Transportation Science

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

A New Formulation for the Dial-a-Ride Problem

Yannik Rist, Michael A. Forbes

To cite this article:
Yannik Rist, Michael A. Forbes (2021) A New Formulation for the Dial-a-Ride Problem. Transportation Science 55(5):1113-1135. https://doi.org/10.1287/trsc.2021.1044

http://pubsonline.informs.org/journal/trsc

TRANSPORTATION SCIENCE
Vol. 55, No. 5, September–October 2021, pp. 1113–1135
ISSN 0041-1655 (print), ISSN 1526-5447 (online)

A New Formulation for the Dial-a-Ride Problem

Yannik Rist,[a] Michael A. Forbes[a]

[a]School of Maths and Physics, University of Queensland, St. Lucia, Queensland 4072, Australia
Contact: yannik.rist@uq.net.au, https://orcid.org/0000-0001-7589-4705 (YR); m.forbes@uq.edu.au (MAF)

Received: May 5, 2020
Revised: October 15, 2020
Accepted: January 16, 2021
Published Online in Articles in Advance: August 5, 2021

https://doi.org/10.1287/trsc.2021.1044

Abstract. This paper proposes a new mixed integer programming formulation and branch and cut (BC) algorithm to solve the dial-a-ride problem (DARP). The DARP is a route-planning problem where several vehicles must serve a set of customers, each of which has a pickup and delivery location, and includes time window and ride time constraints. We develop "restricted fragments," which are select segments of routes that can represent any DARP route. We show how to enumerate these restricted fragments and prove results on domination between them. The formulation we propose is solved with a BC algorithm, which includes new valid inequalities specific to our restricted fragment formulation. The algorithm is benchmarked on existing and new instances, solving nine existing instances to optimality for the first time. In comparison with current state-of-the-art methods, run times are reduced between one and two orders of magnitude on large instances.

Funding: This research is supported by an Australian Government Research Training Program Scholarship.
Supplemental Material: The online appendix is available at https://doi.org/10.1287/trsc.2021.1044.

Keywords: transportation • vehicle routing • dial-a-ride problem • branch and cut • valid inequalities

1. Introduction

Pickup and delivery problems (PDPs) are routing problems where one or more capacitated vehicles travel through a set of locations to fulfil customers' requests (Savelsbergh and Sol 1995). Each customer specifies a pickup location and a delivery location, which must be visited by the same route. In pickup and delivery problems with time windows (PDPTWs), each location has a time window that restricts when the location may be serviced. The dial-a-ride problem (DARP) is a generalization of the PDPTW, which typically aims to model the transportation of people, particularly in healthcare services (Cordeau and Laporte 2003, Zhang, Liu, and Lim 2015, Detti, Papalini, and de Lara 2017). The purpose of a DARP is to design routes that fulfil a set of pickup and delivery requests made by customers whilst adhering to the vehicle capacity, time window, and ride time restrictions of customers and achieving minimal cost.

As with many variants of the PDP, two formulations of the DARP dominate much of the literature; an arc-flow formulation, with variables indexed by arcs between locations, and a set-partitioning formulation in which variables are indexed by complete routes. The latter is used only with branch and price (BP) and branch and price and cut (BPC) algorithms (Barnhart et al. 1998) as the set of routes is too large to formulate the set-partitioning model directly. BPC algorithms require the solution of a column generation subproblem, which is typically a resource-constrained elementary shortest path problem. The elementarity of paths is sometimes relaxed and enforced in the master problem instead (Ropke 2006, Ropke and Cordeau 2009) to make the subproblem easier to solve. Subproblems are usually solved with labeling algorithms, where labels represent partial routes along with resources describing reduced cost (RC) and feasibility information. These labeling algorithms are typically supported by a label domination criterion, which is used to identify labels whose child labels will always lead to suboptimal solutions and can therefore be discarded.

For the PDPTW, it is straightforward to find conditions under which one partial route (label) dominates another, provided the triangle inequality holds for reduced costs. Although the triangle inequality is usually assumed to hold for travel costs, this is not true in general for the reduced (travel) costs in subproblems, making domination difficult. Ropke and Cordeau (2009) showed that for the PDPTW, reduced costs may be restructured to restore a weak version of the triangle inequality in the reduced costs. In the DARP, on the other hand, even with the triangle inequality, domination is complicated by ride time constraints. These constraints make it nontrivial to determine when the ability to complete a partial route implies the same for another partial route. Unlike with the PDPTW, it no longer suffices to track the earliest time of arrival at a label.

Fortunately, Gschwind and Irnich (2015) provide a strong domination criterion for partial DARP routes, by analyzing the feasibility of onboard customers and establishing resources representing each onboard customer's delivery deadline. A similar analysis of partial route feasibility is done in Gschwind (2015) for the synchronized pickup and delivery problem (SPDP), which generalizes the DARP by including a minimum ride time for each customer. Gschwind (2015) also compares different BPC subproblems for the SPDP, where different constraints in the subproblem are relaxed and enforced in the master problem with infeasible path elimination constraints (IPECs).

Another core component of labeling algorithms is an efficient method of determining when a given path is feasible with respected to timing constraints. As with the feasibility for partial paths, DARP feasibility for a path is not obvious. Hunsaker and Savelsbergh (2002) give an algorithm for determining feasibility of a path, which was later found to be incorrect and subsequently corrected by Tang et al. (2010). Improvements over the corrected $\mathcal{O}(n^2)$ algorithm were later made by Haugland and Ho (2010) and Firat and Woeginger (2011), reducing the running time to $\mathcal{O}(n\log n)$ and $\mathcal{O}(n)$, respectively. Gschwind and Drexl (2019) use the ride time resources proposed by Gschwind and Irnich (2015) to devise a constant time feasibility test for the neighbors of a legal route in an adaptive large neighborhood search algorithm.

The arc-flow formulation for the DARP and similar pickup and delivery problems with time constraints typically includes nonlinear constraints. In some cases (Cordeau 2006, Ropke, Cordeau, and Laporte 2007), these are linearized to give Miller–Tucker–Zemlin (MTZ)-style constraints (Miller, Tucker, and Zemlin 1960). These constraints assign service times and load amounts to each location to handle features like time windows, ride times, and vehicle capacities and also, eliminate subtours. MTZ constraints give notoriously bad linear programming relaxations, leading to branch and cut algorithms seeking to improve lower bounds (LBs) through valid inequalities (Cordeau 2006). Other branch and cut approaches replace the MTZ constraints with IPEC and add valid inequalities as well (Ropke, Cordeau, and Laporte 2007). Depending on whether vehicles are assumed to be homogeneous, the arc-flow formulation has variables indexed by location-location-vehicle (Cordeau 2006) or by location-location (Ropke, Cordeau, and Laporte 2007).

More recently, Munari, Dollevoet, and Spliet (2017) presented so-called p-step formulations for the capacitated vehicle routing problem (CVRP), a family of formulations that encompass the arc-flow and set-partitioning analogues for the CVRP. In a p-step formulation, variables are indexed by paths of locations up to length p. It is clear that $p = 1$ corresponds to the arc formulation, and the authors show that for large-enough p, one recovers the set-partitioning formulation. The purpose of a p-step formulation is to seek a compromise between the strong linear programming (LP) relaxation of set partitioning and the compact size of the arc formulation. It is unclear, however, how the idea of p-step formulations would be extended to PDP, in particular the pairing of pickups and deliveries. The formulation we present in this work is not a p-step formulation but seeks to make the same compromise between the set-partitioning and arc-flow formulations. Alongside p-step formulations, Alyasiry, Forbes, and Bulmer (2019) presented a formulation for the pickup and delivery problems with time window with last in, first out (LIFO) loading (PDPTWL), which used variables indexed by "fragments" of routes. Unlike p-step formulations where the length of the longest path is fixed a priori, the authors use the capacity of the vehicle to implicitly restrict the length, which has the advantage of making explicit capacity constraints redundant.

The contributions of this paper are fourfold. We propose a new type of object from which routes may be constructed, which are "smaller" objects than the fragments of Alyasiry, Forbes, and Bulmer (2019) and provide the basis for a new DARP formulation. We show how to efficiently enumerate these objects and propose two dominance criteria. In addition, we define several new IPECs, valid inequalities, and heuristics applicable to our model. Finally, we demonstrate the effectiveness of our algorithm on existing benchmark instances (Gschwind and Irnich 2015) and introduce a new benchmark. Our algorithm solves 9 instances from the literature that have never been solved exactly, as well as 10 of the new instances, and outperforms the current best-known algorithms for the DARP.

The rest of this paper is structured as follows. Section 2 formally defines the DARP and provides definitions of routes and schedules to be used in later sections. In Section 3, we give a new formulation for the DARP, distinct from the arc-flow and set-partitioning formulations, which borrows ideas from the work of Alyasiry, Forbes, and Bulmer (2019) on the PDPTWL. We discuss how to efficiently construct this model in Sections 4 and 5 and develop IPECs, valid inequalities, and primal heuristics in Section 6. Section 7 demonstrates the computational performance of our algorithm and discusses some possible modifications, and Section 8 is the conclusion.

2. The Dial-a-Ride Problem

The dial-a-ride problem consists of n customers, each with a pickup location and a delivery location. The sets of pickup and delivery locations are denoted by P and D, respectively. Customers are uniquely

identified by their pickup location, so $p \in P$ is used interchangeably to reference a customer or a pickup location. The delivery location of a customer p is denoted by $d(p)$, and the origin and destination depots are denoted by o^+ and o^-. The set of all locations is denoted by $N = \{o^+, o^-\} \cup P \cup D$. Each location $i \in N$ has a time window $[e_i, l_i]$ within which service must start. Online Appendix D summarizes the notation used throughout this paper.

The objective of the DARP is to design a set of routes of minimal cost, such that each customer's pickup and delivery locations lie on the same route (pairing constraints) and the former is visited before the latter (precedence constraints). Each pickup and delivery location must be visited exactly once (cover constraints). The cost of each route is the sum of the costs c_{ij} incurred by traveling between locations. Likewise, the travel time between locations i and j is denoted by t_{ij}. Throughout this work, and as is common in most of the literature, we assume that c_{ij} and t_{ij} satisfy the triangle inequality.

The DARP includes maximum ride time restrictions on customers, where the ride time for customer p is the difference between the start of service at $d(p)$ and p, and it may not exceed the maximum ride time, r_p. Although the ride time restriction is quite reasonable when modeling the transportation of people, the effect on determining route schedules, particularly for partial routes, is significant (Hunsaker and Savelsbergh 2002, Gschwind and Irnich 2015). In addition, each route has a maximum duration, denoted by r_{o^+}. The demand, or load, of customer $p \in P$ is denoted by $q_p \geq 0$, and $q_{d(p)} = -q_p$. The cumulative load on the vehicle may never exceed its capacity, Q. Some applications of the DARP include a service time $s_i \geq 0$ for each location; however, we incorporate this into the ride and travel times by setting $t_{ij} \leftarrow t_{ij} + s_i$ for all $i, j \in N$ and $r_i \leftarrow r_i + s_i$ for all $i \in \{o^+\} \cup P$. Precise definitions of schedules and routes are given.

Definition 1 (DARP Schedule). For a path of locations $(i_1, i_2, \ldots, i_K) \subseteq N$, a DARP *schedule* is an assignment of service start times, $(\tau_1, \tau_2, \ldots, \tau_K)$, that obey the following constraints:

$$\tau_{k+1} - \tau_k \geq t_{i_k, i_{k+1}} \qquad 1 \leq k \leq K-1 \tag{1}$$

$$\tau_l - \tau_k \leq r_{i_k} \qquad 1 \leq k < l \leq K,\ d(i_k) = i_l \tag{2}$$

$$e_{i_k} \leq \tau_k \leq l_{i_k} \qquad 1 \leq k \leq K \tag{3}$$

Constraints (1) ensure that travel times are respected. Constraints (2) ensure that ride times are appropriately restricted, where $d(o^+) := o^-$ so that the maximum route duration constraint is included when both o^+ and o^- are in the path. Constraints (3) are the time windows at each location in the path.

Definition 2 (DARP Route). A *route* for the DARP is a path $(o^+, i_1, i_2, \ldots, i_K, o^-) \subseteq N$ for which the following hold.

1. The path obeys pairing and precedence constraints and does not visit any location more than once.
2. The vehicle capacity is not exceeded anywhere along the path that is, $\sum_{j=1}^{k} q_{i_j} \leq Q$ for $k = 1, \ldots, K$.
3. A schedule exists for the path.

To simplify the exposition to come, it useful to define the following objects.

Definition 3 (Route Path). A *route-defining path*, or *route path* for short, is a path $(i_1, i_2, \ldots, i_K)$ such that $(o^+, i_1, i_2, \ldots, i_K, o-)$ is a route.

The original graph of locations and edges, $G = (N, N \times N)$ can be significantly reduced by eliminating edges that cannot be part of any route in a preprocessing step. The most obvious such reduction is the removal of edges (i, i) and $(d(p), p)$, $i \in N, p \in P$. Other edges are identified as unnecessary by inspecting potential routes stemming from the permutations of $\{p_i, p_j, d(p_i), d(p_j)\}$. The time windows of each location may also be tightened in preprocessing, taking into account the maximum ride time r_i. A detailed description of all preprocessing steps can be found in Cordeau (2006).

3. Restricted Fragment Network Formulation

Alyasiry, Forbes, and Bulmer (2019) presented the idea of *fragments* for the PDPTWL, a special type of route (see Definition 4). These fragments are sufficient to represent any route in the PDPTWL, and the authors construct a network to join these fragments.

Definition 4 (Fragment). A *fragment* is a route path that is only empty at the last location.

It is clear that any route in the PDPTWL can be represented by a series of fragments, simply by splitting the route whenever it becomes empty. The length of such fragments is greatly restricted by the combination of LIFO constraints and time windows, and therefore, the number of fragments is often small enough that they may be enumerated. Nonetheless, the number of fragments grows very quickly as time windows become wider and fragments get longer. Even when vehicles are strongly constrained with regard to their capacity, as in most DARP applications, lax time windows and the lack of LIFO encourage the "leapfrogging" of customers. For example, consider a route path of the form

$$(p_1, p_2, d_1, p_3, d_2, p_4, d_3, p_5, d_4, \ldots, p_k, d_{k-1}, d_k),$$

where $p_m \in P$ and $d_m := d(p_m)$ for $m = 1, \ldots, k$. The route path never exceeds a load of two (assuming unit loads

for pickup locations) and is only empty after visiting d_k because customer p_{m-1} is still on board at location d_m for $1 \le m < k$. The route path is clearly not LIFO, so this is not a PDPTWL fragment, but it is still a DARP fragment. This leapfrogging behavior results in longer fragments, so a different approach is required.

In this route, one may observe that there are many pickup to delivery movements. It may therefore be useful to consider segments of routes that contain exactly one of these movements and attempt to construct routes from these segments instead of fragments. Furthermore, if each of these segments contains one pickup to delivery movement, it follows that they are sequences of pickups followed by sequences of deliveries. The example route path would be broken into k such segments:

$$(p_1, p_2, d_1) + (p_3, d_2) + (p_4, d_3) + (p_5, d_4) + \ldots + (p_k, d_{k-1}, d_k),$$

which illustrates how leapfrogging is avoided within the segments. Given that applications of the DARP usually have tight capacity constraints, we severely restrict the length of these segments as well because they are bounded in length by twice the vehicle capacity. To avoid having redundant segments, we should require that these segments are as long as possible (i.e., that any one is not contained entirely within another). This leads to the following definition.

Definition 5 (Restricted Fragment). Given a route path, a *restricted fragment* is a segment of the path that starts on a pickup location whose predecessor in the route path is not a pickup, ends at a delivery location whose successor is not a delivery, and contains exactly one pickup to delivery movement.

From Definitions 4 and 5, it is clear that a fragment is made up of one or more restricted fragments, so restricted fragments are smaller objects than fragments. Although restricted fragments are much shorter than fragments and therefore, less numerous, this comes at the cost of having to keep track of the customers on board at the start and end of each restricted fragment. For example, consider the two route paths

$$\begin{aligned}(p_1, p_2, d_2, p_3, d_3, p_4, d_1, d_4) &\to (p_1, p_2, d_2) + (p_3, d_3) \\ &\quad + (p_4, d_1, d_4) \\ (p_1, p_2, d_1, p_3, d_3, p_4, d_2, d_4) &\to (p_1, p_2, d_1) + (p_3, d_3) \\ &\quad + (p_4, d_2, d_4),\end{aligned}$$

where both contain restricted fragments that follow the path (p_3, d_3), but the first has customer p_1 on board at the start and end, whereas the second contains customer p_2. In the leapfrogging example, each restricted fragment of the form (p_j, d_{j-1}) starts with p_{j-1} on board and ends with p_j on board. The first restricted fragment would start empty and end with p_2 on board, whereas the last starts with p_{k-1} on board and ends empty. To join restricted fragments together into routes, a network of nodes is built, where a node is defined in Definition 6.

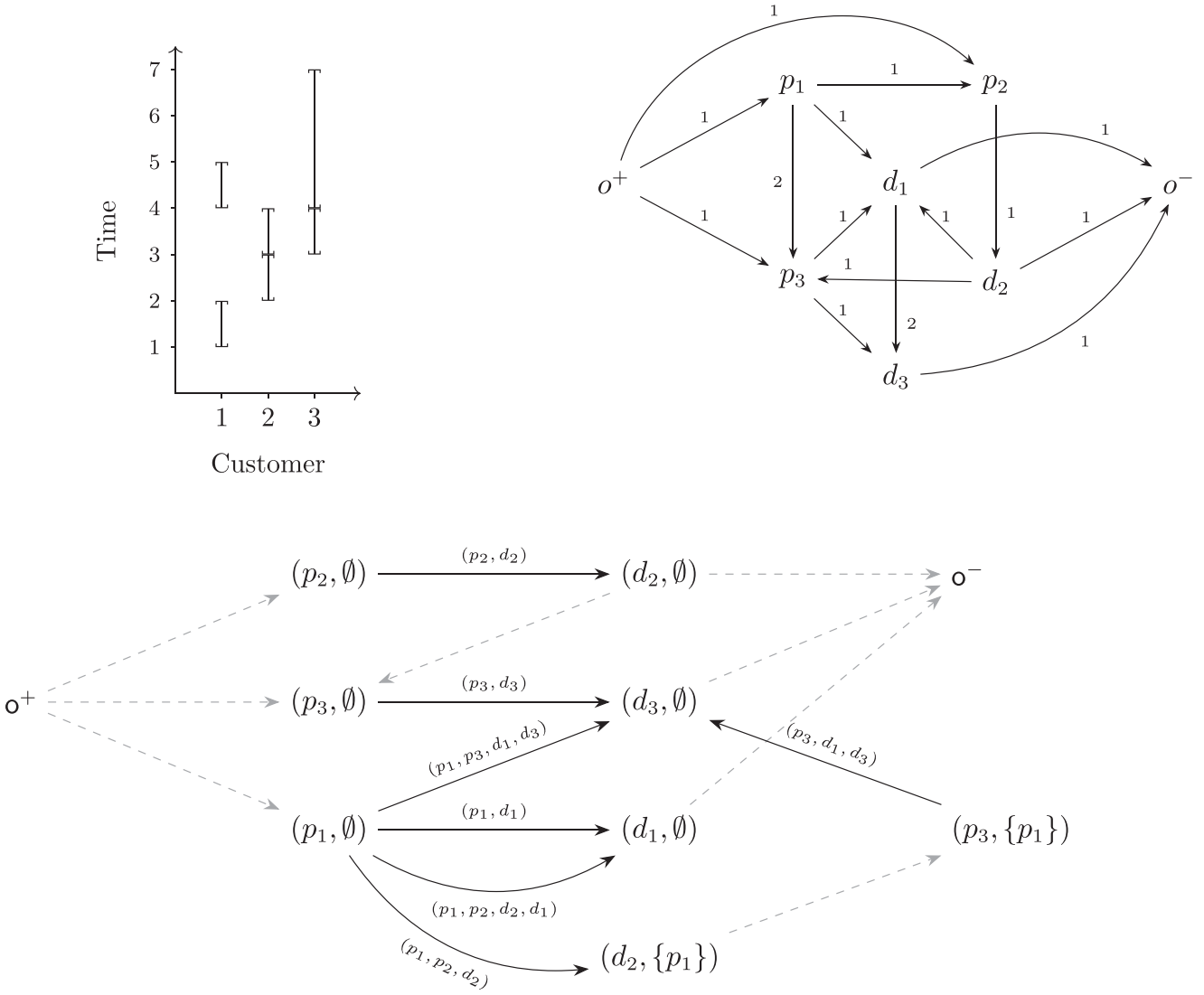
Definition 6 (Node). A *node* is a pair consisting of a location and a set of customers on board.

As with the PDPTWL (see Alyasiry, Forbes, and Bulmer 2019), the algorithm that is developed here can be broadly described as a branch and cut approach to solve a (restricted) fragment network. However, the network of restricted fragments will have many more nodes than just the physical locations N, resulting in additional flow conservation constraints compared with the arc-flow formulation of the DARP. These nodes have partial loads associated with them so precedence information is encoded implicitly in the network. Furthermore, restricted fragments, by their structure, measure the peak load of a route so capacity constraints are also implicit. Each restricted fragment flows from a pickup node to a delivery node; therefore, additional arcs are required that flow from delivery nodes to pickup nodes and to/from depot nodes. Note that there are two depot nodes, one for the origin depot and one for the destination depot, and both have empty loads. These arcs are distinct from the arcs in the arc formulation, as they connect nodes rather than locations. Arcs only flow between nodes with the same load, to keep loads consistent between restricted fragments.

Definition 7 (Arc). An *arc* is a movement connecting a delivery node to a pickup node or destination depot or connecting the origin depot to a pickup node.

As convention, objects in the abstract network shall be denoted by symbols in a Sans Serif font. Let A denote the set of all arcs, F the set of all restricted fragments, and P and D the sets of pickup and delivery nodes, analogous to P and D. The sets P and D contain nodes, pairs of locations, and partial loads, whereas the sets P and D contain locations. The depot locations, o^+, o^-, in the physical network correspond to depot nodes $\mathsf{o}^+ = (o^+, \emptyset), \mathsf{o}^- = (o^-, \emptyset)$ in the abstract network, respectively. The set $\mathsf{N} := \mathsf{P} \cup \mathsf{D} \cup \{\mathsf{o}^+, \mathsf{o}^-\}$ is the set of all nodes. The abstract network is then the digraph $(\mathsf{N}, \mathsf{F} \cup \mathsf{A})$. Let A_n^+ denote the set of arcs that flow to the node $\mathsf{n} \in \mathsf{N}$, and let A_n^- be the set of arcs that leave node n. F_n^+ and F_n^- denote the analogous sets of restricted fragments. Load(n) and Loc(n) give the load and location of node $\mathsf{n} \in \mathsf{N}$, respectively. Start(e) and End(e) denote the start and end nodes of $\mathsf{e} \in \mathsf{F} \cup \mathsf{A}$, respectively, and $\mathrm{Locs}(\mathsf{f}) \subseteq P \cup D$ denotes the ordered set of locations visited by $\mathsf{f} \in \mathsf{F}$. Finally, denote by F_i the fragments that visit $i \in N$. A toy example of a restricted fragment network (RFN) is shown in Figure 1.

Figure 1. A Small Example of a Restricted Fragment Network, with $n = 3$, $Q = 2$, and $q_p = 1$ for Each $p \in P$



Notes. Ride times and the maximum route duration are arbitrarily large. (Upper left panel) The pickup and delivery time windows for customers. Depots have arbitrarily large time windows. (Upper right panel) Physical network with edges labeled by travel time. (Lower panel) Restricted fragment network. Solid edges denote restricted fragments and dashed edges denote arcs.

Using the RFN, we can formulate a mixed integer program (MIP), which solves a relaxation of the DARP. Define c_f as the sum of the travel costs between the locations of f, and likewise for a ∈ A define $c_a = c_{ij}$ where i = Loc(Start(a)) and j = Loc(End(a)). Let $x_f \in \{0,1\}$ be a decision variable, which is one if and only if f is used, and let $y_a \in \{0,1\}$ be one if and only if arc a is used,

$$\min \sum_{f \in F} c_f x_f + \sum_{a \in A} c_a y_a \tag{4}$$

$$\sum_{a \in A_n^+} y_a = \sum_{f \in F_n^-} x_f \qquad \forall\, n \in P \tag{5}$$

$$\sum_{f \in F_n^+} x_f = \sum_{a \in A_n^-} y_a \qquad \forall\, n \in D \tag{6}$$

$$\sum_{f \in F_i} x_f = 1 \qquad i \in P \tag{7}$$

$$\sum_{a \in A_{o^+}^-} y_a \le K \tag{8}$$

$$y_a, x_f \in \{0,1\} \qquad \forall\, a \in A\,, f \in F. \tag{9}$$

The objective (4) is to minimize the sum of travel costs between physical locations. Constraints (5) and (6) maintain the conservation of flow throughout the network. Constraints (7) ensure that every pickup location is visited exactly once, and Constraints (8) ensure that the maximum number of vehicles is not exceeded. Constraints (9) are the standard binary constraints. We refer to Model (4)–(9) as the RFN model.

The flow constraints, together with the structure of the restricted fragment network, ensure that pairing and precedence constraints are obeyed in any feasible solution to the RFN model. The vehicle capacity constraints are also contained within restricted fragments because the peak load of any route occurs at the pickup to delivery movement of a restricted fragment. It remains to ensure that the "chaining" of restricted fragments in the network results yields actual routes (i.e., that the chains may be assigned a schedule). This is done using IPECs, which are added as lazy constraints and are detailed in Section 6.1.

4. Enumerating Restricted Fragments

For restricted fragments to be useful, it must be possible to enumerate them without enumerating all possible routes. Fortunately, by leveraging the triangle inequality for travel times, we can enumerate a subset of routes that will produce a superset of all restricted fragments. Consider a route path of the form

$$(p_{i_1}, \ldots, p_{i_k}, d_{j_1} \ldots, d_{j_k}), \tag{10}$$

which contains a single fragment. Such route paths may be enumerated using a path-extending procedure similar to the one presented in Alyasiry, Forbes, and Bulmer (2019), which generates PDPTWL fragments, with a modification to restrict the paths to contain a single pickup to delivery movement. Customer ride times must also be checked; the details of the algorithm can be found in Online Appendix B.

Using these route paths, a set, F′, of potential restricted fragments can be constructed in the following manner. For each route path having the form (10), we remove up to $k-1$ pickups from the start of the route and up to $k-1$ deliveries from the end, where $2k$ is the length of the route path. The removed pickups determine the starting load, and likewise, the removed deliveries give the end load, yielding k^2 potential restricted fragments. As an example, consider the route path $(p_1, p_2, p_3, d_3, d_2, d_1,)$, which yields nine possible restricted fragments, listed in Table 1.

Table 1. Potential Restricted Fragments Obtained from the Route Path $(p_1, p_2, p_3, d_3, d_2, d_1)$

Path						Initial load	Final load
$(p_1,$	$p_2,$	$p_3,$	$d_3,$	$d_2,$	$d_1)$	$\emptyset$	$\emptyset$
	$(p_2,$	$p_3,$	$d_3,$	$d_2,$	$d_1)$	$\{p_1\}$	$\emptyset$
		$(p_3,$	$d_3,$	$d_2,$	$d_1)$	$\{p_1, p_2\}$	$\emptyset$
$(p_1,$	$p_2,$	$p_3,$	$d_3,$	$d_2)$		$\emptyset$	$\{p_1\}$
	$(p_2,$	$p_3,$	$d_3,$	$d_2)$		$\{p_1\}$	$\{p_1\}$
		$(p_3,$	$d_3,$	$d_2)$		$\{p_1, p_2\}$	$\{p_1\}$
$(p_1,$	$p_2,$	$p_3,$	$d_3)$			$\emptyset$	$\{p_1, p_2\}$
	$(p_2,$	$p_3,$	$d_3)$			$\{p_1\}$	$\{p_1, p_2\}$
		$(p_3,$	$d_3)$			$\{p_1, p_2\}$	$\{p_1, p_2\}$

Proposition 1. F′ *contains all restricted fragments;* $\mathrm{F} \subseteq \mathrm{F}'$.

Proof. Suppose a restricted fragment f exists with initial and final loads $L_1, L_2 \subset P$, respectively, and define $d(L) = \{d(p) | p \in L\}$ for a set $L \subseteq P$. From Definition 5, there must exist a route path that has the form (r_1, f, r_2), where r_1 is a path that visits the pickup locations L_1 but does not visit their deliveries and r_2 is a path that visits the delivery locations $d(L_2)$ but not the corresponding pickups.

Let r_1' be the path obtained from r_1 by deleting any location not in L_1, and let r_2' be the path obtained from r_2 by deleting any location that is not in $d(L_2)$. In other words, r_1' is a permutation of L_1, and r_2' is a permutation of $d(L_2)$. As the triangle inequality for travel times holds, deleting locations from a route will always result in a another route, assuming pairing constraints are still satisfied. Therefore, the route path (r_1', f, r_2') exists. Because r_1' contains only pickup locations and r_2' contains only delivery locations, this route path is of the same form as the 1 in 10. By trimming r_1' and r_2' from the route, we obtain f, so $\mathrm{f} \in \mathrm{F}'$. □

The potential restricted fragments F′, as their name suggests, are not necessarily restricted fragments (i.e., $\mathrm{F} \neq \mathrm{F}'$ in general). For instance, suppose the route path (p_1, p_2, d_1, d_2) is the longest possible route path containing customers 1 and 2. Trimming p_1 from this path gives (p_2, d_1, d_2), but for this path to exist as a restricted fragment, it must be part of a route where p_2 is preceded by a delivery (see Definition 5). Because no route paths of the form $(p, p_1, d(p), p_2, d_1, d_2)$ exist, (p_2, d_1, d_2) is not a restricted fragment. Clearly, any $\mathrm{f} \in \mathrm{F}'$ that starts and ends empty is in F because it is also a route path of form (10). The following definition is useful in identifying the other restricted fragments.

Definition 8 (Minimal Restricted Fragment). For an initial load $L \subseteq P$, a *minimal* restricted fragment f is a shortest restricted fragment (w.r.t. number of locations) that delivers every $p \in L$ and ends empty. For a final load L, f is a minimal restricted fragment if f is a shortest restricted fragment that visits every $p \in L$, starting from an empty load.

Note that a restricted fragment f that ends empty is a minimal restricted fragment if and only if $|\mathrm{f}| = |\mathrm{Load}(\mathrm{Start}(\mathrm{f}))| + 2$ where $|\mathrm{f}|$ is the length of f, as it must visit the delivery location corresponding to the pickup location it started from. Likewise, a restricted fragment f that starts empty is a minimal restricted fragment if and only if $|\mathrm{f}| = |\mathrm{Load}(\mathrm{End}(\mathrm{f}))| + 2$. There are two minimal (potential) restricted fragments listed in Table 1 in the third and seventh rows.

Proposition 2. *Let* $\mathrm{f} \in \mathrm{F}'$ *have nonempty start and end loads. Then* $\mathrm{f} \in \mathrm{F}$ *if and only if there exist minimal restricted fragments* $\mathrm{g}_1, \mathrm{g}_2 \in \mathrm{F}$, *such that* $(\mathrm{g}_1, \mathrm{f}, \mathrm{g}_2)$ *forms a route.*

Proof. Let $f \in F$ have start and end loads of L_1 and L_2, respectively. As in Proposition 1, there exist paths r_1 and r_2 such that (r_1, f, r_2) forms a route path, and r_1 visits all of L_1 and r_2 visits all of $d(L_2)$. By Definition 5, r_1 must end on a delivery $d(p_1)$, and r_2 must begin with a pickup p_2. Consequently, r_1 must visit p_1, and r_2 must visit $d(p_2)$. Let g_1 be r_1 with all locations not in $\{d(p_1), p_1\} \cup L_1$ deleted, and let g_2 be r_2 with all locations not in $\{p_2, d(p_2)\} \cup d(L_2)$ deleted. By the same argument as in Proposition 1, (g_1, f, g_2) forms a route path, which immediately implies that $g_1, f, g_2 \in F$. Because $|g_1| = |L_1| + 2$ and $g_2 = |L_2| + 2$ by construction, these two restricted fragments are minimal. The proof of the other direction follows directly from Definition 5. □

From Proposition 2, it is also clear how one may identify restricted fragments f, which start nonempty but end empty, by considering route paths of the form (g_1, f) where g_1 is minimal. If (g_1, f) forms a route path, then both g_1 and f are restricted fragments. Likewise, restricted fragments that end nonempty and start empty can be identified using route paths of the form (f, g_2). We first check for route paths of the form (g_1, g_2), where $g_1, g_2 \in F'$, $|g_1| = |L_1| + 2$, and $|g_2| = |L_2| + 2$ to identify all minimal restricted fragments first (by proving they are in fact in F). This reduces the number of route paths needed to identify all other restricted fragments.

Algorithm 1 describes the enumeration of restricted fragments and the arcs used to connect them. Step 7 enumerates the necessary arcs between delivery and pickup nodes by checking pairs of minimal restricted fragments. If (f, g) form a route path, then the arc $(\text{End}(f), \text{Start}(g))$ is created. Step 7 at first seems redundant with Step 3. However, consider a group of four minimal restricted fragments $\{f_1, f_2, g_1, g_2\}$ such that (f_i, g_j) forms a route path for $i, j \in \{1, 2\}$ and g_1 and g_2 start at different nodes. In Step 3, we may check (f_1, g_1) and (f_2, g_2), identifying all four as restricted fragments. We still need to check the route path (f_2, g_1) in Step 7 to identify the arc from the end of f_2 to the start of g_1. In our implementation, we store any arcs discovered during Steps 3–6 to avoid checking the same route paths twice.

Algorithm 1 (Restricted Fragment and Arc Enumeration)

1. Generate fragments in the sense of Alyasiry, Forbes, and Bulmer (2019) but restricted to one pickup to delivery movement.
2. Trim fragments to obtain the set of potential restricted fragments F′.
3. Find the minimal restricted fragments in F′ by checking route paths of the form (f, g) where f and g are minimal.
4. Find the nonminimal restricted fragments, $f \in F'$, that start empty by checking route paths of the form (f, g) where $g \in F$ is minimal.
5. Find the nonminimal restricted fragments, $f \in F'$, that end empty by checking route paths of the form (g, f) where $g \in F$ is minimal.
6. Find the nonminimal restricted fragments, $f \in F'$, that start and end nonempty by checking route paths of the form (g_1, f, g_2) where $g_1, g_2 \in F$ are minimal.
7. Identify delivery-pickup arcs $a \in A$ by checking route paths of the form (f, g) where $f, g \in F$ are minimal.
8. Add arcs (d, o^-) for delivery nodes that have $\text{Load}(d) = \emptyset$ and (o^+, p) for pickup nodes with $\text{Load}(p) = \emptyset$.

Further detail on Steps 1 and 2 can be found in Online Appendix B. Step 6 in Algorithm 1 takes $\mathcal{O}(|F'|^3)$ time, although in practice, this is reduced because only a fraction of F′ consists of minimal restricted fragments. It is possible to construct a relaxed RFN model using F′ instead of F, at the cost of a worse LP relaxation and additional variables and cuts in the branch and cut phase. If Step 7 is still followed, then the running time will be reduced to $\mathcal{O}(|F'|^2)$. However, for the instances considered in this work, save for the largest, the time spent recovering F from F′ was insignificant when compared with the overall run time. For brevity, we shall refer to restricted fragments in further sections simply as fragments.

5. Dominating Restricted Fragments

In BPC algorithms for PDP, an important factor is the efficiency of the labeling algorithm used to identify the lowest reduced cost route. This requires a strong domination criterion for dominating labels, which typically represent incomplete routes starting at the origin depot. Gschwind and Irnich (2015) present a domination criterion obtained by analyzing the schedules of such paths that maximize the feasibility of onboard customers' ride time constraints. To make domination more concrete, we define the following.

Definition 9 (Domination of Routes). A DARP route r dominates another, r', if whenever r' is present in a solution, r may replace it at the same or lower cost.

Domination between fragments is defined similarly.

Definition 10 (Domination of Restricted Fragments). A restricted fragment f dominates another, g, if whenever g is present in a solution, f may replace at the same or lower cost.

For routes, it is not hard to see that two conditions ((1) r must have a lower total travel cost than r', and (2) both must visit the same locations) are necessary and sufficient for domination to occur. For fragments, the same two conditions are necessary but definitely

not sufficient. A further, obvious necessary condition is that the two fragments start and end at the same nodes, in order to maintain the flow Constraints (5) and (6). The primary complication in dominating fragments stems from determining when the existence of a schedule for the path $(h_1,\dots,g,\dots,h_l)$ implies a schedule for $(h_1,\dots,f,\dots,h_l)$.

Suppose $(h_1,\dots,g,\dots,h_l)$ has a schedule. To attempt to find a schedule for $(h_1,\dots,f,\dots,h_l)$, we only modify the schedule at locations visited by both f and g and fix the times at locations visited by the fragments $h_1,\dots,h_l$. This imposes a new condition: the time taken to traverse f is at most that taken to traverse g, in any route containing either of them. Unfortunately, the time taken to traverse a fragment depends on the fragments before and after it, so to handle this, we consider specific schedules that have some useful properties.

The *early* schedule for fragment $f=(i_1,i_2,\dots,i_K)$ starting at time t is the schedule where service starts at i_1 no earlier than t and as early as possible for all other locations in f. The *late* schedule for f ending at time t is the schedule in which service starts at i_K no later than t and as late as possible for all other locations in f. The existence of the late schedule was proven by Gschwind and Irnich (2015), and an almost identical proof may be given for the existence of the earliest schedule. In fact, Tang et al. (2010) give an algorithm that explicitly computes the earliest schedule, if it exists. In both cases, it should be noted that paths from the origin depot were considered, rather than restricted fragments, but the same arguments still hold. Let $\boldsymbol{\tau}_f^{\text{early}}(t)=(\tau_{f,1}^{\text{early}}(t),\dots,\tau_{f,K}^{\text{early}}(t))$ be the vector of the service start times in the early schedule, parameterized by t, the earliest start of service at i_1. Similarly, let $\boldsymbol{\tau}_f^{\text{late}}(t)=(\tau_{f,1}^{\text{late}}(t),\dots,\tau_{f,K}^{\text{late}}(t))$ be the service start times in the late schedule, where t is the latest start of service at i_K. It is useful to define

$$T_f(k_1,k_2)=\sum_{j=k_1}^{k_2-1} t_{i_j,i_{j+1}},$$

provided $k_1\le k_2$, and let $T_f:=T_f(1,K)$. Finally, as in Luo, Liu, and Lim (2019) and Sippel (2019), let $E_f=\tau_{f,K}^{\text{early}}(e_{i_1})$ be the earliest time at which service may start at i_K and $L_f=\tau_{f,1}^{\text{late}}(l_{i_K})$ the latest time at which service may start at i_1, across all schedules.

It is worth noting that $\tau_{f,1}^{\text{early}}(t)$ is not equal to t in general, as it may be necessary to wait at i_1 in order to avoid violating the ride time constraint of customer $i_1\in P$ (see Hunsaker and Savelsbergh 2002, Tang et al. 2010, and Gschwind and Irnich 2015 for a more in-depth discussion). The following propositions provide a characterization for the schedules $\boldsymbol{\tau}_f^{\text{early}}(t)$ and $\boldsymbol{\tau}_f^{\text{late}}(t)$.

Proposition 3 (Gschwind and Irnich 2015). *For any fragment* $f=(i_1,\dots,i_K)$ *and* $t\ge E_f$,

$$\tau_{f,k}^{\text{late}}(t)=\min\left\{t+c_1^k,c_2^k\right\}$$

for some constants c_1^k *and* c_2^k, $k=1,\dots,K$.

Proposition 3 is actually a slight modification of the one presented by Gschwind and Irnich (2015). The authors were concerned with paths where pickups and deliveries were not necessarily contiguous and only considered k where i_k is a pickup without a matching delivery. Nonetheless, the proof for this proposition requires only a minor adjustment to their proof. We say there is waiting time in a schedule, $\tau(t)$, between locations i_j and i_k if $\tau_k(t)-\tau_j(t)>T_f(j,k)$. The late schedule additionally has the following property.

Proposition 4. *Let* $f=(i_1,\dots,i_K)$ *be a fragment, and let* $t\ge E_f$. *If* $\boldsymbol{\tau}_f^{\text{late}}(t)$ *contains waiting time, then* $\tau_{f,1}^{\text{late}}(t)=L_f$.

Like the late schedule, the early schedule also has a simple form, which leads to a similar property.

Proposition 5. *For any fragment* $f=(i_1,\dots,i_K)$ *and* $t\le L_f$,

$$\tau_{f,k}^{\text{early}}(t)=\max\left\{t+T_f(1,k),\tau_{f,k}^{\text{early}}(e_{i_1})\right\}$$

for all $k=1,\dots,K$.

Proof. For brevity, we shall denote the earliest schedule $\boldsymbol{\tau}_f^{\text{early}}(e_{i_1})$ by $\boldsymbol{\tau}^0$. Define

$$\begin{aligned}\delta_k(t)&:=\max\left\{t+T_f(1,k)-\tau_k^0,0\right\}\\ \tau_k(t)&:=\tau_k^0+\delta_k(t)\end{aligned}.$$

We first need to check that Conditions (1)–(3) hold for $\boldsymbol{\tau}(t)$. Note that $\boldsymbol{\tau}^0$ is a schedule, so (1)–(3) must be true for $\boldsymbol{\tau}^0$. Clearly, $e_{i_k}\le\tau_k^0\le\tau_k(t)$, and if $\delta_k(t)=0$, then $\tau_k(t)=\tau_k^0\le l_{i_k}$. Otherwise,

$$\tau_k(t)=t+T_f(1,k)\le l_{i_k},$$

where the inequality follows from the assumption that $t\le L_f$ (i.e., at least one schedule starting at t or later must exist). In both cases, (3) is satisfied.

By Constraints (1), it holds $-\tau_{k-1}^0\ge-\tau_k^0+t_{i_{k-1},i_k}$, which implies

$$\begin{aligned}t+T_f(1,k-1)-\tau_{k-1}^0&\ge t+T_f(1,k-1)+t_{i_{k-1},i_k}-\tau_k^0\\&=t+T_f(1,k)-\tau_k^0,\end{aligned}$$

and therefore, $\delta_k(t)\le\delta_{k-1}(t)$. Let $1\le k<l\le K$ be such that $i_k\in P$ and $i_l=d(i_k)$. Observe that

$$\tau_l(t)-\tau_k(t)=\left(\tau_l^0-\tau_k^0\right)+\left(\delta_l(t)-\delta_k(t)\right)\le r_{i_k}-0,$$

so Constraints (2) are satisfied.

For the travel time Constraint (1) between i_{k-1} and i_k, there are two cases to consider.

Case (i): $\delta_{k-1}(t) = 0$. Using $\tau_k(t) \geq \tau_k^0$:

$$\tau_k(t) - \tau_{k-1}(t) \geq \tau_k^0 - \tau_{k-1}^0 \geq t_{i_{k-1},i_k}$$

Case (ii): $\delta_{k-1}(t) > 0$. Using $\tau_k(t) \geq t + T_f(1,k)$:

$$\tau_k(t) - \tau_{k-1}(t) \geq t + T_f(1,k) - t - T_f(1,k-1) = t_{i_{k-1},i_k}$$

Finally, fix $e_{i_1} < t \leq L_f$ and suppose $\boldsymbol{\tau}^*$ is the earliest schedule starting on or after t, where $\boldsymbol{\tau}^* \leq \boldsymbol{\tau}(t)$ and $\tau_k^* < \tau_k(t)$ for some k. Because $\tau_k^0 \leq \tau_k^*$, this implies $\tau_k^0 < \tau_k(t)$, so $\delta_k(t) > 0$. As $\delta_k(t) \leq \delta_{k-1}(t)$, $\delta_{k'}(t) > 0$ for each $k' = 1,\ldots,k$, and therefore, $\tau_{k'}(t) = t + T_f(1,k')$ for such k'. In particular, for $k' = 1$, this reads $\tau_1(t) = t$. Taking the summation of Constraints (1) from 1 to $k-1$ for τ^* yields the inequality

$$\tau_k^* - \tau_1^* \geq T_f(1,k).$$

However, because $t \leq \tau_1^* \leq \tau_1(t) = t$, it must also hold

$$\tau_k^* - \tau_1^* = \tau_k^* - t < \tau_k(t) - t = (t + T_f(1,k)) - t,$$

which is a contradiction. On the other hand, if $t \leq e_{i_1}$, note that $\boldsymbol{\tau}(t) = \boldsymbol{\tau}^0$, and because $\boldsymbol{\tau}^0$ is the earliest schedule, we are done. We have established that $\boldsymbol{\tau}(t)$ is the earliest schedule that starts on or after time t, so the proof is complete.

Corollary 1. *Let* $f = (i_1,\ldots,i_K)$ *be a fragment, and let* $t \leq L_f$. *If* $\boldsymbol{\tau}_f^{\text{early}}(t)$ *contains waiting time, then* $\tau_{f,K}^{\text{early}}(t) = E_f$.

Proof. Note that $E_f := \tau_{f,K}^{\text{early}}(e_{i_1})$, so if $\tau_{f,K}^{\text{early}}(t) > E_f$, then $\tau_{f,K}^{\text{early}}(t) = t + T_f(1,K)$, that is, the fragment contains no waiting time. □

The simple forms of $\boldsymbol{\tau}_f^{\text{early}}(t)$ and $\boldsymbol{\tau}_f^{\text{late}}(t)$ allow for comparison of schedules between different fragments based on only a (very) finite number of time points.

Proposition 6. *Let* $[a,b]$ *be an interval, and take* $f(x) = \max\{x + c_1, c_2\}$ *and* $g(x) = \max\{x + d_1, d_2\}$. *If* $f(a) \geq g(a)$ *and* $f(b) \geq g(b)$, *then* $f(x) \geq g(x)$ *for all* $x \in [a,b]$.

Corollary 2. *Let* $[a,b]$ *be an interval, and take* $f(x) = \min\{x + c_1, c_2\}$ *and* $g(x) = \min\{x + d_1, d_2\}$. *If* $f(a) \leq g(a)$ *and* $f(b) \leq g(b)$, *then* $f(x) \leq g(x)$ *for all* $x \in [a,b]$.

Finally, Proposition 7 provides sufficient conditions to relate the total duration of two fragments for any given service time at the last location.

Proposition 7 (Sippel 2019). *Let* f *and* g *be fragments. If* $T_f \leq T_g$, $E_f \leq E_g$, *and* $L_f \geq L_g$, *then* $\tau_{f,1}^{\text{late}}(t) \geq \tau_{g,1}^{\text{late}}(t)$ *for all* $t \geq E_g$.

We now present two dominance criteria that apply to certain classes of restricted fragments. The first applies to fragments that have no unmatched deliveries (i.e., every customer delivered by the fragment is picked up in the same fragment). Customers that are picked up in the fragment may still be on board at the end of the fragment. For such a fragment, the late schedule ensures that the customers who are delivered in a later fragment have as much of their ride time available as possible.

Theorem 1. *Let* f *and* g *be fragments, and assume the following conditions are true:*

$$c_f \leq c_g \tag{11}$$

$$\text{Start}(f) = \text{Start}(g) \tag{12}$$

$$\text{End}(f) = \text{End}(g) \tag{13}$$

$$\text{Locs}(f) = \text{Locs}(g) \tag{14}$$

$$\text{Load}(\text{Start}(g)) \subseteq \text{Load}(\text{End}(g)) \tag{15}$$

$$T_f \leq T_g \tag{16}$$

$$E_f \leq E_g \tag{17}$$

$$L_f \geq L_g \tag{18}$$

Furthermore, for every pickup i_k *in* g *that does not have a matching delivery in* g, *let* k' *index its position in* f. *For all such* i_k, *if the following conditions are satisfied:*

$$\tau_{f,k'}^{\text{late}}(E_g) \geq \tau_{g,k}^{\text{late}}(E_g) \tag{19}$$

$$\tau_{f,k'}^{\text{late}}(l_{i_k}) \geq \tau_{g,k}^{\text{late}}(l_{i_k}) \tag{20}$$

in addition to (11)–(18), *then* f *dominates* g.

Proof. Conditions (12)–(14) imply that substituting g for f in a route will not break precedence or pairing constraints within the route, and the route still visits the same locations. Condition (11) ensures the cost will not increase as a result of this replacement. Therefore, we only need to check that there is enough time to fit f in and that customers' maximum ride times are not violated. Let t be the time at which service starts at the end of g, and because $E_f \leq E_g \leq t$, there exists a late schedule for f, which ends at t.

To check there is enough time to fit f, it suffices to check $\tau_{f,1}^{\text{late}}(t) \geq \tau_{g,1}^{\text{late}}(t)$, i.e., that we may depart from i_1 with f at the same time or later than with g, waiting at i_1 if necessary to preserve the end time t. This is implied by Conditions (16)–(18) via Proposition 7.

Any customer who is on board at both the start and end of g (or f) will spend the same amount of time on board the vehicle when g is replaced by f because they are delivered after g (or f). Thus, their maximum ride time cannot be violated because g is part of a route.

It remains to show that for each customer $i_k \in P$ who is picked up during g and delivered in a later fragment, their time spent on g is at most that spent on f. This is equivalent to checking that $\tau_{f,k'}^{\text{late}}(t) \geq \tau_{g,k}^{\text{late}}(t)$. Because g forms part of a legal solution, $t \in [E_g, l_{i_k}]$ and by Corollary 2, Conditions (19) and (20) imply $\tau_{f,k'}^{\text{late}}(t) \geq \tau_{g,k}^{\text{late}}(t)$. □

As discussed, Conditions (11)–(14) are obvious. Condition (15) restricts the class of fragments for

which this domination criterion is applicable. Conditions (16)–(18) ensure that the travel time constraints are not broken between the start of f and the preceding node and the end of f and the following node in any route. The ride times of customers in initial or final loads of f are maintained by Conditions (19) and (20), under the assumption (15).

The second domination criterion applies to fragments for which the only customers on board at the end of the fragment were also initially on board. This is equivalent to Condition (25). In this case, the early schedule maximizes ride time feasibility because by delivering early, the customers initially on board will have their ride times made as short as possible.

Theorem 2. *Let* f *and* g *be fragments, and assume the following conditions are true:*

$$c_{\mathrm{f}} \le c_{\mathrm{g}} \tag{21}$$

$$\mathrm{Start}(\mathrm{f}) = \mathrm{Start}(\mathrm{g}) \tag{22}$$

$$\mathrm{End}(\mathrm{f}) = \mathrm{End}(\mathrm{g}) \tag{23}$$

$$\mathrm{Locs}(\mathrm{f}) = \mathrm{Locs}(\mathrm{g}) \tag{24}$$

$$\mathrm{Load}(\mathrm{Start}(\mathrm{g})) \supseteq \mathrm{Load}(\mathrm{End}(\mathrm{g})) \tag{25}$$

$$T_{\mathrm{f}} \le T_{\mathrm{g}} \tag{26}$$

$$E_{\mathrm{f}} \le E_{\mathrm{g}} \tag{27}$$

$$L_{\mathrm{f}} \ge L_{\mathrm{g}} \tag{28}$$

Furthermore, for every delivery i_k *in* g *that does not have a matching pickup in* g, *let* k' *index its position in* f. *For all such* i_k, *if the following conditions are satisfied*

$$\tau^{\mathrm{early}}_{\mathrm{f},k'}(e_{i_1}) \le \tau^{\mathrm{early}}_{\mathrm{g},k}(e_{i_1}) \tag{29}$$

$$\tau^{\mathrm{early}}_{\mathrm{f},k'}(L_{\mathrm{g}}) \le \tau^{\mathrm{early}}_{\mathrm{g},k}(L_{\mathrm{g}}) \tag{30}$$

in addition to (11)–(18), *then* f *dominates* g.

Proof. Let t be the time at which service starts at i_1 in g. We proceed in a similar manner to the proof in Theorem 1, first showing there is enough time to fit f in the place of g and then checking the ride times. Cover, pairing, and precedence are still maintained by Conditions (22)–(24).

For there to be enough time to fit f, we need to check $\tau^{\mathrm{early}}_{\mathrm{f},K}(t) \le \tau^{\mathrm{early}}_{\mathrm{g},K}(t)$. Suppose there is waiting time in $\boldsymbol{\tau}^{\mathrm{early}}_{\mathrm{f}}(t)$, then

$$\tau^{\mathrm{early}}_{\mathrm{f},K}(t) = E_{\mathrm{f}} \le E_{\mathrm{g}} \le \tau^{\mathrm{early}}_{\mathrm{g},K}(t),$$

where the equality follows from Corollary 1. On the other hand, if there is no waiting time in $\boldsymbol{\tau}^{\mathrm{early}}_{\mathrm{f}}(t)$, then the time taken to traverse f is simply T_{f}. Consequently,

$$\tau^{\mathrm{early}}_{\mathrm{f},K}(t) = t + T_{\mathrm{f}} \le t + T_{\mathrm{g}} \le \tau^{\mathrm{early}}_{\mathrm{g},K}(t),$$

so there is enough time to fit f.

As in Theorem 1, customers on board at the start and end of g will not see a change in their ride times. Unlike in Theorem 1, f and g have unmatched deliveries, not unmatched pickups (see (25)). Therefore, in order to ensure that the ride times of these customers are not violated, we require that the start of service at these deliveries with f be earlier or the same as with g. In other words, we require that $\tau^{\mathrm{early}}_{\mathrm{f},k'}(t) \le \tau^{\mathrm{early}}_{\mathrm{f},k}(t)$ for every unmatched delivery i_k in g. Because $t \in [e_{i_1}, L_{\mathrm{g}}]$, this follows easily from (29) and (30) by Proposition 6. □

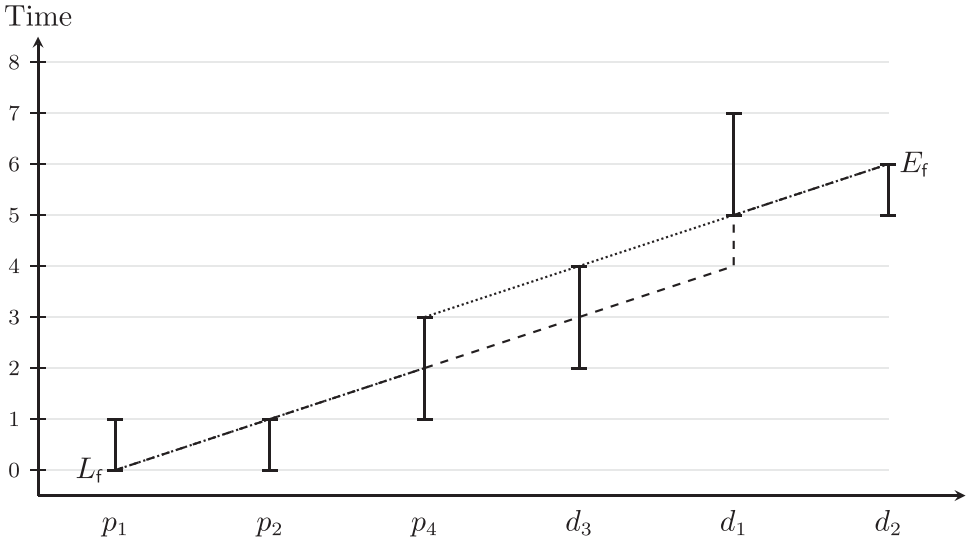
The applicability of Theorems 1 and 2 is restricted by Conditions (15) and (25), respectively. The difficulty in relaxing either of these conditions is in the conflicting "objectives" that occur when determining a schedule for a fragment that starts with customer i on board and drops i off, as well as picking up j to be dropped off in a later fragment. Both theorems rely on the existence of fragment schedules that maximize feasibility with regard to the external constraints (the ride times of customers originating from or destined for another fragment), subject to the internal constraints (time windows, travel times, and ride times of customers contained within the fragment).

For the aforementioned fragment, such a schedule would require customer i to be dropped off as early as possible, whereas customer j should be picked up as late as possible. However, because a fragment is a sequence of pickups followed by deliveries, customer j is picked up before customer i is dropped off, so delaying j's pickup time may affect i's delivery time.

As a more concrete example, consider a restricted fragment $\mathrm{f} = (p_1, p_2, p_4, d_3, d_1, d_2)$, with initial load $\{p_3\}$ and final load $\{p_4\}$. Because the initial and final loads are disjoint, neither Theorem 1 nor Theorem 2 is applicable. Assume that $t_{ij} = 1$ for all i, j, and every customer has a maximum ride time of eight. The time windows for the locations visited by f are shown in Figure 2. Also shown are the early and late schedules, $\boldsymbol{\tau}^{\mathrm{early}}_{\mathrm{f}}(L_{\mathrm{f}})$ and $\boldsymbol{\tau}^{\mathrm{late}}_{\mathrm{f}}(E_{\mathrm{f}})$. Because of the time window at p_2, it is not possible to leave after e_{p_1}, so $L_{\mathrm{f}} = e_{p_1}$. Similarly, it is not possible to arrive later than E_{f} at d_2, because of the time window bound d_2. This implies there is only one early schedule and one late schedule across all t, simplifying the comparison. In the late schedule, customer p_3 is on board for two units of time, and customer p_4 is on board for three units of time. On the other hand, the early schedule assigns two units of ride time to customer p_4 and three units to customer p_3. Note that both schedules take the same total amount of time to traverse f. Neither the late nor early schedule are able to maximize the feasibility of both customers p_3 and p_4 simultaneously that is, minimize the time they spend on board during f.

In order to generalize Theorems 1 and 2 to apply to all fragments, we require a schedule that minimizes

Figure 2. Time Windows and Schedules for f = $(p_1, p_2, p_4, d_3, d_1, d_2)$ Starting from Node $(p_1, \{p_3\})$ and Ending at Node $(d_2, \{p_4\})$



Notes. The black intervals represent time windows, the dotted line represents the late schedule $\tau_f^{\text{late}}(E_f)$, and the dashed line represents the early schedule $\tau_f^{\text{early}}(L_f)$. Vertical sections in the schedules correspond to waiting time.

ride time simultaneously for all customers who are not self-contained within the fragment. Gschwind (2015) encounters a similar problem when attempting to generalize the resource extension functions for the DARP (Gschwind and Irnich 2015) to apply to the synchronized pickup and delivery problem. The resource extension functions are used in a labeling algorithm, which iteratively extends partial routes from the origin depot. As such, they are akin to the late schedules used here for fragments with no unmatched deliveries (cf. Conditions (15), (19), and (20) in Theorem 1). The SPDP generalizes the DARP by including a minimum ride time for each customer. This additional constraint means that when considering completions of partial routes, it is no longer enough to simply consider the latest possible pickup time, as with the late schedule. Instead, for each onboard customer, the author considers the latest possible delivery time that does not impact other onboard customers' delivery time.

A similar idea could be applied to fragments. For example, suppose the time window for d_3 in Figure 2 was translated forward in time by 2, from $[2,4]$ to $[4,6]$. In this case, the late schedule would always be preferable because customer p_3 would be completed at $t = 4$ in both the early and late schedules, whereas customer p_4 still spends one less unit of time with the late schedule.

6. Branch and Cut

In this section, we detail the aspects of our branch and cut algorithm, specifically IPEC and cycle elimination constraints, strong valid inequalities, and a set of primal heuristics to obtain good upper bounds (UBs) early. Algorithm 2 at the end of this section summarizes how these different techniques are combined.

6.1. Infeasible Path Elimination Constraints

The RFN model alone does not ensure that travel or ride time constraints between fragments are respected, nor does it ensure the maximum route time is respected. Furthermore, cyclic paths are also possible. We handle the time-related constraints using IPEC, as in Ropke and Cordeau (2009) and Alyasiry, Forbes, and Bulmer (2019). Whenever we encounter an incumbent solution in the branch and bound tree, we construct a set of fragment chains where the first fragment in a chain is connected to the origin depot by an arc and likewise, for the last fragment and destination depot. For each such chain, we attempt to find a legal schedule, and if none exist, the chain cannot be a route. These chains must be eliminated using IPECs, which are added as lazy constraints.

Definition 11 (Chain). A chain is a sequence of fragments $(f_1, \ldots, f_l)$ of length $l > 1$, satisfying Load(End(f_k)) = Load(Start(f_{k+1})) for $1 \le k \le l-1$. Such a chain defines a sequence of arcs $(a_1, \ldots, a_{l-1})$ where a_k = (End(f_k), Start(f_{k+1})).

Definition 12 (Illegal Chain). A chain is said to be *illegal* if all paths containing that chain are not routes. A chain that is not illegal is *legal*.

In order to define IPECs that are applicable to more than one illegal chain, the following definition is necessary.

Definition 13 (Minimal Illegal Chain). An illegal chain $(f_1, \ldots, f_l)$ is said to be *minimal* if, for any integers j, k

such that $1 \le j < k \le l$ and $k - j < l - 1$, the chain $(f_j, f_{j+1}, \ldots, f_k)$ is legal.

Note that an illegal chain does not necessarily violate any constraints itself. For example, if the chain (f_1, f_2) does not violate any DARP constraints but f_2 does not end empty and (f_1, f_2, g) is illegal for any $g \in F$, then any route containing (f_1, f_2) chain is also illegal because it could only appear in a route before at least one other fragment. However, if a chain starts and ends empty and violates no DARP constraint, then it itself is a legal route and is therefore legal.

In the following work, let $c = (f_1, \ldots, f_l)$ be a minimal illegal chain with arcs $(a_1, \ldots, a_{l-1})$. It is trivial to define a valid cut (31) that is violated if and only if c is present in the solution

$$\sum_{k=1}^{l} x_{f_k} + \sum_{k=1}^{l-1} y_{a_k} \le 2l - 2. \tag{31}$$

Constraint (31) can be strengthened easily, by adding to the left-hand side any fragment g such that the chain $(f_1, \ldots, f_{l-1}, g)$ is also illegal and g starts at the same node as f_l (i.e., $g \in F^-_{a_{l-1}}$). Let the set F^-_c contain all fragments $g \in F^-_{a_{l-1}}$ such that the chain $(f_1, \ldots, f_{l-1}, g)$ is legal. With this notation and the aforementioned observation, the lifted version of (31) reads

$$\sum_{k=1}^{l-1} x_{f_k} + \sum_{k=1}^{l-1} y_{a_k} + \sum_{f \in F^-_{a_{l-1}} \setminus F^-_c} x_f \le 2l - 2. \tag{32}$$

Proposition 8. *Inequality* (32) *is a valid cut for the RFN model.*

Proof. By the cover (7) and flow Constraints (5) and (6), the third summation has a maximum value of 1, so the entire left-hand side has a maximum value of $2l - 1$. Thus, the inequality is violated if and only if the left-hand side is equal to $2l - 1$, in which case $x_{f_k} = y_{a_k} = 1$ for $k = 1, \ldots, l-1$ and $x_f = 1$ for a single $f \in F^-_{a_{l-1}} \setminus F^-_c$. This means the illegal chain $(f_1, \ldots, f_{l-1}, f)$ appears in the solution. Because $f_l \in F^-_{a_{l-1}} \setminus F^-_c$, the original chain c is also cut off. □

Conversely, one may define the cut in terms of fragments that cause legal chains, rather than forbid fragments that cause illegal ones. The cut is equivalent for integer values of (x, y) but tighter in the LP relaxation because of the smaller constant on the right-hand side:

$$\sum_{k=1}^{l-1} x_{f_k} + \sum_{k=1}^{l-1} y_{a_k} \le 2l - 3 + \sum_{f \in F^-_c} x_f. \tag{33}$$

As will be shown, other inequalities may be tightened using the same principle that is detailed in the following lemma.

Lemma 1. *Let* $P = \{x \in [0,1]^n | Ax \ge b\} \subset \mathbb{R}^n$, *and let* $P_{IP} = P \cap \{0,1\}^n$. *Suppose* $S, Z \subset \{1, \ldots, n\}$ *are index sets such that the following inequalities hold for all* $x \in P$ *and some* $i_0 \in S$:

$$\sum_{i \in Z} x_i \le 1 \tag{34}$$

$$x_{i_0} \le \sum_{i \in Z} x_i. \tag{35}$$

For any partition (X, Y) *of* Z, *define the following inequalities*:

$$\sum_{i \in S} x_i + \sum_{i \in X} x_i \le |S| \tag{36}$$

$$\sum_{i \in S} x_i \le |S| - 1 + \sum_{i \in Y} x_i. \tag{37}$$

The following are true:
1. *For each* $x \in P$, (37) *implies* (36).
2. *For each* $x \in P_{IP}$, (37) *is equivalent to* (36).

Proof. Fix $x \in P$. Because $\sum_{i \in Z} x_i = \sum_{i \in X} x_i + \sum_{i \in Y} x_i \le 1$, it holds $\sum_{i \in Y} x_i \le 1 - \sum_{i \in X} x_i$. Therefore, assuming (37) holds,

$$\sum_{i \in S} x_i \le |S| - 1 + \sum_{i \in Y} x_i \le |S| - \sum_{i \in X} x_i,$$

which when rearranged, gives (36). It remains to show that when $x \in P_{IP}$ and (37) is violated, so too is (36). Suppose (37) is violated, and note there are $|S|$ terms on the left-hand side. The right-hand side is always at least $|S| - 1$ so for the inequality to be violated, the left-hand side must be equal to $|S|$ and the right-hand side must be equal to $|S| - 1$. Therefore, $x_i = 1$ for all $i \in S$, in particular i_0, and $x_i = 0$ for all $i \in Y$. By (35), $x_{i_Z} = 1$ for some $i_Z \in Z$, which implies $i_Z \in X$. Thus, the left-hand side of (36) is equal to $|S| + 1$, violating the inequality. □

Proposition 9.
1. *Inequality* (33) *is a valid cut for the RFN model.*
2. *For any point* (x, y) *in the LP hull of the RFN model,* (33) *implies* (32), *but the converse is not true.*

IPEC (33) can be lifted further in the special case where $F^-_c = \emptyset$, based on the observation that the arc a_{l-1} can never be placed after the chain $(f_1, \ldots, f_{l-1})$ because doing so would lead to an illegal chain. Similar to the legal/illegal sets of fragments, define $A^-_c \subset A^-_{f_{l-1}}$ as the set of arcs a for which a goes to the depot, or there exists a fragment $g \in F^-_a$, where the chain $(f_1, \ldots, f_{l-1}, g)$ is legal. We lift (33) by replacing the empty summation on the right-hand side with a summation over these legal arcs:

$$\sum_{k=1}^{l-1} x_{f_k} + \sum_{k=1}^{l-2} y_{a_k} \le 2l - 4 + \sum_{a \in A^-_c} y_a. \tag{38}$$

Proposition 10. *Let* $c = (f_1, \ldots, f_l)$ *be a minimal illegal chain with* $F_c^- = \emptyset$.

1. *Inequality* (38) *is a valid cut for the RFN model.*
2. *For any point* (x, y) *in the LP hull of the RFN model,* (38) *implies* (33)*, but the converse is not true.*

The cuts presented so far have all targeted the placement of fragments/arcs *after* a chain. Analogous cuts to (33) and (38) can be found by considering the fragments/arcs that can be placed *before* the chain: in other words, considering the fragments that may be placed in lieu of f_1, rather than f_l. In this case, we define $F_c^+ \subset F_{a_1}^+$ as the set of fragments that make the chain $(g, f_2, \ldots, f_l)$ legal and $A_c^+ \subseteq A_{f_2}^+$ as the arcs to f_2, which either start at the depot or start at some fragment g where $(g, f_2, \ldots, f_l)$ is legal. The analogues to (33) and (38) are given in (39) and (40), respectively:

$$\sum_{k=2}^{l} x_{f_k} + \sum_{k=1}^{l-1} y_{a_k} \le 2l - 3 + \sum_{f \in F_c^+} x_f \tag{39}$$

$$\sum_{k=2}^{l} x_{f_k} + \sum_{k=2}^{l-1} y_{a_k} \le 2l - 4 + \sum_{a \in A_c^+} y_a. \tag{40}$$

One might wonder whether (38) can be lifted further when $A_c^- = \emptyset$. However, this would imply that the chain $(f_1, \ldots, f_{l-1})$ is illegal, as there is no g such that $(f_1, \ldots, f_{l-1}, g)$ is legal, and f_{l-1} cannot end empty; otherwise, there would be a depot arc in A_c^-. This contradicts the assumption that c is a *minimal* illegal chain.

6.2. Cycle Elimination

It is also necessary to add cuts that eliminate cyclic chains of fragments. The RFN contains some travel time information (implicit inside fragments) and all precedence information, so one can expect multiple fragment cycles to occur rarely in incumbent solutions. Thus, for a cycle of length $l > 1$ with fragments $(f_1, \ldots, f_l)$ and arcs $(a_1, \ldots, a_l)$, the cut given below is sufficient. The cut is violated if and only if this cycle is present in the solution

$$\sum_{k=1}^{l} x_{f_k} + \sum_{k=1}^{l} y_{a_k} \le 2l - 1. \tag{41}$$

More common are single-fragment cycles (consisting of one arc and one fragment), which can be cut off using the fragment-arc and arc-fragment valid inequalities described in Section 6.3. These cuts are tighter in the LP relaxation (see Propositions 15 and 16 in Section 6.3) than (41).

6.3. Valid Inequalities

The RFN model contains no information on travel or ride times between fragments, and therefore, we expect its LP relaxation to be poor. Although this information is added as needed using lazy Constraints (33) and (38), these do little to improve the LP relaxation, particularly when the minimal illegal chain is longer than two fragments. Fortunately, there are several strong valid inequalities available that encode such information, as well as subset-row (SR) inequalities similar to those available for set-partitioning formulations (Jepsen et al. 2008). To describe these, we require the following sets:

$$\begin{aligned}
F_{(f,\cdot)} &= \{g \in F \mid (f, g) \text{ is legal } \} \\
F_{(\cdot,f)} &= \{g \in F \mid (g, f) \text{ is legal } \} \\
A_{(f,\cdot)} &= \{a \in A_f^- \mid \mathrm{End}(a) = o^- \text{ or } F_{(f,\cdot)} \cap F_a^- \neq \emptyset\} \\
A_{(\cdot,f)} &= \{a \in A_f^+ \mid \mathrm{Start}(a) = o^+ \text{ or } F_{(\cdot,f)} \cap F_a^+ \neq \emptyset\} \\
F_{(a,\cdot)} &= \{f \in F_a^- \mid F_{(\cdot,f)} \cap F_a^+ \neq \emptyset\} \\
F_{(\cdot,a)} &= \{f \in F_a^+ \mid F_{(f,\cdot)} \cap F_a^- \neq \emptyset\}
\end{aligned}$$

$A_{(f,\cdot)}$ ($A_{(\cdot,f)}$) can be described more intuitively as the set of arcs that may appear after (before) the fragment f in a legal solution. Likewise, $F_{(a,\cdot)}$ ($F_{(\cdot,a)}$) is the set of fragments that may be placed after (before) the arc a in a legal solution. The last two sets are only defined for arcs that neither start nor end at depot nodes. An integer solution (x, y) to the RFN model is said to be *legal* if it satisfies all IPEC and cycle elimination constraints.

6.3.1. Subset-Row Inequalities.

Given any set of three nondepot locations, fragments that cover at least two of these are mutually exclusive with one another. This observation leads to a special case of subset-row inequality (Jepsen et al. 2008). Let $F_{i,j,k}$ be the set of fragments that visit at least two of $\{i, j, k\} \subset P \cup D$, where i, j, and k are distinct. The SR inequality for these three locations is defined as

$$\sum_{f \in F_{i,j,k}} x_f \le 1. \tag{42}$$

Proposition 11. *Inequality* (42) *is a valid inequality for the RFN model.*

Proof. The statement is a special case of proposition 1 in Jepsen et al. (2008), where $\kappa = 2$ and S is the set of three cover Constraints (7) indexed by i, j, and k. Here, κ has the meaning of k in Jepsen et al. (2008). □

These inequalities are separated by calculating $M_{i,j} = \sum_{f \in F\, i \cap F\, j} \bar{x}_f$, where $\bar{x}$ denotes the values of the x variables, for every pair of nondepot locations $\{i, j\}$. This is done quickly with sparse computations. We then find triples (i, j, k) of these locations for which

$$M_{i,j} + M_{j,k} + M_{i,k} - 2 \sum_{f \in F_i \cap F_j \cap F_k} \bar{x}_f > 1,$$

by checking pairs of location pairs of the form $\{i, j\}$, $\{i, k\}$ where $M_{i,j}, M_{i,k} > 0$. Each of these triples will lead to a violated subset-row inequality.

6.3.2. Fragment-Arc (FA) Inequalities. These inequalities, given by (43) and (44), simply state that for each fragment, the arc preceding or succeeding it must be legal. That is, the arc either leads to or from a depot or compatible fragment. Although we ensure that arc set A contains only arcs that may be part of some route, these inequalities are not implied by the structure of the network. Consider a simple example where for some arc $a = (d, p)$, $F_d^+ = \{f_1, f_2\}$, $F_p^- = \{g\}$, (f_1, g) is a legal chain, and (f_2, g) is an illegal chain. The arc a must be in the RFN, as (f_1, g) is part of a route, but a may never follow f_2 because this implies the illegal chain (f_2, g) is part of the solution. The illegal chain (f_2, g) is cut off by either one of (43) or (44). If $\text{Start}(f_2) = \text{End}(g)$, then this example also shows why single-fragment cycles cannot be eliminated during network construction:

$$x_f \le \sum_{a \in A_{(f,\cdot)}} y_a \qquad f \in F \tag{43}$$

$$x_f \le \sum_{a \in A_{(\cdot,f)}} y_a \qquad f \in F. \tag{44}$$

Proposition 12. *Inequalities* (43) *and* (44) *are valid inequalities for the RFN model.*

6.3.3. Fragment-Fragment (FF) Inequalities. Rather than considering legal arcs after a fragment, one may compare fragments directly, by considering the set of compatible fragments for a given fragment f. However, this assumes that f is not the last fragment in a route, which is true if f ends with a nonempty load:

$$x_f \le \sum_{g \in F_{(f,\cdot)}} x_g \qquad f \in F,\ \text{Load}(\text{End}(f)) \neq \emptyset \tag{45}$$

$$x_f \le \sum_{g \in F_{(\cdot,f)}} x_g \qquad f \in F,\ \text{Load}(\text{Start}(f)) \neq \emptyset. \tag{46}$$

Proposition 13. *Inequalities* (45) *and* (46) *are valid inequalities for the RFN model.*

6.3.4. Arc-Fragment (AF) Inequalities. The converse to the fragment-arc cuts, these cuts result from considering the set of fragments that may be placed after a given arc. Naturally, this is not defined for arcs that lead to or from depot nodes:

$$y_a \le \sum_{f \in F_{(a,\cdot)}} x_f \quad a \in A,\ \text{Start}(a) \neq o^+,\ \text{End}(a) \neq o^- \tag{47}$$

$$y_a \le \sum_{f \in F_{(\cdot,a)}} x_f \quad a \in A,\ \text{Start}(a) \neq o^+,\ \text{End}(a) \neq o^-. \tag{48}$$

Proposition 14. *Inequalities* (47) *and* (48) *are valid inequalities for the RFN model.*

The fragment-arc, fragment-fragment, and arc-fragment valid inequalities all bear the same structure: a single variable on the left-hand side and a summation of variables over a set S on the right-hand side. These valid inequalities are all separated in the same manner. If value of the left-hand side is nonzero, the set S is directly computed (and cached for future use) before evaluating the right-hand side.

As mentioned in the previous section, we use the arc-fragment inequalities and fragment-arc inequalities to remove single-fragment cycles. However, in the validity proofs of these inequalities presented (Propositions 12–14), we implicitly relied on cycle elimination constraints. Therefore, in order to use these inequalities as cuts, it is necessary to show that they are only violated by cyclic solutions, at least when assuming the IPECs (33) and (38) are satisfied.

Proposition 15. *Let* $(\boldsymbol{x}, \boldsymbol{y})$ *be a feasible solution to the RFN model that satisfies all IPECs of the form* (33) *and* (38) *but not necessarily* (41). *If* (47) *or* (43) *is violated, then the solution contains a single-fragment cycle.*

Proof. Let (43) be violated, so that $x_f = 1$ for some f and $y_a = 1$ for some $a \in A_f^- \backslash A_{(f,\cdot)}$. Note that a cannot go to or from a depot node. This implies that for all $g \in F_a^-$, (f, g) is illegal. Let $g \in F_a^-$ be such that $x_g = 1$. There are three possibilities.

1. $g \neq f$ and the fragments share some locations, violating (7).
2. $g \neq f$ and (f, g) violates an IPEC ((33) or (38)).
3. $g = f$.

The first two are made impossible by assumption, leaving the last. This implies that $a = (\text{End}(f), \text{Start}(f))$, so the solution contains a single-fragment cycle.

Now suppose (47) is violated, letting a and $f \in F_a^- \backslash F_{(a,\cdot)}$ correspond to the $x_f = y_a = 1$, which violates the inequality. Let $g \in F_a^+$ be such that $x_g = 1$, and note that $F_{(\cdot,f)} \cap F_a^+ = \emptyset$, so (g, f) must be illegal. By the same reasoning, this implies $f = g$, so f and a form a single-fragment cycle in the solution. □

Proposition 15 allows us to use either (47) or (43) to cut off single-fragment cycles. The next proposition shows why these cuts are preferable over the more general cycle cuts (41).

Proposition 16. *Let* $(\boldsymbol{x}, \boldsymbol{y})$ *be a point in the LP hull of the RFN model, and let* c *be the single-fragment cycle. Then, Inequalities* (47) *and* (43) *both imply* (41), *but the converse is not true.*

Proof. Let f and a form the single-fragment cycle. It suffices to show $f \notin F_{(a,\cdot)}$ and $a \notin A_{(f,\cdot)}$ for the implication to be true. For any $g \in F_a^-$, it holds $\text{Start}(g) = \text{End}(a) = \text{Start}(f)$, which means the chain (f, g) violates the cover constraint. Therefore, $F_{(f,\cdot)} \cap F_a^- = \emptyset$, so $a \notin A_{(f,\cdot)}$. For the second statement,

consider $g \in F_{(f,\cdot)}$. Clearly, End(g) ≠ End(f); otherwise, (f, g) would be illegal. As End(f) = Start(a), this means that $g \notin F_a^+$, implying $F_{(f,\cdot)} \cap F_a^+ = \emptyset$, and therefore, $f \notin F_{(a,\cdot)}$.

To construct a case where (43) and (47) outperform (41), take a single-fragment cycle where $x_f = y_a = 1/2$. (43) and (47) are both violated, but (41) is not. □

Valid inequalities are added at two points in our algorithm. We first solve the LP relaxation of the RFN model and add all valid inequalities that are violated by 0.01 or more. This is repeated until no more valid inequalities are added. In the branch and cut tree, we separate and add inequalities in the same manner at fractional nodes when the optimality gap is more than 0.1% and either (1) the total number of nodes explored is not more than 10 or (2) the following condition is met:

$$\frac{\mathrm{LB}_{\mathrm{node}} - \mathrm{LB}}{\mathrm{LB}_{\mathrm{node}}} < \min\{0.0025, \mathrm{OptGap}\}. \tag{49}$$

Here, OptGap is the optimality gap between global upper bound and LB, and $\mathrm{LB}_{\mathrm{node}}$ is the objective at the current node. These criteria attempt to ensure we add inequalities at nodes with shallow depth, in order for the cuts to be applicable to larger search subtrees. In this manner, we avoid increasing the size of the LP relaxation unnecessarily by adding too many inequalities.

6.4. Primal Heuristics

In this section, we detail a heuristic used to obtain an initial solution and two solution polishing/repairing heuristics used throughout the branch and cut tree. For a set of locations $S \subseteq N$, define $\mathrm{F}(S) := \{f \in \mathrm{F} | \mathrm{Locs}(f) \subseteq S\}$, and let F^* be the fragments belonging to the integer solution under consideration. The use of heuristics was found to be crucial to finding a good upper bound quickly and subsequently reducing the size of the search tree.

6.4.1. Reduced Cost Heuristic. Before starting the branch and cut process on the full RFN model, we order the x variables in ascending order by their optimal reduced costs in the root node LP relaxation. The first 10% of x variables are kept in the model, and the rest are set to zero. This results in a restricted model that is solved to within an optimality gap of 1%, using the full branch and cut algorithm, including valid inequalities and the other two heuristics. If an incumbent is found, the incumbent in the full model is initialized to the solution of the restricted model. Furthermore, all IPEC/cycle cuts and valid inequalities generated during this branch and cut stage are added to the full model before the LP relaxation of full model is solved again. Then, any x variable whose reduced cost is larger than the optimality gap is set to zero, a procedure commonly referred to as variable fixing (see proposition 1 of Desaulniers, Gschwind, and Irnich 2020).

6.4.2. Local Neighborhood Search (LNS) Heuristic. When a new integer solution is found, we construct a new network from a reduced set of fragments $\overline{\mathrm{F}}$, where

$$\overline{\mathrm{F}} := \bigcup_{f \in \mathrm{F}^*} \mathrm{F}(\mathrm{Locs}(f)) \cup \bigcup_{f,g \in \mathrm{F}^{*,2}} \mathrm{F}\Big(\mathrm{Locs}(f) \cup \mathrm{Locs}(g)\Big),$$

and $\mathrm{F}^{*,2}$ are those fragments in F^* that have a length of two. The idea is that $\overline{\mathrm{F}}$ contains fragments that are similar to those in the current solution F^*, with respect to locations visited. In addition, $\overline{\mathrm{F}}$ may also contain fragments that result from merging fragments of length two. $\overline{\mathrm{F}}$ naturally defines a set of nodes $\overline{\mathrm{N}} \subseteq \mathrm{N}$, which contain only depot nodes and the starts/ends of fragments in $\overline{\mathrm{F}}$. $\overline{\mathrm{N}}$ defines a set of arcs $\overline{\mathrm{A}} \subseteq \mathrm{A}$, which contains only those arcs that start or end in $\overline{\mathrm{N}}$. The RFN model is then constructed for the network $(\overline{\mathrm{N}}, \overline{\mathrm{F}} \cup \overline{\mathrm{A}})$ and solved with branch and cut but without valid inequalities or further heuristics.

6.4.3. Fixed Route Heuristic. Often when a proposed integer solution is not DARP feasible, through the violation of either IPEC or cycle constraints, there are only a few bad paths. If the valid paths (those that are routes) have low costs, then it may prove useful to fix these routes and optimize only for the remaining locations. Let $\mathrm{F}^{*,r} \subset \mathrm{F}^*$ contain those fragments that are part of routes in the current solution, and let $N^r \subset P \cup D$ contain those locations visited by fragments in $\mathrm{F}^{*,r}$. For this heuristic, the restricted set of fragments is given by

$$\overline{\mathrm{F}} := \mathrm{F}^{*,r} \cup \mathrm{F}((\mathrm{P} \cup \mathrm{D}) \setminus \mathrm{N}_r).$$

As in the local neighborhood search heuristic, we construct a reduced network $(\overline{\mathrm{N}}, \overline{\mathrm{F}} \cup \overline{\mathrm{A}})$ and solve the resulting RFN model, using branch and cut without valid inequalities or heuristics. Note that for each $i \in N^r$, there is exactly one fragment in $\mathrm{F}^{*,r} \subset \overline{\mathrm{F}}$ that visits i, so the routes from the original proposed integer solution are held fixed. The fixed route (FR) heuristic is only run when the routes in the solution are "good," which is evaluated using the criterion

$$\sum_{f \in \mathrm{F}^{*,r}} c_f < \frac{\mathrm{UB} \times |N_r|}{2n}, \tag{50}$$

where UB is the current upper bound.

The FR and LNS heuristics can take a significant amount of time to complete if the network resulting from $\overline{\mathrm{F}}$ becomes too large. This can slow the branch and cut algorithm down dramatically, as they are run at every integer solution. We therefore only run these heuristics if $|\overline{\mathrm{F}}|$ is below a fixed threshold, which we set to 1,500 fragments. This rule of thumb resulted in

either heuristic typically finishing in less than 0.5 second. The LNS heuristic was run first, and the FR heuristic was attempted only when the LNS failed.

Algorithm 2 (Proposed Top-Level Algorithm)

procedure SOLVE(USE_VI, USE_HEUR)
Generate restricted fragments and arcs (Algorithm 1)
Apply domination criteria (Theorems 1 and 2)
if USE_VI **then**
repeat
Solve LP relaxation of RFN model
Separate and add valid Inequalities (42)–(48) with violation > 0.01
until no more violated valid inequalities found
end if
if USE_HEUR **then**
Sort x variables in ascending order of reduced cost, and fix the upper 90% to zero
BRANCH-CUT(USE_VI, USE_HEUR) until OptGap < 0.01
Unfix the x variables
Add any IPEC, cycle constraints, and valid inequalities generated during BRANCH-CUT
if incumbent (UB) found **then**
Solve LP relaxation of full model (LB)
Fix any x variable with reduced cost > UB − LB to zero
end if
Initialize incumbent to the best solution found during BRANCH-CUT
end if
BRANCH-CUT(USE_VI, USE_HEUR) until optimal solution found
end procedure

The Branch-Cut Procedure is detailed in Online Appendix C.

7. Computational Experiments

7.1. Benchmark Instances

Cordeau (2006) created a DARP benchmark, split across A and B classes with each class containing 24 instances. The same instances are also used in previous studies for benchmarking branch and cut (Ropke, Cordeau, and Laporte 2007) and branch and price and cut algorithms (Gschwind and Irnich 2015). In some cases, authors only include results on a subset of the instances. The full set is available at neumann.hec.ca/chairedistributique/data/darp/. The A instances all have unit demand for customers and a vehicle capacity of three, whereas the B instances have a vehicle capacity of six with customer demands ranging from one to six. The full set of 48 instances was solved in under a minute, so we followed Gschwind and Irnich (2015) and modified the instances by delaying l_i by 15 minutes for pickup and delivery locations. This doubles the width of time windows, making the problems much more difficult to solve. The results we present in this section are only for these extended instances, which are denoted with an X suffix. The instances follow the naming convention [ab]K-n-X, where n is the number of customers and K is the maximum number of vehicles available.

Throughout this section, with the exception of Figure 3, the results shown were from benchmarks run on a 2.6-GHz processor with 50 GB of memory, limited to four threads and a time limit of one hour. The code is written in Python 3.7 using Gurobi 9.0.1 as the MIP solver and is available at https://github.com/ykrist/darp-rf/. For the sake of comparison with branch and price algorithms, which are typically single threaded, complete results on a single thread are provided in Online Appendix E. Figure 3 shows the impact of the number of threads on solve times for the five largest B instances; four threads seem to be the point of diminishing returns. Note that only the Gurobi solver is multithreaded; our own fragment and network generation code is not. The numerical seed in modern solvers can have a large effect on the solution time; as our algorithm spends the majority of time in the Gurobi solver, each benchmark is run on five seeds, and the results are aggregated. We observed a spread in solve times of up to 51% of the average time, although all benchmark runs solved within the one-hour time limit. The results presented in Figures 3–5 are all averages.

7.2. Bounds

Lower and upper bounds of the 10 largest B instances are shown in Figure 4. We compare two lower bounds from our algorithm: the optimal objective of the LP relaxation for the RFN model and that of the LP relaxation at the end of the cut loop in Algorithm 2 (lifted LP relaxation). The upper bound shown is that obtained by the RC heuristic, which provides the initial incumbent for the main branch and cut search tree.

The strength of the valid Inequalities (42)–(48) is demonstrated in Figure 4, where the lifted LP relaxation closes more than half the integrality gap in most cases. The RC heuristic is also justified for larger instances, where it can take a significant amount of time to run, usually finding an upper bound within 0.5% of the optimal solution. For all the A instances, the RC heuristic failed because it resulted in an infeasible model. This is because the size of the fragment network in these instances (in particular, the number of fragments) is much smaller, by more than an order of magnitude. On the other hand, the smaller networks produced by the A instances were found to be impacted more by the valid inequalities in the cut loop, as lifted LP relaxation was, on average, 99.5% of the optimal objective (across all A instances).

Figure 3. Impact of Number of Threads on Performance for Large Instances

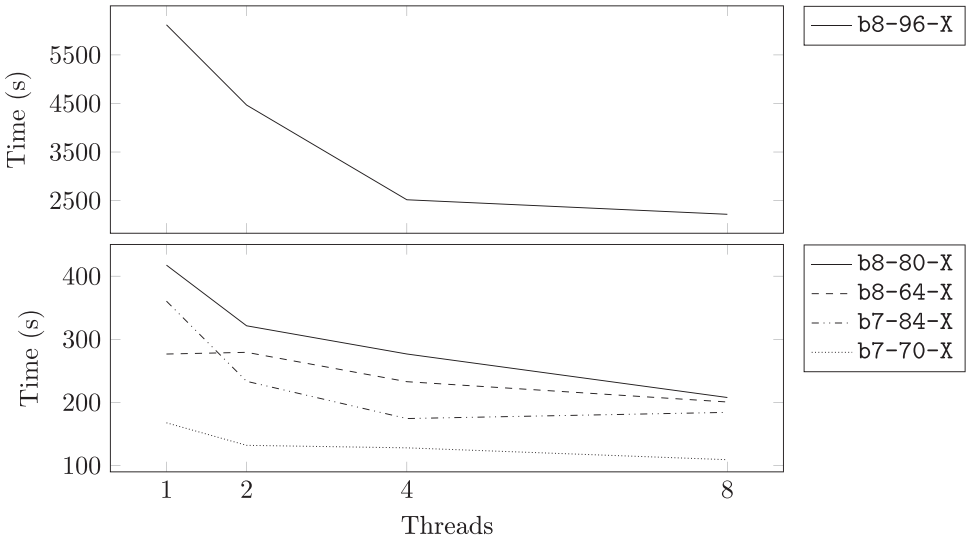
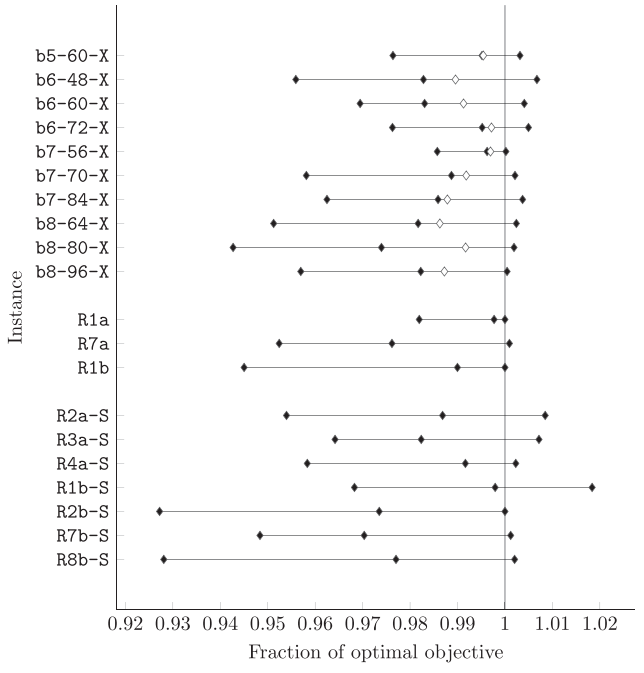


Figure 4 also shows the lower bound at the root node obtained by Gschwind and Irnich (2015), who used a set-partitioning model with valid inequalities. Set-partitioning models are known to give very strong LP bounds for pickup and delivery problems. The lower bounds of the fragment model (with valid inequalities) are comparable with the set-partitioning model, which demonstrates the strength of the fragment formulation despite its more compact size.

Figure 4. Various Bounds Obtained Through the Algorithm Relative to the Optimal Objective



Notes. Each interval shown contains three black marks representing optimal objective values: from left to right: root LP relaxation, lifted LP relaxation, and reduced cost heuristic. The white mark is the lower bound at the root node of the set-partitioning model IMP^{cut}_{strg} from Gschwind and Irnich (2015). Values are expressed as a fraction of the optimal objective for each instance.

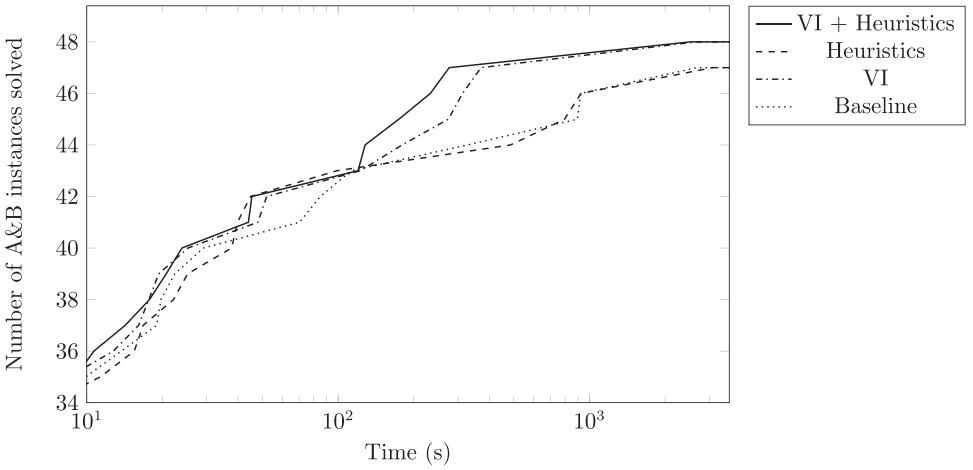
7.3. Network Size

The Network size columns in Table 2 show the size of the fragment network $(N, F \cup A)$ for each instance. The values for |F| shown are the numbers of undominated fragments; domination using Theorems 1 and 2 removed up to 22% of fragments, with larger instances being affected the most. The number of constraints in the RFN model relates directly to the number of nodes |N|, and the number of variables is given by |F| + |A|. Therefore, the largest instance resulted in an initial MIP of around 8,100 constraints and 91,000 integer variables, which easily fit into memory, given the sparsity of flow Constraints (5) and (6). For the smallest B instance, b2-16-X, we generated both restricted fragments and fragments (by Definition 4). The instance produced 118,885 fragments but only 464 restricted fragments (without domination), highlighting that there are vastly fewer restricted fragments than fragments.

7.4. Branch and Cut Tree

The Nodes column in Table 2 shows that many instances were solved at the root node or immediately afterward. This is credited to the lower bounds at the root node, which were similar in strength to those of set-partitioning formulations for instances that resulted in fewer fragments. On the other hand, for cases where the number of nodes is higher, the solve time

Figure 5. The Effects of Valid Inequalities (VIs) and Heuristics on Run Times for Extended A and B Instances



Notes. Heuristics refers to the combination of reduced cost, neighborhood search, and fixed route heuristics. VIs refer to those discussed in Section 6.3. The number of A and B instances solved within a given time limit is shown.

increased much less dramatically when compared with the BPC algorithm of Gschwind and Irnich (2015) (cf column $\text{IMP}^{\text{cut}}_{\text{strg}}$ in Table 2). This may be because of differences in the implementation of the branch and bound tree in a BPC algorithm compared with a commercial MIP solver. Another advantage provided by the valid inequalities is that the number of cuts added is low (see the Cuts column in Table 2), even for the largest instances, because the FF, AF, and FA inequalities cut off illegal chains of length two, which are the most common.

7.5. Solution Times

Compared with the current best-performing exact methods for solving DARP instances (Gschwind and Irnich 2015), our method provides a speedup of up to two orders of magnitude on large A instances. For smaller instances (in both A and B classes), the speedup is less dramatic, with a speedup factor of between 1 and 10. The greater speedup on A instances is primarily because of the smaller vehicle capacity in these problems, which greatly restricts the number of fragments and in turn, keeps the number of nodes in the network manageable. It is difficult to estimate the speedup on large B instances, as the algorithm of Gschwind and Irnich (2015) did not terminate within the time limit in many cases. However, it is expected to be significant, as the large B instances typically have larger integrality gaps. The BPC model $\text{IMP}^{\text{cut}}_{\text{strg}}$ solves instances b7-56-X and b6-72-X in less time than b6-48-X and b6-60-X, despite the latter instances being smaller in size. This is because the former two have very small integrality gaps at the root node (see Figure 4). On the other hand, our fragment model does not show much of a difference in solutions times for these problems, suggesting it is slowed less by larger integrality gaps than BPC.

7.6. Impact of Valid Inequalities and Heuristics

In Figure 5, a comparison of run times is shown for variants of our algorithm with and without the valid inequalities from Section 6.3 and heuristics from Section 6.4. The results suggest that the heuristics are only worthwhile after the lower bounds have been strengthened through valid inequalities. The efficacy of heuristics is limited to the larger, more difficult instances because as mentioned previously, smaller instances will frequently lead to infeasible subnetworks in the RC and LNS heuristics.

7.7. Hard Instances

To test the limits of our algorithm, we ran further benchmarks on another DARP data set introduced by Cordeau and Laporte (2003), the R instances. These instances are characterized by large numbers of customers (up to 144) and wide time windows. They are split into two subclasses "Ra" and "Rb," the former containing time windows ranging in width from 15 to 45 minutes and the latter ranging between 30 and 90 minutes. Customer ride times are all set to 90 minutes. Compared with the extended A and B instances, which have ride times of 30 and 45 minutes, respectively, and time window widths of 30 minutes, these are much harder problems. The capacity of the vehicles in the R instances is six, as in the B instances, but the customer loads are all one in the R instances, whereas the B instances have loads between one and six. Consequently, the vehicle capacity is less restrictive for the R instances.

In previous literature (Cordeau 2006, Parragh and Schmid 2013, Braekers, Caris, and Janssens 2014,

Table 2. Results for Extended A and B Instances

	Network size			Branch and Cut			Valid inequalities				Solve time (seconds)			
Instance	|F|	|A|	|N|	Nodes	Cuts	Obj.	AF	FA	FF	SR	Network	Tree	Total	IMP^{cut}_{strg}
a2-16-X	44	154	38	1.0	0.0	278.235	3.0	3.0	0.0	0.0	0.0	0.0	0.1	0.1
a2-20-X	68	236	50	0.0	2.0	330.691	8.0	8.0	0.0	0.0	0.0	0.1	0.1	0.1
a2-24-X	111	342	77	1.0	13.6	389.095	10.0	10.2	0.0	8.0	0.1	0.5	0.6	0.2
a3-18-X	149	225	80	0.0	7.0	272.705	7.0	7.0	0.0	2.0	0.1	0.1	0.2	—
a3-24-X	275	396	132	1.0	7.0	289.570	8.0	8.0	4.0	0.0	0.1	0.1	0.3	0.9
a3-30-X	163	526	104	0.0	8.0	452.829	13.0	10.0	0.0	8.0	0.1	0.2	0.3	0.6
a3-36-X	248	754	143	1.0	0.0	501.022	1.0	1.0	0.0	0.0	0.1	0.2	0.3	2.4
a4-16-X	160	190	79	1.0	16.0	235.174	5.6	5.6	0.0	16.0	0.1	0.2	0.3	—
a4-24-X	172	362	104	0.0	4.0	359.340	3.0	3.0	0.0	4.0	0.1	0.1	0.2	—
a4-32-X	341	673	185	1.0	13.2	447.278	3.0	3.0	0.0	13.6	0.2	0.3	0.5	0.8
a4-40-X	567	1,048	240	0.0	0.0	509.018	10.0	10.0	4.0	4.0	0.3	0.3	0.6	1.3
a4-48-X	681	1,444	301	5.4	16.4	618.958	20.4	22.8	6.4	26.2	0.4	1.6	2.0	10.4
a5-40-X	662	1,067	302	0.0	7.0	464.066	10.0	10.0	4.0	36.0	0.3	0.3	0.6	1.4
a5-50-X	848	1,636	377	1.0	5.6	621.957	8.2	8.2	4.0	26.4	0.5	0.8	1.2	4.2
a5-60-X	967	2,196	424	1.0	14.8	745.398	13.2	14.0	0.4	8.8	0.5	0.9	1.4	83.8
a6-48-X	1,341	1,814	529	1.0	10.4	572.513	21.8	22.6	17.4	11.2	0.7	0.8	1.5	8.9
a6-60-X	1,429	2,411	554	1.0	15.6	757.876	19.2	22.6	9.4	59.6	1.1	1.4	2.5	5.8
a6-72-X	1,727	3,301	674	1.0	15.0	868.288	26.0	22.6	9.8	15.6	1.1	1.7	2.8	284.1
a7-56-X	1,346	2,163	525	1.0	8.8	663.428	33.8	36.2	19.8	23.8	0.8	1.7	2.4	17.2
a7-70-X	2,271	3,496	814	1.0	10.0	824.428	33.4	36.4	21.2	35.0	1.6	1.8	3.4	59.4
a7-84-X	2,588	4,687	949	69.6	19.8	950.577	62.0	61.4	39.6	41.2	1.8	7.1	9.0	1,422.2
a8-64-X	2,551	3,184	867	1.0	10.0	701.151	33.6	35.6	27.6	38.0	1.9	2.1	4.0	75.3
a8-80-X	2,426	4,437	947	1.0	29.2	880.288	42.6	42.6	33.8	56.8	1.7	3.7	5.4	386.9
a8-96-X	3,693	6,339	1,300	370.0	46.6	1115.093	57.4	57.8	44.8	72.6	2.6	15.2	17.8	1,564.7
b2-16-X	420	224	112	0.0	4.0	244.319	5.0	7.0	8.0	6.0	0.2	0.1	0.4	0.1
b2-20-X	334	292	116	1.0	11.0	280.211	1.0	1.0	0.0	0.0	0.2	0.1	0.3	0.1
b2-24-X	382	421	155	1.0	8.2	372.589	9.0	7.0	9.0	5.0	0.2	0.2	0.4	0.7
b3-18-X	1,356	560	309	1.0	13.2	251.949	12.0	16.0	1.0	15.0	0.7	0.5	1.2	—
b3-24-X	986	582	280	1.0	22.0	321.209	8.6	8.6	4.0	2.0	0.5	0.5	1.1	2.9
b3-30-X	991	759	304	1.0	12.4	433.282	15.0	12.2	5.2	19.8	0.5	0.5	0.9	1.7
b3-36-X	1,255	1,012	348	1.0	10.0	486.014	10.0	9.0	2.0	8.0	0.6	0.4	1.0	1.1
b4-16-X	632	430	220	10.8	16.8	233.172	14.2	16.6	6.8	9.6	0.3	0.8	1.1	—
b4-24-X	1,594	771	362	1.0	17.6	292.969	27.2	24.6	12.8	20.6	1.0	1.0	2.0	—
b4-32-X	1,185	899	371	1.0	15.4	410.142	8.8	7.8	7.0	5.0	0.6	0.5	1.0	1.6
b4-40-X	3,639	1,759	728	0.0	5.6	478.675	7.0	7.0	10.0	8.0	2.5	0.9	3.4	2.2
b4-48-X	7,891	2,930	1,228	1.0	15.4	543.252	24.8	24.2	15.6	36.6	6.9	3.8	10.7	25.1
b5-40-X	5,084	2,209	997	1.0	8.8	494.781	34.0	41.2	36.2	42.2	3.2	3.3	6.6	18.9
b5-50-X	8,081	4,055	1,603	19.8	9.2	606.820	64.4	68.4	82.2	65.6	7.0	13.7	20.7	801.5
b5-60-X	8,523	4,097	1,632	1.0	7.6	704.518	41.8	44.6	53.2	66.4	7.5	6.7	14.2	489.6
b6-48-X	7,719	3,416	1,414	290.2	23.2	577.822	62.0	62.2	81.2	61.6	6.1	17.8	23.9	1,153.4
b6-60-X	12,696	5,117	2,110	327.2	36.6	703.240	94.4	97.6	121.8	75.4	11.9	33.5	45.4	1h
b6-72-X	17,308	7,355	2,794	1.0	26.6	789.145	88.6	86.6	124.2	58.2	17.1	26.8	44.0	696.8
b7-56-X	44,278	8,665	3,868	0.8	2.8	603.941	79.0	78.6	144.6	90.4	76.2	43.8	120.1	347.4
b7-70-X	18,599	8,318	3,087	3,300.6	66.4	740.946	206.4	219.8	286.0	110.8	22.5	105.5	128.0	1h
b7-84-X	24,903	9,943	3,852	2,665.2	38.4	961.569	220.2	235.4	324.6	156.8	25.9	148.5	174.5	1h
b8-64-X	39,136	11,805	4,774	1,726.6	38.2	638.461	215.2	226.4	358.8	129.2	55.9	176.9	232.8	1h
b8-80-X	21,288	10,659	3,680	7,440.6	117.0	822.222	262.2	260.4	302.6	126.2	21.0	255.7	276.7	1h
b8-96-X	71,670	19,729	8,081	47,914.4	79.4	951.952	614.6	666.4	1,130.6	349.2	101.0	2,415.5	2,516.6	1h

Notes. Nodes refers to the number of nodes explored in the main MIP search tree (heuristic search tree nodes are not included), and Cuts is the total number of IPEC and cycle elimination constraints added to the RFN model, across all stages of Algorithm 2. Obj. is the optimal objective. Under Solve time, Network refers to time spent generating fragments and building the network, whereas Tree is the time spent in branch and cut, including the time spent solving MIPs in heuristics. The IMP^{cut}_{strg} column is the time that Gschwind and Irnich (2015) report for the extended A and B instances using their best-performing BP variant (see tables EC.8 and EC.9, column "6/3" in the online appendix of Gschwind and Irnich 2015). A dash (—) is shown where the authors did not report results, and "1h" is shown where their algorithm did not terminate within the 1-hour time limit.

Chassaing, Duhamel, and Lacomme 2016, Gschwind and Drexl 2019), these problems have been solved using inexact methods. To the best of our knowledge, there are no solutions published that have been proven to optimality. We were able to solve the three smallest instances within the one-hour time limit, and they are shown at the top of Table 3. Cordeau and Laporte (2003) were the first to find these solutions

Table 3. Results for RS Instances and Selected R Instances

		Network size			Branch and Cut					Valid inequalities				Solve time (seconds)		
Instance	n	|F|	|A|	|N|	Nodes	Cuts	LB	UB	Gap, %	AF	FA	FF	SR	Network	Tree	Total
R1a	24	62,301	4,206	2,590	0.0	0.0	*	190.019	*	27.0	27.0	37.0	44.0	131.0	17.1	148.1
R7a	36	259,009	29,534	12,433	391.6	149.6	*	291.711	*	312.0	328.8	600.2	96.8	841.6	1,099.2	1,940.8
R1b	24	280,041	14,713	7,483	1.0	11.4	*	164.460	*	83.4	84.4	219.4	69.8	1,937.2	432.0	2,369.2
R1a-S	24	555	512	219	0.0	0.0	*	206.725	*	0.0	0.0	0.0	0.0	0.3	0.1	0.4
R2a-S	48	7,105	4,401	1,575	180.4	99.0	*	333.549	*	112.8	108.0	57.2	96.8	9.1	30.6	39.7
R3a-S	72	6,131	6,702	1,669	4,356.6	136.8	*	618.363	*	141.6	148.6	61.4	271.0	7.2	55.6	62.8
R4a-S	96	22,360	19,855	4,615	407.6	29.2	*	675.724	*	173.0	167.6	171.6	157.6	41.3	101.3	142.7
R5a-S	120	79,086	58,151	14,062	6,245.2	231.8	730.279	735.407	0.697	448.2	497.2	542.0	505.8	221.1	3378.9	1h
R6a-S	144	73,492	65,109	13,347	959.6	289.2	917.542	926.764	0.995	567.8	610.4	647.8	527.0	216.3	3,383.7	1h
R7a-S	36	1,287	1,369	460	1,177.8	136.8	*	329.364	*	60.6	60.2	25.2	44.0	0.9	28.5	29.3
R8a-S	72	9,266	8,699	2,212	20,660.8	508.6	*	585.619	*	423.0	458.8	286.2	307.0	12.8	686.2	699.0
R9a-S	108	27,322	27,011	5,671	4,865.0	3,455.4	766.676	—	—	638.8	598.2	472.2	489.6	60.7	3,539.3	1h
R10a-S	144	66,816	60,834	12,065	1,080.8	2,085.8	981.251	1,032.975	5.007	732.2	766.0	625.0	524.8	205.9	3,394.1	1h
R1b-S	24	1,454	950	387	1.0	6.0	*	185.165	*	4.0	4.0	2.0	4.0	1.5	0.5	2.0
R2b-S	48	13,271	8,226	2,538	5,278.8	147.0	*	335.791	*	194.0	236.4	208.4	129.8	28.5	118.2	146.7
R3b-S	72	25,525	21,892	4,741	16,868.4	553.8	573.240	581.314	1.389	547.8	614.2	527.8	474.8	89.3	3,510.7	1h
R4b-S	96	102,964	80,572	15,697	303.6	646.0	611.641	630.723	3.025	524.6	566.0	698.8	295.6	652.0	2,948.0	1h
R5b-S	120	300,737	211,146	41,493	0.0	5.2	162.280	—	—	146.0	162.0	192.4	81.2	2,707.6	892.4	1h
R6b-S	144	292,293	234,070	42,401	0.0	0.0	—	—	—	0.0	0.0	0.0	0.0	2,656.5	943.5	1h
R7b-S	36	4,600	3,706	1,052	1,090.4	84.2	*	290.341	*	91.2	93.6	46.6	62.2	7.8	23.6	31.4
R8b-S	72	21,825	17,850	4,187	11,975.8	2,70.4	*	532.284	*	335.4	370.0	276.8	376.0	58.6	690.7	749.3
R9b-S	108	90,328	74,589	14,356	877.2	1,522.0	686.029	725.532	5.445	636.2	622.0	645.8	426.6	430.3	3,169.7	1h
R10b-S	144	238,473	208,725	34,831	0.6	6.0	894.969	—	—	524.2	570.4	505.8	288.4	2,074.6	1,525.4	1h

Notes. n is the number of customers in the instance. LB is the final lower bound on the objective, which is averaged across the five runs for each instance. UB is the best final objective upper bound across five runs, as not all runs produced a solution within the time limit. Gap is LB expressed as a percentage of UB. All other columns have the same meaning as in Table 2. A dash (—) is shown for instances where no integer solution and/or LB was found, and "1h" is shown where their algorithm did not terminate within the 1-hour time limit. For LB and Gap, an asterisk (*) denotes instances solved to optimality.

using a tabu search but with no indication of optimality. For all other instances, we failed to generate all fragments within the time limit.

The number of fragments generated in the R instances is much larger because of increased numbers of customers and wider time windows. Potential fragments (elements of F′) are generated by extending paths starting at the origin depot, but ride time constraints are not guaranteed to be satisfied when extending paths to new locations, as the existence of a schedule is determined only after the path is complete (see Online Appendix B). Using resource extension functions as in Gschwind and Irnich (2015) to handle ride time constraints may provide speedups by reducing the number of paths that are extended. Furthermore, because we perform domination after all fragments have been generated, unlike in a labeling algorithm, it is straightforward to parallelize the generation of paths, where each parallel task consists of generating all paths starting from a given pickup node $p \in P$.

In order to investigate other limitations of the RFN model, we created a new benchmark set from the R instances by halving the ride times whilst leaving the time windows unmodified. These instances are named "RS" and denoted with a –S (S for short) suffix in the instance names. Shorter ride times will result in shorter and therefore, fewer fragments. Table 3 gives the results on these instances, showing that even for the largest RS instances, the number of fragments was at most 300,737. On the other hand, the number of nodes, |N|, is much larger when compared with A and B instances because of the larger number of customer requests.

The RS instances also require substantially more IPECs and tend to contain more violated valid inequalities. This is most likely because of the wider time windows, which make illegal chains more likely. The fragment network $(N, F \cup A)$ may be viewed as a (very) coarse, implicit time discretization, where each node $n = (i, L)$ is associated with a single point in time: the start of the time window at i, e_i. As in the explicit time discretization in Alyasiry, Forbes, and Bulmer (2019), by eliminating illegal chains, the IPECs correct "rounding-down errors" that result from traversing multiple fragments. When the time windows widen, these errors become larger, and the likelihood of an IPEC being violated increases. Figure 4 shows the lower bounds for some RS instances and the three solved R instances. The root LP integrality gap is worse for the RS instances than for the A and B instances, likely because of wider time windows. This gap closes significantly at the lifted LP relaxation (i.e., after valid inequalities are included at the root node), showing that our valid inequalities are effective for harder instances as well.

Table 4. Results for RS Instances Using a Relaxed RFN Model Based on F′

	Network size		Branch and Cut			Solve time (seconds)			
Instance	|F′|	|F′| – |F|	LB	UB	Gap, %	Network	Δ Network	Total	Δ Total
R1a-S	993	438	*	206.725	*	0.3	0.0	0.5	0.1
R2a-S	15,218	8,113	*	333.549	*	7.3	−1.8	86.4	46.6
R3a-S	12,061	5,930	*	618.363	*	6.1	−1.0	172.2	109.4
R4a-S	47,042	24,682	*	675.724	*	33.9	−7.4	579.9	437.2
R5a-S	180,292	101,206	—	—	—	352.6	131.5	1h	—
R6a-S	161,114	87,622	—	—	—	348.7	132.4	1h	—
R7a-S	2,121	834	*	329.364	*	0.7	−0.1	37.4	8.0
R8a-S	18,877	9,611	*	585.619	*	10.2	−2.6	1,212.3	513.3
R9a-S	58,361	31,039	764.180	—	—	51.2	−9.4	1h	—
R10a-S	146,974	80,158	975.209	—	—	199.6	−6.3	1h	—
R1b-S	3,088	1,634	*	185.165	*	1.2	−0.4	3.3	1.4
R2b-S	29,149	15,878	*	335.791	*	15.8	−12.7	745.9	599.2
R3b-S	58,820	33,295	570.060	581.399	1.950	62.7	−26.6	1h	—
R4b-S	256,879	153,915	—	—	—	500.2	−151.9	1h	—
R5b-S	879,160	578,423	—	—	—	2,897.6	190.0	1h	—
R6b-S	753,511	461,218	—	—	—	3,486.6	830.1	1h	—
R7b-S	10,085	5,485	*	290.341	*	5.3	−2.6	92.7	61.3
R8b-S	49,464	27,639	*	532.284	*	42.9	−15.6	2,949.7	2,200.4
R9b-S	218,488	128,160	—	—	—	411.9	−18.4	1h	—
R10b-S	606,217	367,744	—	—	—	2,811.5	736.9	1h	—

Notes. |F′| – |F| is the number of additional fragments compared with Table 3. Δ Network and Δ Total are the differences in network construction and total solve times, respectively, when compared with Table 3. All other columns have the same meaning as in Tables 2 and 3. A dash (—) is shown for instances where no integer solution and/or LB was found, and "1h" is shown where the algorithm did not terminate within the 1-hour time limit. For LB and Gap, an asterisk (*) denotes instances solved to optimality.

As mentioned in Section 4, it is possible to use the set of potential restricted fragments F′ directly with our algorithm, rather than extracting F from F′. We still stipulate that only necessary arcs are added to the RFN (see Step 7 in Algorithm 1), so the complexity of the network construction decreases from $\mathcal{O}(|F'|^3)$ to $\mathcal{O}(|F'|^2)$. This may lead to an RFN where some fragments are "dead ends" (e.g., a delivery node with no arcs leaving it). These fragments and nodes are quickly found and removed with a simple linear search.

The effect of this modification on RS instances is shown in Table 4. In most cases, there are almost twice as many fragments in the network. The numbers of arcs and nodes remain unchanged because Step 7 in Algorithm 1 is still followed, so they are not shown. Unexpectedly, the time spent on network construction increases for the largest RS instances. Of the minimal potential restricted fragments in F′, most are not restricted fragments; between 65% and 83% can never be part of any route. Therefore, when arcs are constructed and minimal fragments are paired, the time spent finding the necessary arcs grows dramatically when using potential minimal fragments, as opposed to minimal fragments. On the other hand, the $\mathcal{O}(|F'|^3)$ running time of Algorithm 1 is because of checking fragments, which start and end nonempty (Step 6). These fragments are uncommon, making up 10%–17% of F′ in RS instances. The $\mathcal{O}(|F'|^3)$ time bound of checking these fragments is pessimistic in practice and is dominated by the run time of arc construction.

Of course, it is also possible to add arcs without checking whether they are required, reducing the $\mathcal{O}(|F'|^2)$ running time to $\mathcal{O}(|F'|)$. However, even when the network generation time decreases in Table 4, the difference is eclipsed by the increase in running time during branch and cut. In particular, a large amount of time is spent at the root node obtaining the initial lower bound with valid inequalities. This highlights the shortcomings of this approach; the LP relaxation suffers greatly when unnecessary fragments are in the model. This will only worsen when additional arcs are included as well.

To mitigate the effect of wide time windows, an explicit time discretization may be required, where nodes at indexed by different time points as well as by location and load, but this would further increase the number of flow constraints in the RFN model. Dynamic discretization discovery (He et al. 2018) may alleviate this problem, by excluding unnecessary time points from the discretization. Time discretization would also improve the lower bound further, as many of the valid inequalities become implicit in the network structure.

Instead of building routes from restricted fragments, one could also use paths that contain only pickup or only delivery locations. These objects would essentially be half of a restricted fragment and much less numerous than restricted fragments for large instances. The disadvantage is that the number of network nodes would grow, and it is not clear how to enumerate these paths, without first enumerating restricted fragments. The lower bounds are also expected to worsen, but this may be solved by combining this approach with a time discretization. These possible improvements are left to future research.

8. Conclusion

This paper introduced a new formulation for the DARP based on route segments called restricted fragments. These restricted fragments are more fundamental route building blocks than the fragments of Alyasiry, Forbes, and Bulmer (2019) and therefore, less numerous, allowing them to model DARP routes more effectively. We proposed a set of lifted IPECs and strong valid inequalities for use in a branch and cut algorithm as well as two domination criteria for reducing the number of restricted fragments in the model, leveraging previous work on DARP route feasibility by Gschwind and Irnich (2015). Computational results showed that this method is superior to current branch and price and cut algorithms based on set-partitioning models, obtaining similar lower bounds with vastly fewer variables. The idea of restricted fragments can be applied to other pickup and delivery problems such as the PDPTWL with handling (Veenstra et al. 2017), where the fragments of Alyasiry, Forbes, and Bulmer (2019) become too numerous to use without delayed column generation because of relaxed LIFO constraints. As a generalization of the PDPTW, the algorithm presented in this paper is almost directly applicable to the PDPTW, with the advantage that domination of restricted fragments is simpler and more powerful. In the broader scope of p-step formulations (Munari, Dollevoet, and Spliet 2017) and other hybrid set-partitioning models, we hope that further research will lead to new formulations for other pickup and delivery problems.

Acknowledgments

The authors would like to thank the two anonymous referees for their suggestions and comments on the paper.

References

Alyasiry AM, Forbes M, Bulmer M (2019) An exact algorithm for the pickup and delivery problem with time windows and last-in-first-out loading. *Transportation Sci.* 53(6):1695–1705.

Barnhart C, Johnson EL, Nemhauser GL, Savelsbergh MWP, Vance PH (1998) Branch-and-price: Column generation for solving huge integer programs. *Oper. Res.* 46(3):316–329.

Braekers K, Caris A, Janssens GK (2014) Exact and meta-heuristic approach for a general heterogeneous dial-a-ride problem with multiple depots. *Transportation Res. Part B: Methodological* 67(2014):166–186.

Chassaing M, Duhamel C, Lacomme P (2016) An ELS-based approach with dynamic probabilities management in local search for the dial-a-ride problem. *Engrg. Appl. Artificial Intelligence* 48(2016):119–133.

Cordeau JF (2006) A branch-and-cut algorithm for the dial-a-ride problem. *Oper. Res.* 54(3):573–586.

Cordeau JF, Laporte G (2003) A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Res. Part B: Methodological* 37(6):579–594.

Desaulniers G, Gschwind T, Irnich S (2020) Variable fixing for two-arc sequences in branch-price-and-cut algorithms on path-based models. *Transportation Sci.* 54(5):1170–1188.

Detti P, Papalini F, de Lara GZM (2017) A multi-depot dial-a-ride problem with heterogeneous vehicles and compatibility constraints in healthcare. *Omega* 70(2017):1–14.

Firat M, Woeginger GJ (2011) Analysis of the dial-a-ride problem of Hunsaker and Savelsbergh. *Oper. Res.* 39(1):32–35.

Gschwind T (2015) A comparison of column-generation approaches to the synchronized pickup and delivery problem. *Eur. J. Oper. Res.* 247(1):60–71.

Gschwind T, Drexl M (2019) Adaptive large neighborhood search with a constant-time feasibility test for the dial-a-ride problem. *Transportation Sci.* 53(2):480–491.

Gschwind T, Irnich S (2015) Effective handling of dynamic time windows and its application to solving the dial-a-ride problem. *Transportation Sci.* 49(2):335–354.

Haugland D, Ho SC (2010) Feasibility testing for dial-a-ride problems. Chen B, ed. *Algorithmic Aspects in Information and Management.* AAIM 2010, Lecture Notes in Computer Science, vol. 6124 (Springer, Berlin), 170–179.

He E, Boland N, Nemhauser G, Savelsbergh M (2018) A dynamic discretization discovery algorithm for the minimum duration time-dependent shortest path problem. van Hoeve WJ, ed. *Integration of Constraint Programming, Artificial Intelligence, and Operations Research.* CPAIOR 2018, Lecture Notes in Computer Science, vol. 10848 (Springer, Cham, Switzerland), 289–297.

Hunsaker B, Savelsbergh M (2002) Efficient feasibility testing for dial-a-ride problems. *Oper. Res. Lett.* 30(3):169–173.

Jepsen M, Petersen B, Spoorendonk S, Pisinger D (2008) Subset-row inequalities applied to the vehicle-routing problem with time windows. *Oper. Res.* 56(2):497–511.

Luo Z, Liu M, Lim A (2019) A two-phase branch-and-price-and-cut for a dial-a-ride problem in patient transportation. *Transportation Sci.* 53(1):113–130.

Miller CE, Tucker AW, Zemlin RA (1960) Integer programming formulation of traveling salesman problems. *J. ACM* 7(4):326–329.

Munari P, Dollevoet T, Spliet R (2017) A generalized formulation for vehicle routing problems. Preprint, submitted September, https://www.researchgate.net/publication/303840066_A_generalized_formulation_for_vehicle_routing_problems.

Parragh SN, Schmid V (2013) Hybrid column generation and large neighborhood search for the dial-a-ride problem. *Comput. Oper. Res.* 40(1):490–497.

Ropke S (2006) Heuristic and exact algorithms for vehicle routing problems. PhD thesis, University of Copenhagen, Copenhagen.

Ropke S, Cordeau JF (2009) Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Sci.* 43(3):267–286.

Ropke S, Cordeau JF, Laporte G (2007) Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks* 49(4):258–272.

Savelsbergh MWP, Sol M (1995) The general pickup and delivery problem. *Transportation Sci.* 29(1):17–29.

Sippel L (2019) An exact algorithm for the ER-DARP. Unpublished honours thesis, University of Queensland, Brisbane, Australia.

Tang J, Kong Y, Lau H, Ip AW (2010) A note on "efficient feasibility testing for dial-a-ride problems." *Oper. Res. Lett.* 38(5):405–407.

Veenstra M, Cherkesly M, Desaulniers G, Laporte G (2017) The pickup and delivery problem with time windows and handling operations. *Comput. Oper. Res.* 77(2017):127–140.

Zhang Z, Liu M, Lim A (2015) A memetic algorithm for the patient transportation problem. *Omega* 54(2015):60–71.