set $U(s) = \{(i,k) : \text{request } i \text{ is assigned to vehicle } k\}$. The neighbourhood $N(s)$ of a solution s is composed of all solutions that can be obtained from s by applying a simple operator that removes an attribute (i,k) from $U(s)$ and replaces it with another attribute (i,k'), where $k \neq k'$. When attribute (i,k) is removed from the solution, vertices v_i and v_{i+n} are removed from route k which is reconnected by linking the respective predecessor and successor of each deleted vertex. Insertion of vertices v_i and v_{i+n} in route k' is then performed so as to minimize the total increase in $f(s)$ by using simple insertions (i.e., the ordering of the vertices already in route k' remains unchanged).

Attributes are also used to a large extent to control tabu tenures and implement a diversification strategy. When request i is removed from route k, its reinsertion in that route is forbidden for the next θ iteration by assigning a tabu status to the attribute (i,k). Through an aspiration criterion, the tabu status of an attribute can, however, be revoked if that would allow the search process to reach a solution of smaller cost than that of the best known solution *having that attribute*.

4.3. Diversification strategy

To diversify the search, any solution $\bar{s} \in N(s)$ such that $f(\bar{s}) \geqslant f(s)$ is penalized by a factor proportional to the frequency of addition of its distinguishing attributes and of a scaling factor. More precisely, let ρ_{ik} be the number of times attribute (i,k) has been added to the solution during the search. If (i,k) denotes the attribute that must be added to the current solution s in order to obtain the new solution $\bar{s}$, a penalty

$$p(\bar{s}) = \lambda c(\bar{s})\sqrt{nm}\rho_{ik}$$

is thus added to $f(\bar{s})$ when evaluating the cost of $\bar{s}$. The scaling factor $c(\bar{s})\sqrt{nm}$ introduces a correction that adjusts the penalties with respect to the total solution cost and the problem size as measured by the number of possible attributes. The more attributes there are, the higher a frequently added attribute should be penalized. This type of scaling factor was proposed by Taillard (1993) and has been used successfully in several other applications of tabu search to vehicle routing problems (see, e.g., Gendreau et al., 1994; Cordeau et al., 1997). Finally, the parameter λ is used to control the intensity of the diversification. These penalties have the effect of driving the search process toward less explored regions of the solution space whenever a local optimum is reached.

4.4. Construction of an initial solution

An initial solution s_0 is constructed by assigning every request i to a randomly selected vehicle and inserting the associated vertices v_i and v_{i+n} sequentially at the end of the partially constructed routes. This procedure ensures that constraints (i) and (ii) are satisfied. All other constraints may, however, be violated by the initial solution.

4.5. Tabu search iterations

The tabu search algorithm starts from the initial solution s_0 and chooses, at iteration t, the best non-tabu solution in $N(s_t)$ with respect to the objective function $f(s) + p(s)$. After each iteration, the values of the parameters α, β, γ and τ are modified by a factor $1 + \delta$, where $\delta > 0$. If the current solution is feasible with respect to load constraints, the value of α is divided by $1 + \delta$; otherwise, it is multiplied by $1 + \delta$. The same rule applies to β, γ and τ with respect to duration, time window and ride time constraints, respectively. This process is repeated for η iterations and the best feasible solution s^* identified during the search becomes the final solution.

4.6. Route optimization

Every κ iterations, intra-route exchanges are performed. Every vertex $v_1, \ldots, v_{2n}$ is then sequentially removed from its current route and reinserted in the best position so as to minimize the value of $f(s)$. Intra-route exchanges are also performed whenever a new best solution is identified during the search so as to provide some form of intensification.

4.7. Neighbourhood evaluation

Removing the two vertices associated with request i from route k and inserting them in route k' may affect the maximum load and total duration of both routes involved in the exchange as well as the feasibility of time window and ride time constraints for all requests assigned to these routes. As a result, evaluating the impact on $f(s)$ of removing attribute (i, k) from $U(s)$ and replacing it with attribute (i, k') involves substantial computations.

Consider a particular ordered route $k = (v_0, \ldots, v_i, \ldots, v_q)$, where v_0 and v_q both represent the depot. It is clear that sequentially setting $D_0 = e_0$ and $B_i = \max\{e_i, A_i\}$ for $i = 1, \ldots, q$ is optimal in terms of minimizing time window violations because the vehicle leaves the depot as early as possible and the service at each vertex also begins as early as possible. However, because of route duration and ride time constraints, a solution that is infeasible if $D_0 = e_0$ and $B_i = \max\{A_i, e_i\}$ for every vertex v_i can in fact be feasible (or less infeasible) if the departure from the depot as well as the beginning of service at some vertices are voluntarily delayed. Of course, a simple adjustment that will reduce route duration is obtained by setting $D_0 = \max\{e_0, e_j - t_{0j}\}$, where v_j denotes the first vertex visited after leaving the depot. However, it may sometimes be possible to further delay the departure from the depot, especially when the time window associated with vertex v_j is wide.

Assuming $d_i = 0$ (and hence $D_i = B_i$), Savelsbergh (1992), defines the *forward time slack* F_i of vertex v_i as

$$F_i = \min_{i \leqslant j \leqslant q} \left\{ l_j - \left(B_i + \sum_{i \leqslant p < j} t_{p,p+1} \right) \right\}. \tag{1}$$

Using the fact that

$$B_j = B_i + \sum_{i \leqslant p < j} t_{p,p+1} + \sum_{i < p \leqslant j} W_p, \tag{2}$$

one can rewrite (1) as

$$F_i = \min_{i \leqslant j \leqslant q} \left\{ l_j - \left(B_j - \sum_{i<p\leqslant j} W_p \right) \right\}, \tag{3}$$

$$= \min_{i \leqslant j \leqslant q} \left\{ \sum_{i<p\leqslant j} W_p + (l_j - B_j) \right\}. \tag{4}$$

The latter form emphasizes the fact that the slack at vertex v_j is the cumulative waiting time up to vertex v_j, plus the difference between the end of the time window and the beginning of service at vertex v_j. It also generalizes directly to the case of non-zero service times.

When feasibility must be maintained through exchanges, the forward time slack is the largest increase in the beginning of service at vertex v_i that will not cause any time window violation. In our case, since infeasible solutions are allowed during the search, the notion of forward time slack must be slightly modified to represent the largest increase in the beginning of service at vertex v_i that will not cause any increase in time window violations. Hence, the term $(l_j - B_j)$ should be replaced with $(l_j - B_j)^+$ in (4) because even if the time window for vertex v_j cannot be satisfied in the current route, one can nevertheless increase the beginning of service at vertex v_i by as much as $\sum_{i<p\leqslant j} W_p$ without increasing the violation of the time window constraint at vertex v_j.

When computing the forward time slack of a vertex $v_i \neq v_0$, care must also be taken not to increase the violation of ride time constraints. Indeed, by delaying the beginning of service at vertex v_i, one may increase the ride time for a request whose origin vertex is before v_i and whose destination vertex is at or after v_i. As a result, Eq. (4) becomes

$$F_i = \min_{i \leqslant j \leqslant q} \left\{ \sum_{i<p\leqslant j} W_p + \left(\min\{l_j - B_j, L - P_j\} \right)^+ \right\}, \tag{5}$$

where P_j denotes the ride time of the user whose destination vertex is v_j if $n+1 \leqslant j \leqslant 2n$, and $P_j = 0$ otherwise.

Setting $D_0 = e_0 + F_0$ instead of $D_0 = e_0$ will thus yield a modified route of minimal total duration with equal violations of time window constraints and equal or smaller violations of ride time constraints. Observe that delaying the departure time from the depot by $\sum_{0<p<q} W_p$ does not affect the arrival time A_q at the end of the route whereas delaying the departure by more would simply increase A_q by as much. As a result, the minimal route duration that does not increase constraint violations is given by

$$A_q - \left(e_0 + \min \left\{ F_0, \sum_{0<p<q} W_p \right\} \right).$$

The notion of forward time slack can also be used to delay the beginning of service B_i (and thus the departure time D_i) at the origin vertex v_i of a request i in the hope of reducing the ride time associated with this request. This might not only improve the feasibility of a route but also the quality of service as measured by the average ride time of users. Since the forward time slack is computed in such a way as to never increase the violation of time window or ride time constraints, delaying the beginning of service at an origin node can only help improve feasibility.

The impact of deleting vertices v_i and v_{i+n} from route k and inserting them at pre-specified locations in route k' can thus be assessed by performing the desired insertions and deletions and then applying the following procedure to each of the routes involved in the exchange:

1. Set $D_0 := e_0$.
2. Compute A_i, W_i, B_i and D_i and for each vertex v_i in the route.
3. Compute F_0.
4. Set $D_0 := e_0 + \min\{F_0, \sum_{0<p<q} W_p\}$.
5. Update A_i, W_i, B_i and D_i for each vertex v_i in the route.
6. Compute L_i for each request assigned to the route.
7. For every vertex v_j that corresponds to the origin of a request j
   (a) Compute F_j.
   (b) Set $B_j := B_j + \min\{F_j, \sum_{j<p<q} W_p\}$; $D_j := B_j + d_j$.
   (c) Update A_i, W_i, B_i and D_i, for each vertex v_i that comes after v_j in the route.
   (d) Update the ride time L_i for each request i whose destination vertex is after vertex v_j.
8. Compute changes in violations of vehicle load, route duration, time window and ride time constraints.

This procedure first minimizes time window constraints violations in steps (1) and (2). It then minimizes route duration without increasing time window constraints violations in steps (3)–(6). Finally, in step (7), it sequentially minimizes ride times by delaying the beginning of service at each origin node as much as possible without increasing route duration, time window or ride time constraints violations. When applied to a route for which time windows can be satisfied, the procedure will yield departure and arrival times that minimize route duration and then minimize the total violations of ride time constraints. Since minimizing route duration can only help reduce ride times, it is not suboptimal to perform these two steps sequentially. In addition, treating requests sequentially in step (7) is optimal for minimizing the violation of ride time constraints because delaying the beginning of service at a given vertex will never increase the violation of ride time constraints. It could, however, increase the ride time of a request for which the constraint is satisfied.

It is worth mentioning that although this procedure yields optimal departure and arrival times for the hierarchical objective of first minimizing $w(s)$ followed by $d(s)$ and $t(s)$ for a given solution s, it does not necessarily minimize the value of $f(s)$. Finding an optimal solution to this problem would require taking into account the tradeoffs between the different types of violation as well as the relative weights β, γ and τ. This could be achieved by solving a linear program in the variables A_i, W_i, B_i, D_i and L_i but computational efficiency would worsen considerably as a result.

4.8. Neighbourhood reduction

A vertex v_i is called critical if $e_i \neq 0$ or $l_i \neq T$. Given our previous assumptions, there is at most one critical vertex for every pair of vertices (v_i, v_{i+n}) associated with request i. If no time window is specified by the user, then any of the two vertices can be designated as critical. To reduce the size of the neighbourhood considered at every iteration of the tabu search algorithm, the following

rule is used to evaluate the impact of inserting vertices v_i and v_{i+n} in route k. First, the best insertion position is determined for the critical vertex. Then, holding the critical vertex in its best position, the best insertion position is determined for the non-critical vertex. This rule has a dramatic effect on computing times as it reduces the maximum number of possible exchanges involving request i from $O(r^2)$ to $O(r)$, where r is the number of vertices in route k.

Of course, insertion and deletion costs need not be fully recomputed at every iteration. Since each exchange involves only two routes, insertion and deletion costs for all remaining routes are still valid after performing the exchange. Since only a small portion of the total information needs to be recomputed at each iteration when the number of vehicles is large, it would not be useful to perform a partial evaluation of the neighbourhood. With 20 vehicles, for example, selecting only 10% of the requests and computing everything from scratch at every iteration is equivalent in terms of computation time to evaluating the complete neighbourhood and updating 10% of the information after each iteration.

5. Computational experiments

To our knowledge, no test instances are available in the literature for the version of the DARP studied in this paper. To analyze the behaviour of the tabu search heuristic, we have thus generated a set of 20 instances according to realistic assumptions. Information regarding time window widths, vehicle capacity, route duration and maximum ride time was provided by the Montreal Transit Commission (MTC). We have also tested our approach on six real-life datasets provided by a Danish transporter.

The randomly generated instances contain between 24 and 144 requests. For an n request instance, requests $1,\ldots,n/2$ are assumed to be outbound while requests $n/2+1,\ldots,n$ are assumed to be inbound. For each instance, origin and destination locations were generated by using a procedure, previously described by Cordeau et al. (1997), that creates clusters of vertices around a certain number of seed points. For an instance with n requests, $2n$ vertices are generated in the square $[-10,10]^2$ with this procedure. For $i=1,\ldots,n$, vertex v_i is the origin of request i while vertex v_{n+i} is its destination. For each vertex, the service time d_i is equal to 10 and the load q_i is equal either to 1 or -1 depending on whether the vertex corresponds to the origin or the destination of a request. The location of the depot is equal to the average location of the seed points used to generate origin and destination locations. For every arc $(v_i,v_j)\in A$, the routing cost c_{ij} and travel time t_{ij} are equal to the Euclidean distance between the two ver-tices.

A time window $[e_i,l_i]$ is also associated to each vertex. As mentioned in Section 4, origin vertices of outbound requests and destination vertices of inbound requests have trivial time windows $[0,T]$, where T denotes the end of the planning horizon (equal to $24\times 60=1440$ in our experiments). Two groups of instances were created by using different parameters for generating time windows. In the first group (a), narrow time windows were generated by first choosing a uniform random number e_i in the interval [60, 480] and then choosing a uniform random number l_i in the interval $[e_i+15,e_i+45]$. In the second group (b), wider time windows were created by choosing the random numbers e_i and l_i in the intervals [60, 480] and $[e_i+30,e_i+90]$, respectively. In all instances, maximum route duration is set to 480 while the capacity of a vehicle is equal to 6. Finally, the maximum ride time L is equal to 90.

Table 1
Comparison of procedures for neighbourhood evaluation

	Size		Steps (1)–(2)		Steps (1)–(6)		Full procedure	
	n	m	Cost	CPU[a]	Cost	CPU	Cost	CPU
R1a	24	3	204.85	0.37	193.63	0.63	190.79	1.90
R2a	48	5	321.65	1.35	314.46	2.25	303.60	8.06
R3a	72	7	561.06	2.75	556.28	4.46	541.25	17.18
R4a	96	9	642.09	4.76	614.51	8.45	596.94	28.77
R5a	120	11	692.56	6.81	664.62	10.99	663.10	46.24
R6a	144	13	870.85	9.35	841.03	17.36	823.52	53.87
R7a	36	4	306.26	0.79	295.07	1.26	294.77	4.39
R8a	72	6	531.63	3.34	515.75	5.62	495.71	20.44
R9a	108	8	717.59	7.41	699.56	12.66	683.15	50.51
R10a	144	10	934.18	12.63	919.15	24.38	884.10	87.53
R1b	24	3	172.34	0.38	167.14	0.63	164.98	1.93
R2b	48	5	318.64	1.54	312.93	2.52	302.91	8.29
R3b	72	7	525.35	3.14	510.09	4.99	501.17	18.54
R4b	96	9	580.01	5.15	560.24	7.81	557.93	31.18
R5b	120	11	615.99	8.36	601.58	12.70	591.60	54.33
R6b	144	13	813.15	10.51	787.70	16.65	775.24	73.70
R7b	36	4	258.10	0.85	256.36	1.33	250.88	4.23
R8b	72	6	514.45	3.53	482.73	5.78	472.69	22.86
R9b	108	8	657.28	7.48	633.43	12.43	607.85	51.28
R10b	144	10	866.97	12.68	826.09	21.36	831.10	92.41
Avg.			555.25	5.16	537.62	8.71	526.66	33.88

[a] CPU times are in minutes on a Pentium 4, 2 GHz computer.

Table 1 provides the number of requests n and the number of vehicles m in each of the 20 randomly generated instances. For instances R1a–R6a and R1b–R6b, the number of vehicles was set so as to yield moderately full routes whereas instances R7a–R10a and R7b–R10b are more tightly constrained and are probably infeasible with fewer vehicles. All test instances are available from the Internet at http://www.crt.umontreal.ca/~cordeau/data.

Following sensitivity analyses performed previously with a similar solution methodology (Cordeau et al., 2001) we first set $\delta := 0.5$, $\lambda = 0.015$ and $\theta := 7.5 \log_{10} n$. Although these values seemed to provide a proper diversification effect during the search, we noticed that periods of intensification sometimes allowed the identification of even better solutions. As a result, we chose to update the values of these three parameters in a random fashion during the search. Every 10 iterations, the values of δ, λ and θ are chosen randomly according to uniform distributions in the intervals [0, 0.5], [0, 0.015] and $[0, 7.5 \log_{10} n]$, respectively. This mechanism produces alternate periods of diversification and intensification during the search. The value of κ, which controls the frequency of intra-route reoptimizations, was also set to 10 iterations.

We performed initial experiments to assess the benefits of applying the procedure described in Section 4.7 in comparison with more simple procedures for evaluating candidate solutions in the neighbourhood $N(s)$. More specifically, we compared three approaches:

P1: perform steps (1) and (2) only but set $D_0 = \max\{e_0, e_j - t_{0j}\}$, where v_j is the first vertex visited by the vehicle;
P2: perform steps (1)–(6) so as to minimize route duration;
P3: apply the full procedure so as to minimize both route duration and ride times.

The results obtained for one execution with $\eta = 10^4$ iterations are reported in Table 1. CPU times are in minutes on a Pentium 4, 2 GHz computer. As expected, these results show that procedures **P1** and **P2** are faster but produce solutions of lower quality than the application of the full procedure. In particular, procedure **P1** is approximately 6 times faster than **P3** but yields solutions whose average cost is 5.4% higher. Procedure **P2** is only slightly slower than **P1** but generates much better solutions. These solutions have an average cost approximately 2% higher than the average for **P3**. These results show that when using the simpler procedures, the tabu search heuristic often treats as infeasible some solutions that can be made feasible by adjusting the departure time from the depot and the beginning of service at the origin vertices of the requests.

Using procedure **P3** for neighbourhood evaluation, we then executed the algorithm once on every instance with $\eta = 10^5$ iterations. The cost of the best solution found after 10^3, 10^4 and 10^5 iterations is reported in Table 2. In another set of experiments, we executed the algorithm 10 times for 10^4 iterations on each instance with different randomly generated initial solutions. The cost of the best solutions identified during these experiments is reported in column 10×10^4. Next to the solution costs, we also indicate in parentheses the percentage deviation from the cost of the best solution identified during all experiments. For each instance, the latter value is indicated in the last column.

Table 2
Best solutions identified

	n	m	10^3	10^4	10^5	10×10^4	Best
R1a	24	3	191.05 (0.54)	190.79 (0.41)	190.02 (0.00)	190.02 (0.00)	190.02
R2a	48	5	304.04 (0.65)	303.87 (0.59)	302.08 (0.00)	302.44 (0.12)	302.08
R3a	72	7	550.48 (3.46)	535.60 (0.66)	532.08 (0.00)	534.95 (0.54)	532.08
R4a	96	9	597.32 (4.28)	587.95 (2.65)	576.87 (0.71)	572.78 (0.00)	572.78
R5a	120	11	691.55 (8.57)	652.73 (2.47)	636.97 (0.00)	644.15 (1.13)	636.97
R6a	144	13	870.66 (8.64)	828.45 (3.38)	801.40 (0.00)	803.20 (0.22)	801.40
R7a	36	4	292.80 (0.37)	292.80 (0.37)	291.86 (0.05)	291.71 (0.00)	291.71
R8a	72	6	506.62 (2.37)	497.62 (0.55)	495.74 (0.17)	494.89 (0.00)	494.89
R9a	108	8	732.12 (8.88)	689.89 (2.60)	672.44 (0.00)	678.09 (0.84)	672.44
R10a	144	10	933.22 (6.20)	894.73 (1.82)	878.76 (0.00)	880.25 (0.17)	878.76
R1b	24	3	165.31 (0.52)	164.72 (0.16)	164.46 (0.00)	164.58 (0.07)	164.46
R2b	48	5	304.73 (2.93)	301.28 (1.76)	296.06 (0.00)	299.55 (1.18)	296.06
R3b	72	7	510.86 (3.56)	498.20 (0.99)	494.58 (0.26)	493.30 (0.00)	493.30
R4b	96	9	563.24 (5.10)	548.89 (2.42)	540.48 (0.85)	535.90 (0.00)	535.90
R5b	120	11	615.36 (4.34)	592.65 (0.49)	589.74 (0.00)	591.60 (0.32)	589.74
R6b	144	13	810.65 (9.02)	766.55 (3.09)	743.60 (0.00)	754.71 (1.49)	743.60
R7b	36	4	253.04 (1.95)	248.46 (0.10)	248.21 (0.00)	248.47 (0.10)	248.21
R8b	72	6	495.31 (7.05)	471.31 (1.86)	467.79 (1.10)	462.69 (0.00)	462.69
R9b	108	8	657.96 (9.30)	611.43 (1.57)	601.96 (0.00)	607.85 (0.98)	601.96
R10b	144	10	909.58 (13.89)	820.18 (2.70)	798.63 (0.00)	820.16 (2.70)	798.63
Avg.			547.80 (5.08)	524.91 (1.53)	516.19 (0.16)	518.56 (0.49)	515.38

Since optimal solutions to these problems are unknown, evaluating the performance of the heuristic is difficult. However, when using the best known solution as a benchmark, it seems that solutions of very good quality are obtained by performing 10^4 iterations. On the 20 test problems, the average deviation from the best known solution was slightly more than 1.5%. For this number of iterations, one can see from Table 1 that the CPU time varies between 2 and 90 min approximately. The latter figure is reasonable for a problem that needs to be solved daily. Since the time needed to compute an initial solution is negligible, total CPU time is directly proportional to the number of iterations performed. The results reported in Table 2 also show that for the same computing time, performing 10^5 iterations from a single initial solution yields better solutions, on average, than performing 10^4 iterations on each of 10 different initial solutions. The average deviation from the best known solution is 0.16% in the former case compared with 0.49% in the latter case.

We finally tested our solution methodology on a set of six real-life instances from a door-to-door transport service in Denmark. Two basic instances were considered (D1 and D2). Instance D1 contains 200 requests and D2 (a superset of D1) contains 295 requests. For each instance, three scenarios (a, b and c) corresponding to different time window widths were considered. All time windows are centered around a desired visit time specified by the user. Time window width is 30 min in scenario a, 60 min in scenario b and 90 min in scenario c. As before, outbound requests have a time window on the arrival time whereas inbound requests have a time window on the departure time.

In these instances, distances and travel times do not correspond to the Euclidean distance between the points but were instead computed by a geographical information system. Service times d_i are 3 min for every vertex and the maximal duration of a route is set to 12 h. The capacity of each vehicle is 8. Most users use only one unit of capacity but a few must be transported alone, i.e., they use the whole vehicle capacity. This was handled by setting $q_i = 8$ and $q_{i+n} = -8$ for each such request i. There are 30 such requests in instance D1 and 40 in instance D2. The maximum ride time is 60 min for regular requests and 120 min for requests with $q_i = 8$.

Table 3 presents the size of each instance and the results obtained by executing the algorithm once with 10^4 iterations. Again, CPU times are in minutes of a Pentium 4, 2 GHz computer. As in previous experiments, we observe that the full procedure produces better solutions than **P1** and

Table 3
Comparisons of routing cost on real-life instances

	Size		Procedure **P1**		Procedure **P2**		Procedure **P3**	
	n	m	Cost	CPU[a]	Cost	CPU	Cost	CPU
D1a	200	15	4084.95	13.21	4014.87	21.57	3935.06	104.48
D1b	200	15	3610.05	17.43	3650.10	22.21	3499.91	115.53
D1c	200	15	3436.40	14.95	3361.72	23.81	3301.47	105.02
D2a	295	20	7566.20	24.09	7074.79	44.52	6999.85	192.12
D2b	295	20	6104.40	27.83	5902.58	46.87	5938.35	211.73
D2c	295	20	5967.30	28.40	5735.99	49.69	5564.40	267.82
Avg.			5128.22	20.99	4956.68	34.78	4873.17	166.12

[a] CPU times are in minutes on a Pentium 4, 2 GHz computer.

Table 4
Comparisons of duration, waiting time and ride time on real-life instances

	Procedure **P1**			Procedure **P2**			Procedure **P3**		
	T_{tot}	W_{tot}	L_{tot}	T_{tot}	W_{tot}	L_{tot}	T_{tot}	W_{tot}	L_{tot}
D1a	8087.38	3243.25	5401.00	7658.05	2882.85	4695.63	7598.03	2903.65	4499.00
D1b	8210.72	3781.98	5061.80	7342.43	2893.25	4844.50	7153.88	2833.68	4703.48
D1c	8179.12	3914.70	5434.98	6947.02	2753.98	4830.38	6907.22	2747.92	4375.98
D2a	13486.65	5102.85	7743.25	13487.57	5520.82	7950.55	13553.28	5630.40	7799.48
D2b	12956.87	5803.42	8163.35	12904.07	5922.35	8117.77	12982.58	5985.63	8021.43
D2c	13151.68	6149.80	8598.78	12646.42	5831.12	7654.42	12773.67	6105.85	7722.42
Avg.	10678.74	4666.00	6733.86	10164.26	4300.73	6348.88	10161.44	4367.86	6186.97

P2. CPU times are, however, rather large, especially for instances D2a–D2c. One can also notice that the width of the time windows has an important impact on solution cost.

Finally, Table 4 reports additional statistics related to solution quality for the results obtained on instances D1a–D2c. In this table, columns T_{tot}, W_{tot} and L_{tot} refer to the total duration of all routes, total waiting time at all vertices and total ride time of all users, respectively. The latter results show that by purposely delaying the departure from the depot, one can often decrease the total waiting time and duration of the routes while also decreasing the routing cost. However, the greatest benefit to the users comes from the reduction in ride times. An added benefit obtained by applying procedure **P3** is that waiting time tends to be concentrated at origins, before users get on the vehicle, and rarely occurs at destinations where users have to wait aboard the vehicle. Similar results were obtained on instances R1a–R10b.

It is worth mentioning that procedure **P1** sometimes produces solutions that exhibit smaller total durations or total ride times than those produced by procedure **P3**. While this may seem surprising at first sight, one must remember that the primary objective of the heuristic is to minimize routing cost. Minimizing route duration and ride times thus helps finding feasible solutions with small cost but is not part of the objective function per se. On instance D2a, for example, the total waiting time was smaller with procedure **P1** but the cost of the solution was almost 10% higher, as can be seen from Table 3.

6. Conclusion

We have described a tabu search heuristic for the static multi-vehicle DARP and we have proposed a procedure for neighbourhood evaluation that adjusts the visit time of the vertices on the routes so as to minimize route duration and ride times. The procedure not only facilitates the identification of feasible solutions but also helps improve the overall quality of the solutions produced. The solution methodology is flexible and can easily be adapted to handle multiple depots or multiple vehicle types, by following the general framework used by Cordeau et al. (2001) for various vehicle routing problems with time windows. In addition, because route duration, waiting times and ride times are computed explicitly when evaluating the neighbourhood of a solution, the approach can easily be adapted to deal with more sophisticated objective functions incorporating these performance measures.

Acknowledgements

This work was partly supported by a Strategic research grant provided by HEC Montréal, by the Quebec Government FCAR research program under grant 2002-GR-73080, and by the Canadian Natural Sciences and Engineering Research Council under grants 227837-00 and OGP0039682. This support is gratefully acknowledged. Thanks are also due to two anonymous referees for their valuable comments.

References

Bodin, L.D., Sexton, T., 1986. The multi-vehicle subscriber dial-a-ride problem. TIMS Studies in Management Science 26, 73–86.

Borndörfer, R., Grötschel, M., Klostermeier F., Küttner, C., 1997. Telebus Berlin: Vehicle scheduling in a dial-a-ride system. Technical Report SC 97-23, Konrad-Zuse-Zentrum für Informationstechnik Berlin.

Cordeau, J.-F., Gendreau, M., Laporte, G., 1997. A tabu search heuristic for periodic and multi-depot vehicle routing problems. Networks 30, 105–119.

Cordeau, J.-F., Laporte, G., 2002. The dial-a-ride problem: Variants, modeling issues and algorithms. Quarterly Journal of the Belgian, French and Italian Operations Research Societies, forthcoming.

Cordeau, J.-F., Laporte, G., Mercier, A., 2001. A unified tabu search heuristic for vehicle routing problems with time windows. Journal of the Operational Research Society 52, 928–936.

Desaulniers, G., Desrosiers, J., Erdmann, A., Solomon, M.M., Soumis, F., 2002. VRP with pickup and delivery. In: Toth, P., Vigo, D. (Eds.), The Vehicle Routing Problem. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia.

Desrosiers, J., Dumas, Y., Soumis, F., 1986. A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows. American Journal of Mathematical and Management Sciences 6, 301–325.

Desrosiers, J., Dumas, Y., Soumis, F., Taillefer, S., Villeneuve, D., 1991. An algorithm for mini-clustering in handicapped transport. Cahier du GERAD G-91-22, École des Hautes Études Commerciales, Montreal.

Dumas, Y., Desrosiers, J., Soumis, F., 1989. Large scale multi-vehicle dial-a-ride problems. Cahier du GERAD G-89-30, École des Hautes Études Commerciales, Montreal.

Dumas, Y., Desrosiers, J., Soumis, F., 1991. The pickup and delivery problem with time windows. European Journal of Operational Research 54, 7–22.

Fu, L., 2002. Scheduling dial-a-ride paratransit under time-varying, stochastic congestion. Transportation Research B 36, 485–506.

Gendreau, M., Hertz, A., Laporte, G., 1994. A tabu search heuristic for the vehicle routing problem. Management Science 40, 1276–1290.

Glover, F., 1986. Future paths for integer programming and links to artificial intelligence. Computers and Operations Research 13, 533–549.

Glover, F., Laguna, M., 1997. Tabu Search. Kluwer, Boston.

Ioachim, I., Desrosiers, J., Dumas, Y., Solomon, M.M., 1995. A request clustering algorithm for door-to-door handicapped transportation. Transportation Science 29, 63–78.

Jaw, J., Odoni, A.R., Psaraftis, H.N., Wilson, N.H.M., 1986. A heuristic algorithm for the multi-vehicle advance-request dial-a-ride problem with time windows. Transportation Research B 20, 243–257.

Madsen, O.B.G., Ravn, H.F., Rygaard, J.M., 1995. A heuristic algorithm for the a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. Annals of Operations Research 60, 193–208.

Nanry, W.P., Barnes, J.W., 2000. Solving the pickup and delivery problem with time windows using reactive tabu search. Transportation Research B 34, 107–121.

Psaraftis, H.N., 1980. A dynamic programming approach to the single-vehicle, many-to-many immediate request dial-a-ride problem. Transportation Science 14, 130–154.

Psaraftis, H.N., 1983. An exact algorithm for the single-vehicle many-to-many dial-a-ride problem with time windows. Transportation Science 17, 351–357.

Savelsbergh, M.W.P., 1992. The vehicle routing problem with time windows: Minimizing route duration. ORSA Journal on Computing 4, 146–154.

Sexton, T., Bodin, L.D., 1985a. Optimizing single vehicle many-to-many operations with desired delivery times: I. Scheduling. Transportation Science 19, 378–410.

Sexton, T., Bodin, L.D., 1985b. Optimizing single vehicle many-to-many operations with desired delivery times: II. Routing. Transportation Science 19, 411–435.

Taillard, É.D., 1993. Parallel iterative search methods for vehicle routing problems. Networks 23, 661–673.

Toth, P., Vigo, D., 1996. Fast local search algorithms for the handicapped persons transportation problem. In: Osman, I.H., Kelly, J.P. (Eds.), Meta-Heuristics: Theory and Applications. Kluwer, Boston, pp. 677–690.

Toth, P., Vigo, D., 1997. Heuristic algorithms for the handicapped persons transportation problem. Transportation Science 31, 60–71.

Toth, P., Vigo, D., 2002. The vehicle routing problem. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia.

Wilson, N.H.M., Colvin, N., 1976. Computer control of the Rochester dial-a-ride system. Technical Report R77-31, Department of Civil Engineering, Massachusetts Institute of Technology, Cambridge, MA.

Wilson, N.H.M., Sussman, J., Wong, H., Higonnet, B., 1971. Scheduling Algorithms for dial-a-ride systems. Technical Report USL TR-70-13, Urban Systems Laboratory, Massachusetts Institute of Technology, Cambridge, MA.

Wilson, N.H.M., Weissberg, H., 1976. Advanced dial-a-ride algorithms research project: Final report. Technical Report R76-20, Department of Civil Engineering, Massachusetts Institute of Technology, Cambridge, MA.