ELSEVIER

Transportation Research Part B 38 (2004) 635–655

TRANSPORTATION RESEARCH PART B

www.elsevier.com/locate/trb

Waiting strategies for the dynamic pickup and delivery problem with time windows

Snežana Mitrović-Minić [a,*], Gilbert Laporte [b]

[a] *School of Computing Science, Simon Fraser University, Burnaby, BC, Canada V5A 1S6*

[b] *Canada Research Chair in Distribution Management, École des Hautes Études Commerciales, 3000 Chemin de la, Côte-Sainte-Catherine, Montreal, Canada H3T 2A7*

Received 1 September 2002; received in revised form 1 March 2003; accepted 26 September 2003

Abstract

The dynamic pickup and delivery problem with time windows arises in courier companies making same-day pickup and delivery of letters and small parcels. In this problem solution quality is affected by the way waiting time is distributed along vehicle routes. This article defines and compares four waiting strategies. An extensive empirical study is carried out on instances generated using real-life data.
© 2003 Elsevier Ltd. All rights reserved.

Keywords: Scheduling; Dynamic pickup and delivery problem with time windows; Heuristic algorithm

1. Introduction

The purpose of this article is to describe and compare various waiting (scheduling) strategies for the Dynamic *Pickup and Delivery Problem with Time Windows* (PDPTW), arising in the management of courier companies that operate same-day pickup and delivery of letters and small parcels.

One important aspect of dynamic problems is that since information on the problem data arrives in real-time, decisions made at an early stage of the planning horizon may affect the planner's ability to make good decisions at a later stage. In the dynamic PDPTW, the presence of time windows means that vehicles may have to wait at various locations along their routes. Adequately distributing this waiting time may affect the overall solution quality. The contribution of this article is to examine this question which has been relatively neglected in the past.

* Corresponding author. Tel.: +1-604-291-5938; fax: +1-604-291-3045.
E-mail address: smitrovi@cs.sfu.ca (S. Mitrović-Minić).

0191-2615/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/j.trb.2003.09.002

The article is organized as follows. In Section 2, we formally describe two versions of the PDPTW: the *static* case in which all problem data are known in advance, and the *dynamic* case in which pickup and delivery requests are revealed in a real-time fashion. The relevant literature for each version of the problem will be summarized. Section 3 describes our solution methodology. Section 4 introduces terminology used in scheduling, and Section 5 describes four waiting strategies. Computational results are presented in Section 6, followed by conclusions in Section 7.

2. The pickup and delivery problem with time windows

We formally define in this section the static and dynamic versions of the PDPTW, followed by a short literature review for each case.

2.1. The static PDPTW

In the static PDPTW, n users make pickup and delivery requests to be served by a fleet of m vehicles whose initial locations are specified. Denote the ith pickup location by i^+ and the associated delivery location by i^-. Denote by $d_{i,j}$ the distance from location i to location j, by $t_{i,j}$ the travel time from i to j, by s_i the service time at location i, and by $[a_i, b_i]$ the time window on a pickup or delivery location i. (The parameter a_i is also called the release time of location i, while b_i is called the deadline.) The static PDPTW consists of determining m vehicle routes and the corresponding schedules such that:

(1) each route starts at a given initial location,
(2) a pickup and its associated delivery are satisfied by the same vehicle (pairing constraint),
(3) a pickup is always made before its associated delivery (precedence constraint),
(4) all time windows are satisfied,
(5) a vehicle is allowed to wait at its initial location or at any pickup or delivery location,
(6) the total distance traveled by vehicles is minimized.

Routing consists of determining an ordered sequence of locations on each vehicle route. *Scheduling* consists of determining arrival and departure times (A_i, D_i) for each route location i.

It is implicitly assumed that the number of vehicles is always sufficient to satisfy all requests. This is in line with our observation of courier companies operating in the Vancouver area (Mitrović-Minić, 2001). Rather than lose business, these firms prefer resorting to part-time drivers, taxis, and even competitors whenever the need arises. Also, contrary to what is frequently observed in the vehicle routing literature (see, e.g., Toth and Vigo, 2002) vehicle routes may start at different locations and not at a single depot. This is explained by the fact that drivers working for courier companies are private operators who often start their work day at home. The return time to the drivers' home locations is not considered in the problem. In other words, a solution is a set of m open paths. Vehicle capacities are not constraining given the small size of letters and parcels, and maximal route durations are automatically taken care of by time windows, i.e., only those requests whose time windows fall within normal operating hours are accepted.

There exists a rich literature on the PDPTW. A large proportion of studies have dealt with the *Dial-a-Ride Problem* (DARP), a problem related with people transportation, and on which a recent survey can be found in Cordeau and Laporte (2003b). These problems are characterized by tight capacity constraints, narrow time windows, and an upper bound imposed on the customer riding time. In addition, a vehicle is typically not allowed to wait while carrying a customer, but there are some exceptions to this rule (Toth and Vigo, 1997). Another important related problem is the *Pickup and Delivery Problem with Time Windows and with Capacity constraints* (PDPTWC) arising in the transportation of goods. Optimization algorithms for these capacity constrained PDPTWs include dynamic programming methods applied to the single-vehicle DARP (Psaraftis, 1983; Desrosiers et al., 1986), and the Dantzig–Wolfe decomposition method (column generation) which can also solve the multiple-vehicle version of the PDPTWC (Dumas et al., 1991). Since only modest size instances (up to 55 requests) have been solved optimally by such methods, several types of heuristics have been explored: decomposition methods, construction methods, improvement methods, and incomplete optimization based on mathematical programming methods. One of the first heuristics was a three-phase approach (Jaw et al., 1986) that included clustering in time, clustering in space and routing. It was tested on instances with 2617 customers. One of the most successful methods is incomplete column generation, capable of solving instances involving more than 2400 requests (Desrosiers et al., 1991; Ioachim et al., 1995). Unfortunately, the approach is very sensitive to the tightness of the time windows and capacity constraints, and tends not to perform well when these constraints are loose (Dumas et al., 1995). Other recent studies are those of Borndörfer et al. (1997), based on branch-and-cut algorithm, and of Wolfler Calvo and Colorni (submitted for publication) based on the solution of an assignment problem. Modern heuristics applied to the capacity constrained PDPTW include a variable-depth arc-exchange procedure combined with a simulated annealing (Van der Bruggen et al., 1993) and tested on the DARP with 38 and 50 customers, a constraint-directed search algorithm (Potvin and Rousseau, 1992) applied on the DARP with around 100 requests, tabu with thresholds (Toth and Vigo, 1997) applied to the DARP with 312 requests, reactive tabu search (Nanry and Barnes, 2000) tested on PDPTWC instances with 100 requests, and classical tabu search (Cordeau and Laporte, 2003a) applied to DARP instances with up to 295 requests. The quality of solutions produced by the modern heuristics tends to be strongly related to running time.

The scheduling aspect has been addressed in the context of people transportation where a number of "people specific" constraints have led to the introduction of inconvenience functions. The review by Desrosiers et al. (1995) mentions two approaches (Sexton and Bodin, 1985; Dumas et al., 1990) for minimizing total customer inconvenience. Also, an inconvenience function has been used in the solution of the pickup and delivery problem with soft time windows (Sexton and Choi, 1986).

The pickup and delivery problem with time windows without capacity constraints (PDPTW) has been much less studied. There are no reports on exact algorithms for this problem, and only a few heuristics have been developed, all for the dynamic case.

2.2. The dynamic PDPTW

The definition of the static PDPTW implicitly assumes that all information is available at time of planning. This is a realistic assumption in contexts where users specify transportation requests

one or two days in advance, as is often the case in dial-a-ride services for the elderly and the handicapped (see, e.g., Cordeau and Laporte, 2003b). In courier operations, requests are typically made dynamically, in real-time, so that a priori planning of the day's operation is no longer possible. The problem here is to allocate requests to vehicles on an on-going basis. It is also possible, at least in principle, to transfer a pickup and delivery request from a vehicle to another, although this was not done in the cases we have observed. There are important differences with the static PDPTW. One is that the number of users is not an input data in the dynamic case. Another difference is the importance of scheduling in a dynamic environment. An important question is to decide how to adequately distribute a vehicle's idle time in order to achieve a good solution. Should a vehicle delay as much as possible its departure from its current location, should it move as soon as it can to its next location, or do some intermediate waiting strategies perform better? Finally note that even in a dynamic context a batch of pickup and delivery requests will often be available at the start of planning as a number of users may have made advance bookings. In the cases we have observed, such requests make up around 10% of all requests.

The literature on the dynamic PDPTW is less developed than for the static case. We mention reports by Psaraftis (1988), Powell (1991), Psaraftis (1995), and Gendreau and Potvin (1998) specifically focused on dynamic vehicle routing problems. A rare case study on the DARP in a dynamic environment is described by Madsen et al. (1995) who solved instances with 300 customers and 24 vehicles. All known methods for the dynamic PDPTW use modern heuristics. Shen et al. (1995) have developed an expert system using a neural network as a learning module. The system was tested on real-life instances with 140 requests, 12 vehicles, and a 6-h service period. Thirty minutes were allowed for a pickup, and 90 min for a delivery, from the time a request came in. Another decision support system with learning capabilities uses a utility function pre-constructed by a genetic programming technique, aimed at approximating the decisions of a dispatcher (Benyahia and Potvin, 1998). The same real-life instances were used for testing. Gendreau et al. (1998) have solved the dynamic pickup and delivery problem with soft time windows by means of a tabu search procedure. Ejection chains were used to define neighborhoods, and an adaptive memory (Rochat and Taillard, 1995)—a repository of routes associated with the best visited solutions—was used to diversify the search, while a decomposition technique was implemented to intensify the search. The method was tested on instances with 24 requests per hour, over the 6-h service period, and on instances with 33 requests per hour, over the 4-h service period. The authors also reported on a parallel method implementation. Finally, Mitrović-Minić et al. (in press) have recently proposed double-horizon based heuristics for the dynamic PDPTW. These make use of two different planning horizons (short term and long term) to which different goals are applied. The idea is that in the short term, it is important to concentrate on minimizing total route length, whereas in the long term, it is more crucial to preserve a certain amount of flexibility in order to better accommodate future requests. Several double-horizon based heuristics were tested on instances containing 100, 500 and 1000 requests.

In a dynamic environment, the routing and scheduling decisions are intertwined. Yet, we are not aware of any study on the scheduling aspect of the PDPTW. This article attempts to fill this gap. It concentrates only on the scheduling aspect of the problem. More specifically, it is concerned with the design of a good *waiting strategy*, i.e., a way of organizing and distributing the waiting time along a dynamically constructed route.

3. Our on-line solution methodology

This section gives general description of our on-line solution methodology that deals with routing and scheduling subproblems in a sequential manner. Routing is solved by a cheapest insertion procedure and optionally improved by a tabu search procedure. The scheduling of fixed routes is handled through various strategies described in Sections 4 and 5. The cheapest insertion procedure which handles new requests is an extension of the procedure introduced by Rosenkrantz et al. (1977). Our procedure determines the overall best insertions for the two locations of a request by examining all possible pairs of feasible slots in each route. The cost is the increase in total route length. The procedure starts by inserting the pickup location i^+ in a feasible slot, it updates the slack times of all route locations after i^+, and it then checks all feasible slots for the delivery location i^-. The best pair of slots is stored, and another feasible slot for the pickup location is explored. (The *slack time of a route location* is the difference between the latest departure time and the earliest departure time from the location.) By appropriately updating the slack times, the feasibility of a location insertion may be checked in constant time. Therefore, the overall complexity of a request insertion in a route with l locations is $O(l^2)$, which is also equal to the number of possible pairs of slots in the route.

The cheapest insertion procedure may be applied immediately upon the arrival of a request, or every k minutes. When applied periodically the procedure may assign all new requests or only requests with an impending pickup deadline. Assuming that $b_{i^+} = \min\{b_{i^+}, b_{i^-} - t_{i^+,i^-}\}$, requests with an impending pickup deadline are those for which $b_{i^+} \leqslant t + c$, where t is the current time and c is pre-specified constant. Before their assignment in a route, unassigned requests may be sorted, either by their slack time $(b_{i^-} - a_{i^+}) - t_{i^+,i^-}$ or by their delivery deadline. Therefore, the on-line insertion procedure is defined by the following three criteria: a criterion that determines when to start inserting the unassigned requests, a criterion that determines which requests are eligible for insertion, and a criterion for sorting the eligible requests.

The tabu search procedure is a simplified version of the heuristic introduced by Gendreau et al. (1998), with neighborhoods defined by means of ejection chains. A greedy approach is used to extend the current ejection chain. Solution feasibility is maintained at all times, and a solution already visited is tabu for a random number of iterations. The tabu search procedure may be used or omitted. When the algorithm includes tabu search, the cheapest insertion procedure is applied every k minutes, and the tabu search procedure is applied to the current solution in the meantime.

A more systematic description of our on-line algorithm is provided in the pseudo-code given below. Note also that while the algorithm runs new requests are accumulated.

On-line Algorithm

```
begin
  while (true)
    while (criterion for starting insertion of requests is not satisfied)
     wait
    end while
    update the earliest times and slack times of all route locations, if necessary
    select unassigned requests eligible to be assigned to routes
    sort eligible requests (optional step)
```

```
        for each eligible request (i+, i−) do
            if (there are vehicles that can feasibly serve request (i+, i−)) then
                find the best insertion of the request
                insert the request
            else
                assign the request to a new vehicle
            end if
            make a schedule for the changed route
        end for
        if (the tabu search procedure is to be applied) then
            t ← current time
            k ← time between request insertions
            ε ← time needed for concatenating two solutions
            S ← current solution
            S_f ← first portion of solution S, from now until time t + k
            S_0 ← portion of S starting from time t + k
            S'_0 ← S_0 improved by applying tabu search for time k − ε
            S ← concatenate solutions S_f and S'_0
            make a schedule for S
        end if
    end while
end
```

A route comprises unserved locations. Each location is removed from the route once it has been served. The update of the earliest arrival time, the earliest departure time and the slack time of route locations (line 6 of the algorithm) is needed only if the scheduling subproblem is solved by a strategy that allows a vehicle to wait at location i after serving it. In this case, the earliest arrival time at any location following location i will change as long as the vehicle waits. More precisely, the earliest arrival time at the next location to be served depends on the time when the vehicle leaves its current position (location i) as given in Eq. (1) of Section 4. The change in the earliest arrival time at a route location may cause changes in the earliest departure time from the location, thus causing changes in the slack time of the location.

4. Scheduling

In this section we define some notation similar to that introduced by Savelsbergh (1992). Consider a fixed route for which a schedule has to be determined (without any change in the location sequence). For the sake of simplicity we denote the starting position of a vehicle by 0 and its final location by l. The scheduled arrival time A_i at location i may be any time between the earliest arrival time $\underline{A}_i$ and the latest arrival time $\overline{A}_i$ (Fig. 1), i.e., $A_i \in [\underline{A}_i, \overline{A}_i]$. Similarly, the scheduled departure time D_i from location i may be between the earliest departure time $\underline{D}_i$ and the latest departure time $\overline{D}_i$, i.e., $D_i \in [\underline{D}_i, \overline{D}_i]$. The *earliest arrival time* $\underline{A}_i$ is the earliest time when location i *may* be reached, assuming that all locations preceding i were served within their time

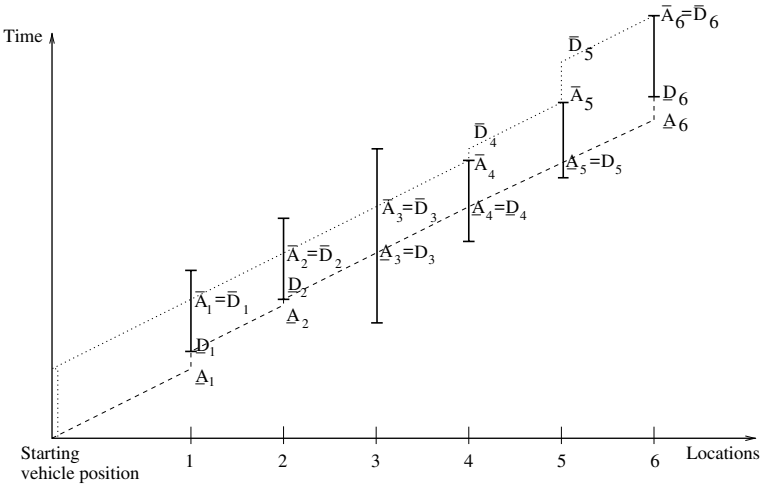

Fig. 1. The earliest arrival times $\underline{A}_i$, the latest arrival times $\overline{A}_i$, the earliest departure times $\underline{D}_i$ and the latest departure times $\overline{D}_i$ at locations of one route. The vertical intervals shown are the time windows for each of six locations in the route. The route locations are positioned along the horizontal axis, and service time at each location is 0. The dashed line represents the route through time if it is scheduled according to the earliest times. The dotted line represents the route through time if it is scheduled according to the latest times. The slope of each line represents driving the vehicle, while vertical portions correspond to waiting periods.

windows. If the vehicle serving location i leaves its current position 0 at time t, then the following relationships hold:

$$\begin{aligned} &\underline{A}_1 = t + t_{0,1}, \\ &\underline{D}_i = \max\{a_i, \underline{A}_i\} + s_i \quad \forall i \in \{1, \ldots, l\}, \\ &\underline{A}_i = \underline{D}_{i-1} + t_{i-1,i} \quad \forall i \in \{2, \ldots, l\}. \end{aligned} \tag{1}$$

Note that $\underline{A}_1$ is function of t. When t is fixed $\underline{A}_1$ is a constant.

The *latest departure time* $\overline{D}_i$ from location i is the latest possible time when the vehicle can leave location i, in order to serve on time all locations following i. The latest arrival times and the latest departure times may be calculated by the formulas:

$$\begin{aligned} &\overline{A}_l = b_l - s_l, \\ &\overline{D}_i = \overline{A}_{i+1} - t_{i,i+1} \quad \forall i \in \{1, \ldots, l-1\}, \\ &\overline{A}_i = \min\{\overline{D}_i, b_i\} - s_i \quad \forall i \in \{1, \ldots, l-1\}. \end{aligned}$$

5. Waiting strategies

Two simple waiting strategies, called *Drive-First* (DF) and *Wait-First* (WF), reflect the two most extreme situations. The other two strategies introduced in this article, *Dynamic Waiting* (DW) and *Advanced Dynamic Waiting* (ADW), are combinations of the two simple strategies.

The best empirical results have been achieved with ADW, while all other strategies have advantages and disadvantages which are discussed below.

5.1. The drive-first waiting strategy

The drive-first waiting strategy requires a vehicle to drive as soon as it is feasible, i.e., to drive from its current location at the earliest departure time. The scheduled arrival time A_i at location i is thus the earliest arrival time $\underline{A}_i$. A vehicle may wait before serving a location if it arrives before the location release time. The waiting time at location i is equal to $\max\{0, a_i - \underline{A}_i\}$, while the waiting time after service is 0. Fig. 2 shows the route of a vehicle serving three locations.

The DF strategy is the most commonly used waiting strategy when transporting letters, parcels or goods (excluding hazardous and easy-perishable loads), e.g., see Benyahia and Potvin (1998), Gendreau et al. (1998). This may be because DF is the only appropriate strategy for the corresponding static vehicle routing. It is also worth mentioning that in a dynamic environment, maintaining the earliest arrival and departure times of route locations is relatively easy with DF since the earliest times and the slack times remain the same if no new location is inserted in the route.

5.2. The wait-first waiting strategy

The wait-first waiting strategy requires a vehicle to wait at its current location for as long as it is feasible, i.e., the vehicle leaves its current location at the latest possible departure time. The scheduled arrival time A_i at location i is the latest arrival time $\overline{A}_i$. A location is served immediately after being reached but a vehicle may wait after serving the location. The waiting time at location i is $\max\{0, \overline{D}_i - b_i\}$, while the waiting time before service is 0. Fig. 3 shows the route of a vehicle serving three locations by using WF.

The WF strategy has a potential for building shorter routes, compared to DF. Waiting of vehicles at their starting positions results in more requests being known at the time they leave and in more potential for optimizing the routes serving these locations. This was confirmed by our empirical study. Unfortunately, WF requires more vehicles than DF for serving the same set of locations. The explanation lies in a tendency for WF to concentrate long waiting times in the first

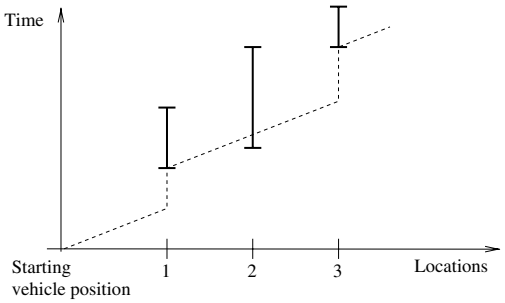

Fig. 2. The drive-first waiting strategy: the vehicle drives as soon as possible and waits only if it reaches a location before its release time. The route locations are positioned along the horizontal axis. The three vertical intervals are the location's time windows. The dashed line shows the vehicle route in time.

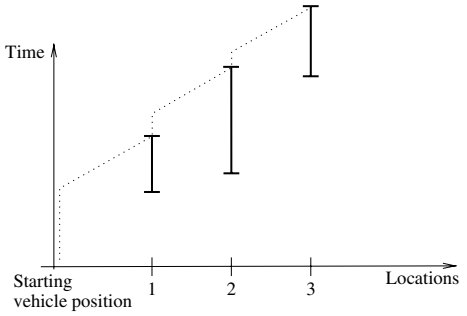

Fig. 3. The wait-first waiting strategy: the vehicle waits at the current location for as long as it is feasible. The route locations are positioned along the horizontal axis. The three vertical intervals are the location's time windows. The dashed line shows the vehicle route in time.

part of the route, leaving too little waiting time in the remainder of the route. Fig. 4 shows an example of the waiting times generated by WF. While the vehicle waits at the starting position for new locations to appear in the hope of constructing better route, it stands idle for too long. When the vehicle finally leaves its starting position, at time t, it may spend the most of its remaining time driving. As a result, new requests appearing after time t are unlikely to be served by this vehicle.

It is more difficult to implement WF than DF. The WF strategy requires updating of the earliest arrival and departure times because their values constantly change while the vehicle is waiting when it is feasible to drive.

5.3. Dynamic partitioning of a route

In order to describe the next two waiting strategies it is necessary to introduce the concept of dynamic partitioning of a route. We are interested in partitioning route locations in such a way that each partition contains a number of consecutive locations close to each other. A partition can

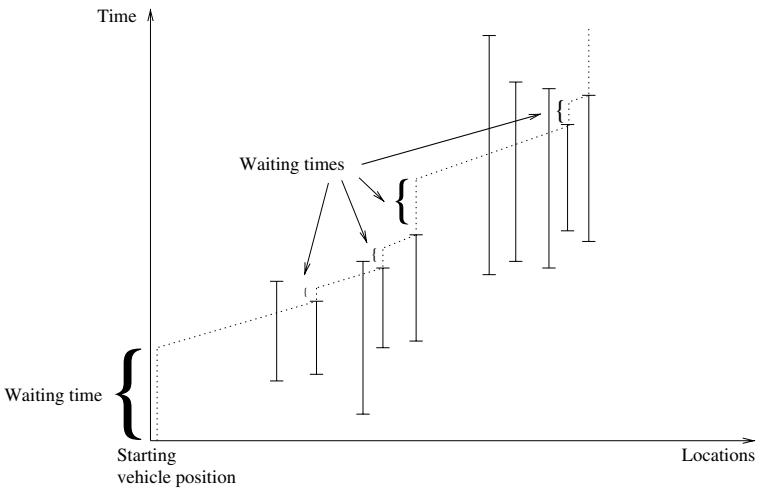

Fig. 4. Route and waiting times generated by the wait-first waiting strategy.

be represented by a three-dimensional box called the *service zone*. Geometrically, a service zone is a cylinder or a prism, whose *base* is a circle or a polygon enclosing the corresponding locations. We choose the minimal enclosing rectangle for the base, although the minimal enclosing circle and the convex hull may be used. The height of a service zone, the *time span*, represents the time spent serving the locations of the zone. Ideally, one could set the zone time span equal to the interval (A_{z_1}, D_{z_k}), where z_1 is the first location and z_k is the last location in the zone. If the scheduled arrival and departure times are not available, the time span may be set to $(\underline{A}_{z_1}, \underline{D}_{z_k})$. We use $\underline{D}_{z_k}$ instead of $\overline{D}_{z_k}$ as the end of the time span, in order to avoid overlapping of the time spans of two consecutive service zones of a route. (The value $\overline{D}_i$ may be larger than $\underline{A}_{i+1}$, as it is, for example, when $\overline{D}_i = b_i$, $\underline{A}_{i+1} = a_{i+1}$ and $b_i > a_{i+1}$.)

A route may be represented by a series of service zones. Fig. 5 depicts a route partitioned in two service zones each containing five locations. (The starting vehicle position does not belong to the first service zone.) All the locations are positioned along a straight line. Therefore, the zone bases are line segments and the zones are rectangles. In general, route partitioning is *dynamic* because the service zones change over time. Changes occur when locations are served, when locations are inserted or removed, or simply due to the passing of time. For example, because of the pre-specified maximal size u of a zone base, the insertion of a new location in the route may cause one of the following situations:

(a) The new location becomes part of an existing service zone, possibly causing an expansion of the zone. The zone base expands when the new location does not lie within the base. The zone time span expands as it has been defined—based on the first and last locations.
(b) The new location becomes part of an existing service zone, causing the splitting of the zone. The split is necessary when the area of newly created zone base is larger than pre-specified maximal size of a zone base.
(c) A new service zone is created containing the new location, in the case when route was previously empty.

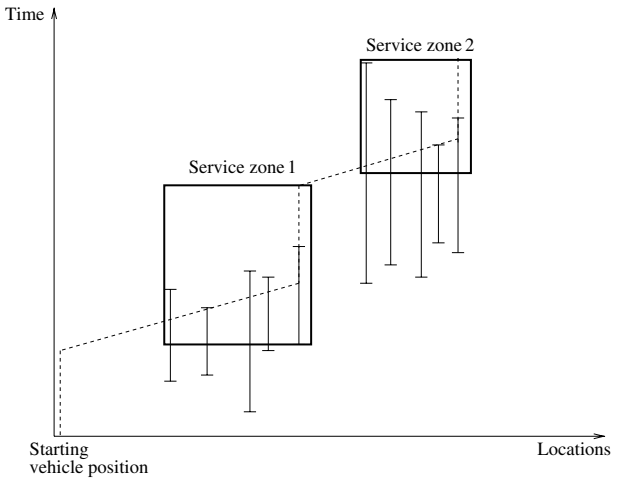

Fig. 5. Route consisting of two service zones. The route locations are positioned along the horizontal axis.

The following pseudo-code provides details of the procedure *addLocInARouteServiceZone* that updates the service zones of a route. The procedure is always called after the insertion of a location q after location p in a route. Procedure *addLocInARouteServiceZone* increases the zone base or the zone time span. It handles zone splitting and zone merging, if necessary. Since a route consists of only unserved locations, a route may be empty even though its vehicle has served some locations in the past.

addLocInARouteServiceZone(q)

```
begin
  if (q is the only location in the route) then
    create an empty service zone and insert q in it
    return
  end if
  if (q is the first location in the route) then
    Z1 ← first route service zone
  else
    Z1 ← zone to which p belongs
    if ((p is the last in Z1) and
     (Z1 is not the last service zone in the route)) then
          Z2 ← zone after Z1
          d1 ← distance between q and Z1
          d2 ← distance between q and Z2
          if (d2 < d1) then Z1 ← Z2
    end if
  end if
  if (area of Z1's base will not be larger than u after insertion of q) then
    insert q in Z1 and update its base and time span
  else
    if (q would be the first (last) location in Z1) then
     create an empty zone before (after) Z1 and insert q in it
    else
     split Z1 into two zones Z1' and Z1''' such that p is the last location in Z1'
     create an empty zone Z1'' between Z1' and Z1'''
     insert q in Z1'''
     if (Z1' and Z1'' may be merged) then merge Z1' and Z1''
     else if (Z1'' and Z1''' may be merged) then merge Z1'' and Z1'''
    end if
  end if
  return
end
```

The distance between a location and a service zone is sum of two distances: a plane distance and a time distance. The *plane distance* is the minimum distance between the location and the zone base. The *time distance* is the distance between the location's time window and the zone's time

span. More precisely, if the two time intervals overlap, the time distance is 0, and if they do not overlap, the distance is equal to the minimum distance between their end points. Before adding the two distances, the time distance is transformed into the distance that may be traveled during that time.

Similarly, the removal of a location from a route requires some updates of the route service zones. Two possible cases are handled differently in our implementation. When a location is removed from a route because it has already been served, no changes are made to the route service zones. The second case of a location removal is a result of our tabu search procedure which is an inter-route improvement procedure. The removal is handled in the following way: the whole portion S_0 of the solution over which the tabu search has been applied is removed, and the re-optimized solution S_0' is concatenated into S_f by inserting locations one at a time at the end of each corresponding route. (See the notation in Section 3.) The insertion of each location is followed by a call to procedure *addLocInARouteServiceZone*.

In the dynamic route partitioning scheme, two new waiting strategies are built: the dynamic waiting strategy and the advanced dynamic waiting strategy. These arrange waiting along the route in a few larger time intervals.

5.4. The dynamic waiting strategy

The dynamic waiting strategy is a combination of DF and WF. It constructs a schedule for a fixed route in the following manner: (1) the vehicle drives within each service zone according to DF, and (2) when the vehicle finishes serving all locations in one service zone, the vehicle uses WF. Consequently, the vehicle drives as soon as it is feasible while serving close locations; when all such locations are served and the vehicle has to serve the first far location, it waits for as long as it is feasible. Another way to look at DW is to coalesce all route locations of one service zone into a super-location. Then, DW is equivalent to scheduling all super-locations using WF, while the scheduling of all locations within one super-location is done by DF.

A route scheduled by DW is shown in Fig. 6. (The problem instance is the same as that of Fig. 4.) The locations marked i_1 and i_2 dictate the WF strategy for the route super-locations. The deadline of location i_1 determines the latest departure time from the starting vehicle position. The deadline of location i_2 determines the latest departure time from service zone 1.

Empirically, DW has achieved shorter routes than DF and used less vehicles than WF. Compared with DF, the DW strategy increases waiting at some locations along the route, and thus increases the possibility of assigning new locations to the closest service zone. Compared with WF, the DW strategy divides the waiting time available in the route into a few larger intervals, and distributes them along the entire route (see Figs. 4 and 6). This leaves more time for the insertion of a new location in the second half of the service period. A disadvantage of DW is in the number of vehicles used: in many instances, the number of vehicles used is still large compared with the solutions achieved by DF.

An analysis of the experimental results reveals that the reason for this behaviour lies in the characteristics of the instances which are very similar to real-world instances. None of the locations is known in advance, requests arise uniformly throughout the service period, and time windows are wide. In this environment, DW tends to generate schedules closely related to those produced by WF. Most of the waiting time is concentrated and used up after the first service zone

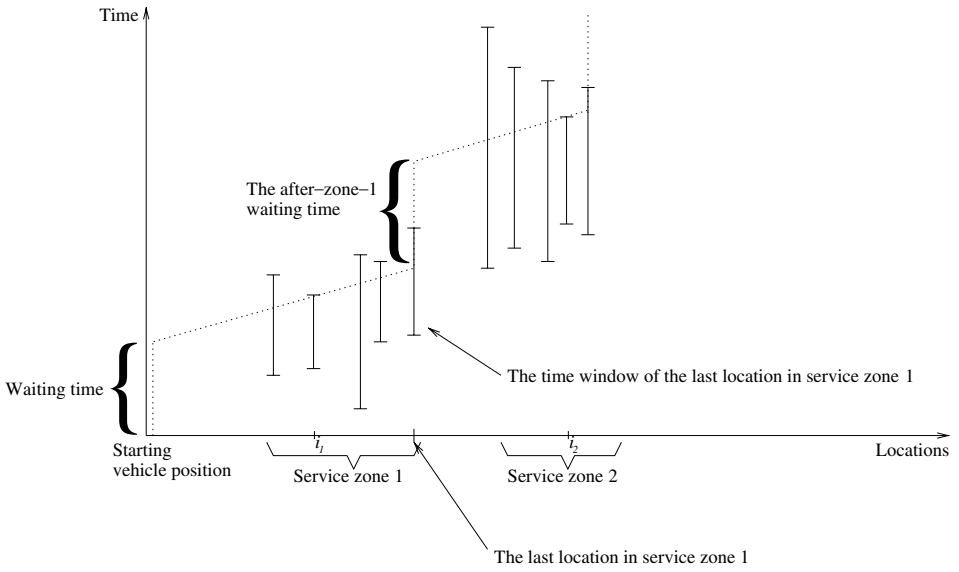

Fig. 6. Waiting times generated by the dynamic waiting strategy.

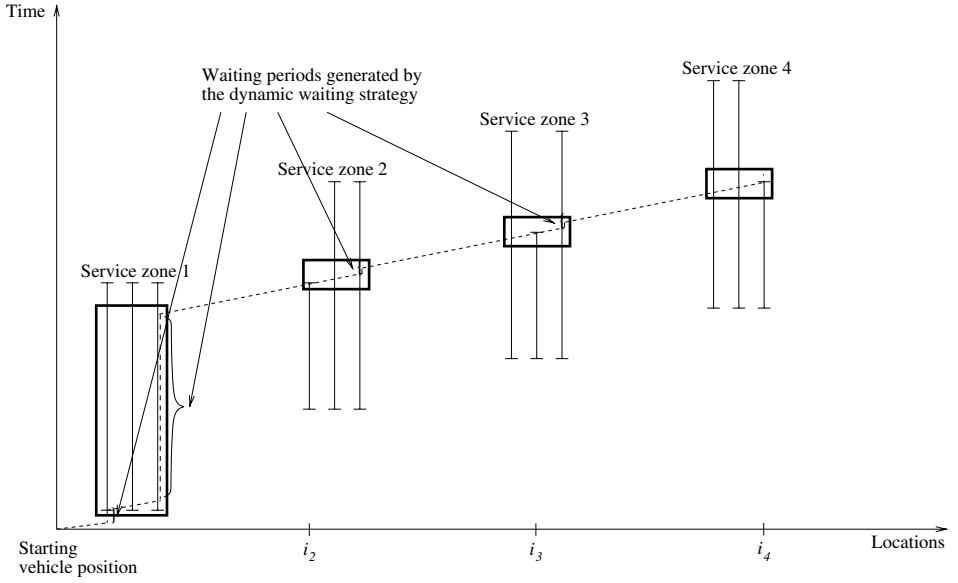

Fig. 7. Waiting times generated by the dynamic waiting strategy. The route consists of four service zones.

of the route, leaving very little waiting time along the remainder of the route. An example is given in Fig. 7, whereas the same route scheduled by WF is given in Fig. 8. The advanced dynamic waiting strategy, introduced in the next subsection, attempts to correct this behaviour.

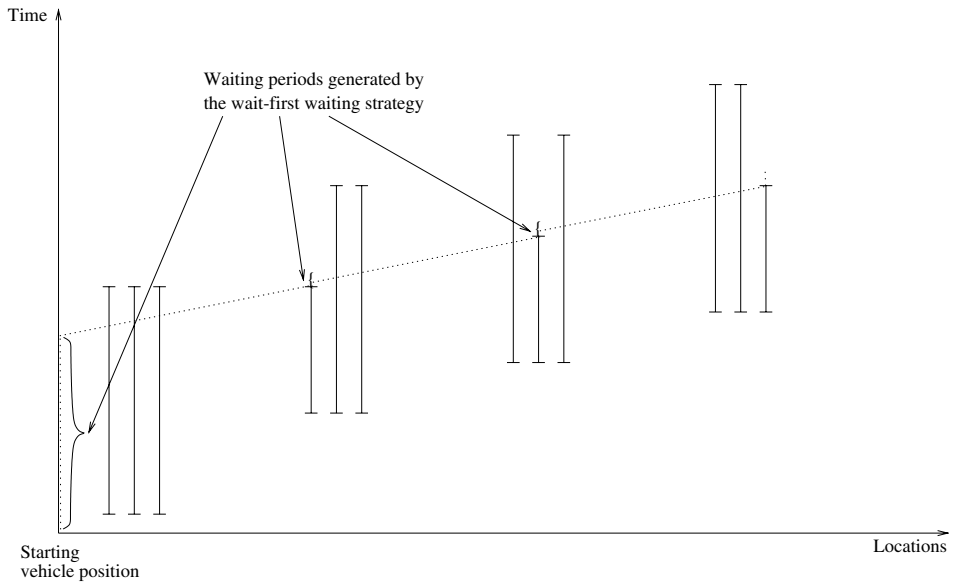

Fig. 8. Route and waiting times generated by the wait-first waiting strategy.

5.5. The advanced dynamic waiting strategy

The idea behind the advanced dynamic waiting strategy is to propagate the total waiting time available in the route along the entire route (Fig. 9). It relies on the dynamic partitioning of the

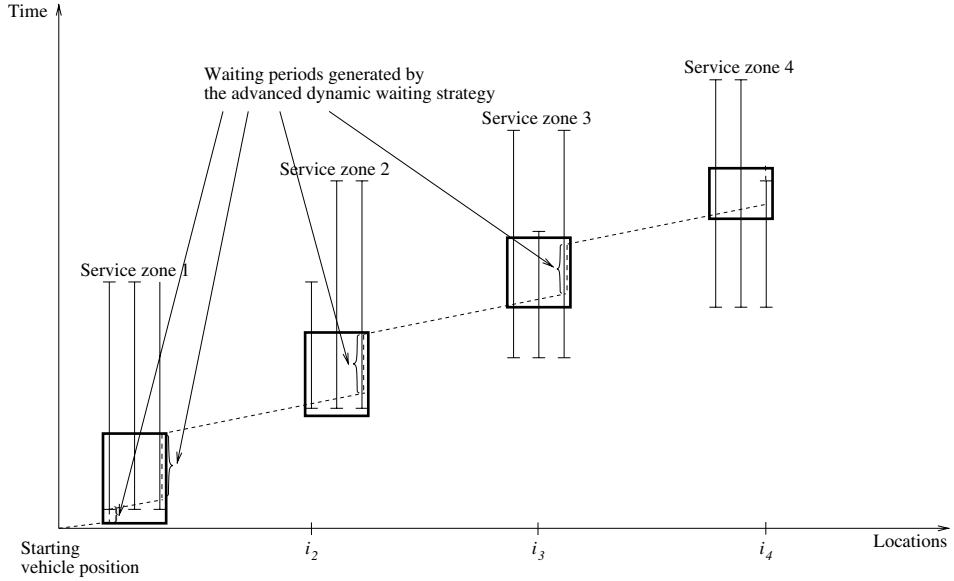

Fig. 9. Waiting times generated by the advanced dynamic waiting strategy. The route locations are positioned along the horizontal axis and the route consists of four service zones.

route. Within each service zone, the scheduling is the same as in DW—as per DF. However, the waiting time at the last location of a service zone—the *after-zone waiting time*—is determined differently. It is taken to be just a portion of the longest feasible waiting time. This portion may be defined in several ways. In our experiments, the after-zone waiting is proportional to the time span of the zone. It is the ratio of the zone time span to the sum of time spans of all service zones of the route. Another approach could be to set the after-zone waiting time proportional to the sum of slack times available in the zone. (The slack time of location i is the time available for the insertion of a new location immediately after location i.) In our tests the ADW strategy has generated the best solutions (with respect to total route length and the number of vehicles used) compared with all waiting strategies introduced in this article.

6. Empirical study and computational results

This section describes an empirical study whose purpose was to compare the four waiting strategies just described. We have carried out several sets of experiments. In all the experiments routing was solved first, followed by scheduling of the fixed routes. The routing subproblem was initially solved by means of a cheapest insertion heuristic, and later a tabu search improvement procedure was added. The scheduling subproblem was solved by each of the four waiting strategies. The performance evaluation of the waiting strategies was done with respect to the drive-first waiting strategy.

6.1. Test instances

To our knowledge, no test instances are available in the literature for the version of the dynamic PDPTW studied in this article. To analyze behaviour of the waiting strategies we have generated forty instances with 100, 300, 500, and 1000 requests. There are 10 instances for each problem size. The time windows of requests were generated to emulate real-life data. In all instances, the service period is 10 h, the service area is 60×60 km^2, and vehicle speed is 60 km/h. Requests occur during the service period according to a continuous uniform distribution, and no requests are known in advance. The time windows depend on the maximal time allowed to serve the whole request, assuming that the request is picked up at the pickup release time. If this maximal time is k hours, the request is called a k-hour request.

Based on real-life data collected in two medium-to-large courier companies operating in Vancouver, the distribution of requests is taken to be: 20% 1-h requests, 30% 2-h requests, and 50% 4-h requests. A request i is created by: (1) generating the time of its occurrence, (2) generating random positions of the pickup and delivery locations, and (3) generating k in order to decide whether request i is a 1-h, a 2-h, or a 4-h request. The time windows of the pickup location i^+ and the delivery location i^- are determined as follows: the pickup release time a_{i^+} is the time when the request occurs, the delivery deadline is $b_{i^-} = a_{i^+} + k$, the pickup deadline is $b_{i^+} = b_{i^-} - t^*_{i^+,i^-}$, and the start of the delivery time window is $a_{i^-} = a_{i^+} + t^*_{i^+,i^-}$. The time $t^*_{i^+,i^-}$ is the minimum travel time between i^+ and i^-. All requests can be feasibly served within the service period.

Rejection of requests and violation of time windows are not allowed. Consequently, the fleet size is assumed to be unbounded, which is consistent with practice. The initial fleet is either five

vehicles for all problem sizes, or 20, 40, 60, 80 for instances with 100, 300, 500, 1000 requests, respectively. We have performed no-depot experiments, and one-depot experiments with the depot located at (20, 30 km). In the no-depot experiments, the starting positions of vehicles are taken from pre-generated random positions.

All experiments were carried out using simulation speed in excess of one, in order to solve many more instances of the dynamic problem in less time. A simulation speed s means that 1 h of real-life operations is simulated in $1/s$ hours of computer time. Note also that a simulation speed s used for solving a problem instance of size n indicates that even an instance of size $s \times n$ can be solved without any significant difficulty.

6.2. The waiting strategies behaviour when routing is solved by cheapest insertion

This section reports on the performance of the all four waiting strategies when the routing subproblem is solved by the cheapest insertion procedure described in Section 3. The dynamic partitioning of a route (see Section 5.3) is done by using value of 100 km^2 for u—the maximal area of a service zone base. We have tested following seven variations of the cheapest insertion procedure:

(1) insertion is done immediately upon new request arrival,
(2) insertion is done every 15 min:
  (a) all new requests are eligible for insertion:
    (i) requests are not sorted,
    (ii) requests are sorted by their slack time,
    (iii) requests are sorted by their delivery deadlines.
  (b) only requests whose pickup deadline is within next 2 h are eligible for insertion:
    (i) requests are not sorted,
    (ii) requests are sorted by their slack time,
    (iii) requests are sorted by their delivery deadlines.

Instances with 100 and 300 requests were tested with a simulation speed of 60, while instances with 500 and 1000 requests were tested with a simulation speed of 30. Three groups of experiments were carried out. In the first group (Table 1), no depot was used and five vehicles were initially active. The second group of experiments (Table 2) also had no depot but contained a larger number of initially active vehicles: 20 for 100-request instances, 40 for 300-request instances, 60 for 500-request instances and 80 for 1000-request instances. The third group of experiments (Table 3) introduced a depot, and the number of initially active vehicles were the same as in the second group of experiments. These three tables provide average experimental results over 70 test runs (10 instances times seven variations of the cheapest insertion procedure). Boldface entries represent average improvements obtained by the three waiting strategies (WF, DW, and ADW), compared with DF.

The WF, DW, and ADW strategies have achieved shorter routes (in total length) on almost all instance sizes, with respect to the routes achieved by DF. The WF strategy achieves solutions with route length 1.27–9.19% smaller than with DF (with the exception of experiments on 500-request instances in Table 1), but it uses up to 17.06% more vehicles. The DW strategy was tested only in

Table 1
No-depot experiments with five initially active vehicles

Scheduling	100 requests		300 requests		500 requests		1000 requests	
	Distance	m	Distance	m	Distance	m	Distance	m
DF	2822.96	11.43	6397.47	22.71	9208.96	34.1	15 688.90	54.7
WF	2563.57	13.38	6112.41	25.59	9246.90	37.26	14 732.66	55.96
WF vs DF (%)	**9.19**	−17.06	**4.46**	−12.68	**−0.41**	−9.27	**6.10**	−2.30
DW	2587.45	12.35	6193.82	23.67	9325.38	35.06	15 415.29	55.35
DW vs DF (%)	**8.34**	−8.05	**3.18**	−4.23	**−1.26**	−2.82	**1.74**	−1.19
ADW	2620.41	11.45	6087.52	22.82	9026.27	33.04	14 721.21	52.25
ADW vs DF (%)	**7.17**	−0.17	**4.84**	−0.48	**1.98**	3.11	**6.17**	4.48

The table reports total distance traveled and number of vehicles used (m). It also shows improvements achieved by each of the three waiting strategies—wait-first (WF), dynamic waiting (DW), and advanced dynamic waiting (ADW)—compared with drive-first (DF).

Table 2
No-depot experiments with larger number of initially active vehicles

Scheduling	100 requests		300 requests		500 requests		1000 requests	
	Distance	m	Distance	m	Distance	m	Distance	m
DF	2825.66	11.21	6403.20	23.07	9410.63	34.46	15 603.99	55.43
WF	2571.03	13.06	6118.63	25.37	9176.99	37.63	14 876.47	57.67
WF vs DF (%)	**9.01**	−16.50	**4.44**	−9.97	**2.48**	−9.20	**4.66**	−4.04
ADW	2624.72	11.42	6178.74	22.73	9099.66	33.37	15 417.30	54.13
ADW vs DF (%)	**7.11**	−1.87	**3.51**	1.47	**3.30**	3.16	**1.20**	2.35

The table reports total distance traveled and number of vehicles used (m). It also shows improvements achieved with wait-first (WF) and advanced dynamic waiting (ADW) compared with drive-first (DF).

Table 3
One-depot experiments with larger number of initially active vehicles

Scheduling	100 requests		300 requests		500 requests		1000 requests	
	Distance	m	Distance	m	Distance	m	Distance	m
DF	2641.11	14.23	6193.09	25.31	9596.96	43.24	14 965.55	55.64
WF	2453.61	13.76	5874.11	25.95	9034.82	41.31	14 776.21	58.11
WF vs DF (%)	**7.10**	3.30	**5.15**	−2.53	**5.86**	4.46	**1.27**	−4.44
ADW	2518.14	13.32	5882.26	24.92	8824.20	35.31	14 539.68	53.69
ADW vs DF (%)	**4.66**	6.39	**5.02**	1.54	**8.05**	18.34	**2.85**	3.50

The table reports total distance traveled and number of vehicles used (m). It also shows improvements achieved with wait-first (WF) and advanced dynamic waiting (ADW) compared with drive-first (DF).

the first set of experiments (Table 1). It produces routes that are up to 8.34% shorter, but the number of vehicles is still high (up to 8.05% more vehicles are used). Compared with WF, the number of vehicles is smaller but the total distance traveled is larger. In addition, DW is outperformed by ADW (in both aspects), which resulted in the decision to drop the DW strategy from further testing. The DF and WF strategies remained in experiments as two simple opposing

Table 4
Second set of instances with slightly larger time windows: 30% 2-h requests, 40% 4-h requests, and 30% 6-h requests, using one depot and a larger number of initially active vehicles

Scheduling	100 requests		300 requests		500 requests		1000 requests	
	Distance	m	Distance	m	Distance	m	Distance	m
DF	2999.38	12.20	7018.95	19.36	10 315.30	23.68	16 754.55	35.32
WF	2827.16	11.42	6782.03	18.24	10 443.81	25.76	17 159.39	37.4
WF vs DF (%)	**5.74**	6.39	**3.38**	5.79	**−1.25**	−8.78	**−2.42**	−5.89
DW	2760.87	10.92	6733.88	17.92	10 245.41	24.68	16 883.68	36.28
DW vs DF (%)	**7.95**	10.49	**4.06**	7.44	**0.68**	−4.22	**−0.77**	−2.72
ADW	2796.51	11.20	6596.49	16.82	10 037.76	23.07	16 357.50	34.2
ADW vs DF (%)	**6.76**	8.20	**6.02**	13.12	**2.69**	2.58	**2.37**	3.17

The table reports total distance traveled and number of vehicles used (m). It also shows improvements achieved by each of the three waiting strategies—wait-first (WF), dynamic waiting (DW), and advanced dynamic waiting (ADW)—compared with drive-first (DF).

strategies. The ADW strategy produces solutions with up to 8.05% shorter total route length compared with DF, and either close to or shorter than the solutions obtained by WF. The number of vehicles is either close to the number of vehicles used in DF, or better.

We have observed that the one-depot experiments were fairer than the no-depot experiments because in the latter case each heuristic chooses its own set of most suitable starting vehicle positions, despite the pre-generated set of points for the starting vehicle positions. The reason for this lies in the difference in the two heuristics when assigning a new request. Since a given request may be inserted into two different routes, it may be infeasible for one heuristic to serve a new request using any active vehicle, while the same new request may be feasibly served with the other heuristic. Consequently, experiments were later performed only on one-depot instances.

A second set of random instances was generated to check whether wider time windows influence the comparative performance of the waiting strategies. The distribution of requests were: 30% 2-h requests, 40% 4-h requests, and 30% 6-h requests. The results given in Table 4, show that the relative performance of the waiting strategies remains unchanged.

6.3. The waiting strategies behaviour when routing is enhanced by tabu search

This section reports on the performance of the waiting strategies when the routing subproblem is solved by a more involved procedure. This ascertains that the superiority of the ADW strategy holds when a more involved algorithm is used for the routing subproblem.

The routing scheme consists of the cheapest insertion procedure followed by an improvement procedure (Section 3). The cheapest insertion procedure is applied every 15 min. All new requests are ordered by their slack time and inserted into routes. The remaining time (12 min) between the runs of the insertion procedure is devoted to improving the current solution by means of a tabu search procedure. Our tabu list contains moves of the following form: 'vehicle v serves request r'. A move is tabu for a random number of iterations, where the number is chosen from interval $[5, 10]$. The scheduling subproblem is solved by the DF and ADW strategies. The dynamic par-

Table 5
Improvements achieved by advanced dynamic waiting (ADW) when routing is solved by the two-phase approach with a tabu search (TS) improvement procedure

Scheduling	100 requests Routing						500 requests Routing					
	CI		TS1		TS2		CI		TS1		TS2	
	Distance	*m*	Distance	*m*	Distance	*m*	Distance	*m*	Distance	*m*	Distance	*m*
DF	2619.54	13.78	2487.31	13.70	2515.32	13.77	9266.08	40.70	8477.20	34.11	8445.77	36.30
ADW	2505.89	13.00	2377.50	13.30	2365.95	12.70	8775.00	35.44	8263.62	34.20	8176.18	33.20
ADW vs DF (%)	**4.34**	5.66	**4.41**	2.92	**5.94**	7.77	**5.30**	12.92	**2.52**	−0.26	**3.19**	8.54

In the TS2 experiments, the tabu search procedure runs 2.5 times longer than in the TS1 experiments.

titioning of a route (see Section 5.3) is again done by using 100 km^2 as the maximal area of the service zone base.

We have solved the one-depot instances with 100 and 500 requests, each with two different simulation speeds. The 100-request instances were tested with simulation speeds of 10 and 4, resulting the tabu search taking 1.2 and 3 min, respectively. The 500-request instances were tested with simulation speeds of 5 and 2, with the tabu search improvement procedure running for 2.4 and 6 min, respectively. The number of vehicles initially active were 20 for 100-request instances and 60 for 500-request instances.

Table 5 compares results averaged over 10 test runs. Boldface entries show average improvements obtained with ADW compared with DF. With respect to total route length, improvements range from 4.34% to 5.30%, when routing was done by the cheapest insertion procedure (CI), and from 2.52% to 4.41% when routing was enhanced by the tabu search improvement procedure (TS1).

The TS2 columns display solution values when the tabu search improvement procedure is applied 2.5 times longer, i.e., when simulation speed is 2.5 times slower. The routes are slightly shorter compared with TS1. In this case, ADW also generated shorter routes (from 3.19% to 5.94%) compared with DF. The relative superiority of ADW therefore seems to remain unaltered if a different procedure is used to solve the routing subproblem.

7. Conclusion

We have presented four waiting strategies for the dynamic pickup and delivery problem with time windows. The two simplest waiting strategies are the drive-first and the wait-first strategies. The wait-first waiting strategy builds better (shorter) routes because of the increased idle of a vehicle before it leaves its current position. During this time more new requests appear allowing the building of better routes. Unfortunately, the wait-first waiting strategy uses a much larger number of vehicles for the same set of requests. In order to explore whether an intermediate balance between waiting and driving could achieve better results, we have developed two new

waiting strategies: the dynamic waiting strategy and the advanced dynamic waiting strategy. Both attempt to better distribute the waiting times along routes in order to facilitate future request insertions. They were built by partitioning of each of the current vehicle routes. An extensive empirical study has shown that, in terms of total route length, the newly introduced waiting strategies outperform the commonly used drive-first waiting strategy. The advanced dynamic waiting strategy seems to be the most efficient.

Acknowledgements

This research was partly supported by the Canadian Natural Sciences and Engineering Research Council under grants RGPIN36809 and OGP0039682. This support is gratefully acknowledged. Thanks are due to a referee for his/her valuable comments.

References

Benyahia, I., Potvin, J.-Y., 1998. Decision support for vehicle dispatching using genetic programming. IEEE Transactions on Systems, Man and Cybernetics—Part A: Systems and Humans 28, 306–314.

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

Cordeau, J.-F., Laporte, G., 2003a. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. Transportation Research B 37, 579–594.

Cordeau, J.-F., Laporte, G., 2003b. The dial-a-ride problem (DARP): Variants, modeling issues and algorithms. 4OR-Quarterly Journal of the Belgian French and Italian Operations Research Societies 1, 89–101.

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. Les cahiers du GERAD G-91-02, École des Hautes Études Commerciales, Montreal.

Desrosiers, J., Dumas, Y., Solomon, M.M., Soumis, F., 1995. Time constrained routing and scheduling. In: Ball, M.O., Magnanti, T.L., Monma, C.L., Nemhauser, G.L. (Eds.), Network Routing. In: Handbooks in Operations Research and Management Science, vol. 8. North-Holland, Amsterdam, pp. 35–139.

Dumas, Y., Soumis, F., Desrosiers, J., 1990. Optimizing the schedule for a fixed vehicle path with convex inconvenience costs. Transportation Science 24, 145–152.

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

Dumas, Y., Desrosiers, J., Gélinas, E., Solomon, M.M., 1995. An optimal algorithm for the traveling salesman problem with time windows. Operations Research 43, 367–371.

Gendreau, M., Potvin, J.-Y., 1998. Dynamic vehicle routing and dispatching. In: Crainic, T.G., Laporte, G. (Eds.), Fleet Management and Logistics. Kluwer, Boston, pp. 115–126.

Gendreau, M., Guertin, F., Potvin, J.-Y., Séguin, R., 1998. Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. Report CRT-98-10, Centre de recherche sur les transports, Montreal.

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

Jaw, J.-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 a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. Annals of Operations Research 60, 193–208.

Mitrović-Minić, S., 2001. The dynamic pickup and delivery problem with time windows. Ph.D. Dissertation, School of Computing Science, Simon Fraser University, Burnaby, Canada.

Mitrović-Minić, S., Krishnamurti, R., Laporte, G., in press. Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows. Transportation Research B. doi:10.1016/j.trb.2003.09.001.

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.

Potvin, J.-Y., Rousseau, J.M., 1992. Constraint-directed search for the advanced request dial-a-ride problem with service quality constraints. In: Balci, O., Shrada, R., Zenios, S.A. (Eds.), Computer Science and Operations Research: New Developments in Their Interfaces. Pergamon Press, Oxford, pp. 457–474.

Powell, W.B., 1991. Optimization models and algorithms: an emerging technology for the motor carrier industry. IEEE Transactions on Vehicular Technology 40, 68–80.

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.

Psaraftis, H.N., 1988. Dynamic vehicle routing problems. In: Golden, B.L., Assad, A.A. (Eds.), Vehicle Routing: Methods and Studies. Studies in Management Science and Systems, vol. 16. North-Holland, Amsterdam, pp. 223–248.

Psaraftis, H.N., 1995. Dynamic vehicle routing: status and prospects. Annals of Operations Research 61, 143–164.

Rochat, Y., Taillard, É.D., 1995. Probabilistic diversification and intensification in local search for vehicle routing. Journal of Heuristics 1, 147–167.

Rosenkrantz, D.J., Stearns, R.E., Lewis II, P.M., 1977. An analysis of several heuristics for the traveling salesman problem. SIAM Journal on Computing 6, 563–581.

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

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

Sexton, T.R., Choi, Y., 1986. Pickup and delivery of partial loads with time windows. American Journal of Mathematical and Management Sciences 6, 369–398.

Shen, Y., Potvin, J.-Y., Rousseau, J.-M., Roy, S., 1995. A computer assistant for vehicle dispatching with learning capabilities. Annals of Operations Research 61, 189–211.

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 Pouting Problem. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia.

Van der Bruggen, L.J.J., Lenstra, J.K., Schuur, P.C., 1993. Variable-depth search for the single-vehicle pickup and delivery problem with time windows. Transportation Science 27, 298–311.

Wolfler Calvo, R., Colorni, A., submitted for publication. An approximation algorithm for the dial-a-ride problem.