

1 A Hybrid Tabu Search and Constraint
2 Programming Algorithm for the Dynamic
3 Dial-a-Ride Problem

4 Gerardo Berbeglia¹
 Jean-François Cordeau²
 Gilbert Laporte¹

¹*Canada Research Chair in Distribution Management, HEC Montréal
3000 chemin de la Côte-Sainte-Catherine, Montréal, Canada H3T 2A7
{gerardo, gilbert}@crt.umontreal.ca*

²*Canada Research Chair in Logistics and Transportation, HEC Montréal
3000 chemin de la Côte-Sainte-Catherine, Montréal, Canada H3T 2A7
cordeau@crt.umontreal.ca*

5 November 7, 2009

6 **Abstract**

7 This paper introduces a hybrid algorithm for the dynamic dial-a-ride problem in which
8 service requests arrive in real time. The hybrid algorithm combines an exact constraint
9 programming algorithm and a tabu search heuristic. An important component of the tabu
10 search heuristic consists of three scheduling procedures which are executed sequentially.
11 Experiments show that the constraint programming algorithm is sometimes able to accept
12 or reject incoming requests, and that the hybrid method outperforms each of the two
13 algorithms when they are executed alone.

14 **Keywords:** dial-a-ride problem, dynamic, constraint programming, tabu search, scheduling

1 Introduction

In the Dial-a-Ride Problem (DARP), a fleet of vehicles must serve transportation requests between given origins and destinations. The main application of the DARP arises in door-to-door transportation services offered to elderly and handicapped people in many cities. Case studies have been described for the cities of Toronto (Desrosiers et al., 1986), Berlin (Borndörfer et al., 1997), Bologna (Toth and Vigo, 1996), Copenhagen (Madsen, Ravn, and Rygaard, 1995), and Brussels (Rekiek et al., 2006). The minimization of user inconvenience often has to be balanced with operation costs since these objectives usually conflict. User inconvenience is taken into consideration, for instance, by assigning time windows to pickups or deliveries and by imposing a maximum ride time for each user.

An important dimension of the DARP relates to the availability of information. In the *static* DARP, all requests are assumed to be known *a priori*, before routes are constructed. A solution therefore consists of a static output specifying the routing and scheduling information. In the *dynamic* DARP, some or all requests for service are received in real time, while routing operations take place. Instead of a static output, a solution to a dynamic DARP consists of a solution strategy specifying which routing and scheduling actions should be performed in the light of newly received service requests and of the current state of the system.

Over the past 30 years, most studies on the DARP have focused on the static version (see the recent survey of Cordeau and Laporte (2007)). In this article we develop a hybrid algorithm for the dynamic DARP, which has been less studied, but has recently attracted some interest. One of the first studies on the dynamic DARP was carried out by Psaraftis (1980) who considered the single vehicle case. The author developed an exact $O(n^23^n)$ dynamic programming algorithm for the static DARP. Whenever a new request arrives, the static instance is updated and reoptimized by fixing the partial route already performed. Madsen et al. (1995) have presented an insertion based algorithm for a real-life multi-vehicle dynamic DARP for the transportation of elderly and handicapped people in Copenhagen. An algorithm for demand-responsive passenger services such as taxis, including time window restrictions for the dynamic requests, capacity constraints and booking cancellations has been developed by Horn (2002). A parallel algorithm for the Dynamic DARP, by Attanasio et al. (2004), works as follows. When a new request arrives, each of the parallel threads inserts the request randomly in the current solution and runs a tabu search algorithm to obtain a feasible solution. Another algorithm for a

48 dynamic DARP was developed by Coslovich et al. (2006). In the problem considered by
49 these authors, a driver may unexpectedly receive a trip demand by a person located at a
50 stop and must decide quickly whether to accept it or not. An efficient insertion algorithm
51 attempts to insert incoming requests in at least one of the solutions in the repository, and
52 a request is accepted only if the insertion algorithm succeeds. A two-phase algorithm for
53 solving a complex dynamic DARP arising in the transportation of patients in hospitals
54 was proposed by Beaudry et al. (2010). In the first phase, a fast insertion scheme is
55 used, and the second phase involves a tabu search which attempts to improve the current
56 solution. Finally, Xiang et al. (2008) have studied a sophisticated dynamic DARP in
57 which travel and service times have a stochastic component. New requests are inserted
58 into the established routes by means of a local search procedure based on simple inter-trip
59 moves. See Berbeglia et al. (2010) for a recent survey of the dynamic DARP and of other
60 dynamic pickup and delivery problems.

61 The dynamic DARP studied in this article can be described as follows. Let $G = (V, A)$
62 be a complete and directed graph with vertex set $V = \{0\} \cup R$, where vertex 0 is the depot,
63 and R represent the customer vertices. The set R is partitioned into $R^+ = \{1, \dots, n\}$
64 (pickup vertices) and $R^- = \{n+1, \dots, 2n\}$ (delivery vertices). Let $H = \{1, \dots, n\}$ be the
65 set of requests, and let T be the end of the planning horizon. Request i has an associated
66 pickup vertex $i^+ = i \in R^+$, a delivery vertex $i^- = n + i \in R^-$, and a time t_i at which
67 it is received. With each vertex $i \in V$ are associated a time window $[e_i, l_i]$, a service
68 duration D_i , and a load q_i (with $D_0 = 0$, $q_0 = 0$ and $q_{n+j} = -q_j$ for $j = 1, \dots, n$). If
69 request i is outbound (i.e., from home to a destination) the time window associated to
70 the pickup vertex i is $[0, T]$, whereas if it is inbound the time window associated to the
71 delivery vertex $n + i$ is $[0, T]$. The delivery vertex of an outbound request and the pickup
72 vertex of an inbound request are called *critical*. The maximum allowed ride time of a
73 user, defined as the difference between the arrival time at destination and the departure
74 time at origin, is L . Each arc (i, j) has a non-negative routing cost c_{ij} and a routing time
75 T_{ij} both satisfying the triangular inequality.

76 A *route* is a circuit over some vertices, starting and finishing at the depot. A request
77 is said to be *served* when it is part of a route. The set of routes must satisfy the following
78 constraints:

79 (i) the pickup and delivery vertices of any request are either both in the same route or
80 none of them are;

- 81 (ii) all requests known at the beginning of the time horizon must be served;
- 82 (iii) the pickup vertex of a request must precede its delivery vertex;
- 83 (iv) the load of any vehicle may never exceed the vehicle maximum load capacity, denoted
- 84 by Q ;
- 85 (v) the ride time of each served request cannot exceed L ;
- 86 (vi) the pickup and delivery of each served request are performed in their respective time
- 87 windows.

88 Our solution strategy for the dynamic DARP is as follows. An initial solution to serve
89 the known requests is obtained by first assigning every request to a randomly selected
90 vehicle and inserting the pickup and delivery vertices of the request at end of the partially
91 constructed routes. Then, using this solution as a seed, the tabu search procedure finds
92 a feasible solution. As time evolves, service requests are received and a quick decision on
93 whether to accept or reject each of them has to be made. This decision is final, meaning
94 that no rejected request can later be accepted and all accepted requests must be served.
95 The algorithm must

- 96 (i) decide whether or not to accept an incoming request;
- 97 (ii) serve the accepted requests in such a way that at the end of the time horizon all
- 98 routes respect the properties just described.

99 The hybrid algorithm we have developed consists of a *tabu search* (TS) heuristic pro-
100 cedure combined with an exact *constraint programming* (CP) algorithm which is able to
101 determine whether a given instance of the DARP is feasible or not. The role of the TS
102 heuristic is to continually optimize the current solution and to try and insert incoming
103 requests into the current solution. When an incoming request is received the constraint
104 programming algorithm is also executed, in parallel to the tabu procedure, in the hope
105 of finding a feasible solution or to prove that no feasible solution compatible with the
106 past actions exists. The incoming request is accepted only when either the TS or the CP
107 algorithm identifies a feasible solution. The request is rejected when the CP algorithm
108 proves the infeasibility or after a preset time limit, generally of one or two minutes.

109 As a rule, the TS algorithm can easily insert a new request in the current solution
110 when it is not too tightly constrained. In contrast, CP is rather effective in proving
111 that no insertion is feasible in very tight scenarios. Our goal is to develop an algorithm
112 that combines the advantages of these two solution methodologies. There are two main
113 benefits in applying CP in conjunction with TS. First, CP is sometimes able to find a

114 feasible solution when the TS cannot or takes longer to do so. Second, in many instances
115 when the TS has not found a solution, it can actually prove that no feasible insertion
116 exists. From a quality of service point of view, proving that a given request cannot be
117 inserted is a more convincing statement than simply stating that no solution has been
118 found.

119 The remainder of this article is organized as follows. In Section 2 we present the
120 main components of the TS heuristic. In Section 3, we give a brief description of the
121 constraint programming paradigm and we present a model of the DARP as a constraint
122 satisfaction problem. The three scheduling algorithms used by the TS heuristic are then
123 described in Section 4. The main scheme of the proposed hybrid algorithm is presented
124 in Section 5, and computational results are given in Section 6. We close this article with
125 some conclusions in Section 7.

126 **2 Tabu search**

127 Tabu search is a metaheuristic that combines local search with a memory scheme in
128 order to avoid visiting the same solutions repetitively (Glover and Laguna, 1997). It
129 has been proved to be very successful in vehicle routing (see, e.g., Cordeau et al. 2001
130 and Gendreau et al. 1994). In this section we present the basic TS concepts applied to
131 our algorithm for the dynamic DARP. The algorithm we have developed is based on the
132 TS procedure for the static DARP developed by Cordeau and Laporte (2003). We will
133 provide a summary of the main features and dynamic aspects of the procedure. We refer
134 the reader to the original article for a more extensive description.

135 One of the important characteristics of the TS algorithm is the allowance of infeasible
136 solutions during the search. A solution is represented by a set of m routes such that

- 137 *(i)* each routes starts and ends at the depot;
- 138 *(ii)* each accepted request is assigned to exactly one route;
- 139 *(iii)* for each accepted request, the pickup vertex precedes the delivery vertex.

140 Therefore, an intermediate solution may violate the ride time constraints and the
141 time window constraints associated to the requests, as well as the capacity constraints
142 associated to the vehicles.

143 2.1 Relaxation mechanism and objective function

144 Let $r = (i_0, \dots, i_k)$ be a route of a given solution s , and let $c(r)$, $q(r)$, $w(r)$ and $t(r)$
145 denote the routing cost, load violation, time window violation, and ride time violation of
146 route r , respectively. Formally, $c(r) = \sum_{u=0}^{k-1} c_{i_u, i_{u+1}}$ and $q(r) = \sum_{u=1}^k (q_{i_u} - Q)^+$, where
147 $x^+ = \max\{0, x\}$. The time window violation is defined as $w(r) = \sum_{u=0}^k (BT_{i_u} - l_{i_u})^+$,
148 where BT_i specifies the start of service at vertex i . Finally, the ride time violation is given
149 by $t(r) = \sum_{u=1}^k (L_{i_u} - L)^+$, where $L_i = 0$ if i is a pickup vertex and is equal to the ride
150 time of the request associated to vertex i in case i is a delivery vertex. The total routing
151 cost of a given solution s with routes $\{r_1, \dots, r_m\}$ is $c(s) = \sum_{u=1}^m c(r_u)$. Similarly, the
152 total load violation, total time window violation and total ride time violation of solution
153 s are equal to the sum of their respective violations for each route.

154 The total cost of a solution s is equal to $f(s) = c(s) + \alpha q(s) + \gamma w(s) + \tau t(s)$. Initially,
155 The parameters α , γ , τ are set equal to 1. They are dynamically adjusted after each
156 iteration as follows. If the current solution respects the load constraint, the value α is
157 divided by $1 + \delta$; otherwise it is multiplied by $1 + \delta$ where δ , is a uniformly distributed ran-
158 dom number between 0 and 0.5, and is updated every 10 iterations. The same procedure
159 applies to γ and τ , regarding the time window and ride time violations, respectively.

160 2.2 Neighbourhood definition and evaluation

161 A request i is said to be *fixed* in a solution s and at current time t if the request cannot
162 be moved to another route. This happens when either the pickup vertex has already been
163 served at time t or the vehicle has already left the vertex preceding the pickup vertex of
164 i , since diversion of vehicles is not allowed.

165 A solution s is characterized by the set $U(s) = \{(i, k) : \text{request } i \text{ is assigned to vehicle}$
166 $k\}$. The neighbourhood $N(s, t)$ of a solution s at time t consists of all solutions that can
167 be reached by removing an attribute (i, k) from $U(s)$ whose request is not fixed at time
168 t , and replacing it with a new attribute (i, k') with $k' \neq k$. When a request is removed
169 from a route, the order of the remaining vertices in the route is unchanged. When the
170 insertion of a request into a route r takes place, the order of the other vertices in r remains
171 unchanged and the pickup and delivery are located in order to minimize the total cost
172 function described in Section 2.4.

173 After the removal or insertion of a request, the cost of the route must be updated.

174 Computing the new routing cost as well as the capacity violations can be achieved easily
 175 in linear time. More complex computations are needed to update the time window and
 176 ride time violations. To compute these two violations, a route scheduling algorithm is
 177 required. We have developed three scheduling algorithms which are described in Section
 178 4.

179 **2.3 Route optimization**

180 Intra-route optimization is performed every κ iterations by sequentially removing one
 181 vertex at a time and reinserting it in a position that minimizes $f(s)$. As an additional
 182 search intensification, this procedure is also performed whenever a new incumbent is
 183 identified. In our implementation κ was set to 10.

184 **2.4 Tabu control, aspiration, and diversification**

185 To avoid repeating solutions, a request i removed from a route r cannot be inserted
 186 back into this route for the next θ iterations. The value of θ is a random number uniformly
 187 distributed between 0 and $7.5 \log_{10} n$, and updated every 10 iterations. As an aspiration
 188 mechanism, the tabu prohibition is disabled when the reinsertion would produce a solution
 189 with smaller cost than the best known solution having request i in route r .

190 The tabu search algorithm evaluates a solution s using the objective function $f(s) +$
 191 $p(s)$, where $p(s)$ is used to diversify the search and penalizes a neighbour solution s' of s ,
 192 only when $f(s') > f(s)$. This penalty is proportional to the frequency of addition of its
 193 distinguishing attributes and of a scaling factor. More precisely, suppose that (i, k) is the
 194 attribute that must be added to the current solution s in order to obtain the new solution
 195 \bar{s} , and let ρ_{ik} denote the number of times attribute (i, k) has been added to the solution
 196 during the search. The penalty term used to evaluate solution \bar{s} is then

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

197 where μ is a random number uniformly distributed between 0 and 0.015, and it is also
 198 updated every 10 iterations.

3 Constraint programming

We now provide a brief introduction to constraint programming and we present a model of the DARP as a constraint satisfaction problem. Constraint programming is a programming paradigm based on reasoning and search techniques, which is applied to the solution of combinatorial problems. It originally emerged from the artificial intelligence community in the 1970s when the concept of a constraint satisfaction problem was formulated. In the 1980s, logic programming researchers have developed several constraint solving algorithms which have led to the development of *constraint logic programming*. This paradigm extends the *logic programming* concept through the use of constraints. Constraint programming then appeared in the 1990s through a transformation of constraint logic programming, in which a constraint orientated view and more sophisticated propagation techniques were developed. For an introduction to these concepts, see Van Hentenryck (1989).

In CP, a problem is modeled as a *Constraint Satisfaction Problem* (CSP). Informally, a CSP consists of a set of variables and a set of restrictions, called *constraints*, over the variables. A constraint on a sequence of variables is a relation on the variable domains. It states which combinations of values from the variable domains are permitted and which of them are not. Once we have modeled a problem as a CSP, we proceed to solve it. Constraint programming solves a model using inference algorithms to reduce the search space, as well as search methods. The inference algorithms, called *constraint propagation algorithms* or *filtering algorithms*, try to simplify the problem by removing values from variable domains while preserving the same set of solutions. Search methods generally consist of backtracking or branch-and-bound combined with constraint propagation. Constraint programming has been successfully applied to scheduling, planning, molecular biology, finance, and numerical analysis. These and other applications of CP are surveyed in van Hoes and Katriel (2006).

We now give a formulation of the static DARP as a constraint satisfaction problem based on successor variables presented by Berbeglia et al. (2009). In Section 5 we show how to use this model for the dynamic version of the problem. We first extend the graph G as follows. Vertex 0, corresponding to the depot, is replaced by the depot set $V = V^+ \cup V^-$ with $|V^+| = |V^-| = m$. The new graph G has $|V| + |R| = 2m + 2n$ vertices. Vehicle $k \in K = \{1, \dots, m\}$ is represented by vertices $start(k) \in V^+$ (starting depot) and $end(k) \in V^-$ (ending depot). Under this transformation, the route of vehicle

232 k is represented by the circuit $(start(k)) : S_i : (end(k))$, where S_k is a sequence, possibly
 233 empty, of client vertices.

234 We list the variables for the constraint programming formulation. For each vertex
 235 $i \in V \cup R$,

236 (i) $s[i] \in V \cup R$ identifies the direct successor of vertex i ;

237 (ii) $\ell[i] \in [0, Q]$ states the vehicle load just after performing the pickup or delivery at
 238 vertex i ;

239 (iii) $v[i] \in K$ indicates the vehicle serving vertex i ;

240 (iv) $t[i] \in [e_i, l_i]$ represents the time at which vertex i is served.

241 The constraints for the DARP are the following.

242 Basic constraints:

243 (i) For each vehicle $j \in K$, $s[end(j)] = start(j)$;

244 (ii) for each vehicle $j \in K$, $v[end(j)] = v[start(j)] = j$;

245 (iii) *allDifferent*(s);

246 (iv) for each request $i \in H$, $v[i^+] = v[i^-]$;

247 (v) for each vertex i , $v[i] = v[s[i]]$;

248

249 Precedence and time windows constraints:

250 (vi) for each request $i \in H$, $t[i^+] \leq t[i^-] - T_{i^+,i^-} - D_{i^+}$;

251 (vii) for each vertex $j \in V^+ \cup R$, $t[j] \leq t[s[j]] - T_{j,s[j]} - D_j$;

252

253 Capacity constraints:

254 (viii) for each vehicle i , $\ell[start(i)] = 0$;

255 (ix) for each client vertex $j \in R$, $\ell = [s[j]] = \ell[j] + q_{s[j]}$ and $\ell[j] \leq Q$;

256

257 Ride time constraints:

258 (x) for each request $i \in H$, $t[i^-] - (t[i^+] + D_{i^+}) \leq L$.

259

260 This CSP is solved with the constraint programming algorithm proposed by Berbeglia
 261 et al. (2009), which contains filtering methods, symmetry breaking strategies, and variable
 262 fixing techniques for improving the efficiency.

4 Scheduling

An important aspect of an algorithm for the dynamic DARP consists in deciding at which time the vehicles arrive, start service, and depart from each vertex. As will be shown in Section 6, the scheduling strategy alone has a considerable impact on algorithm performance.

In this section we present three scheduling algorithms which we call *basic scheduling*, *lazy scheduling* and *eager scheduling*. Given a fixed route r and a current time t , these algorithms output the arrival time, start of service time and departure time for each vertex in r , without modifying the actions taken before time t .

Consider a vehicle route $r = (0, \dots, q)$ with 0 and q being the depot vertex. We define the following scheduling variables for each vertex $j = 0, \dots, q$:

AT_j : the arrival time at vertex j ;

BT_j : the start of service at vertex j ;

DT_j : the departure time at vertex j ;

WT_j : the waiting time at vertex j before service ($WT_j = BT_j - AT_j$).

For clarity of exposition, we assume that the service duration D_j for each vertex is equal to zero. The algorithms presented in this section can easily be adapted to the case where the service duration has a positive value. At vertex 0, which represents the depot at the start of the route, $BT_0 = DT_0$, $AT_0 = 0$, and $e_0 = e_q = 0$. For the vertex q which is the depot at the end of the route, $A_q = B_q = DT_q$ represents the arrival time.

A *schedule* for route r consists of an assignment of values to the variables AT_{j+1} , BT_j and DT_j for $0 \leq j \leq q - 1$. It is assumed that $e_j \leq BT_j$ for $0 \leq j \leq q$. Observe that a schedule must also satisfy

$$AT_{j+1} = DT_j + T_{j,j+1}, \text{ for all } 0 \leq j \leq q - 1 \quad (1)$$

and

$$DT_j \geq BT_j, \text{ for all } 0 \leq j \leq q. \quad (2)$$

Thus, to define a schedule it is sufficient to fix either BT_0, \dots, BT_{q-1} and AT_1, \dots, AT_q , or BT_0, \dots, BT_{q-1} and DT_0, \dots, DT_{q-1} .

A schedule is *feasible* if

(i) $e_j \leq BT_j \leq l_j$ for $0 \leq j \leq q$ and,

(ii) given any request i such that the pickup vertex and the delivery vertex are in r , i.e.,

$i^+ \in r$ and $i^- \in r$, then $BT_{i^-} - BT_{i^+} \leq L$.

293 Assume we are given a time value $t < B_{q-1}$, a route r , and a schedule for the route.
 294 We are interested in modifying the schedule for route r without altering the arrival, the
 295 start of service and the departure time of any vertex that was served before time t . Three
 296 cases can be distinguished:

297 (i) If the vehicle has not yet started the route (i.e., $t < BT_0$), then the departure time at
 298 the depot can be modified but cannot occur before t .

299 (ii) If the vehicle is moving towards a vertex (i.e., $DT_j \leq t < AT_{j+1}$ for some $0 \leq j \leq$
 300 $q - 1$), then the arrival time at vertex $j + 1$ cannot be modified.

301 (iii) If the vehicle is waiting to serve a customer (i.e., $AT_j \leq t < BT_j$ for some $1 \leq j \leq$
 302 $q - 1$), then the start of service at the vertex can be modified with the restriction that
 303 the new time x for the start of service must satisfy $t \leq x$.

304 Let $k + 1$ (with $0 \leq k + 1 \leq q - 1$) be the first vertex at which it is possible to modify
 305 the start of service (or the departure time when $k + 1$ represents the depot), i.e. BT_{k+1} .
 306 Formally, $k = \max \{ \min \{j \in \{1, \dots, q\} : BT_{j+1} > t\} \cup \{-1\}$. Therefore, BT_j , DT_j , and
 307 AT_{j+1} cannot be modified for all $0 \leq j \leq k$.

308 4.1 Basic scheduling

309 The basic scheduling procedure is described by Algorithm 1.

Algorithm 1 Basic scheduling algorithm

Input: A route $r = (0, \dots, q)$, a time t , a number $k \in \{-1, \dots, q - 2\}$ and a schedule for route
 r .

$$BT_{k+1} = \max \{e_{k+1}, AT_{k+1}, t\}$$

$$DT_{k+1} = BT_{k+1}$$

for $j = k + 2$ to $q - 1$ **do**

$$AT_j = BT_{j-1} + T_{j-1,j}$$

$$BT_j = \max \{e_j, AT_j\}$$

$$DT_j = BT_j$$

end for

$$AT_q = BT_{q-1} + T_{q-1,q}$$

310

311 The resulting schedule, which may not always be feasible, has the following properties:

312 (i) It minimizes the time window violation for any vertex $j = k + 1, \dots, q$ defined as
313 $(BT_j - l_j)^+$, and thus, it also minimizes the total violation $\sum_{j=k+1}^q (BT_j - l_j)^+$.

314 (ii) It minimizes the start of service time B_j of any vertex j with $k + 1 \leq j \leq q$.

315 The algorithm serves each vertex as early as possible but always ensures that service
316 at vertex j cannot begin before e_j . As stated in Cordeau and Laporte (2003), the sched-
317 ule produced by the basic scheduling algorithm may not be feasible, even though there
318 actually exists a feasible schedule. This is because it may sometimes be worthwhile to
319 delay the service of a vertex in order to reduce the ride time of the associated request.
320 The algorithm presented in the following section, called lazy scheduling, overcomes this
321 problem.

322 4.2 Lazy scheduling algorithm

323 We present here a procedure called the lazy scheduling algorithm, which is the dynamic
324 version of an algorithm for the static DARP proposed by Cordeau and Laporte (2003).
325 The algorithm transforms a schedule into another schedule called *lazy*, which minimizes
326 the ride time violation of every request without increasing the time window violation of
327 any vertex. The idea behind the lazy scheduling algorithm is to delay as much as possible
328 the time BT_j at which service starts at vertex j , starting with vertex $k + 1$ and finishing
329 with vertex $q - 1$. This is the reason why the algorithm is called *lazy*. The maximum
330 possible delay at any vertex $j \in \{k + 1, \dots, q - 1\}$ will be constrained so that there is
331 no increase in the time window violation or in the ride time violation of any vertex of
332 the route. Since the delay of the pickup vertex of every request precedes the delay of the
333 delivery vertex, the procedure will sequentially minimize the ride time violation of each
334 request.

335 When the input schedule is generated by the basic scheduling algorithm, the schedule
336 produced by the lazy algorithm will be infeasible if and only if no feasible schedule ac-
337 tually exists. Thus, by applying the lazy scheduling algorithm after the basic scheduling
338 algorithm we can determine whether or not a given route possesses a feasible schedule.

339 The ride time of a request i is defined as $P_i = BT_{i-} - BT_{i+}$. We now derive the
340 algorithm for producing the lazy schedule. We assume in this derivation that the variables
341 AT_j , BT_j , WT_j and DT_j contain the scheduling values of the input schedule and we
342 denote by BT'_j the new departure time at vertex j . We wish to determine the latest
343 time at which service at vertex $k + 1$ can start without increasing the time window

344 violation of any vertex and without increasing the ride time violation of any request. Let
 345 $J_k^- = \{i^- \in \{k+1, \dots, q\} \text{ be such that } i^+ \in \{1, \dots, k\}\}$. In order not to increase the ride
 346 time violation of a request i with $i^- \in J_k^-$, the start of service at vertex $k+1$ cannot be
 347 performed later than

$$AT_{k+1} + \sum_{j=k+1}^{i^-} WT_j + (L - P_i)^+.$$

348 Thus,

$$BT'_{k+1} \leq AT_{k+1} + \min_{i: i^- \in J_k^-} \left\{ \sum_{j=k+1}^{i^-} WT_j + (L - P_i)^+ \right\}.$$

349 For any vertex $j \in \{k+1, \dots, q\}$ the start of the service at vertex $k+1$ cannot be
 350 later than

$$AT_{k+1} + \sum_{u=k+1}^j WT_u + (l_j - BT_j)^+.$$

351 Thus,

$$BT'_{k+1} \leq AT_{k+1} + \min_{j \in \{k+1, \dots, q\}} \left\{ \sum_{u=k+1}^j WT_u + (l_j - BT_j)^+ \right\}.$$

352 Therefore, the latest time at which it is possible to serve vertex $k+1$ without increasing
 353 the time window violation of any vertex and without increasing the ride time violation of
 354 any request i with $i^- \in J_k^-$ is

$$BT'_{k+1} = AT_{k+1} + \min \left\{ \min_{j \in \{k+1, \dots, q\}} \left\{ \sum_{u=k+1}^j WT_u + (l_j - BT_j)^+ \right\}, \right. \\ \left. \min_{i: i^- \in J_k^-} \left\{ \sum_{u=k+1}^{i^-} WT_u + (L - P_i)^+ \right\} \right\}.$$

355 The lazy scheduling algorithm is presented in Algorithm 2.
 356

357 Once B_{k+1} is computed, we set $DT_{k+1} = B_{k+1}$ and the arrival time at vertex $k+2$
 358 (A_{k+2}) is obtained by Algorithm 1. This delay on the service at vertex $k+1$ propagates
 359 along all the following vertices as shown in lines 7 to 11. Once the effects of delaying
 360 vertex $k+1$ have been computed, the algorithm proceeds with delaying vertices $k+2$ up
 361 to $q-1$.

362 A potential problem of the lazy schedule is that it may become hard to insert a new
 363 request into the route. This is because the vehicle serves the vertices as late as possible,

Algorithm 2 Lazy scheduling algorithm

```
1: Input: A route  $r = (0, \dots, q)$ , a number  $k \in \{-1, \dots, q - 2\}$  and a schedule for route  $r$ .
2: for  $h = k + 1$  to  $q - 1$  do
3:    $BT_h = AT_h + \min\{\min_{j \in \{h, \dots, q\}} \{\sum_{u=h}^j WT_u + (l_j - BT_j)^+\}, \min_{i: i^- \in J_{h-1}^-} \{\sum_{u=h}^{i^-} WT_u + (L - P_i)^+\}\}$ 
4:    $DT_h = BT_h$ 
5:    $AT_{h+1} = DT_h + T_{h,h+1}$ 
6:    $WT_h = BT_h - AT_h$ 
7:   for  $f = h + 1$  to  $q - 1$  do
8:      $BT_f = \max\{e_f, AT_f\}$ 
9:      $AT_{f+1} = B_f + T_{f,f+1}$ 
10:     $WT_f = BT_f - AT_f$ 
11:   end for
12:   for each request  $i$  such that  $\{i^+, i^-\} \subseteq \{0, \dots, q\}$  do
13:      $P_i = BT_{i^-} - BT_{i^+}$ 
14:   end for
15: end for
```

364 which means that when a new request arrives, there may be not sufficient available slack
365 time to insert it. In contrast, it may be easier to perform the insertion if the vertices
366 were served earlier. In the next section we present the eager scheduling algorithm which
367 produces a schedule in which each vertex is served as early as possible without increasing
368 the time window and ride time violations.

369 4.3 Eager scheduling algorithm

370 The eager scheduling algorithm transforms a given schedule into another one that
371 minimizes the start of service time B_i of every vertex i without increasing the time win-
372 dow violation of any vertex and without increasing the ride time violation of any request.
373 Unlike the lazy scheduling algorithm, this procedure does not minimize ride time viola-
374 tions, but only ensures that they will not be increased. Thus, to obtain a schedule that
375 first minimizes the time window violations, second the ride time violations, and third the
376 service starting time of each vertex, we can apply first the basic scheduling algorithm,
377 then the lazy scheduling algorithm to its output, and finally use this schedule as an input
378 to the eager scheduling algorithm.

379 Yuen et al. (2009) have developed a scheduling algorithm called Drive First (DF) for
380 the dynamic DARP in which the vehicles serve vertices as soon as possible. However, they
381 have modeled the problem in such a way that the maximum ride time restrictions can be
382 expressed through the time window constraints. This is not possible in our definition of
383 the DARP. As a result, their algorithm for minimizing the start of service time at each
384 vertex is much simpler than the one we present here and is equivalent to the methods
385 employed for the solution of pickup and delivery problems in which there are no ride time
386 constraints (see, e.g., Mitrović-Minić and Laporte, 2004).

387 The idea of the algorithm is the following. Starting from the last vertex of the route,
388 we compute the minimum time required to arrive at that vertex. Once this value is
389 determined, we move to the previous vertex, until we finish with vertex $k + 1$. Let $\tilde{A}T_{j+1}$
390 and $\tilde{B}T_j$ be the arrival time at vertex j and the start time of the service at vertex j in
391 the basic schedule for $k + 1 \leq j \leq q - 1$, respectively. The sequences $(\tilde{A}T_{k+2}, \dots, \tilde{A}T_q)$
392 and $(\tilde{B}T_{k+1}, \dots, \tilde{B}T_{q-1})$ can be computed using Algorithm 1. Consider again the route
393 $r = (0, \dots, q)$ and a schedule for r . The amount of time by which it is possible to antepone
394 the arrival at h with the only restriction of serving each vertex j with $0 \leq j \leq h - 1$ not
395 before e_j , is equal to $AT_h - \tilde{A}_h$. Assume that $\{BT_h, \dots, BT_{q-1}\}$ and $\{AT_{h+1}, \dots, AT_q\}$ are
396 fixed, i.e., the service time of vertices h up to $q - 1$ cannot be changed (with $0 \leq h \leq q - 1$).
397 Assume also that the starting time of the vertices in $\{0, \dots, h - 1\}$ cannot be increased.
398 This is coherent with our objective of minimizing the starting time BT_j of every vertex j .
399 Therefore, at time t , in order not to increase the ride time of a request i with $i^- \in J_{h-1}^-$,
400 the arrival time at vertex h cannot be earlier than

$$AT_h - \left((L - P_i)^+ + \sum_{j=\lambda}^{h-1} (BT_j - \max\{e_j, AT_j, t\}) \right), \quad (3)$$

401 where $\lambda = \max\{i^+ + 1, k + 1\}$. The validity of this inequality can be explained as follows.
402 The ride time of request i is measured by $P_i = BT_{i^-} - BT_{i^+}$. It is assumed that BT_{i^-}
403 cannot be modified and that BT_{i^+} cannot be increased. Thus, without increasing the
404 ride time P_i , the only feasible time margin for arrival at vertex h is equal to the sum
405 of the waiting times which we can potentially reduce. These are the waiting times over
406 the vertices $\{\lambda, \dots, h - 1\}$, whose sum is equal to $\sum_{j=\lambda}^{h-1} BT_j - \max\{e_j, AT_j, t\}$. This is an
407 upper bound on the total time that it is possible to gain by serving vertices $\{\lambda, \dots, h - 1\}$
408 earlier. Since it is sometimes possible to increase the ride time, we add the term $(L - P_i)^+$
409 which states by how much the ride time can be increased without producing a violation.

410 Therefore, the following inequality must hold

$$AT'_h \geq AT_h - \min_{i:i^- \in J_{h-1}^-} \left\{ (L - P_i)^+ + \sum_{j=\lambda}^{h-1} (BT_j - \max\{e_j, AT_j, t\}) \right\}.$$

411 The eager scheduling procedure is described in Algorithm 3. Proceeding backwards
 412 from vertex $h = q$ to vertex $k + 2$, the algorithm computes the earliest arrival time at
 413 vertex h using (3) and then sets the departure time and service time at the previous vertex
 414 in lines 5 and 6. This advance in the departure at vertex $h - 1$ propagates backwards
 415 into an update of the arrival and start of service of the vertices between $h - 1$ and $k + 1$.
 416 Basically, the new arrival time at a vertex j is equal to the minimum between the previous
 417 arrival time and the new start of service time. At the end of each step in the main cycle
 418 which iterates on h , defined between lines 2 to 16, the algorithm has computed the final
 419 value of the arrival time at vertex h and the start of service at vertex $h - 1$.

Algorithm 3 Eager scheduling algorithm

- 1: Input: A route $r = (0, \dots, q)$, a number $k \in \{-1, \dots, q - 2\}$ and a schedule for route r .
 - 2: **for** $h = q$ to $k + 2$ **do**
 - 3: $\Delta_h = \min \{ AT_h - \tilde{A}T_h, \min_{i:i^- \in J_{h-1}^-} \{ (L - P_i)^+ + \sum_{j=\max\{i^++1, k+1\}}^{h-1} (B_j - \max\{e_j, AT_j, t\}) \} \}$
 - 4: $AT_h = AT_h - \Delta_h$
 - 5: $BT_{h-1} = AT_h - T_{h-1, h}$
 - 6: $DT_{h-1} = BT_{h-1}$
 - 7: $j = h - 1$
 - 8: **while** $j \geq k + 2$ **do**
 - 9: $AT_j = \min \{ BT_j, AT_j \}$
 - 10: $BT_{j-1} = AT_j - T_{j-1, j}$
 - 11: $j = j - 1$
 - 12: **end while**
 - 13: **for** each request i such that $\{i^+, i^-\} \subseteq \{0, \dots, q\}$ **do**
 - 14: $P_i = BT_{i^-} - BT_{i^+}$
 - 15: **end for**
 - 16: **end for**
-

420

4.3.1 Delaying the departure

In the three scheduling algorithms just described, the vehicle departs from a vertex immediately after service takes place, i.e. $DT_j = BT_j$ for all $j = k + 1, \dots, q - 1$. It is possible, however, to modify this by applying equations (4) and (5) to the schedule produced by any of the three scheduling algorithms:

$$DT_j = BT_{j+1} - T_{j,j+1}, \text{ for all } k + 1 \leq j \leq q - 1 \quad (4)$$

$$AT_j = BT_j, \text{ for all } k + 1 \leq j \leq q - 1. \quad (5)$$

This modification does not change the properties of the output schedules of any of the three algorithms. The advantage in delaying the departure time is that this creates a waiting period which allows the TS and CP algorithms to change the next vertex to visit and thus increase the space in which to find a feasible solution.

5 A hybrid algorithm

We now present the most important aspects of the hybrid algorithm combining the TS heuristic described in Sections 2 and 4 and an exact CP algorithm proposed by Berbeglia et al. (2009). We recall that given an instance I of the static DARP, the constraint programming algorithm returns either a feasible solution for I or proves that none exists. Our purpose, however, is slightly different. We wish to determine whether it is possible or not, in a dynamic context, to accept and satisfy an incoming request by updating the current solution. We explain below how the CP algorithm was adapted for this purpose.

When a new request is received at time t , a new instance I of the DARP is created, containing all the static and accepted requests up to time t , as well as the new request. Naturally, if the CP algorithm is executed with instance I as input and no additional constraints, it may find a feasible solution whose routing and scheduling actions up to time t do not correspond to the ones that were actually implemented. This difficulty is resolved through the introduction of additional constraints in the constraint programming model, which state that the solution must respect the partial routes followed up to time t .

Observe that in the CP model there are no variables to represent the arrival and departure times at each of the vertices. However, one must take the departure times of

449 the current solution into account in order to properly fix the CP variables and thus avoid
 450 inconsistencies. The pseudo-code of this procedure is given in Algorithm 4. It considers
 451 one route at a time and can be divided into two parts. In the *while* cycle (i.e., lines 6
 452 to 17), it either sets a lower bound or fixes the service time for the relevant vertices. In
 453 the *for* cycle (i.e., lines 18 to 20), it fixes the successor variables up to time t in the given
 solution.

Algorithm 4 Procedure for fixing a partial solution to the CP algorithm

```

1: Input: DARP Instance with new request  $I$ , current solution  $s$  (without the new request)
   and actual time  $t$ .
2: Load the CP model for instance  $I$ 
3: for each of route  $r = (i_0, \dots, i_k)$  of solution  $s$  do
4:    $isFixed = 1$ 
5:    $j = 0$ 
6:   while  $j \leq k$  AND  $isFixed = 1$  do
7:     if  $BT_{i_j} \leq t$  then
8:        $t[i_j] = BT_{i_j}$ 
9:       if  $DT_{i_j} < t$  then
10:         $t[s[i_j]] \geq DT_{i_j} + T_{i_j, s[i_j]}$ 
11:       else
12:         $t[s[i_j]] \geq t + T_{i_j, s[i_j]}$ 
13:         $isFixed = 0$ 
14:       end if
15:     else
16:        $t[i_j] \geq t$ 
17:     end if
18:   end while
19:   for Each  $u$  from 0 to  $j - 1$  do
20:      $s[i_u] = i_{u+1}$ 
21:   end for

```

454
 455

456 The main template of the hybrid algorithm is provided in Algorithm 5. First, a feasible
 457 solution is obtained by the TS algorithm, considering only the static requests. While no
 458 new incoming requests arrive, the solution is optimized using the tabu search algorithm.

459 This requires special attention since any new optimized solution should not be different
460 with respect to the previous solution up to the time at which the new solution is obtained.
461 To this end, the tabu search is performed for a fixed duration of φ minutes, and the input
462 solution is frozen up to φ minutes in advance of the current time, where φ is a parameter
463 fixed to 2 in our implementation. When this period of time has elapsed, the current
464 solution is updated. This procedure is repeated until a new request arrives, in which
465 case the optimization is interrupted. After a new request is received, a new instance I'
466 is created which contains all the data of the static and previously accepted requests, as
467 well as the new request. The tabu search and the CP algorithm are then executed in
468 parallel with the input instance I' , freezing the partial routes up to the current time, plus
469 φ minutes. Both procedures are terminated when one of them has found a solution, when
470 the CP algorithm has proved that the instance I' is infeasible subject to the fixed partial
471 routes, or when the time limit of φ minutes of computing time has elapsed. Naturally,
472 the incoming request is accepted only when a feasible solution has been found by any of
473 the two algorithms, and rejected otherwise.

474

475 **6 Computational results**

476 We have conducted a series of tests on two sets of instances for assessing the perfor-
477 mance of the hybrid algorithm.

478 **6.1 Instance generation**

479 The first set of dynamic instances were based on the set of static instances a and b used
480 in Ropke et al. (2007). In the instance subset a , vertices are located in a 20×20 square,
481 taking floating point values, and with a uniform random distribution. The distances are
482 Euclidean and are measured in minutes, the time horizon is 12 hours, the time windows
483 of critical vertices have a 15 minute length, and $Q = 3$. The instance subset b is similar,
484 except that $Q = 6$. We have only used the instances with at least 40 requests. The
485 instance labels are of the form ‘ $am-n$ ’ or ‘ $bm-n$ ’. The letter a and b state whether the
486 instance is from the subset a or b , the number m corresponds to the number of vehicles,
487 and the number n is the number of requests. More details of these instances can be found
488 in Cordeau (2006).

Algorithm 5 Main scheme of the hybrid algorithm

- 1: Obtain a solution s considering the instance I that only has the static requests using the tabu algorithm.
 - 2: **while** Time horizon has not been reached **do**
 - 3: **while** No new requests **do**
 - 4: Reoptimize actual solution s of I using the tabu search algorithm
 - 5: **end while**
 - 6: Create a new DARP instance I' by adding the new request
 - 7: Execute in parallel the tabu search procedure and the constraint programming algorithm with I and time limit φ and freeze all partial routes up to time $t + \varphi$
 - 8: **if** Either the tabu or the constraint programming procedures have found a solution s' **then**
 - 9: ACCEPT request
 - 10: $I = I', s = s'$
 - 11: **else**
 - 12: **if** Infeasibility was proved by CP or time limit φ has passed **then**
 - 13: REJECT request
 - 14: **end if**
 - 15: **end if**
 - 16: **end while**
-

489 These static instances were converted into dynamic ones by using a pair of parameters
 490 (α, β) . The value $\alpha \in [0, 1]$ gives the ratio of the requests which are known at the beginning
 491 of the time horizon. Thus, if $\alpha = 1$ the instance is completely static, while setting $\alpha = 0$
 492 yields an instance with no requests known a priori. Given a request i , the value $\mathcal{U}(i)$ is
 493 an upper bound on the time at which the request must be known in order to be able to
 494 serve it. It is defined as $\mathcal{U}(i) = \min\{l_{i+}, l_{i-} - T_{i+i-} - D_{i+}\}$. The parameter β states how
 495 much time before $\mathcal{U}(i)$ request i is known. If $\mathcal{U}(i) < \beta$, then request i is known at time
 496 zero.

497 The static instances of the subsets ‘ a ’ and ‘ b ’ were transformed into dynamic instances
 498 with the parameters $(\alpha = 0.25, \beta = 60)$, i.e., 25 % of the requests are static, and each
 499 dynamic request i becomes known 60 minutes before $\mathcal{U}(i)$. The hybrid algorithm was
 500 tested using the lazy scheduling algorithm and the eager scheduling algorithm presented
 501 in Section 4. Table 1 gives the number of accepted requests by the tabu search and by the
 502 CP algorithm, the number of rejected requests because of a time out of two minutes, and
 503 the number of infeasible requests identified by the CP algorithm. These results show that
 504 the number of dynamic requests that were accepted using the eager scheduling algorithm
 505 compared to those accepted with the lazy algorithm was increased by 270%. Our results
 506 also show that around 77% of all the rejected requests were proved to be infeasible by the
 507 CP algorithm.

508 The second set of instances are based on the 20 static instances of Cordeau and
 509 Laporte (2003) which contain between 24 and 144 requests and between three and 13
 510 vehicles. In these instances each request has a load of one unit and a maximum ride time
 511 of 90 minutes, and the vehicles have a capacity of six. The critical vertices have a time
 512 windows of length varying between 15 and 90 minutes. This set of static instances was
 513 transformed into a set of dynamic instances with the parameters $\alpha = 0.25$, and β being
 514 a random number uniformly distributed between 60 and 240. This means that 25% of
 515 the requests are static and each dynamic request i becomes known between one and four
 516 hours before its deadline $\mathcal{U}(i)$.

517 6.2 Results

518 In Table 2 we compare the performance of the dynamic DARP on these instances using
 519 the lazy and the eager scheduling algorithms. We can see that the eager algorithm still
 520 performs better than the lazy algorithm, but the difference between the two, although

Instance	Lazy scheduling				Eager scheduling			
	Accepted requests		Rejected requests		Accepted requests		Rejected requests	
	By tabu	By CP	Time out	Proved	By tabu	By CP	Time out	Proved
a4-40	23	0	0	5	5	17	1	5
a4-48	12	0	1	20	29	4	0	0
a5-40	15	0	0	12	25	2	0	0
a5-50	18	0	0	18	34	1	0	1
a5-60	29	0	0	14	36	1	1	5
a6-48	13	0	0	21	34	0	0	0
a6-60	10	0	8	22	37	2	0	1
a6-72	31	1	2	19	51	2	0	0
a7-56	19	0	3	14	34	2	0	0
a7-70	18	0	9	21	30	18	0	0
a7-84	24	1	16	19	50	3	0	7
a8-64	25	0	1	17	39	4	0	0
a8-80	4	0	11	38	48	0	0	5
a8-96	33	0	15	21	58	2	2	7
b4-40	14	0	0	14	28	0	0	0
b4-48	20	1	0	13	32	2	0	0
b5-40	9	0	1	16	24	2	0	0
b5-50	22	0	0	12	32	0	0	2
b5-60	29	0	9	6	36	1	0	7
b6-48	23	0	1	8	32	0	0	0
b6-60	28	0	8	8	41	2	0	1
b6-72	31	0	6	11	46	1	0	1
b7-56	12	0	10	16	31	0	1	6
b7-70	12	0	9	29	2	27	5	16
b7-84	6	0	8	42	53	3	0	0
b8-64	19	0	6	21	42	1	3	0
b8-80	19	1	6	29	52	0	0	3
b8-96	9	2	9	50	56	6	7	1
Total	527	6	139	536	1017	103	20	68

Table 1: Comparison of the number of accepted requests using the eager and the lazy scheduling algorithms

521 significant, is smaller than on the first set of instances. On average, the number of accepted
522 requests using the eager algorithm has increased by 34% compared to the number of
523 accepted requests with the lazy algorithm. On these instances the CP algorithm had
524 more difficulty proving the infeasibility of the rejected requests. On average, around 10%
525 of the rejected requests were proven to be infeasible by the CP algorithm in the available
526 running time of two minutes.

527 6.3 Modification of the objective function

528 We have also performed some experiments with a modified version of the tabu search
529 algorithm in which the objective function is changed. We have added a term we call
530 *slack(s)* to the objective function $f(s)$. This new term rewards solutions whose route
531 schedules can easily be modified and penalizes solutions whose routes have a rigid schedule.
532 The idea is that an incoming request is unlikely to be inserted in a route whose schedule is
533 very rigid, and therefore it is preferable to have solutions whose routes are more ‘schedule

Instance	Lazy scheduling				Eager scheduling			
	Accepted requests		Rejected requests		Accepted requests		Rejected requests	
	By tabu	By CP	Time out	Proved	By tabu	By CP	Time out	Proved
pr01	14	4	0	0	18	0	0	0
pr02	28	4	0	0	32	0	0	0
pr03	31	14	7	0	52	0	0	0
pr04	33	11	15	0	58	1	0	0
pr05	52	10	24	0	83	1	0	2
pr06	38	12	41	2	93	0	0	0
pr07	17	5	1	0	23	0	0	0
pr08	23	11	13	0	47	0	0	0
pr09	34	8	28	0	68	2	0	0
pr10	42	13	42	0	88	2	0	7
pr11	17	1	0	0	18	0	0	0
pr12	31	3	0	0	34	0	0	0
pr13	42	4	7	0	53	0	0	0
pr14	39	10	18	0	67	0	0	0
pr15	39	0	13	33	83	0	2	0
pr16	56	9	36	0	101	0	0	0
pr17	20	3	3	0	26	0	0	0
pr18	38	9	3	0	50	0	0	0
pr19	47	9	20	0	76	0	0	0
pr20	47	12	43	0	101	0	0	1
Total	688	152	314	35	1171	6	2	10

Table 2: Comparison of the number of accepted requests using the eager and the lazy scheduling algorithms.

flexible'. Let $r = (i_1, \dots, i_k)$ be a route and let BT_j^e and BT_j^l denote the start of service at vertex i_j using the eager and lazy schedules, respectively. We define the slack time of a route r as $slack(r) = \max \{BT_j^l - BT_j^e : j = 1, \dots, k\}$. The *slack* of a solution is equal to the sum of the slacks of each route. Although not perfect, this measure is global in the sense that it takes all requests of the route into account. For instance, if the slack of a route is 30 minutes this means that, at least for a given period of time, there are at least 30 extra minutes available to serve a new request without increasing the time window and ride time violation on any of the requests on the route.

Table 3 shows the results for the second set of instances when the objective function was modified to include the slack of the routes. When the lazy schedule is used, there is a slight increase, of around 4% on average, in the number of accepted requests compared to the algorithm without the slack measure in the objective function. However, this improvement is not observed when the eager algorithm is applied. In this case, the number of accepted requests is almost the same for both versions of the objective function.

Instance	Lazy scheduling				Eager scheduling			
	Accepted requests		Rejected requests		Accepted requests		Rejected requests	
	By tabu	By CP	Time out	Proved	By tabu	By CP	Time out	Proved
pr01	16	2	0	0	18	0	0	0
pr02	28	4	0	0	32	0	0	0
pr03	33	16	3	0	52	0	0	0
pr04	29	16	0	14	59	0	0	0
pr05	47	13	26	0	80	1	5	0
pr06	34	20	39	0	93	0	0	0
pr07	18	4	1	0	23	0	0	0
pr08	27	9	11	0	47	0	0	0
pr09	33	8	15	14	66	3	1	0
pr10	42	18	45	0	91	1	5	0
pr11	17	1	0	0	18	0	0	0
pr12	25	8	1	0	34	0	0	0
pr13	33	10	10	0	52	1	0	0
pr14	40	8	19	0	67	0	0	0
pr15	43	19	23	0	82	1	0	2
pr16	50	21	30	0	101	0	0	0
pr17	22	2	2	0	26	0	0	0
pr18	35	9	6	0	49	1	0	0
pr19	37	14	25	0	75	1	0	0
pr20	57	2	43	0	102	0	0	0
Total	666	204	299	28	1167	9	11	2

Table 3: Comparison of the number of accepted requests using the eager and the lazy scheduling algorithms with the slack time objective

7 Conclusions

We have developed a new hybrid algorithm for the dynamic DARP, combining a tabu search procedure and an exact constraint programming algorithm. Experiments performed on dynamic instances created from static instances have shown that the CP algorithm is sometimes able to accept or reject incoming requests. On the other hand, the tabu search tends to accept requests faster. This shows that the hybrid method outperforms any of the two algorithms when they are executed alone.

The capability of the CP procedure to prove infeasibility varies considerably, depending on the type of instance. Results have shown that on the first set of instances, around 77% of all the rejected requests were proved to be infeasible. However, this rate falls to 10% on the second set. An explanation for this difference is that in the first set of instances, the critical time windows are much smaller and therefore the solution space is reduced considerably.

Given a fixed route, we have developed scheduling algorithms to determine the times at which the arrival and the start of service at each vertex should take place. The basic and lazy scheduling algorithms are the natural dynamic extensions of the procedures presented by Cordeau and Laporte (2003) for the static problem. We have then developed a new scheduling algorithm called *eager*, which serves each vertex as early as possible without

566 increasing the time window or the ride time violation of any request. Results have shown
567 that the eager algorithm leads to the acceptance of considerably more requests than is
568 possible with the lazy algorithm.

569 **Acknowledgements**

570 This work was supported by the Canadian Natural Sciences and Engineering Research
571 Council under grants 227837-04 and 39682-05. This support is gratefully acknowledged.

572 **References**

- 573 A. Attanasio, J.-F. Cordeau, G. Ghiani, and G. Laporte. Parallel tabu search heuristics
574 for the dynamic multi-vehicle dial-a-ride problem. *Parallel Computing*, 30:377–387,
575 2004.
- 576 A. Beaudry, G. Laporte, T. Melo, and S. Nickel. Dynamic transportation of patients in
577 hospitals. *OR Spectrum*, 2010. Forthcoming.
- 578 G. Berbeglia, G. Pesant, and L.-M. Rousseau. Checking feasibility of the dial-a-ride
579 problem using constraint programming. Submitted for publication, 2009.
- 580 G. Berbeglia, J.-F. Cordeau, and G. Laporte. Dynamic pickup and delivery problems.
581 *European Journal of Operational Research*, 202:8–15, 2010.
- 582 R. Borndörfer, F. Klostermeier, M. Grötschel, and C. Küttner. Telebus Berlin: Vehicle
583 scheduling in a dial-a-ride system. Technical Report SC 97-23, Konrad-Zuse-Zentrum
584 für Informationstechnik, Berlin, 1997.
- 585 J.-F. Cordeau. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Re-*
586 *search*, 54:573–586, 2006.
- 587 J.-F. Cordeau and G. Laporte. A tabu search heuristic for the static multi-vehicle dial-a-
588 ride problem. *Transportation Research Part B*, 37:579–594, 2003.
- 589 J.-F. Cordeau and G. Laporte. The dial-a-ride problem: models and algorithms. *Annals*
590 *of Operations Research*, 153:29–46, 2007.

- 591 J.-F. Cordeau, G. Laporte, and A. Mercier. A unified tabu search heuristic for vehicle
592 routing problems with time windows. *Journal of the Operational Research Society*, 52:
593 928–936, 2001.
- 594 L. Coslovich, R. Pesenti, and W. Ukovich. A two-phase insertion technique of unexpected
595 customers for a dynamic dial-a-ride problem. *European Journal of Operational Research*,
596 175:1605–1615, 2006.
- 597 J. Desrosiers, Y. Dumas, and F. Soumis. A dynamic programming solution of the large-
598 scale single-vehicle dial-a-ride problem with time windows. *American Journal of Math-
599 ematical and Management Sciences*, 6:301–325, 1986.
- 600 M. Gendreau, A. Hertz, and G. Laporte. A tabu search heuristic for the vehicle routing
601 problem. *Management Science*, 40:1276–1290, 1994.
- 602 F. Glover and M. Laguna. *Tabu Search*. Kluwer, Boston, 1997.
- 603 M. E. T. Horn. Fleet scheduling and dispatching for demand-responsive passenger services.
604 *Transportation Research Part C*, 10:35–63, 2002.
- 605 O. B. G. Madsen, H. F. Ravn, and J. M. Rygaard. A heuristic algorithm for a dial-a-ride
606 problem with time windows, multiple capacities, and multiple objectives. *Annals of
607 Operations Research*, 60:193–208, 1995.
- 608 S. Mitrović-Minić and G. Laporte. Waiting strategies for the dynamic pickup and delivery
609 problem with time windows. *Transportation Research Part B*, 38:635–655, 2004.
- 610 H. N. Psaraftis. A dynamic programming approach to the single-vehicle, many-to-many
611 immediate request dial-a-ride problem. *Transportation Science*, 14:130–154, 1980.
- 612 B. Rekiek, A. Delchambre, and H. A. Saleh. Handicapped person transportation: An
613 application of the grouping genetic algorithm. *Engineering Applications of Artificial
614 Intelligence*, 19:511–520, 2006.
- 615 S. Ropke, J.-F. Cordeau, and G. Laporte. Models and branch-and-cut algorithm for
616 pickup and delivery problems with time windows. *Networks*, 49:258–272, 2007.

- 617 P. Toth and D. Vigo. Fast local search algorithms for the handicapped persons trans-
618 portation problem. In H.I. Osman and J.P. Kelly, editors, *Meta-heuristics Theory and*
619 *Applications*, pages 677–690. Kluwer, Boston, 1996.
- 620 P. Van Hentenryck. *Constraint Satisfaction in Logic Programming*. The MIT Press, 1989.
621 Cambridge, MA.
- 622 W.-J. van Hoesve and I. Katriel. Global constraints. In F. Rossi, P. Van Beek, and
623 T. Walsh, editors, *Handbook of Constraint Programming*, pages 169–208. Elsevier, Am-
624 sterdam, 2006.
- 625 Z. Xiang, C. Chu, and H. Chen. The study of a dynamic dial-a-ride problem under time-
626 dependent and stochastic environments. *European Journal of Operational Research*,
627 185:534–551, 2008.
- 628 C. W. Yuen, K. I. Wong, and A. F. Han. Waiting strategies for the dynamic dial-a-
629 ride problem. *International Journal of Environment and Sustainable Development*, 8:
630 314–329, 2009.