Effective Pruning Heuristics for the Fixed Route Dial-a-Ride Problem

Tal Grinshpoun · Elad Shufan · Hagai Ilani · Vadim Levit · Haya Brama

Abstract The fixed route dial-a-ride problem (FRDARP) is a variant of the famous dial-a-ride problem, in which all the requests are chosen between terminals that are located along a fixed route. A reduction to the shortest path problem has been proposed for finding an FRDARP optimal solution. However, the basic graph construction ends up with a huge graph, which makes the reduction impractical due to its memory consumption. To this end, we propose several pruning heuristics that enable us to considerably reduce the size of the graph through its dynamic construction. Our experiments show that each of the proposed heuristics on its own improves the practical solvability of FRDARP. Moreover, using them together is considerably more efficient than any single heuristic.

Keywords Timetabling in Transport · DARP · Pruning Heuristics

Tal Grinshpoun
Department of Industrial Engineering and Management, Ariel University, Ariel, Israel
E-mail: talgr@ariel.ac.il

Elad Shufan
Physics Department, SCE – Shamoon College of Engineering, Ashdod, Israel
E-mail: elads@sce.ac.il

Hagai Ilani
Department of Industrial Engineering and Management, SCE – Shamoon College of Engineering, Ashdod, Israel
E-mail: hagai@sce.ac.il

Vadim Levit
Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel
E-mail: vadim@bgu.ac.il

Haya Brama
Department of Industrial Engineering and Management, Ariel University, Ariel, Israel
E-mail: hayahartuv@gmail.com

1 Introduction

Dial-a ride is a flexible demand responsive transportation solution, in which a fleet of vehicles fulfills a set of transport requests [9,2]. The flexibility is usually in both the adjustable departure and pick-up times, as well as in the vehicle routes. A usual customer request in all the variants of the Dial-a-Ride Problem (DARP) contains a pick-up location, a destination station, and a desired pick-up or arrival time. The scheduler groups customers into specific vehicles, and determines the vehicle routes and timetable. The objective function is usually related to minimizing the solution cost, and may also include the aim to increase customer satisfaction. The problem of route determination in DARP makes the problem hard. Vehicle routing problems, for example the traveling salesman problem, are known to be NP-hard [5].

In a previous work, we have presented a DARP transportation solution that neutralizes the difficulty in finding the vehicle routes by determining the route in advance [7]. The customers can be picked-up from stations that are located along a known route. We termed this DARP variant a Fixed Route DARP (FRDARP). With a fixed route, the remaining problem is of grouping customers together and scheduling the timetable. The DARP solutions, and in particular the solution we offer, are suitable for a variety of transportation needs, including dedicated solutions for low-populated areas, and also for customers with special needs, such as children or elderly [2,4].

In the FRDARP problem, each customer requests to be transported between two terminals along the fixed route, at requested times. There are two types of requests, called s-type and r-type. S-type requests have a deadline. An s-type customer who requested to leave at a certain time cannot leave later than that time, but could leave earlier. In the latter case, the customer will reach the destination sooner than ideally wanted. R-type requests have a release time. An r-type customer who requested to leave at a certain time cannot leave before the requested time, but can leave at a later time. The (positive) difference between the requested time and the real departure time is called the waiting time of the passenger. Naturally, the passengers expect it to be as minimal as possible. The aim in the presented FRDARP is to minimize the sum of all the waiting times, for a given operational cost. The cost is determined by the number of times a vehicle leaves the depot for its round trip. The scheduler receives as input the requests, the vehicles' working hours, and the number of round transports. She is then expected to find a schedule that will increase customer satisfaction by minimizing the total waiting time.

In [7] we have presented a polynomial algorithm to solve the FRDARP by a reduction to the shortest path problem [8]. Based on the problem input, we dynamically construct a graph. A shortest weighted path in this graph, which starts at a source node and ends at a goal node, corresponds to an optimal schedule. In Section 2 we fully describe the FRDARP and the reduction.

Though the presented method for solving the FRDARP is polynomial, its implementation involves construction of huge graphs. As elaborated in Section 3, this makes it difficult to find an optimal solution when the number

of requests becomes large. The main contribution of this work is in obtaining heuristic algorithms for improving the implementation of the solution method. Using these heuristics considerably reduces the graph size, without compromising the optimality of the solution. Improvement is done by setting pruning rules that avoid creating duplicate representations of transports, or pruning rules that do not develop an arc, when it is clear that it will not participate in an optimal path. In Section 4 we present the pruning heuristics. Section 5 is dedicated to experiments showing the improvement that is obtained by using these heuristics.

2 FRDARP: the model and its relation to the shortest path problem

2.1 Problem input

The problem input consists of:

- A vehicle fleet of size M. Vehicle m has capacity C_m ($m = 1, 2, \ldots, M$). It starts working at time a_m and finishes working at time b_m.
- A circular route with L terminals, including the depot. In each round, which is also termed transport, a vehicle starts at a depot A_0, goes through terminals $A_1, A_2, \ldots, A_{L-1}$ in this order, and finally returns back to the depot, which is also denoted A_L. The ride time from A_i to A_j is denoted D_{ij}. The total transport time is $D = D_{0L}$.
- A declared number of transports, which is denoted K. The cost of a transport is assumed to be vehicle-independent.
- Ride requests. A request is determined by its type, either s-type or r-type, a pick-up terminal A_i, a destination terminal A_j, and a desired pick-up time, which is denoted s^{ij} or r^{ij}, for s-type and r-type requests, respectively. The total number of requests is called N.

The ride requests are collected into sets. The set of all the s-type requests from terminal A_i to terminal A_j is denoted S^{ij}. The size of the set is denoted $N_S^{ij} = |S^{ij}|$. An element $s^{ij} \in S^{ij}$ has a value, which is the desired pick-up time of this request. We also define a normalized value $\bar{s}^{ij} = s^{ij} - D_{0i}$; in order to pick a customer from A_i at time s^{ij} the vehicle should depart from the depot at time $\bar{s}^{ij}$. The normalized values form the set $\bar{S}^{ij}$. Note that the elements in the described sets are the requests; two different requests are considered as two different elements, even when they have the same value. The elements of each set are indexed according to their value in a non-increasing order, e.g., if $S^{ij} = \{s_1^{ij}, s_2^{ij}, \ldots, s_{N_S^{ij}}^{ij}\}$ then $s_1^{ij} \le s_2^{ij} \le \cdots \le s_{N_S^{ij}}^{ij}$. The letter l is usually used for the index of an s-type request: s_l^{ij}. We similarly define R^{ij}, $\bar{R}^{ij}$, N_R^{ij}, and the indexed elements r_h^{ij} and $\bar{r}_h^{ij}$, with $h = 1, 2, \ldots, N_R^{ij}$. Finally, we define the sets $R = \cup_{i,j} R^{ij}$, $S = \cup_{i,j} S^{ij}$, and $Q = S \cup R$.

2.2 A feasible solution for FRDARP

An FRDARP solution $\mathcal{S}$ is a triplet $(\mathcal{P}, \mathcal{V}, \mathcal{T})$, where:

- $\mathcal{P} = (P_1, P_2, \ldots, P_K)$ is a partition of the set of Q into K disjoint sets. The partition part P_i represents the requests that are fulfilled by the ith transport. For simplicity, a partition part is also called a transport.
- $\mathcal{V} = (v_1, v_2, \ldots, v_K)$ is a list of vehicle indices. Each vehicle is numbered from 1 to M and, according to $\mathcal{V}$, the ith transport is operated by vehicle number v_i.
- $\mathcal{T} = (t_1, t_2, \ldots, t_K)$ is a list of departure times. The ith transport departs from A_0 at t_i.

A solution $\mathcal{S} = (\mathcal{P}, \mathcal{V}, \mathcal{T})$ is feasible if it satisfies the following capacity, partitioning, and time constraints:

1. Capacity constraint: The number of passengers on vehicle v_i that operates transport P_i, at any part of the ride, cannot exceed C_{v_i}, the vehicle's capacity. Note that $|P_i|$, the total number of passengers that share a transport, may be larger than C_{v_i} because passengers are using only a part of the circular route.
2. SR constraint: A request $\bar{s}^{ij}$ can share a ride with a request $\bar{r}^{i'j'}$ only if $\bar{r}^{i'j'} \leq \bar{s}^{ij}$. This leads to the following partitioning constraint, for a mixed transport, i.e., one that contains both s-type and r-type requests. Consider a mixed transport P_i. Denote the latest normalized r-type request in P_i by $r_{\texttt{last}} = \max_{\bar{r} \in P_i}\{\bar{r}\}$, and the earliest s-type normalized request by $s_{\texttt{first}} = \min_{\bar{s} \in P_i}\{\bar{s}\}$. The transport is feasible only if $r_{\texttt{last}} \leq s_{\texttt{first}}$.
3. Time constraints: The departure time t_i of a feasible transport P_i must satisfy $r_{\texttt{last}} \leq t_i \leq s_{\texttt{first}}$. An additional time constraint regards the working hours of the vehicle, i.e., $a_{v_i} \leq t_i \leq b_{v_i} - D$. Another time constraint regards two transports that are operated by the same vehicle. If $v_i = v_j$ (with $i \neq j$), then the respective departure times must satisfy $|t_i - t_j| \geq D$.

In the next section we define the objective function and discuss properties of an optimal solution. We close this section by an example.

Example 1 Consider an FRDARP problem with $M = 2$ vehicles of capacity $C_1 = C_2 = 2$, which are available from $a_1 = a_2 = \texttt{8:00}$ until $b_1 = b_2 = \texttt{16:00}$. The route consists of $L = 3$ terminals (including the depot), $A_0 \rightarrow A_1 \rightarrow A_2 \rightarrow A_3$, and takes 90 minutes to complete. The number of transports is set to $K = 3$. The normalized requests are shown in Table 1.

$\bar{s}_1^{01}$	$\bar{s}_1^{02}$	$\bar{s}_1^{13}$	$\bar{r}_1^{01}$	$\bar{r}_1^{12}$	$\bar{s}_1^{23}$	$\bar{r}_1^{23}$	$\bar{s}_2^{13}$	$\bar{r}_1^{13}$	$\bar{r}_2^{23}$
8:45	9:45	10:15	10:15	10:30	10:45	12:00	12:45	12:45	13:15

Table 1 Ride requests normalized to the respective departure times from the depot

The transport $\{s_1^{01}, s_1^{02}, s_1^{13}\}$ is an example of a feasible transport. Though the capacity of the vehicle is two, it can accommodate these three requests,

because s_1^{01} frees her seat at terminal A_1, which allows s_1^{13} to join the ride. This transport should leave the depot not later than `8:45` in order to meet the deadline requirement of $s_{\texttt{first}} = \bar{s}_1^{01}$. The transport $\{s_1^{13}, s_1^{23}, s_2^{13}\}$ is not feasible, as it violates the capacity constraint. The transport $\{s_1^{13}, r_1^{01}, r_1^{12}\}$ is an example of a feasible mixed transport that must depart exactly at `10:15`. The transport $\{s_1^{23}, r_1^{23}\}$ is an example of a non-feasible transport, because it violates the SR constraint. An example of a feasible solution is $\mathcal{S} = (\mathcal{P}, \mathcal{V}, \mathcal{T})$, with $\mathcal{P} = (\{s_1^{01}, s_1^{02}, s_1^{13}, s_1^{23}\}, \{r_1^{01}, r_1^{12}, r_1^{23}, s_2^{13}\}, \{r_1^{13}, r_2^{23}\})$, $\mathcal{V} = \{1, 2, 1\}$, and $\mathcal{T} = \{\texttt{8:45}, \texttt{12:00}, \texttt{13:15}\}$.

2.3 The objective function and optimal solution properties

A given solution defines for each passenger, either s^{ij} or r^{ij}, an actual departure time, denoted $\texttt{adt}(q^{ij})$, where q may be either s or r. The waiting time of a request is the difference between the desired and the actual departure times: $w(q^{ij}) = |q^{ij} - \texttt{adt}(q^{ij})|$. Our aim is to minimize the sum over the waiting times of all the passengers, i.e., the objective function is $W = \sum_{q \in Q} w(q)$.

Definition 1 A solution is called S-ordered if for any two terminals A_i and A_j, and any two requests $s_l^{ij}, s_{l'}^{ij} \in S^{ij}$ between these two terminals, $s_l^{ij} < s_{l'}^{ij} \Rightarrow \texttt{adt}(s_l^{ij}) \leq \texttt{adt}(s_{l'}^{ij})$.

We similarly define an R-ordered solution.

Definition 2 Given an FRDARP problem, we define *the set of optional departure times* (ODT) as a set whose elements are the following values:

1. $a_m + nD$, with $m = 1, 2, \ldots, M$.
2. $\bar{s}_l^{ij} \pm nD$, with $0 \leq i < j \leq L$, $1 \leq l \leq N_S^{ij}$.
3. $\bar{r}_h^{ij} \pm nD$, with $0 \leq i < j \leq L$, $1 \leq h \leq N_R^{ij}$.
4. $b_m - (n+1)D$, with $m = 1, 2, \ldots, M$.

In all of the above $n = 0, 1, \ldots, (K-1)$. In addition, the value in ODT must fit the working hours of the vehicles, i.e., they have to be larger than $\min_{1 \leq m \leq M}\{a_m\}$, and smaller than $\max_{1 \leq m \leq M}\{b_m\} - D$. In case one of the above values is not, it is deleted from the set.

In [7] we have shown that if FRDARP has a feasible solution, then it is always possible to find an optimal solution with the following properties:

1. It is both S-ordered and R-ordered.
2. For every $t \in \mathcal{T}$, $t \in ODT$, i.e, every vehicle leaves the depot in a departure time, which is an element of ODT.

The solution presented in Example 1 of Section 2.2 is an optimal solution with these two properties. We have reached it by a reduction of the FRDARP to the shortest path problem. The reduction procedure, which is based on these two properties, is described in the following section.

2.4 Reduction of FRDARP to a shortest path problem

In this section we show how to solve the FRDARP by a reduction to the problem of finding a shortest path in a directed and weighted graph G. The reduction has been previously introduced [7], and is inspired by a similar approach that we applied to an FRDARP with two terminals [8]. The graph is constructed dynamically, starting from a source node. Two nodes that are connected by an arc correspond to a possible transport. A path from the source node to one of several goal nodes corresponds to a feasible schedule of the respective FRDARP. The path that has the lowest total weight corresponds to an optimal schedule. The algorithm is polynomial in the number of requests, for a given number of terminals, vehicles and transports.

A node in G has $(L-1)(L+2)+M$ coordinates. The first $(L-1)(L+2)$ coordinates are indices, which are related to the $(L-1)(L+2)$ sets: R^{ij} and S^{ij}, where $0 \leq i < j \leq L$. The other M coordinates are related to the M vehicles. A node is a sequence $(\mathbf{h}^\frown\mathbf{l};\boldsymbol{\tau})$. The symbol $^\frown$ is used as a concatenation sign. $\mathbf{h} = (h^{ij})$ is a vector of indices that regard to r-type requests. For example, if $L = 3$ then $\mathbf{h} = (h^{01}, h^{02}, h^{12}, h^{13}, h^{23})$. Similarly, $\mathbf{l} = (l^{ij})$ is vector of indices that regard to the s-type requests. Finally, $\boldsymbol{\tau} = (\tau_1, \tau_2, \ldots, \tau_M)$ is vector of the availability times of the vehicles. A node $(\mathbf{h}^\frown\mathbf{l};\boldsymbol{\tau})$ indicates that all requests $\{r_1^{ij}, r_2^{ij}, \ldots, r_{h^{ij}}^{ij}\}$ and $\{s_1^{ij}, s_2^{ij}, \ldots, s_{l^{ij}}^{ij}\}$, for the respective i and j, have been handled, and that the vehicle m $(m = 1, 2, \ldots, M)$ is available for the next transport at time τ_m.

An arc connecting $(\mathbf{h}^\frown\mathbf{l};\boldsymbol{\tau})$ and $((\mathbf{h}+\boldsymbol{\Delta}\mathbf{h})^\frown(\mathbf{l}+\boldsymbol{\Delta}\mathbf{l});\boldsymbol{\tau}')$ represents a possible transport shared by $\{r_{h^{ij}+1}^{ij}, \ldots, r_{h^{ij}+\Delta h^{ij}}^{ij}\}$ and $\{s_{l^{ij}+1}^{ij}, \ldots, s_{l^{ij}+\Delta l^{ij}}^{ij}\}$, for all relevant pairs of i and j. The vector $\boldsymbol{\tau}'$ is different from $\boldsymbol{\tau}$ by only one component, the one that represents the available time of the vehicle, which operates this transport. The weight of the arc is its cost, i.e., the total waiting time of the passengers in the transport.

In order to describe the structure and the construction of the graph, we return to the FRDARP problem of Example 1. The source node in the constructed graph is $\texttt{node}_0 = (0,0,0,0,0,0,0,0,0,0;\texttt{8:00},\texttt{8:00})$, which means that none of the requests has been handled yet, and that both vehicles are available at `8:00`. Note that the first five (zero) coordinates are related to r-type requests, for the following ordered pairs of terminals: (01,02,12,13,23). This order is kept also for the other five coordinates, which are related to s-type requests. A goal node is of the form $(1,0,1,1,2,1,1,0,2,1;\tau^*,\tau^{**})$, where τ^* and τ^{**} might have any valid value smaller than b_1 and b_2, respectively. Arcs and nodes are created, starting from the source node, by two steps: (1) Choose a vehicle, create all the feasible transports, and represent each by the index coordinates; (2) Consider all relevant departure times and represent each possibility by a weighted arc leading to a node (with updated available time for the operating vehicle). For example, let us choose vehicle v_1 as a first operating vehicle, and consider a transport handling $\{s_1^{01}, s_1^{02}, s_1^{13}, s_1^{23}\}$. The representative 10 index coordinates are $(0,0,0,0,0,1,1,0,1,1)$. This transport should leave the depot between `8:00` to `8:45`. If there are more s-type requests

in the transport (as in this transport), than there is a dilemma. On one hand, the weight of the arc will be minimum if the transport will leave at the latest possible time. On the other hand, leaving earlier might be beneficial for the next considered transports. So, in the case of a majority of s-type requests, we need to generate nodes with all possible departure times from the ODT set, in the relevant time slot, which is, in the example, between `8:00` and `8:45`. When the vehicle departs at the latter time, as an example, it will complete its 90 minute route at `10:15`. Passengers s_1^{02}, s_1^{13} and s_1^{23} arrive at their destinations 60, 90 and 120 minutes ahead of time, respectively, thus their accumulated waiting time is 270 minutes. The node in the graph that represents the state after handling this transport is $\texttt{node}_1 = (0, 0, 0, 0, 0, 1, 1, 0, 1, 1; \texttt{10:15}, \texttt{8:00})$ and the cost of the arc $\texttt{node}_0 \rightarrow \texttt{node}_1$ is 270.

The described generating process (steps 1 and 2 above) continues from each node, which is connected to the source node by less than K arcs. When the graph is completed, a path of minimal weighted length from the source node to any goal node, using at most K arcs, represents an optimal schedule.

We continue the example by considering a possible second transport that accommodates requests r_1^{01}, r_1^{12}, r_1^{23} and s_2^{13} using vehicle v_2. This transport should leave the depot between `12:00` and `12:45` in order to meet the respective release and deadline requirements of $r_{\texttt{last}} = r_1^{23}$ and $s_{\texttt{first}} = s_2^{13}$. In this example, there are more r-type than s-type requests. When this is the case, there is no dilemma regarding the departure time of the vehicle. It should depart as early as possible, i.e., at `12:00` and finish at `13:30`. Passengers r_1^{01} and r_1^{12} will wait at their source terminals for 105 and 90 minutes, respectively, whereas s_2^{13} will arrive at her destination 45 minutes ahead of time. The node that represents the state after handling this ride is $\texttt{node}_2 = (1, 0, 1, 0, 1, 1, 1, 0, 2, 1; \texttt{10:15}, \texttt{13:30})$ and the cost of the arc $\texttt{node}_1 \rightarrow \texttt{node}_2$ is 240.

The careful reader might have noticed that the above described transports belong to the optimal schedule, which was presented in Section 2.2. The third transport accommodates the remaining requests r_1^{13} and r_2^{23} using vehicle v_1. It will depart from the depot at `13:15` (as early as possible). The node that represents the state after handling this ride is a goal node $\texttt{node}_3 = (1, 0, 1, 1, 2, 1, 1, 0, 2, 1; \texttt{14:45}, \texttt{13:30})$ and the cost of the arc $\texttt{node}_2 \rightarrow \texttt{node}_3$ is 30. The presented path $\texttt{node}_0 \rightarrow \texttt{node}_1 \rightarrow \texttt{node}_2 \rightarrow \texttt{node}_3$ is optimal for the FRDARP problem of Example 1 and represents a solution of cost 540.

3 Graph traversal

The original idea for solving FRDARP was to use the constructed graph of Section 2.4, and to run on it K iterations of the Bellman-Ford algorithm [1] in order to find the shortest path [7]. The Bellman-Ford algorithm traverses the graph similarly to *breadth-first search* (BFS) [3], and needs to maintain all the graph nodes in memory at all times.

We have recently attempted to implement the above solution method for a new transportation solution for the elderly that is based on FRDARP. We have found out that elderly people living in nursing facilities have transportation needs of rather limited dispersity, e.g., medical clinic, community center, and supermarket [4]. Therefore, a transportation solution with a flexible schedule and a circular route going through these places, starting and ending in the nursing facility (depot), may very well fit their needs. However, even though the FRDARPs to be solved were of rather limited size ($N \leq 20$ daily requests), the resulting graphs (on which the Bellman-Ford algorithm should run) turned out to be very large and the solver ran out of memory.

To this end, we propose to traverse the graph in *depth-first search* (DFS) [3] manner by using a classical Branch & Bound procedure. A major advantage of DFS traversal lies in its memory usage – in DFS there is no need to maintain the graph nodes in memory. Moreover, the Branch & Bound procedure reaches very quickly a first valid solution (not necessarily optimal) and can consequently use this solution as an initial bound. In order to understand the merits of holding such a bound, let us, for the sake of clarity, term a path consisting of less than K arcs that has not reached a goal node, as a *partial* path. Now, every time the cost of a partial path reaches the bound, the remainder of this path can be pruned. The bound is updated every time a new better solution is discovered.

The above basic pruning of the Branch & Bound procedure helps in some extent to reduce the search space, and serves as a good first step for reducing the actual graph size. In the next section, we propose several domain-specific heuristic that considerably enhance the effectiveness of pruning.

4 Pruning heuristics

We propose herein various heuristics for pruning the FRDARP search space. We separate the heuristics into two types according to their main motif of pruning – redundancy-removal heuristics and path-removal heuristics. Broadly speaking, the redundancy-removal heuristics (Sections 4.1 and 4.2) attempt to a priori identify multiple paths in the graph that ultimately lead to equivalent solutions, and consequently remove redundant paths. On the other hand, the path-removal heuristics (Sections 4.3 and 4.4) aim to a priori identify paths that lead to either infeasible solutions or solutions that are not optimal, and consequently completely remove them.

4.1 SR redundancy

Consider $\texttt{node}_{SR}$ that represents a state in which there are both s-type and r-type unhandled requests, and the earliest normalized unhandled request is an s-type request, denoted $s_{\texttt{first}}$. In this situation a single transport cannot possibly handle all the remaining requests, because $s_{\texttt{first}}$ prevents a mixed

transport with the r-type requests (SR constraint in Section 2.2). The *SR redundancy heuristic* prevents at $\texttt{node}_{SR}$ the development of any arc (transport) P_r that contains only r-type requests. It can do so, without compromising the completeness (optimality) of the solution, since developing transport P_r at this stage leads to redundant paths and can therefore be pruned.

Lemma 1 *Developing an arc for transport P_r at $\texttt{node}_{SR}$ is redundant.*

Proof Denote by P_s any arc (transport) that fulfills the early s-type request $s_{\texttt{first}}$. It may potentially include additional s-type requests, but not r-type requests, due to the SR constraint. So clearly any two potential transports P_s and P_r do not share any mutual requests. We consider two cases according to the vehicles v_s and v_r that operate the respective transports P_s and P_r:

- $v_s = v_r$: In case the same vehicle operates both P_s and P_r it will have to perform the P_s transport before P_r. This is because P_s must depart from the depot not later than $s_{\texttt{first}}$, and the r-type passengers in P_r must all depart later than $s_{\texttt{first}}$. Consequently, P_s should be developed before P_r.
- $v_s \neq v_r$: In case different vehicles operate P_s and P_r then the two transports have absolutely no effect on each other. Consequently, developing P_s at $\texttt{node}_{SR}$ followed by the development of P_r is exactly the same as developing P_r at $\texttt{node}_{SR}$ followed by the development of P_s.

Considering both cases, it is redundant to develop arc P_r at $\texttt{node}_{SR}$. □

Corollary 1 *The SR redundancy heuristic does not compromise the optimality of the obtained solution.*

To exemplify the SR redundancy heuristic, consider $\texttt{node}_0$, the source node in Example 1. $\texttt{node}_0$ is an example of $\texttt{node}_{SR}$, since it has unhandled requests of both types and the earliest is an s-type request ($s_{\texttt{first}} = s_1^{01}$). Now consider two possible transports, $P_s = \{s_1^{01}, s_1^{02}, s_1^{13}, s_1^{23}\}$ to be operated by v_1 and $P_r = \{r_1^{01}, r_1^{12}, r_1^{23}\}$ to be operated by v_2. In case we handle ride P_s first we reach $\texttt{node}_1 = (0,0,0,0,0,1,1,0,1,1;\texttt{10:15},\texttt{8:00})$, and after also handling ride P_r we reach $\texttt{node}_{2'} = (1,0,1,0,1,1,1,0,1,1;\texttt{10:15},\texttt{13:30})$. Conversely, if we start with ride P_r we reach $\texttt{node}_{1'} = (1,0,1,0,1,0,0,0,0,0;\texttt{8:00},\texttt{13:30})$, and after handling ride P_r we reach exactly the same $\texttt{node}_{2'}$ with exactly the same costs as in the first case. This redundancy is illustrated in Figure 1.

4.2 Equal capacity redundancy

Consider $\texttt{node}_{EC}$ in the graph and a potential transport P that can be operated by two or more vehicles of equal capacity C_{EC}. Denote by v_{first} the earliest available vehicle of capacity C_{EC}, with ties broken according to the vehicle index. The *equal capacity redundancy heuristic* prevents at $\texttt{node}_{EC}$ the development of more than a single arc for transport P with equal capacity vehicles, i.e., it considers at most a single vehicle v_{first} of each capacity

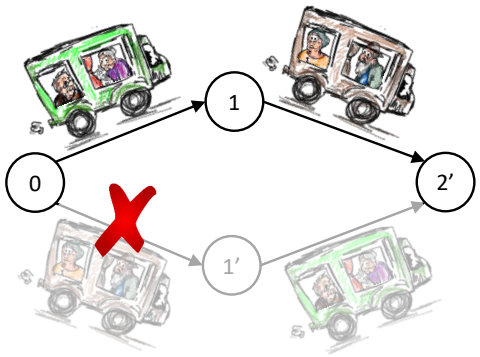

Fig. 1 SR redundancy illustration

C_{EC} to operate transport P. It can do so, without compromising the completeness (optimality) of the solution, since developing at this stage additional arcs of the same transport, operated by vehicles of the same capacity, leads to redundant paths and can therefore be pruned.

Lemma 2 *Developing an arc for transport P operated by a vehicle $v_{other} \neq v_{first}$ with capacity C_{EC} at $\texttt{node}_{EC}$ is redundant.*

Proof We consider two cases according to the availability times τ_{other} and τ_{first} of the respective vehicles v_{other} and v_{first}:

- $\tau_{other} = \tau_{first}$: The redundancy in this case is trivial, because there is absolutely no difference between vehicles v_{other} and v_{first},
- $\tau_{other} > \tau_{first}$: The only reason to prefer the later-available v_{other} over v_{first} is to free up vehicle v_{first} for operating some other transport P' that needs to departure earlier. However, this possibility would be accounted for anyway, because when the arc for transport P' will be considered it would be operated by vehicle v_{first} (for exactly the same reason that transport P is operated by v_{first}). As a consequence, transport P would be considered to be operated by vehicle v_{other} later on along that path. Thus, considering v_{other} for operating transport P at $\texttt{node}_{EC}$ is redundant.

According to the way that v_{first} was selected it holds that $\tau_{first} \leq \tau_{other}$, thus considering both above cases, it is redundant to develop an arc operated by v_{other} for transport P at $\texttt{node}_{EC}$. □

Corollary 2 *The equal capacity redundancy heuristic does not compromise the optimality of the obtained solution.*

To exemplify the equal capacity redundancy heuristic, consider $\texttt{node}_0$, the source node in Example 1. $\texttt{node}_0$ is an example of $\texttt{node}_{EC}$, since it has two vehicles of equal capacity ($C_1 = C_2 = 2$), both available ($\tau_1 = \tau_2 = \texttt{8:00}$). Now consider a possible transport $P = \{s_1^{01}, s_1^{02}, s_1^{13}, s_1^{23}\}$ to be operated by either v_1 (v_{first}) or v_2 (v_{other}). In case vehicle v_1 operates transport P we

reach $\texttt{node}_1 = (0,0,0,0,0,1,1,0,1,1;\texttt{10:15},\texttt{8:00})$. In case vehicle v_2 operates transport P we reach $\texttt{node}_{1'} = (0,0,0,0,0,1,1,0,1,1;\texttt{8:00},\texttt{10:15})$. Nodes $\texttt{node}_1$ and $\texttt{node}_{1'}$ are clearly equivalent. This redundancy is illustrated in Figure 2.

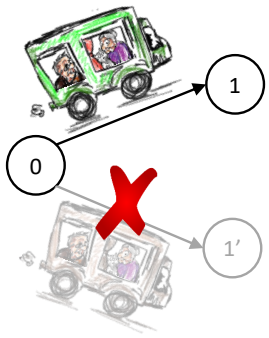

Fig. 2 Equal capacity redundancy illustration

4.3 Infeasible capacity path removal

Consider an arc $\texttt{node}_A \rightarrow \texttt{node}_B$, which is developed during the DFS traversal. We assume that the arc represents a feasible transport, i.e., it satisfies the SR and capacity constraints, and that it is not a goal node. The removal criterion that is described in this section accounts for arcs that, though feasible, will necessarily not be a part of a feasible solution due to capacity considerations.

The idea behind the *infeasible capacity path removal heuristic* is that for each developed arc there is an upper bound for the number of requests that can be fulfilled, in between any two consecutive terminals, by the later developed arcs. The heuristic prevents the development of arcs for which this bound exceeded. More specifically, the bound is determined by the number of remaining transports, and by the maximal capacity $C^* = \max\{C_1, C_2, \ldots, C_M\}$. When considering an arc $\texttt{node}_A \rightarrow \texttt{node}_B$ during DFS construction, we know the number of arcs leading to $\texttt{node}_B$ from the source node in the considered path. Denote this number by K_{path}. This defines the number of remaining transports $\tilde{K} = K - K_{path}$. The maximal number of requests that can be fulfilled in any particular segment of the route is restricted by $N_{max} = \tilde{K} \cdot C^*$.

Given a developed arc, we also know the number of remaining requests that should be fulfilled. Denote the integer coordinates of $\texttt{node}_B$ by $(\mathbf{h}^\frown\mathbf{l})$, where $\mathbf{h} = (h^{01}, h^{02}, \ldots, h^{L-1,L})$ and similarly $\mathbf{l} = (l^{01}, l^{02}, \ldots, l^{L-1,L})$. Then the number of *remaining* requests from terminal A_i to terminal A_j is given by $\tilde{N}^{ij} = N_S^{ij} + N_R^{ij} - (h^{ij} + l^{ij})$. Finally, the number of remaining requests between two consecutive terminals A_k and A_{k+1} is given by $\tilde{N}_k = \sum_{i=0}^{k} \sum_{j=k+1}^{L} \tilde{N}^{ij}$. If

the considered arc is part of a feasible path, then it must hold that $\tilde{N}_k \leq N_{max}$, for each $k = 0, 1, 2, \ldots, L-1$. This proves the following lemma.

Lemma 3 *An arc for which* $\max\{\tilde{N}_0, \tilde{N}_1, \ldots, \tilde{N}_{L-1}\} > N_{max}$ *can be removed from the graph that represents the respective FRDARP.*

Corollary 3 *The infeasible capacity path removal heuristic does not compromise the optimality of the obtained solution.*

To exemplify the infeasible capacity path removal heuristic, consider $\texttt{node}_0$, the source node in Example 1. $\texttt{node}_0$ is an example of $\texttt{node}_A$. Now consider $\texttt{node}_B$ whose integer coordinates are $(0, 0, 0, 0, 0, 1, 1, 0, 1, 0)$. The arc $\texttt{node}_A \rightarrow \texttt{node}_B$ represents the transport of $\{s_1^{01}, s_1^{02}, s_1^{13}\}$. The number of remaining transports is $\tilde{K} = 2$ and the maximal capacity is $C^* = 2$. Hence, the maximal number of requests that could be treated in a given route segment is $N_{max} = 4$. However, among the remaining seven requests, there are five requests that use the route segment between terminals A_2 and A_3, i.e., $\tilde{N}_2 = 5$. These are $s_1^{23}, r_1^{23}, s_2^{13}, r_1^{13}$, and r_2^{23}. Since $\tilde{N}_2 > N_{max}$ this arc can be removed from the graph.

4.4 A*-based path removal

The A* algorithm [6] is a classical graph traversal algorithm, often used for minimal path search. The classical A* traverses the graph in a *best-first search* [10] manner, which we do not use herein due to its space requirements (see Section 3). However, we can adopt the node expansion strategy of A* as an additional pruning heuristic.

When the A* algorithm needs to decide whether to expand a node in the graph, say $\texttt{node}_i$, it considers two values – the cost $g(\texttt{node}_i)$ of the path from $\texttt{node}_0$ (the source node) to $\texttt{node}_i$, and a heuristic estimate $h(\texttt{node}_i)$ of the cost of the optimal path from $\texttt{node}_i$ to any goal node. A heuristic $h(\cdot)$ is called *admissible* if its estimate is never larger than the real cost of the optimal path from $\texttt{node}_i$ to a goal. Therefore, when $h(\cdot)$ is admissible, the sum $g(\texttt{node}_i) + h(\texttt{node}_i)$ can serve as an optimistic view for the potential of the current partial path going through $\texttt{node}_i$. When $\texttt{node}_i$ is expanded, $g(\texttt{node}_i)$ is already known. Consequently, the challenge in achieving efficient pruning lies in the development of an admissible heuristic $h(\cdot)$ that is both fast to compute and tight, in the sense that its estimate is as close to the actual optimal value as possible.

We propose herein a heuristic $h(\cdot)$ that focuses on the set of yet unhandled requests $\tilde{Q} \subseteq Q$. The main idea is to look at the differences between the request times in $\tilde{Q}$. The differences between requests correspond to costs that would definitely be applied in case these requests share the same ride. However, considering all combinations of unhandled requests, including taking into account their types and pick-up/destination terminals, is computationally heavy. Therefore, we completely disregard the request types and pick-up/destination

terminals, and consider just a sequence of the unhandled ride requests $\tilde{Q}$, ordered according to their requested times. Relating only to consecutive elements is optimistic, as is required for the admissibility of the heuristic, for the following reasons. On one hand, such pairs of elements (passengers sharing a ride) are generally "cheaper" than non-consecutive elements. On the other hand, they may turn out to be infeasible, due to the disregard of request types (e.g., SR constraint) and terminals (e.g., capacity constraint).

```
   Input: set of unhandled requests Q̃, number of remaining transports K̃
   Output: value of h(·)
 1 sort Q̃ according to the request times
 2 Δ = multiset of time differences between consecutive elements in Q̃
 3 sort Δ in reverse order
 4 remove the first (highest) K̃ − 1 elements from Δ
 5 initialize h_val and factor to zero
 6 for index = 0 . . . (|Δ| − 1) do
 7 |   if index mod 2 · K̃ = 0 then
 8 |   |   increment factor by 1
 9 |   end
10 |   increment h_val by Δ(index) · factor
11 end
12 return h_val
```

Algorithm 1: Computation of $h(\cdot)$

The computation of our proposed $h(\cdot)$ is given in Algorithm 1. It starts with the sorting of $\tilde{Q}$ (line 1). Then, in line 2, a multiset Δ is generated, consisting of the request time differences between every pair of consecutive elements in $\tilde{Q}$. (Δ is a multiset because it may contain several repetitions of the same time difference.) Next, Δ is sorted in reverse order (line 3) followed by the removal of its first (highest) $\tilde{K} - 1$ elements (line 4). $\tilde{K}$ remaining transports mean that $\tilde{K} - 1$ time-request differences can be removed without cost, because their respective passengers can be separated to different transports. For the sake of admissibility we remove the most costly $\tilde{K} - 1$ differences.

Two variables are introduced and initialized in line 5 – h_{val}, the value of $h(\cdot)$ that needs to be computed, and $factor$ that represents the minimal number of times a cost difference should be accounted for in the computation of h_{val}. To understand the concept of $factor$ consider an `adt` of a ride that is earlier than the ride time requests of three (s-type) passengers on that ride. The difference between the `adt` and the latest of these three request times will be accounted for three times, once for the cost of each passenger, i.e., the $factor$ should be 3 in this case. However, the difference between the request times of the first and second passengers will only be accounted for once for the cost of the first passenger, i.e., the $factor$ should be 1 in this case.

Now we continue to the main loop (lines 6-11), which visits all the remaining elements of Δ (after the removal in line 4). In $index = 0$ and later in every $2 \cdot \tilde{K}$ iterations the $factor$ increases by 1 (lines 7-8). Consider again the time

difference between the requests of the first and second passengers from the last example. It has a $factor$ of 1 because it is in the "border" of the ride (no passenger with a later request than the first). However, there may also be a similar border for r-type requests of earlier time than `adt`. Moreover, any remaining transport of the $\tilde{K}$ can have two such borders. Thus, for sake of admissibility we consider the maximal number of possible time differences accounted for before the $factor$ grows. Next, in line 10, each time-difference value in Δ is multiplied by the corresponding $factor$ at that time of the computation, before being added to h_{val}. Naturally, for the optimistic scenario needed herein, we must consider that the high differences (high costs) are multiplied by low $factor$ values, and vice versa, and this is indeed achieved by starting with the highest values in Δ (recall the reverse ordering in line 3). Finally, in line 12, the accumulated h_{val} is returned as the output of the computation.

Lemma 4 *The $h(\cdot)$ heuristic of Algorithm 1 is admissible.*

Proof In Algorithm 1 we relate to the unhandled requests $\tilde{Q}$ without regarding their types and terminals. By disregarding the terminals, the problem becomes a simple case of the *two-campus transport problem* (TCTP) [8]. It has been proven that there always exists an optimal solution that groups together in a ride only consecutive request times (according to each type) [8, Theorem 1]. In $\tilde{Q}$ we also disregard the types, so now grouping together only consecutive request times is definitely optimistic (though probably infeasible). As a consequence, we can relate only to the costs of consecutive ride requests, i.e., their time differences (Δ). A best-case optimal solution may (i) potentially rule-out the most costly differences, and (ii) repeat as few times as possible the costs of the remaining costly differences. This is exactly what happens in line 4, and in the loop of lines 6-11, respectively. Consequently, the returned h_{val} serves as an optimistic lower bound to the actual cost for the remaining requests $\tilde{Q}$ and transports $\tilde{K}$. □

Following the admissibility of $h(\cdot)$, the A*-*based path removal heuristic* does not expand any $\texttt{node}_i$ for which $g(\texttt{node}_i) + h(\texttt{node}_i) \geq UB$, where UB is the cost of the best solution found so far. The upper bound UB is maintained as part of the Branch & Bound procedure (Section 3).

Corollary 4 *The* A*-*based path removal heuristic does not compromise the optimality of the obtained solution.*

To exemplify the computation of $h(\cdot)$ in the A*-based path removal heuristic, consider $\texttt{node}_0$, the source node in Example 1. Given that $\texttt{node}_0$ is a source node, we have $\tilde{Q} = Q$ and $\tilde{K} = K = 3$. The ride requests Q are already ordered (line 1) in the presentation of Table 1, so we continue to present in Table 2 the corresponding Δ and ordered Δ (lines 2 and 3).

After the execution of line 4, the two $(\tilde{K} - 1)$ highest costs (75 and 60) are removed from Δ. Their remain 7 elements in Δ to be considered in the main loop (lines 6-11). The $factor$ is incremented every six $(2 \cdot \tilde{K})$ iterations,

Δ	60	30	0	15	15	75	45	0	30
Ordered Δ	75	60	45	30	30	15	15	0	0

Table 2 Δ and ordered Δ in the computation of $h(\texttt{node}_0)$

so only the last element, which is zero anyway, is multiplied by $factor = 2$. So the computed h_{val} is $(45 + 30 + 30 + 15 + 15 + 0) \cdot 1 + 0 \cdot 2$, resulting in $h(\texttt{node}_0) = 135$. Note that in case this was not a source node, the cost $g(\texttt{node})$ of the path until `node` would have been added for the comparison with the current UB.

5 Experimental evaluation

We have implemented an FRDARP solver based on Branch & Bound (Section 3) and all the presented pruning heuristics (Section 4) in Java. In the experiments we evaluate six versions of the algorithm – a basic version without any pruning heuristic, four versions that apply a single heuristic each, and a full version that incorporates all four heuristics. By that we can learn about the effectiveness of each of the heuristics, and also examine their combined effect. The experiments were executed on a hardware comprised of an Intel i7-6820HQ processor and 32GB memory.

In order to evaluate the efficiency of the proposed pruning heuristics we chose a setting that is based on a transportation solution for elderly people living in nursing facilities [4], see Section 3. The *basic setting* includes $N = 15$ ride requests, $M = 2$ vehicles of capacity $C_1 = C_2 = 5$, which are available from $a_1 = a_2 =$ `7:00` until $b_1 = b_2 =$ `19:00`. The route consists of $L = 4$ terminals (including the depot), and takes 60 minutes to complete. The number of transports is set to $K = 3$.

We create 50 instances for each setting in each of the experiments, and present the mean values of these 50 instances. The instances differ in the ride requests. For each ride request in an instance we randomly choose its type (s-type or r-type), its two terminals (pick-up and destination) and its desired pick-up time (every 15 minutes within the working hours of the vehicles).

Our experiments examine the percentage of problems that can be (optimally) solved within a short period of time, where the timeout is set to one minute. In each experiment we focus on some parameter in order to observe its effect.

We begin with the number of ride requests parameter, which varies in the range $9 \leq N \leq 21$, while all other parameters remain fixed according to the basic setting. The results are shown in Figure 3.

Next, we focus on the vehicles' capacity parameter, which we vary in the range $3 \leq C \leq 7$. However, changing only the capacity may result in unsatisfiable problems (when C is too small) or in rather easy problems (when C is too large). To this end, we would like to maintain a balance between the number of

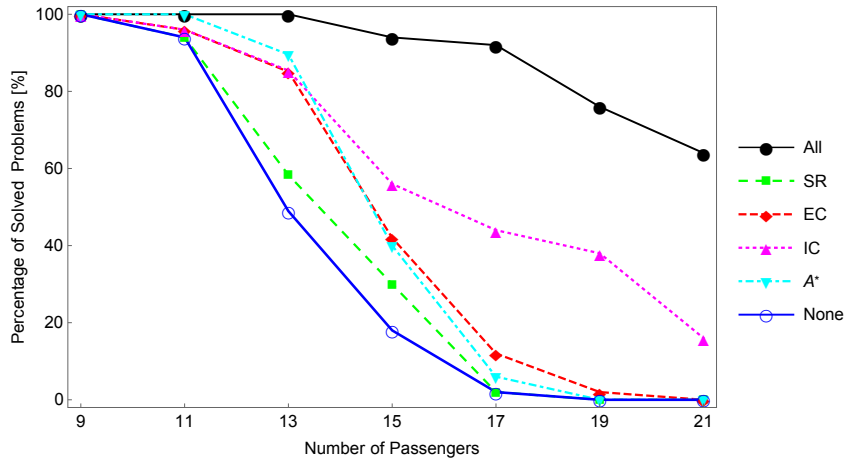

Fig. 3 Percentage of 1-minute solved problems with varying number of ride requests N

ride requests and the overall capacity (considering both the number of transports and the vehicles' capacity), according to the formula $N = K \cdot C$. Note that since passengers free up places in intermediate terminals, this initially tight-looking ratio usually corresponds to satisfiable problems. Thus, in this experiment, shown in Figure 4, we also accordingly vary the number of ride requests in the range $9 \leq N \leq 21$, where the number of transports remains $K = 3$ as in the basic setting.

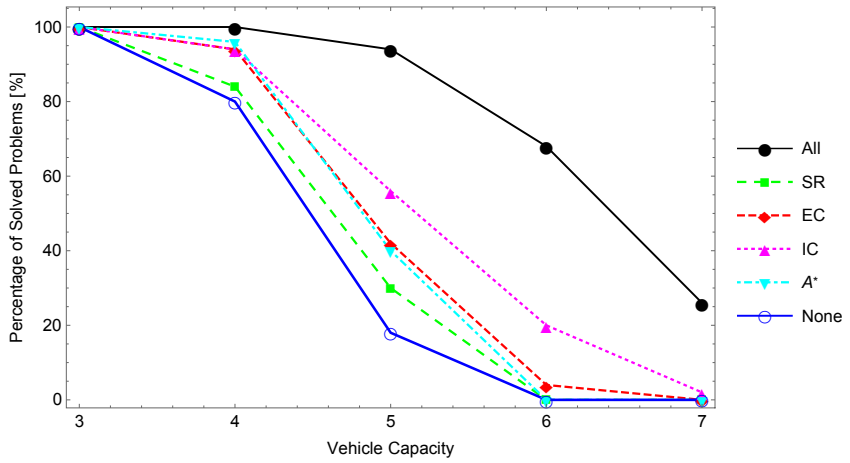

Fig. 4 Percentage of 1-minute solved problems with varying vehicles' capacity C

The last parameter we study is the number of transports, which we vary in the range $2 \leq K \leq 4$. Again, we maintain the balance $N = K \cdot C$. Consequently, in this experiment, depicted in Figure 5, we also vary the number of ride

requests in the range $10 \leq N \leq 20$, where the vehicles' capacity remains $C = 5$ as in the basic setting.

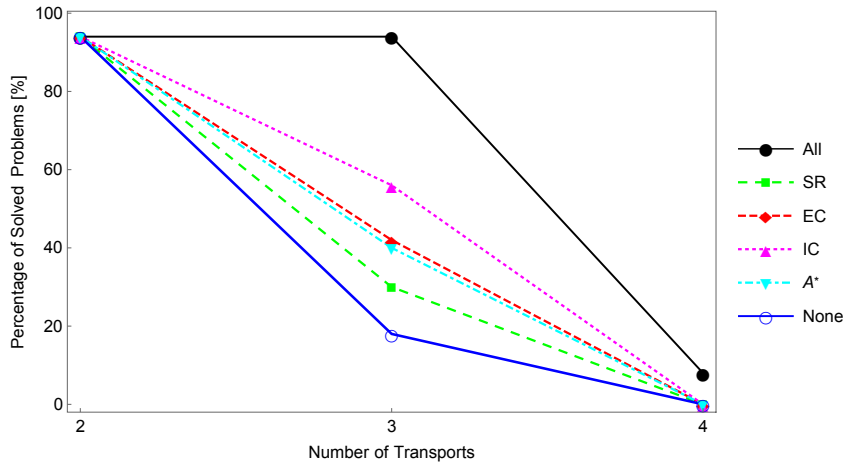

Fig. 5 Percentage of 1-minute solved problems with varying number of transports K

The presented results shed light regarding the performance of the proposed heuristics. Each of the heuristics on its own improves the practical solvability of FRDARP, but interestingly their combination is considerably more efficient than any single heuristic. This phenomenon suggests that the heuristics do not overlap much and are able to prune different parts of the constructed graph. This comes as no surprise given that the four heuristics use completely different ideas that lead their pruning strategies. Further insight regarding the combined effectiveness of the four heuristics can be gained by observing the sizes of obtained graphs without and with the heuristics. For example, the number of developed nodes for instances of the basic setting, which were solved within the 1-minute timeout, reduced from $\sim 300K$ to $\sim 5.5K$ on average.

Out of the four heuristics, the infeasible capacity heuristic performs best in most settings. It is actually only outperformed by the A*-based heuristic in rather easy settings ($N = 11$,$N = 13$ in Figure 3 and $C = 4$ in Figure 4).

Turning the spotlight to the A*-based heuristic, its performance seems to deteriorate as the problems become harder (in terms of number of passengers or vehicle capacity, see Figures 3 and 4). This suggests that although the basic idea of the heuristic has potential (as witnessed in its performance in the easy settings), the value of $h(\cdot)$ for hard instances is not tight enough, i.e., not close enough to the actual cost. Indeed, in the computation of $h(\cdot)$ we completely disregard the types and terminals of unhandled ride requests (Algorithm 1, line 1). The idea was to make the computation fast. However, for harder problems we may consider performing a more complex and tighter computation of $h(\cdot)$ that may result in increased overheads, but at the same time significantly reduce the size of the constructed graph.

6 Conclusion

The main contribution of this paper is in presenting and investigating pruning heuristics for the FRDARP model. In Section 2 we have presented the FRDARP model, including the reduction to the shortest path problem. A side contribution of this paper lies in the compactness of this presentation. We also believe it is more fluent and accessible, and in addition, it includes an illustrating example, which was missing in the first paper on this subject [7].

Regarding the pruning heuristics, we have presented four heuristics. Two of the heuristics attempt to identify and remove redundant paths, whereas the objective of the other two is to rule-out paths that are provably infeasible or non-optimal. According to our experimental evaluation, all the heuristics perform well and improve the practical solvability of FRDARP. However, the most interesting phenomenon is that their combined effort is considerably more efficient than any single heuristic, probably due to their completely different pruning strategies.

With the help of these pruning heuristics, the model is now more applicable for use in real-life problems, as well as in more complex variants of the model. For example, one can regard adding the option for vehicles to wait at intermediate terminals. There are some real-life scenarios in which such waiting is acceptable by the passengers, e.g., train connections in major stations. This option may reduce the value of the objective function, because new scheduling possibilities arise. For example, a vehicle that previously could not peak an r-type passenger from an intermediate station, can now wait in order to pick up this passenger. This can reduce the waiting time of passengers and may also lead to better vehicle availability. However, the option to wait at intermediate terminals is expected to increase the graph size, because of these new possibilities. Therefore, pruning heuristics are crucial for the applicability of such a complex variant.

Acknowledgements This research was supported by the Ministry of Science & Technology, Israel.

References

1. Richard E Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.
2. Jean-François Cordeau and Gilbert Laporte. The dial-a-ride problem: models and algorithms. *Annals of Operations Research*, 153(1):29–46, 2007.
3. Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT Press, 3rd edition, 2009.
4. Svetlana Daichman, Hillel Bar-Gera, and Tal Grinshpoun. Mobility habits and transportation needs of inhabitants of elderly homes in Israel. In *14th International Conference on Industrial Logistics (ICIL)*, 2018.
5. Willem E de Paepe, Jan Karel Lenstra, Jiri Sgall, René A Sitters, and Leen Stougie. Computer-aided complexity classification of dial-a-ride problems. *INFORMS Journal on Computing*, 16(2):120–132, 2004.

6. Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.
7. Hagai Ilani, Elad Shufan, and Tal Grinshpoun. A fixed route dial-a-ride problem. In *Proceedings of the 7th Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA)*, pages 313–324, 2015.
8. Hagai Ilani, Elad Shufan, Tal Grinshpoun, Aviad Belulu, and Alex Fainberg. A reduction approach to the two-campus transport problem. *Journal of Scheduling*, 17(6):587–599, 2014.
9. Yves Molenbruch, Kris Braekers, and An Caris. Typology and literature review for dial-a-ride problems. *Annals of Operations Research*, 259(1-2):295–325, 2017.
10. Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited, 3rd edition, 2016.