TRANSPORTATION SCIENCE
Vol. 00, No. 0, Xxxxx 0000, pp. 000–000
ISSN 0041-1655 | EISSN 1526-5447 | 00 | 0000 | 0001

INFORMS
DOI 10.1287/xxxx.0000.0000

A Maximum Cluster Algorithm for Checking the Feasibility of Dial-A-Ride Instances

Lauri Häme
Aalto University School of Science, lauri.hame@tkk.fi,

Harri Hakula
Aalto University School of Science, harri.hakula@tkk.fi,

The Dial-A-Ride Problem (DARP) involves the dispatching of a fleet of vehicles in order to transport customers requesting service and is one of the most challenging tasks of combinatorial optimization. Berbeglia et al. (2011b) models the DARP as a constraint satisfaction problem, where the goal is to find a feasible solution with respect to the time, capacity and precedence constraints, or to prove infeasibility. The main contribution of our work is a new exact method for this problem formulation. The algorithm is based on a dynamic subroutine which finds for any set of customers a maximal set of customers that can be served by a single vehicle. The performance of the algorithm is analyzed and evaluated by means of computational experiments, justifying the efficiency of the solution method. In addition to the dial-a-ride problem, the algorithm has an important application in food delivery services, where each meal needs to be delivered within guaranteed time limits.

Key words: Dial-A-Ride Problem, Algorithms, Dynamic Programming

Introduction

The dial-a-ride problem (DARP) is a generalization of the vehicle routing problem (VRP) arising in contexts where passengers are transported, either in groups or individually, between specified origins and destinations, as defined by Cordeau et al. (2007a).

In most studies related to the dial-a-ride problem, the customers' trips are restricted by *time windows* for pick-up and drop-off, see for example Psaraftis (1983), Jaw et al. (1986), Madsen et al. (1995), Toth and Vigo (1997), Hunsaker and Savelsbergh (2002), Cordeau and Laporte (2003a), Diana and Dessouky (2004), Wong and Bell (2006), Cordeau (2006), Häme (2011), Berbeglia (2009), Berbeglia et al. (2011a,b). In these studies it is noted that in dynamic settings, time windows eliminate the possibility of indefinite deferment of customers and strict time limits help provide reliable service. In addition to time windows, the vehicle routes are restricted by *maximum ride time constraints* that limit the time spent by passengers on the vehicle between their pick-up and their drop-off, *capacity* and *precedence* constraints. For an exhaustive summary on the models and algorithms for the DARP, the reader is referred to (Cordeau and Laporte 2007c).

We examine the DARP as a *constraint satisfaction problem*, in which the goal is to find a set of m feasible vehicle routes that serve all customers, where m is the number of vehicles, or to prove that such a set of routes does not exist, as in (Berbeglia et al. 2011b). In this reference, it is noted that an algorithm for checking the feasibility of a DARP instance has two main applications: 1) Determining the feasibility can be the first phase in an optimization algorithm in a *static* setting, where all trip requests are known, for example, one day in advance. 2) In *dynamic* services, a constraint satisfaction algorithm can be used for deciding whether to accept or reject incoming user requests.

This work is partly motivated by the latter application, namely, a dynamic demand-responsive transport (DRT) service currently being planned to operate in Helsinki. The service, run by Helsinki Region Transport, The Finnish Transport Agency and Aalto University, will be deployed by the end of 2012. Similarly as the current service routes (Helsinki Region Traffic 2010), the new DRT service is designed to operate on a demand-responsive basis, that is, each trip is booked in advance and the vehicle routes are modified according to the trips. The main difference to existing services is that no pre-order times for trips are required and

the trips can be booked "on the fly" by means of an interactive user interface. A main goal is to provide high-quality service that could compete with private car traffic.

In addition to passenger transportation, we note that a time-constrained problem similar to the DARP arises in food delivery services. Instead of passengers with pick-up and drop-off time windows, the goal is to transport *meals* from restaurants to specified delivery points within guaranteed time limits. Similarly as in the dynamic DARP, the food delivery service provider needs to decide quickly whether to take an incoming order or not, if there is a guaranteed maximum delivery time, for example, one hour.

In both contexts (passenger transportation and food service), a good level of service means that the problem is highly constrained (Jokinen et al. 2011). In such problems, the number of feasible solutions becomes so limited that often any feasible solution will be relatively close to the optimal solution with respect to the objective function (Psaraftis 1983). Thus, we suggest that in highly constrained cases, solving the consraint satisfaction problem provides, if not the optimal solution, at least a good initial solution.

The main result of this work is an exact algorithm for the constraint satisfaction dial-a-ride problem introduced by Berbeglia et al. (2011b). In contrast to most works related to vehicle routing and dial-a-ride problems, no specific cost function is considered[1]. The algorithm stops as soon as a feasible solution is found regardeless of the quality of the solution. If a feasible solution is not found through exhaustive search, the algorithm proves that the problem instance is infeasible. We also show how some infeasible instances can be detected in advance by studying the feasiblility of routes involving two customers. The computational results obtained by the proposed solution method are compared to the results presented in (Berbeglia et al. 2011b). In addition, we solve three instances, for which results have not been previously reported. The experiments suggest that algorithm is capable of solving large instances efficiently.

This document is organized as follows. In Section 1, we formalize the multi-vehicle DARP with time windows. In Section 2.1, we present a dynamic programming algorithm which produces for any set of customers the maximal set of customers that can be served by a single vehicle. The extension of the algorithm to the multi-vehicle case is discussed in Section 2.2. This is followed by an analysis of the structure and complexity of the solution method in Section 3 and computational experiments in Section 4.

1. Problem formulation

The dial-a-ride problem is defined as follows (Berbeglia et al. 2011b). Let $G = (V, A)$ be a complete and directed graph with node set $V = \{0\} \cup P$, where node 0 represents the depot, and P represents the set of pick-up and drop-off nodes, where ($|P| = 2n$). The set P is partitioned into sets P^+ (pick-up nodes) and P^- (drop-off nodes). Each arc $(i, j) \in A$ has a non-negative travel time T_{ij}. The travel times satisfy the triangle equation, that is, $T_{ij} + T_{jk} \geq T_{ik}$ for all $i, j, k \in V$. With each node $i \in V$ are associated a time window $[E_i, L_i]$, a service duration D_i and a load q_i, where $D_0 = 0$ and $q_0 = 0$. Let $H = \{1, \ldots, n\}$ be the set of customers and let $T^{\max}$ be the maximum ride time for any customer. With each customer i is associated a pickup node $i^+ \in P^+$, a delivery node $i^- \in P^-$ and a load $q_{i^+} = q_{i^-}$. The above parameters are illustrated in Figure 1.

Let $K = \{1, \ldots, m\}$ be the set of available vehicles, each with capacity Q. A *route* is a circuit over a set of nodes in P, starting and finishing at the depot 0. The DARP consists of constructing m vehicle routes (possibly empty) such that: (i) for every customer i the pick-up node and the drop-off node are visited by the same route and the pick-up node is visited before the drop-off node; (ii) the load of the vehicles does not exceed the capacity Q at any time; (iii) the ride time of each customer is at most $T^{\max}$; (iv) the service at node i begins within the interval $[E_i, L_i]$.

2. Problem solution

Our solution to the dial-a-ride problem defined in the previous section is based on a single-vehicle algorithm which produces for any set $X \subset H$ of customers a single route that serves as many customers in X as possible (Section 2.1). This algorithm is extended to solve the multi-vehicle dial-a-ride problem in Section 2.2.

[1] For approaches to vehicle routing and dial-a-ride problems with cost functions, we refer to Bräysy and Gendreau (2005a,b), Groër et al. (2010) and Cordeau and Laporte (2003a), Berbeglia et al. (2011a).

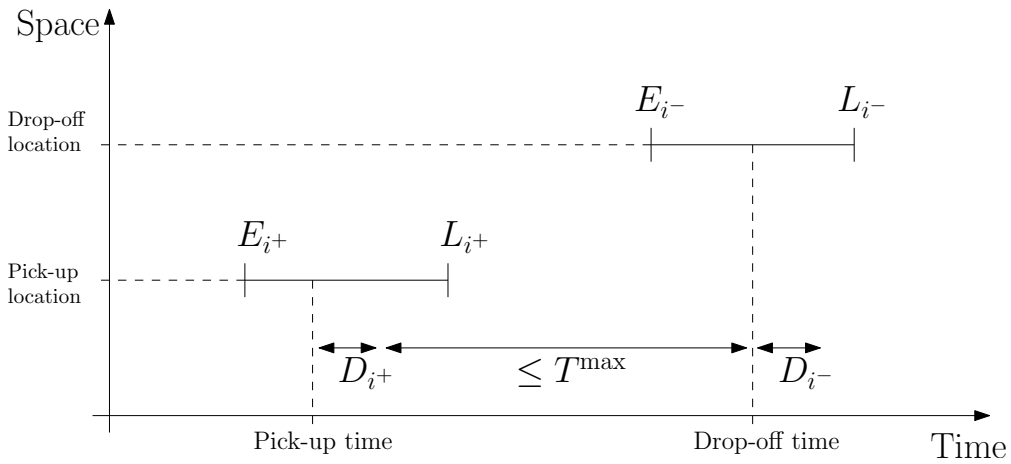

Figure 1 **Time pick-up and drop-off time windows. The pick-up point of customer i is denoted by i^+ and the drop-off point is denoted by i^-. The customer should be picked up at i^+ within the time window $[E_{i^+}, L_{i^+}]$ and the customer should be dropped off at i^- within the time window $[E_{i^-}, L_{i^-}]$. The service times needed for the customer to get on the vehicle and get off the vehicle are denoted by D_{i^+} and D_{i^-}. The time between the drop-off and the pick-up (excluding D_{i^+}) should not exceed the maximum ride time $T^{\max}$.**

The single-vehicle dial-a-ride problem has been previously studied by Psaraftis (1980), Bianco et al. (1994), Hernández-Pérez and Salazar-González (2009), Psaraftis (1983), Desrosiers et al. (1986), Sexton (1979), Sexton and Bodin (1985a,b), Häme (2011). The difference between this work and existing approaches is that we study the problem as a constraint satisfaction problem and the objective is to maximize the number of served customers.

2.1. Maximum cluster algorithm

In this section, we propose a dynamic programming method for the single-vehicle DARP ($m = 1$) with customer set X that produces a feasible solution whenever one exists or proves that the problem is infeasible. In the latter case, the algorithm returns a route that maximizes the number of served customers in X. The main challenge is to find a sequence of pick-up and drop-off nodes for which the time and precedence constraints are satisfied. While capacity constraints are also considered, the focus is on problems, in which time limits are more restrictive.

For clarity, let us denote the customers in X by numbers $\{1,\dots,N\}$ and the sets of pick-up and drop-off nodes by $P^+ = \{1^+,\dots,N^+\}$ and $P^- = \{1^-,\dots,N^-\}$. The goal is to find a feasible service sequence $(0, p_1,\dots,p_{2N}, 0)$ consisting of the pick-up and drop-off nodes of N customers. The number of permutations is $(2N)!$ and the number of feasible permutations with respect to precedence constraints is $(2N)!/2^N$ (Häme 2011). As an example, the 24 possible sequences of the pick-up and drop-off nodes of two customers are represented as a *tree structure* in Figure 2.1. The six sequences that are feasible with respect to the precedence constraints are marked with 'p' and the sequences that are feasible with respect to time and capacity constraints as well are marked with dotted lines. In this example, there is one feasible sequence serving all customers, namely, $(0, 2^+, 1^+, 2^-, 1^-, 0)$. The primary goal is to find this feasible sequence.

Briefly, our approach to the problem is a *depth-first* search, in which the infeasible branches are detected a priori and then discarded. The search begins from node 0 by determining the infeasible branches beginning from 0. In the example case presented in Figure 2.1, the branches beginning from the drop-off points 1^- and 2^- could be discarded since they would not satisfy the precedence constraint. Thus, we would only have to consider the branches beginning from 1^+ and 2^+. Similarly, at state $(0, 2^+)$, the branch beginning with 1^- could be discarded due to the precedence constraint and the search would proceed to the branches beginning from 1^+ and 2^-. In addition to precedence constraints, branches are discarded due to time and capacity constraints (see Section 2.1.3). Since only infeasible branches are discarded, this method finds a feasible solution whenever such a solution exists. If a feasible sequence serving all customers is found, the search can be terminated.

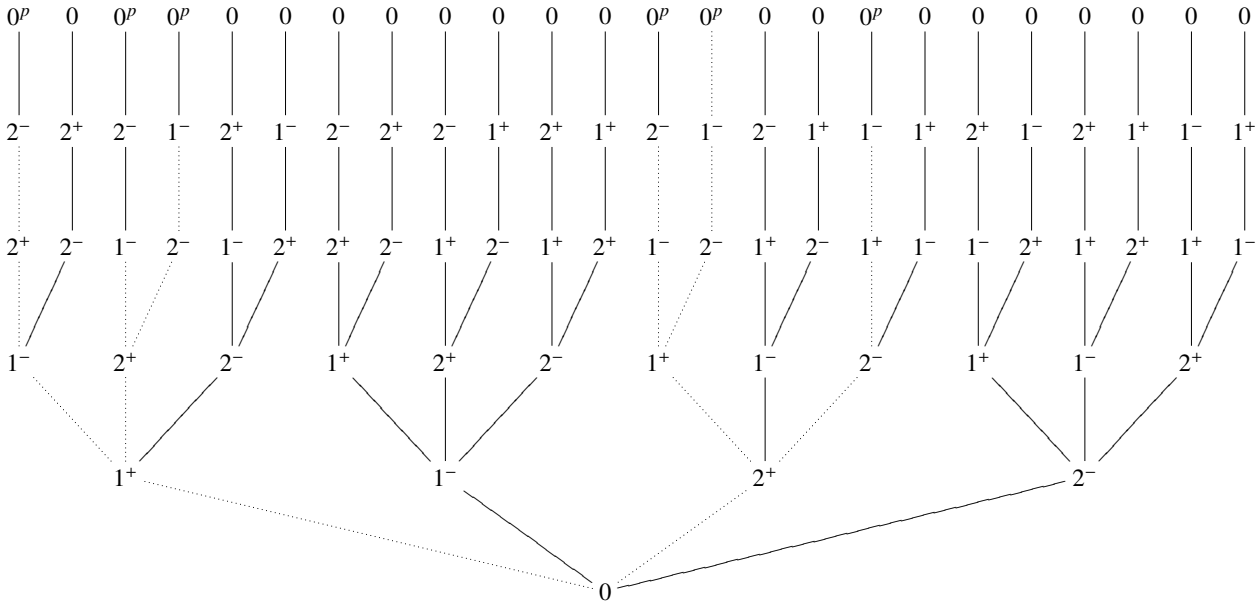

Figure 2 **The single-vehicle DARP as a tree structure. There are** 24 **permutations of the nodes** $1^+, 1^-, 2^+, 2^-$ **and six permutations (marked with 'p') that are feasible with respect to precedence constraints. The goal is to find a sequence that is feasible with respect to time and capacity constraints as well, without going through all permutations.**

2.1.1. Ranking feasible branches In order to find a feasible solution whenever one exists, we have to consider at each state S all branches that are not rendered infeasible (branches $(0, 2^+, 1^+)$ and $(0, 2^+, 2^-)$ in the previous example). However, the order in which these branches are evaluated has a significant effect on the effort needed to find a feasible solution. For example, if the search proceeded by evaluating the branch $(0, 2^+, 1^+)$, producing a feasible solution, the search could be terminated and there would be no need to evaluate the branch $(0, 2^+, 2^-)$ (which would prove to be a dead end).

Given that the search is at state S, our approach is to sort the feasible successor branches (S, i) by means of a two-step procedure as follows:

1. The feasibility of branches (S, i) is evaluated for all successor branches of S.
2. The number of feasible sequences (S, i, j) is calculated for all feasible branches (S, i).

The search proceeds by first evaluating the branches (S, i), for which the number of feasible sequences (S, i, j), where $j \in X$, is greatest.

In the example case (Figure 2.1) with $S = (0, 2^+)$, the search would proceed to branch $(0, 2^+, 1^+)$, since there are two feasible sequences $(S, 1^+, j)$, namely $(0, 2^+, 1^+, 1^-)$ and $(0, 2^+, 1^+, 2^-)$ and only one feasible sequence $(S, 2^-, j)$, that is, $(0, 2^+, 2^-, 1^+)$. A similar example is shown in Figure 3.

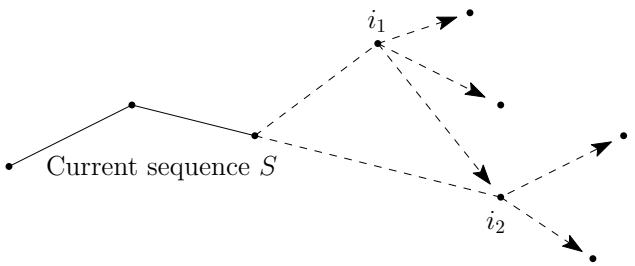

Figure 3 **Ranking feasible branches. The current sequence** S **consisting of three nodes is marked with a solid line. The branches** (S, i_1) **and** (S, i_2) **are ranked by means of the number of feasible sequences** (S, i_1, j) **and** (S, i_2, j) **marked with dashed lines. In this case, the search proceeds to state** (S, i_1) **since there are three feasible sequences** (S, i_1, j) **and two feasible sequences** (S, i_2, j)**.**

2.1.2. Recursive solution Formally, the search is executed by means of the recursion REC(S, R_S) shown in Algorithm 1, where S denotes a sequence of nodes (initially $S = (0)$), R_S denotes the set of remaining nodes at S (initially $R_S = \{1^+, 1^-, \ldots, N^+, N^-, 0\}$) and $S_{\max}$ denotes the sequence that maximizes the number of served customers (initially $S_{\max} = (0)$). $C(S)$ denotes the number of served customers in sequence S, which is equal to the number of drop-off points in S.

Algorithm 1 A recursive solution REC(S, R_S) to the single-vehicle DARP. S denotes a sequence of nodes (initially $S = (0)$), R_S denotes the set of remaining nodes (initially $R_S = \{1^+, 1^-, \ldots, N^+, N^-, 0\}$) and $S_{\max}$ denotes the sequence that maximizes the number of served customers (initially $S_{\max} = (0)$). $C(S)$ denotes the number of served customers in sequence S.

```
if C(S) > C(S_max) then
  S_max ← S;                                   (Store the current sequence S.)
  if C(S) = N
    Terminate recursion;                       (All customers have been served.)
  end if
end if
if R_S = ∅ then
  Return;                                      (Dead end)
end if
Determine the set I of nodes for which (S, i) is infeasible;   (See Section 2.1.3, Algorithm 2.)
Determine the set of remaining nodes R_(S,i) for all i ∈ R_S \ I;
Sort the remaining nodes i ∈ R_S \ I by |R_(S,i)|;
for all i ∈ R_S \ I (in sorted order) do
  REC((S, i),R_(S,i));
end for
```

The main idea of the recursion is that at each step, we have the current service sequence S and the set of remaining nodes R_S (nodes that can possibly be added to the sequence). At first, we check if the current sequence serves more customers than the previously visited sequences. If the maximum number of served customers is improved, the current sequence is stored, $S_{\max} \leftarrow S$. If all customers have been served, that is, if $C(S) = N$, a feasible solution is found and the recursion is terminated. Otherwise, we check if the set of remaining nodes is empty. If $R_S = \emptyset$, no further nodes can be added to the sequence and the search proceeds to the next branch. Otherwise, the feasibility of sequence (S, i) is checked for all $i \in R_S$ and the set of remaining nodes $R_{(S,i)}$ is calculated for all nodes $i \in R_S$ for which (S, i) is feasible (see Section 2.1.3). Then, the set of remaining nodes R_S (excluding the nodes for which (S, i) is infeasible) is sorted by $|R_{(S,i)}|$. Finally, the recursive function REC$((S, i), R_{(S,i)})$ is called for all feasible sequences (S, i) in sorted order.

In the following, we describe the feasibility screening and the calculation of the set of remaining nodes in more detail.

2.1.3. Feasibility considerations The feasibility screening is executed in two steps: In the *first step*, we check if the precedence, time and capacity constraints are satisfied at (S, i) for all remaining nodes $i \in R_S$. In the *second step*, we determine the set of remaining nodes $R_{(S,i)}$ at (S, i) by considering the possibilities of adding nodes $j \in R_S \setminus \{i\}$ to the sequence *after* i.

The first step is similar as in existing dynamic programming algorithms for the single-vehicle DARP with time windows (Psaraftis 1983, Desrosiers et al. 1986). By considering a second step, a *ranking* of the branches (S, i) is obtained by considering the number $|R_{(S,i)}|$ of remaining nodes after the first step. A summary of the feasibility conditions is presented in Table 1.

Table 1 **A summary of feasibility conditions. Conditions** (1), (2) **and** (3) **are related to the feasibility of a successor sequence** (S,i)**, whereas condition** (4) **renders the sequence** (S,i,j) **infeasible.** t_i**,** E_i **and** L_i **represent the arrival time and lower and upper bound of the time window at node** i**.** Q_i **denotes the load on the vehicle after visiting node** i**.**

Step	Condition	Impact	Equation number
1	$t_i > L_i$	(S,i) is infeasible, $R_{(S,j)} \leftarrow R_{(S,j)} \setminus \{i\}$ for all $j \in R_S \setminus \{i\}$	(1)
1	$i = h^- \in P^-$ and $h^+ \notin S$	(S,i) is infeasible	(2)
1	$Q_i > C$	(S,i) is infeasible	(3)
2	$\max(t_i, E_i) + T_{ij} > L_j$	$R_{(S,i)} \leftarrow R_{(S,i)} \setminus \{j\}$	(4)

First step In the following, s denotes the last node in S, t_s denotes the departure time at s and Q_s denotes the load on the vehicle after visiting s. For clarity, service times are assumed equal to zero. However, the modification of the following equations for non-zero service times is straightforward.

First, the set of remaining nodes $R_{(S,i)}$ at (S,i) is initialized, that is, $R_{(S,i)} \leftarrow R_S \setminus \{i\}$ for all $i \in R_S$. The time of arrival t_i at i is given by $t_i = t_s + T_{si}$ and the load Q_i after visiting i is given by $Q_i = Q_s + q_i$.

We note that the sequence (S,i) is infeasible, if

$$t_i > L_i. \tag{1}$$

In this case, there exist no sequences from S to i such that the time constraint at i is satisfied. Thus, the sequence (S,i) is marked infeasible and node i is removed from the set of remaining nodes $R_{(S,j)}$ at (S,j) for all $j \in R_S$. This state elimination criteria is similar to the one presented in (Desrosiers et al. 1986). Maximum ride time constraints are checked similarly with the exception that L_i is modified by means of the procedure described in Section 2.1.4.

In addition, the sequence (S,i) is infeasible, if at least one of the following conditions is true:

$$i = h^- \in P^- \text{ and the node } h^+ \text{ is not included in } S, \tag{2}$$
$$Q_i > Q. \tag{3}$$

In this case, the sequence (S,i) is marked infeasible but i is not removed from the sets of remaining nodes $R_{(S,j)}$, since there may exist feasible sequences (S,j) that can be extended to include node i.

Second step Given that the vehicle is at the last node of sequence S, let us consider the possibilities of adding a node $j \in R_{(S,i)}$ to the sequence (S,i). The time of arrival t_j at j is given by $\max(t_i, E_i) + T_{ij}$.

Similarly as in condition (1), if there exists a node $j \in R_{(S,i)}$ such that

$$\max(t_i, E_i) + T_{ij} > L_j, \tag{4}$$

the sequence (S,i,j) is infeasible and node j is removed from the set of remaining nodes $R_{(S,i)}$ at (S,i). For the checking of maximum ride time constraints, see Section 2.1.4.

This feasibility condition remotely similar to the one studied in (Dumas et al. 1991), in which the set of *admissible arcs* between the pick-up and delivery nodes is constructed as a preprocessing step in the pickup and delivery problem. The set of admissible arcs is made up of arcs which a priori satisfy the precedence, capacity and time constraints of the problem. The difference in our formulation is that we study the feasibility of sequences (S,i,j) during the search of feasible routes (see Figure 4).

Note that even if the precedence or capacity constraint is violated by adding node j to the sequence (S,i), there may exist a feasible sequence $(S,i,k_1,\ldots,k_h,j)$ in which i precedes j. That is, if the sequence (S,i) is not rendered infeasible by condition (4), there *may exist* a feasible sequence in which j is visited after i (not necessarily immediately after i) and thus node j is not removed from $R_{(S,i)}$.

The above two-step feasibility screening procedure is presented in Algorithm 2.

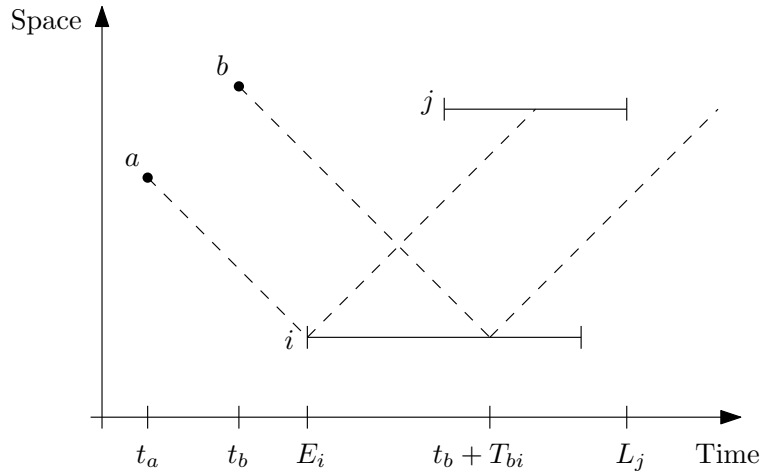

Figure 4 **Admissible arcs. The figure shows the locations and time windows of two nodes i and j. If the vehicle left from a at the instant t_a, visited node i and moved directly to node j, the time constraint L_j would be satisfied. However, assuming that at the instant t_b the vehicle is located at b, the arc (i, j) is seen to be inadmissible.**

Algorithm 2 Two-step feasibility screening. The algorithm returns the set of nodes I for which the sequence (S, i) is infeasible and the set $R_{(S,i)}$ of remaining nodes for each feasible branch (S, i).

$I = \emptyset$, $R_{(S,i)} = R_S \setminus \{i\}$ for all $i \in R_S$; (I = the set of nodes for which (S, i) is infeasible)
for all $i \in R_S$ **do**
 if $t_s + T_{si} > l_i$ (Condition (1))
  $I = I \cup \{i\}$; ((S, i) is infeasible)
  $R_{(S,j)} = R_{(S,j)} \setminus \{i\}$ for all $j \in R_S \setminus \{i\}$;
  continue; (continue to next i)
 end if
 if $(i = h^- \in P^-$ and $h^+ \notin S)$ or $(Q_i > Q)$ (Conditions (2) and (3))
  $I = I \cup \{i\}$; ((S, i) is infeasible)
  continue; (continue to next i)
 end if
 for all $j \in R_{(S,i)}$ **do**
  if $\max(t_i, e_i) + T_{ij} > l_j$ (Condition (4))
   $R_{(S,i)} = R_{(S,i)} \setminus \{j\}$; ($(S, i, j)$ is infeasible)
  end if
 end for
end for
return $(I, \{R_{(S,i)}\}_{i \in R_S})$;

2.1.4. Maximum ride time constraints The feasibility of maximum ride time constraints is checked as follows. Let $T_{\max}$ denote the maximum ride time. When a pick-up node j^+ is added to the sequence, the earliest arrival time t_{j^+} at j^+ is calculated, and the updated latest drop-off time of customer j is calculated by means of the formula

$$L'_{j^-} = \min(L_{j^-}, \max(t_{j^+}, E_{j^+}) + T_{\max}), \tag{5}$$

where $\max(t_{j^+}, E_{j^+})$ denotes the earliest pick-up time of customer j.

The updated latest drop off time L'_{j^-} describes the latest drop-off time given that customer j is picked up at the earliest pick-up time. However, it may sometimes be profitable to delay the beginning of service at

node j^+ so as to reduce the unnecessary in-vehicle waiting time at any node visited between j^+ and j^- and thus, the ride time associated with customer j (Cordeau and Laporte 2003a).

In order to take into account the possibility of delaying the pick-up of customer j, we propose the following procedure. First, at the pick-up node j^+, we define the *time slack* s_j of customer j by $s_j = L_{j^-} - \max(t_{j^+}, E_{j^+})$, where $\max(t_{j^+}, E_{j^+})$ is the earliest pick-up time of customer j. The time slack describes the amount of time the pick-up of customer j can be delayed at most. Since the pick-up of customer can be delayed by at most s_j, we define the *delayed latest drop-off time* L''_j of customer j by

$$L''_{j^-} = \min(L_{j^-}, L'_{j^-} + s_j). \tag{6}$$

In addition, we initialize the *in-vehicle waiting time* w_j of customer j by $w_j = 0$.

Then, at each node h visited between j^+ and j^-, the time slack s_j, in-vehicle waiting time w_j and the delayed latest drop off time L''_{j^-} of customer j are updated:

$$s_j \leftarrow \min(s_j, w_j + L_h - t_h), \tag{7}$$
$$L''_{j^-} \leftarrow \min(L_j, L'_{j^-} + s_j), \tag{8}$$
$$w_j \leftarrow w_j + \max(E_h - t_h, 0), \tag{9}$$

where t_h denotes the earliest time of arrival at node h. Note that if h is a drop-off point, L_h in Equation (7) is replaced by L''_h. Finally, at the drop-off point j^-, the maximum ride time constraint is violated if

$$t_{j^-} > L''_{j^-}, \tag{10}$$

where t_{j^-} is the earliest arrival time at node j^- (see Figure 5).

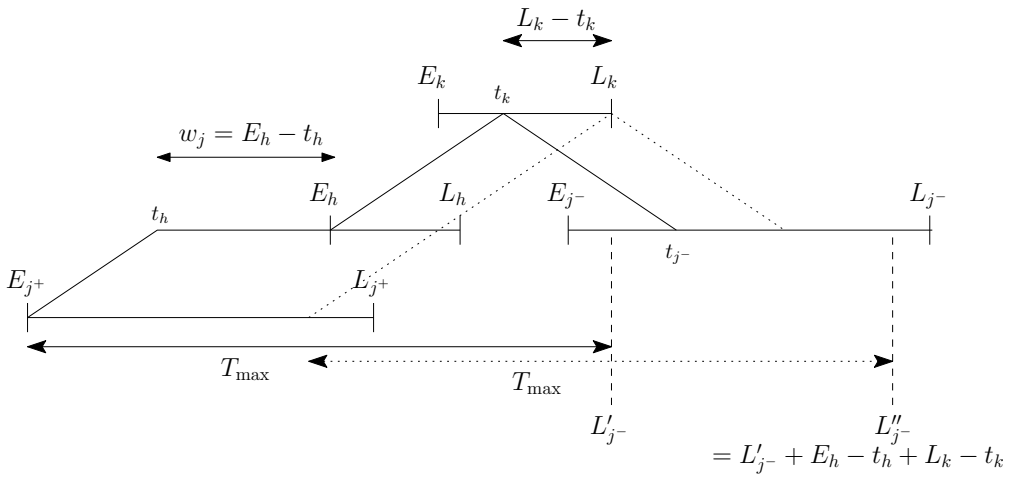

Figure 5 **Maximum ride time constraints. The figure shows the time windows of nodes j^+, h, k, j^-, a non-delayed route (solid line) and a delayed route (dotted line). At the pick-up point j^+, the updated latest drop-off time L'_{j^-} of customer j is calculated by means of Equation (5). At the nodes h and k between j^+ and j^-, the time slack s_j, delayed latest drop-off time L''_{j^-} and in-vehicle waiting time w_j of customer j are updated by Equations (7), (8) and (9). Finally, at j^-, the maximum ride time constraint is checked by Equation (10).**

2.1.5. Discarding suboptimal sequences Since the goal is to maximize the number of served customers, the sequences that can not improve the maximum number of served customers, can be discarded (in addition to infeasible sequences). That is, given a sequence S, set of remaining nodes R_S and the best found solution $S_{\max}$, the sequence S is discarded if

$$C(S) + C(R_S) \leq C(S_{\max}),$$

where $C(S), C(R_S)$ and $C(S_{\max})$ denote the number of drop-off points in S, R_S and $S_{\max}$, respectively.

2.1.6. A heuristic extension Algorithm 1 describes an exact procedure for maximizing the number of served customers in a single route. In order to find the exact solution, the algorithm goes through all branches until no further nodes can be added to the current sequence or the sequence is seen to be suboptimal.

The effort of the algorithm can be controlled by limiting the number of branches that are evaluated by means of a positive parameter L. For example, if $L = 1$, the algorithm constructs a single sequence and stops when the set of remaining nodes in empty. By increasing L the search space is expanded and if $L = (2N)!/2^N$ (the number of permutations that satisfy precedence constraints), the heuristic coincides with the exact algorithm.

2.2. Multi-vehicle solution

In this section, we propose an exact approach to the multi-vehicle DARP as a constraint satisfaction problem. By using Algorithm 1 described in Section 2.1 as a subroutine, the method produces a feasible solution to any instance or proves that the instance is infeasible. The main idea is that the vehicle routes are constructed one by one, each maximizing the number of served customers in the set of remaining customers. The customers are denoted by numbers $1, \dots, n$ and the vehicles are denoted by numbers $1, \dots, m$.

First, a route is constructed for vehicle 1, serving as many customers as possible. Then, the process is repeated with vehicle 2 for the set of customers that were not served by vehicle 1 and so forth (see Figure 6). Letting X denote a set of customers, we denote by $\mathrm{MC}(X)$ the single-vehicle maximum cluster algorithm (Algorithm 1) that returns a route that maximizes number of served customers in set X and the corresponding set of served customers. This set will be referred to as a maximum cluster of X, which is formally defined as follows.

Definition 1. Let X be a set of customers and $M \subset X$ be a subset of X, for which there exists a feasible route serving all customers in M. If $|M| \geq |Y|$ for all sets $Y \subset X$ for which there exists a feasible route serving all customers in Y, then M is a *maximum cluster* of X.

Note that if there exists a feasible route serving all customers in X, we have $M(X) = X$. The solution to the multi-vehicle case is outlined in Algorithm 3.

Algorithm 3 Outline of the multi-vehicle algorithm. X_k denotes the set of customers assigned to vehicle k and $\mathrm{MC}(X_k)$ denotes the single-vehicle subroutine presented in Algorithm 1 that returns a maximum cluster X'_k of X_k and the corresponding route S_k. Initially, all customers are assigned to the first vehicle, that is, $X_1 = \{1, \dots, n\}$ and $X_k = \emptyset$ for all $k \in \{2, \dots, m\}$.

for all $k \in \{1, \dots, m\}$ **do**
  $[S_k, X'_k] \leftarrow \mathrm{MC}(X_k)$; ($S_k$ = route, X'_k = set of served customers)
  $X_{k+1} \leftarrow X_{k+1} \cup (X_k \setminus X'_k)$, where $X_{m+1} = X_1$;
end for

2.2.1. Iteration If a feasible solution is not found directly by using the procedure described in Algorithm 3, the process is repeated. The goal is to find a set of customer-vehicle assignments such that for each vehicle there exists a feasible route serving all customers assigned to the vehicle or to prove infeasibility by going through all possible sets of customer-vehicle assignments.

Formally, we define a *partition* of customers as a disjoint cover of $\{1, \dots, n\}$ consisting of m sets $(X_1, \dots, X_m)$, where X_k denotes the set of customers assigned to vehicle k. The set of all partitions is denoted by $\mathcal{P}$. Since each customer can be assigned to any of the m vehicles, the total number of possible partitions is equal to the Stirling number of the second kind, that is, $|\mathcal{P}| = S_2(n, m) = \frac{1}{m!} \sum_{i=0}^{m} (-1)^i \binom{m}{i} (m - i)^n$. A partition is said to be *feasible*, if $M(X_k) = X_k$ for all $k \in \{1, \dots, m\}$.

Note that if the multi-vehicle instance has a solution, there exists a partition $(X_1^*, X_2^*, \dots, X_m^*)$ for which $M(X_k^*) = X_k^*$ for all $k \in \{1, \dots, m\}$. Thus, infeasibility can be proved by iteratively checking all partitions for feasibility.

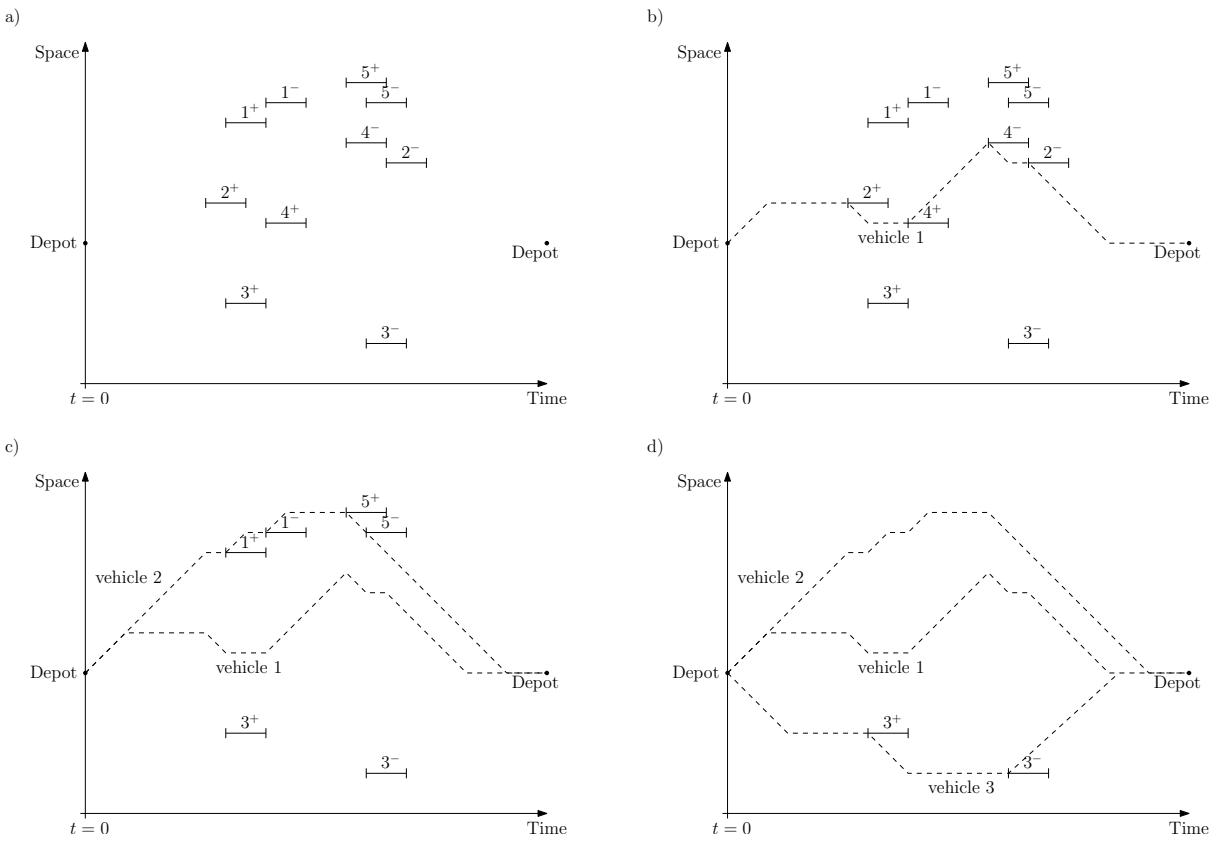

Figure 6 **A one-dimensional example of the approach to the multi-vehicle problem involving five customers and three vehicles. The first route is constructed by maximizing the number of customers that can be served by a single vehicle (Figure b). Then, the customers 2 and 4 served by the first vehicle are removed from the set of remaining customers and the process is repeated for the second vehicle (Figure c). Finally, a route is construced for the third vehicle, serving the last remaining customer 3 (Figure d).**

Each iteration starts with the *candidate* partition $(X_1, \ldots, X_m)$ and results in another partition $(X'_1, \ldots, X'_m)$. If the partition $(X'_1, \ldots, X'_m)$ is not feasible, the candidate partition $(X_1, \ldots, X_m)$ is marked as 'checked'. Otherwise, a feasible solution has been found and the algorithm is terminated.

If $(X'_1, \ldots, X'_m)$ is marked as 'checked', the algorithm proceeds to the next candidate partition that is not marked as 'checked' (see Section 2.2.2). Otherwise, $(X'_1, \ldots, X'_m)$ becomes the candidate partition, $(X_1, \ldots, X_m) \leftarrow (X'_1, \ldots, X'_m)$.

The main idea is that the first iteration (Algorithm 3) produces an *initial candidate partition* of customers, which is improved during later iterations. This procedure is motivated by the fact that the initial vehicle routes are constructed by attempting to serve as many customers as possible from the initial set $\{1, \ldots, n\}$ of all customers. After the initial routes have been determined, we know that at least the customers in X'_k can be served by vehicle k. Thus, for any set X, the number of served customers from the set $X \cup X'_k$ is *at least* equal to the number of customers in the set X'_k. A customer x that has relatively few admissible arcs to all nodes may be in a good position with respect to the customers assigned to a single vehicle k. That is, even if customer x was left out of the initial vehicle routes due to a small number of out admissible arcs to other nodes, x can be added to one of the routes during a later iteration.

2.2.2. A priori screening In order to detect infeasible instances without going through all partitions and to reduce the number of partitions that are considered, we propose the following procedure.

First, all routes involving a single customer are checked for feasibility. If any of these routes is infeasible, there exist no feasible solutions to the problem.

Second, the routes involving two customers are checked for feasibility. If there exist no feasible routes involving customers $\{i, j\}$, an arc between i and j is formed. This means that customers i and j can not be assigned to the same vehicle. Then, we find the *maximum clique* C within the set of customers, that is, the largest set of customers for which there is an arc between all pairs of nodes $i, j \in C$, see (Wood 1997). If the size of the maximum clique is greater than the number m of vehicles, the instance is infeasible since there are $|C| > m$ customers which should all be served by different vehicles.

Otherwise, since the customers in the maximum clique $C = \{c_1, \ldots, c_{|C|}\}$ all have to be assigned to different vehicles, with no loss of generality we may initially assign customer c_j to vehicle j for all $j \in \{1, \ldots, |C|\}$. Then, we determine the set of *feasible vehicles* V_i for each customer i as follows. 1) For $c_j \in \{c_1, \ldots, c_{|C|}\}$, we have $V_{c_j} = \{j\}$. For other customers $i \in \{1, \ldots, n\} \setminus C$, we initially define $V_i = \{1, \ldots, m\}$. 2) If $|V_j| = 1$ and there is an arc between j and i, customer i may not be assigned to the same vehicle as j, that is, $V_i \leftarrow V_i \setminus V_j$. This step is repeated for all $i, j \in \{1, \ldots, n\}$ until convergence.

After the set of feasible vehicles V_i has been determined for each customer $i \in \{1, \ldots, n\}$, the set of *a priori feasible partitions* is defined as follows.

Definition 2. A partition $(X_1, \ldots, X_m)$ of customers is feasible a priori, if $i \in X_k \Rightarrow k \in V_i$ for all $k \in \{1, \ldots, m\}$. The set of a priori feasible partitions is denoted by $\mathcal{P}'$.

Note that assigning each customer i a vehicle number $v_i \in V_i$ defines a unique partition. Since the vehicle number of customer i can be chosen from the set V_i of feasible vehicles, the number of a priori feasible partitions is given by $|\mathcal{P}'| = \prod_{i=1}^{n} |V_i|$.

Theorem 1. *The number of a priori feasible partitions satisfies* $|\mathcal{P}| \leq m^{n-|C|}$.

Proof. Since $|V_i| = 1$ for all $i \in C$, we get $|\mathcal{P}| = \prod_{i=1}^{n} |V_i| = \prod_{i \in \{1,\ldots,n\} \setminus C} \overbrace{|V_i|}^{\leq m} \prod_{i \in C} \overbrace{|V_i|}^{=1} \leq m^{n-|C|}$. □

The set of a priori feasible partitions is represented by the set $V_1 \times \ldots \times V_n$ and each a priori feasible partition is represented by a vector $(v_1, \ldots, v_n) \in V_1 \times \ldots \times V_n$. Information on checked partitions is stored in a sparse array 'CHECKED'. The partitions not marked as checked are included in the set of remaining partitions I (initially $I = V_1 \times \ldots \times V_n$). During run-time, each candidate partition is marked as 'checked' by setting CHECKED$(v_1, \ldots, v_n) = 1$ and removed from the set of remaining partitions, that is, $I \leftarrow I \setminus \{(v_1, \ldots, v_n)\}$. If the iteration results in a partition that is marked as 'checked', the next candidate partition is chosen randomly from the set of remaining partitions I. This way, the iteration described in Section 2.2.1 goes through all feasible partitions.

2.2.3. Filtering The algorithm can be accelerated by using information on previously calculated clusters as follows.

Theorem 2. *Let X denote a set of customers and $M(X)$ a maximum cluster of X. If X' is a subset of X satisfying $M(X) \subset X' \subset X$, the set $M(X)$ is a maximum cluster of X'.*

Proof. If $M(X)$ is not a maximum cluster of X', there exists a set $Y \subset X'$ for which $|Y| > |M(X)|$ and thus $M(X)$ is not a maximum cluster of X. □

In other words, for any set X' satisfying $M(X) \subset X' \subset X$, we do not need to run the single-vehicle maximum cluster algorithm to find a maximum cluster of X'.

Theorem 3. *Let $H = \{1, \ldots, n\}$ denote the set of all customers and $M(H)$ a maximum cluster of H. If Y is a set of customers served in a single route in $X \subset H$ satisfying $|Y| = |M(H)|$, the set Y is a maximum cluster of X.*

Proof. If Y is not a maximum cluster of X, there exists a route that serves customers $Y' \subset X$ for which $|Y'| > |Y| = |M(H)|$ and thus $M(H)$ is not a maximum cluster of H. □

Theorem 3 means that if the maximum cluster algorithm finds a route that serves as many customers as in the maximum cluster $M(H)$ of all customers, the algorithm can be terminated since there exist no larger clusters than $M(H)$.

3. Structure and complexity

The structure of the single-vehicle maximum cluster algorithm (Algorithm 1) is as follows. Briefly described, each recursion step involves checking the the feasibility of sequences (S, i, j), where $i, j \in R_S$.

Clearly, this process has a complexity of order $O(|R|^2)$, where $|R_S|$ is the number of remaining nodes.

The overall complexity of finding a feasible solution in the single-vehicle case is strongly dependent on the number of recursion steps needed to find the solution. Finding a feasible route involving N customers involves at least $2N$ recursion steps: At first, the number of remaining nodes is $2N$ and each time a feasible node is added to the end of the service sequence, the number of remaining nodes is decreased by one (see Figure 6). In the best case, no infeasible states are encountered and thus there are $2N$ recursion steps (one for each node added to the sequence). At step $s \in \{1, \dots, 2N\}$, the number of remaining nodes is $2N + 1 - s$ and thus the complexity is of order $O(\sum_{i=1}^{2N}(2N + 1 - s)^2) = O(N^3)$.

The efficiency of the detection of infeasible branches has a significant effect on the performance of the single-vehicle algorithm. The computational work is linearly increased with the number of dead ends $A(N)$ encountered during the recursion. The overall complexity of the single-vehicle algorithm is thus $O(N^3 + A(N))$. The number of dead ends encountered satisfies $A(N) \leq (2N)!/2^N$. If the heuristic version of the algorithm is used (see Section 2.1.6), we have $A(N) \leq L$.

The multi-vehicle algorithm involves solving the single-vehicle case for each vehicle in r iterations. Thus the complexity is bounded above by $O\left(rm(n^3 + A(n))\right)$. Note that the complexity is in general lower since the size of the set of customers is n only for the first vehicle in the first iteration. For the other vehicles, the number of customers is smaller since the customers that are already included in another vehicle route are not a part of the subproblem. For example, if the customers were divided equally among vehicles a priori, the complexity would be of order $O\left(rm((n/m)^3 + A(n/m))\right) = O\left(r(n^3/m^2) + rA(n/m)\right)$.

Since the total number of possible partitions is equal to the Stirling number of the second kind, the number of iterations r needed to find a feasible solution or to prove infeasibility satisfies $r \leq S_2(n, m) = \frac{1}{m!}\sum_{i=0}^{m}(-1)^i\binom{m}{i}(m-i)^n$. Given the maximum clique C (see section 2.2.2), we have $r \leq |\mathcal{P}'| = \prod_{i=1}^{n}|V_i| \leq m^{n-|C|}$.

In summary, the worst case complexity is high especially when the set of a priori feasible partitions $\mathcal{P}'$ is large. However, as will be seen in the next section, solutions to problems with loose constraints are usually found with little effort. On the other hand, tight constraints reduce the set of a priori feasible partitions $\mathcal{P}'$ and thus also the complexity of the algorithm.

4. Numerical experiments

In the following, we compare the performance of the multi-vehicle maximum cluster (MC) algorithm presented in Section 2.2 with two existing solution methods, namely, a tabu search algorithm (Cordeau and Laporte 2003a) and a constraint programming (CP) algorithm (Berbeglia 2009).

Berbeglia (2009) presents computational tests to compare the tabu and CP algorithms. We repeated these tests for the maximum cluster algorithm and compared our results with the ones presented in (Berbeglia 2009). The maximum cluster algorithm was implemented in Matlab and the tests were performed on a 2.2 GHz Dual Core Intel PC. The tabu and CP algorithms were tested on a 2.5 GHz Dual Core AMD Opteron computer (Berbeglia 2009).

In the studied instances the pick-up and drop-off points are located in a 20×20 square and the ride times between points (in minutes) are equal to Euclidean distances. The time windows have 15 minutes of length.

In the first set of instances marked with a, the capacity of a vehicle equals $Q = 3$, the load of each customer is 1 and the maximum ride time is $RT = 30$ minutes. In the second set marked with b, we have $Q = 6, RT = 45$ and the load associated to each customer is chosen randomly according to a uniform distribution on the set $\{1, \dots, Q\}$. In both sets, the service time is proportional to the number of passengers, namely $d_i = d_{n+i} = q_i$. The instances are described in more detail in (Cordeau 2006, Ropke et al. 2007).

In addition to the original sets a and b, the tests were performed on the modified sets presented in (Berbeglia 2009). In the first and second modifications, the maximum ride time equals $RT = 30$ and $RT = 22$,

respectively. The third modification is obtained by reducing the number of vehicles to 75% of the original number, rounded down to the nearest integer.

The results of the tests are shown in Table 4. The first column shows the instance labels of the form *am-n* or *bm-n*, where *m* indicates the number of vehicles and *n* corresponds to the number of customers. The other columns show the average time (in seconds) needed to solve the instances and the corresponding modifications by using the different solution methods, calculated over ten runs. The results obtained by the first two algorithms, tabu and CP, have been reported previously in (Berbeglia 2009, Berbeglia et al. 2011b)[2]. A number in parentheses indicates that the instance was proven to be infeasible, a dash indicates that a solution was not found in three minutes computing time and a star indicates that results for the instance have not been reported. For comparison, we have added our results obtained by the maximum cluster algorithm to the table. The Matlab-implementations of the maximum cluster algorithm are available at `http://math.tkk.fi/~lehame/exact/`. We first executed the heuristic version of the algorithm (see Section 2.1.6) with $L = 1$. If a solution was not found and the problem was not rendered infeasible by the a priori screening procedure described in Section 2.2.2, the exact version of the algorithm was executed.

By looking at the results obtained by the maximum cluster algorithm we see that most instances were solved within a fraction of a second. Except for a single modified instance (b7-84, 75% of vehicles), the maximum cluster algorithm produced a feasible solution or proved ineasibility in all instances. Note that the maximum cluster algorithm produced a feasible solution or proved infeasibility in the modified instances with six vehicles and 48 customers (b6-48), for which results have not been previously reported.

The CPU times obtained by the maximum cluster algorithm are typically of order ten times smaller compared to the results of the tabu and CP algorithms. The best improvement factor is $109.5/0.04 \approx 2700$ compared to CP (instance b5-60) and $78.5/0.06 \approx 1300$ compared to tabu (instance a4-48, 75% of vehicles). Although the algorithms were tested on different platforms, the results seem to justify the efficiency of the maximum cluster algorithm on the test instances. We acknowledge that there are instances in wich tabu and CP produced a feasible solution or proved infeasibility faster than the maximum cluster algorithm. However, the results suggest that the multi-vehicle maximum cluster algorithm may have practical importance since it is capable of handling large problems in short computation times.

The lower part of Table 4 shows the computational profile of the maximum cluster algorithm for the instance b8-96. The profile shows that most of the run time of the algorithm is consumed in the feasibility screening phase. Thus, in order to further optimize the run times of the maximum cluster algorithm, one might attempt to reduce the computational effort of the feasibility screening phase.

5. Conclusions

In this work, an exact constraint satisfaction algorithm for the multi-vehicle dial-a-ride problem is suggested. The purpose of the method is 1) to produce a solution that satisfies the constraints of the problem or 2) to prove infeasibility. In dynamic demand-responsive transport services, the algorithm can be used for deciding whether to accept or reject incoming user requests in real-time. The algorithm can also be used as a preprocessing step in a static problem setting.

The multi-vehicle solution is based on a recursive single-vehicle algorithm that maximizes the number of served customers in a given set. The single-vehicle algorithm is based on looking two steps ahead at each recursion step: The next decision is determined by the possibilities of making feasible decisions *after* the next one. This single-vehicle subroutine is successively executed for all available vehicles until each customer is assigned to a vehicle route or the problem is seen to be infeasible.

The multi-vehicle algorithm is compared to existing solution approaches (Cordeau and Laporte 2003a, Berbeglia 2009) by means of numerical tests, justifying the efficiency of the solution approach.

Acknowledgments

This work was partly supported by the Finnish Funding Agency for Technology and Innovation, Finnish Ministry of Transport and Communications and Helsinki Region Transport.

[2] The computing time for the instance 'b6-48' has been reported for the tabu algorithm in (Ropke et al. 2007).

Instance	Original			RT=30			RT=22			75 % of vehicles		
	Tabu	CP	MC	Tabu	CP	MC	Tabu	CP	MC	Tabu	CP	MC
a4-40	0.8	0.5	0.02	0.8	0.5	0.02	0.5	0.3	0.05	1.7	0.3	0.64
a4-48	1.0	0.5	0.05	1.0	0.5	0.05	1.3	0.4	0.05	78.5	0.6	0.08
a5-40	0.3	0.3	0.02	0.3	0.3	0.02	0.3	0.3	0.02	1.3	0.3	0.02
a5-50	0.7	0.5	0.04	0.7	0.5	0.04	0.6	0.6	0.03	3.9	1.3	2.0
a5-60	1.4	1.0	0.05	1.4	1.0	0.05	1.5	0.9	0.05	-	24.5	64.7
a6-48	0.4	0.6	0.03	0.4	0.6	0.03	0.4	0.7	0.03	1.1	0.5	0.07
a6-60	1.0	5.6	0.04	1.0	5.6	0.04	-	(1.1)	(0.63)	11.6	6.2	10.7
a6-72	1.9	5.0	0.06	1.9	5.0	0.06	-	(1.7)	(0.77)	5.7	2.0	0.90
a7-56	0.5	1.8	0.04	0.5	1.8	0.04	-	(0.6)	(0.40)	0.9	1.5	0.06
a7-70	1.7	41.5	0.06	1.7	41.5	0.06	1.4	7.6	0.06	3.2	5.1	0.06
a7-84	2.7	3.4	0.08	2.7	3.4	0.08	3.0	4.1	0.08	7.5	3.5	0.07
a8-64	0.8	6.2	0.05	0.8	6.2	0.05	-	(1.9)	(0.84)	1.1	2.5	0.04
a8-80	1.5	8.5	0.07	1.5	8.5	0.07	1.8	6.0	0.07	3.0	3.6	0.07
a8-96	3.5	37.7	0.11	3.5	38.7	0.11	-	(3.7)	(1.72)	8.1	5.3	0.15
b4-40	0.6	0.4	0.02	0.4	0.3	0.05	0.4	0.3	0.05	-	(0.1)	(0.40)
b4-48	1.3	0.4	0.03	1.6	0.4	0.08	-	(0.1)	(0.32)	-	(0.1)	(0.59)
b5-40	0.4	0.3	0.03	0.4	0.4	0.03	-	(0.1)	(0.27)	-	(0.1)	(0.37)
b5-50	1.2	6.8	0.06	0.9	0.8	0.05	1.1	0.9	0.18	-	(0.1)	(0.58)
b5-60	1.5	109.5	0.04	1.8	3.1	0.04	1.6	1.2	0.04	-	(0.1)	(0.78)
b6-48	0.3	*	0.02	*	*	0.02	*	*	0.02	*	*	(0.54)
b6-60	0.9	5.1	0.04	1.5	1.5	0.04	1.0	1.4	0.04	5.3	3.8	3.2
b6-72	2.1	27.3	0.05	2.3	2.4	0.06	2.4	2.4	0.05	9.7	25.6	5.1
b7-56	0.5	13.3	0.04	0.6	1.5	0.03	0.5	1.5	0.04	2.1	3.9	0.19
b7-70	1.4	5.6	0.08	1.6	18.4	0.08	1.3	2.7	0.05	12.6	78.7	4.3
b7-84	3.0	25.8	1.60	2.9	6.3	0.08	-	(0.1)	(0.64)	-	-	-
b8-64	0.7	12.7	0.09	0.8	1.9	0.09	0.8	1.9	0.04	1.8	4.8	0.98
b8-80	1.9	23.9	0.07	1.8	19.2	0.07	-	(0.5)	(1.28)	4.0	13.9	0.27
b8-96	4.1	149.1	0.12	3.9	34.8	0.12	3.9	56.1	0.15	8.2	-	0.74

Computational profile of the maximum cluster algorithm for a5-60, 75% of vehicles	
Single-vehicle subroutine (89.8%)	Other (10.2%)
- Feasibility screening (> 89.1 %)	- Initialization - List operations

Table 2 **Comparison between a tabu search algorithm (Cordeau and Laporte 2003a), a constraint programming algorithm (Berbeglia 2009) and the maximum cluster algorithm. The results obtained by the first two algorithms are given in (Berbeglia 2009, Berbeglia et al. 2011b). The upper table shows the average time (in seconds) needed to solve the instance of the dial-a-ride problem in the first column by using the different solution methods, calculated over ten runs. The times without parentheses indicate that a feasible solution was found and the times in parentheses indicate that the instance was proven to be infeasible. A star (*) indicates that the computing time has not been reported. The lower table shows the computational profile of the maximum cluster algorithm for the instance b8-96. The profile shows that most of the run time of the algorithm is consumed in the feasibility screening phase.**

References

Berbeglia, G. 2009. *Complexity Analyses and Algorithms for Pickup and Delivery Problems*. Ph.D. Thesis, HEC Montreal.

Berbeglia, G., J.-F. Cordeau, G. Laporte. 2011a. A hybrid tabu search and constraint programming algorithm for the dynamic dial-a-ride problem. Forthcoming.

Berbeglia, G., G. Pesant, L.-M. Rousseau. 2011b. Checking feasibility of the dial-a-ride problem using constraint programming. Forthcoming.

Bianco, L., A. Mingozzi, S. Ricciardelli, M. Spadoni. 1994. Exact and heuristic procedures for the traveling salesman problem with precedence constraints, based on dynamic programming. *INFOR* **32** 19–31.

Bräysy, O., M. Gendreau. 2005a. Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation Science* **39**(1) 104118.

Bräysy, O., M. Gendreau. 2005b. Vehicle routing problem with time windows, part ii: Metaheuristics. *Transportation Science* **39**(1) 119139.

Cordeau, J.-F. 2006. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research* **54** 573–586.

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

Cordeau, J.-F., G. Laporte. 2007c. The dial-a-ride problem: Models and algorithms. *Annals of Operations Research* **153** 29–46.

Cordeau, J.-F., G. Laporte, J.-Y. Potvin, M.W.P. Savelsbergh. 2007a. Transportation on demand. *Transportation*. Amsterdam: North-Holland, 429–466.

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

Diana, M., M.M. Dessouky. 2004. A new regret insertion heuristic for solving large-scale dial-a-ride problems with time windows. *Transportation Research Part B* **38** 539–557.

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

Groër, C., B. Golden, E. Wasil. 2010. A library of local search heuristics for the vehicle routing problem. *Mathematical Programming Computation* **2**(2).

Häme, L.E. 2011. An adaptive insertion algorithm for the single-vehicle dial-a-ride problem with narrow time windows. *European Journal of Operational Research.* **209**(1) 11 – 22.

Helsinki Region Traffic. 2010. Jouko neighbourhood routes and service routes. URL `http://www.hsl.fi/EN/passengersguide/ServiceroutesandJoukoroutes/Pages/default.aspx`.

Hernández-Pérez, H., J.-J. Salazar-González. 2009. The multi-commodity one-to-one pickup-and-delivery traveling salesman problem. *European Journal of Operational Research* **196** 987–995.

Hunsaker, B., M. W. P. Savelsbergh. 2002. Efficient feasibility testing for dial-a-ride problems. *Operations Research Letters* **30** 169–173.

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

Jokinen, Jani-Pekka, Teemu Sihvola, Esa Hyytiä, Reijo Sulonen. 2011. Why urban mass demand responsive transport? *IEEE Forum on Integrated and Sustainable Transportation Systems (FISTS)*. Vienna, Austria.

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

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

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

Ropke, S., J.-F. Cordeau, G. Laporte. 2007. Models and branch-and-cut algorithm for pickup and delivery problems with time windows. *Networks* **49** 258–272.

Sexton, T. 1979. *The single vehicle many-to-many routing and scheduling problem*. Ph.D. dissertation, SUNY at Stony Brook.

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

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

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

Wong, K.I., M.G.H. Bell. 2006. Solution of the dial-a-ride problem with multi-dimensional capacity constraints. *International Transactions in Operational Research* **13** 195–208.

Wood, D. R. 1997. An algorithm for finding a maximum clique in a graph. *Operations Research Letters* **21**(5) 211–217.