

# Long-Distance Directional Dial-a-Ride Problems\*

Grzegorz Gutowski<sup>a</sup> and Grzegorz Herman<sup>b</sup>

*Theoretical Computer Science, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland*

**Keywords:** Vehicle Routing, Dial-a-Ride Problem, Local Change Algorithm.

**Abstract:** We consider vehicle routing problems that occur in practice in the context of long-distance ride-sharing. On the one hand, the instances of our problems share the helpful property that the passengers travel in roughly the same geographical direction. On the other, the required cost function has ordering-dependent components. For two such problems, we provide heuristic algorithms employing a dynamic programming optimization of a sliding window in appropriate linear orders. In the first, exemplary problem, we route a single vehicle. In the second, we route a fleet of vehicles with a coordinated stopover and exchange of passengers. The size of the sliding window allows for trade-offs between solution qualities and processing times. Both algorithms are effective and efficient on data sets representing actual travel requests from Hoper, a commercial ride-sharing service operated by Teroplan S.A. in Poland.

## 1 INTRODUCTION

In this paper we describe two algorithms for variants of vehicle routing problems, which are of practical importance for a company providing a long-distance door-to-door ride-sharing service. In this setting, the vehicles and drivers are provided by the operator, while each passenger requests to travel from an arbitrary pickup to an arbitrary delivery location. The requests are primarily long-distance, i.e., the travel distance between pickup and delivery location is in the order of hundreds of kilometers. The tickets are usually sold before the travel date and the operator optimizes the assignment of vehicles and their routes when all the passenger requests are known.

There is a vast literature on similar problems like Capacitated Dial-a-Ride (CDARP), Traveling Salesman with Pickup and Delivery (PDTSP), or more generally Vehicle Routing with Pickup and Delivery (VRPPD), see (Parragh et al., 2008; Molenbruch et al., 2017; Ho et al., 2018; Tafreshian et al., 2021; Ritzinger et al., 2022). All of these problems are versions of the Traveling Salesman Problem with additional constraints and all are inherently difficult to solve optimally. There are numerous practical ap-

proaches to these problems based on integer linear programming (Ropke et al., 2007; Cordeau, 2006; Kikuchi, 1984), dynamic programming (Psaraftis, 1980; Psaraftis, 1983; Desrosiers et al., 1984; Bianco et al., 1994), and local search (Savelsbergh, 1990; Healy and Moll, 1995; Renaud et al., 2000; Renaud et al., 2002).

In the particular case of our application, we have found that the following natural structure occurs in the problem instances. With passengers spending a significant amount of time on board the vehicle, their natural expectation is for the travel not to take much longer than necessary. Choosing a ride-sharing operator, they are implicitly accepting some degree of detours, but should the route stray too much (subjectively) from the direct one, they might become hesitant to choose the same operator again.

On the one hand, this requirement must be represented in the modeling and influence the optimization target. On the other, it suggests an important business level constraint: In order to avoid carrying passengers a significant distance in the direction perpendicular or even completely opposite to the one they requested, it makes sense to assign them initially to a fixed set of **routes**, each based on some travel path with a relatively stable long-term direction and serving pickup and delivery locations not too far from this path. In our setting, passengers are informed of these tentative routes when buying the ticket.

Henceforth, when solving our routing problems,

<sup>a</sup> <https://orcid.org/0000-0003-3313-1237>

<sup>b</sup> <https://orcid.org/0000-0001-6855-8316>

\*This work has been commissioned by Teroplan S.A. and partially financed by European Union funds (grant number: RPMP.01.02.01-12-0572/16-01)

we can assume that the instances are **directional**: all the passengers in a single instance of a problem travel roughly in the same geographical direction. This allows us to propose practical solutions which may not be applicable in general, but turn out to be very effective and efficient in this particular setting.

Traditionally, the cost function being minimized in vehicle routing is modeled as the distance in the graph representing the road network. While the lengths of individual edges might capture something more than simply the road distance, such model has the inherent limitation of being context-insensitive and does not allow to express the requirement of minimizing the detours experienced by individual passengers. In Section 2, we introduce a family of cost functions in which the cost of a segment between successive vehicle stops might depend on the set of passengers on board. This captures not only the detour length, but also other metrics important from the point of view of passenger experience.

In this paper we present solutions for two problems with the above properties. In Section 3, we limit our considerations to routing a single vehicle. This corresponds to a standard Single Vehicle Capacitated Dial-a-Ride Problem. For this problem it is possible to develop algorithms based on the aforementioned techniques. Still, we present our solution as a gentle introduction to our method that works well for both problems. In Section 4 we deal with multiple vehicles and allow for solutions where passengers can change their vehicles during a designated time-synchronized stopover. For this problem we have failed to procure a working solution based on integer linear programming. The main obstacle for that seems to be the structure of the selected cost function which is discussed in detail in Section 2.

Both solutions are based on the local search approach with the following structure. First, we produce a heuristic linear ordering of the objects (route points, or passengers) that is a starting point for the optimization process. The final solution is then obtained using dynamic programming that optimizes every consecutive segment of a fixed-length in this ordering. We call this technique **peephole optimization**, in parallel to a similar technique employed by compilers for code optimization. An important feature of this method is its flexible window size, allowing for a trade-off between solution quality and processing time. This becomes very desirable when the costs of a large number of instances need to be estimated, with only a few of them being actually solved. This happens when we use the single-vehicle algorithm as a sub-procedure in the multiple-vehicle algorithm.

We provide experimental evaluation of our methods on real-world data sets provided by Teroplan S.A., a commercial ride-sharing operator offering the service in Poland. See Figure 1 for the maps of the routes that were used in the evaluations. As these data sets include problems with a very limited number of passengers, we also show that the same algorithms are capable of handling larger, artificially constructed, instances.

## 2 COST FUNCTION

Let us begin by formalizing the optimization criteria. As mentioned in the Introduction, the typical model employed in vehicle routing is that of a graph distance. While elegant in its simplicity and helpful from the point of view of algorithm design, it is unfortunately too simplistic to capture the priorities occurring in practical business operations.

The primary customers of a ride-sharing operator are the passengers. From long-term business perspective it is thus important to consider not only the operating costs, but also the likelihood of passengers choosing the same operator in the future.

Based on data gathered by Teroplan S.A., the following quantities influence the perceived quality of service:

- detour: by how much the distance (or time) traveled by the passenger exceeds the optimal route they could have taken not sharing a ride;
- stops: how many times the vehicle stops while the passenger is on board; and
- leeway: with how many other people the passenger travels.

Quite intuitively, passengers prefer to travel on a direct route, without stops, and on a relatively empty vehicle. Optimization of these goals is very much against the optimization of operating costs.

It turns out that all of the above can be expressed in a model, in which the cost of a route segment between successive vehicle stops is an affine function of the road distance, with coefficients determined by the set of passengers on board. Formally, we let the cost introduced by the segment  $s$  be

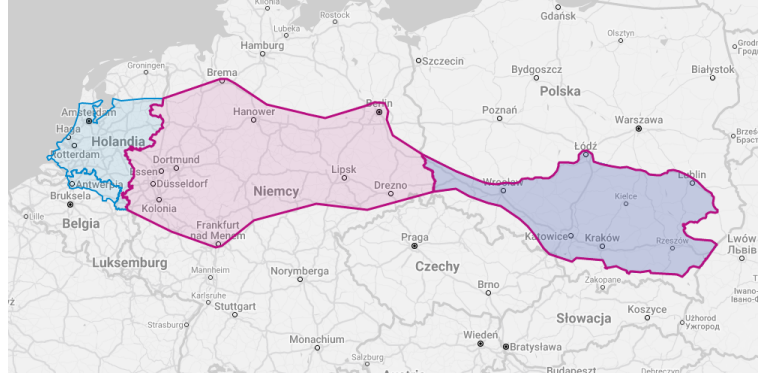
$$c_s := a_{P_s} \cdot d_s + b_{P_s},$$

where  $P_s$  is the set of passengers on board the vehicle (determining the value of coefficients  $a_{P_s}$  and  $b_{P_s}$ ), and  $d_s$  is the distance traveled on segment  $s$ .

Let us show how such formulation captures the above aspects of passenger experience.



(a) Single vehicle problem. Data contained requests for 8 of 17 national routes in Poland gathered for 16 months



(b) Multiple vehicle problem. Data contained requests for an international route spanning parts of Poland, Germany, Netherlands, and Belgium with stopover close to Polish-German border gathered for 14 months

Figure 1: Maps of routes used for evaluation of algorithms.

**Detour.** For each passenger  $p$ , let  $\hat{d}_p$  be the optimal distance between their requested endpoints. Assume that the “discomfort” cost of a detour of length  $d$  can be expressed as  $\delta(\hat{d}_p) \cdot d$  for some arbitrary function  $\delta$  (this might account for the fact that the same absolute detour might be more acceptable when the optimal route is longer). Letting  $S_p$  be the set of route segments on which they have been on board, the detour length experienced by passenger  $p$  is  $\sum_{s \in S_p} d_s - \hat{d}_p$ , and the associated cost equals  $\sum_{s \in S_p} \delta(\hat{d}_p) \cdot d_s - \delta(\hat{d}_p) \cdot \hat{d}_p$ . The first term can be incorporated in the total cost of all segments by making  $\delta(\hat{d}_p)$  contribute to  $a_{P_s}$  whenever  $p \in P_s$ , and the second term does not depend on vehicle routes and thus is irrelevant for the optimization.

**Stops.** The number of “unnecessary” stops experienced by passenger  $p$  is  $|S_p| - 1$ . To make each of them introduce a cost of  $\sigma$ , it is enough to add  $\sigma$  to  $b_{P_s}$  whenever  $p \in P_s$ .

**Leeway.** The experience of (lack of) leeway happens over time. Therefore the appropriate cost component for each passenger  $p$  over the segment  $s$  can be expressed as some constant  $\lambda$  (possibly dependent on the capacity of the vehicle), multiplied by  $d_s \cdot (|P_s| - 1)$ . Again, this cost can be incorporated into our model by increasing  $a_{P_s}$  by  $\lambda \cdot |P_s|^2$ .

Overall, all required aspects of the cost can be modeled by taking

$$a_{P_s} := \sum_{p \in P_s} \delta(\hat{d}_p) + \lambda \cdot |P_s|^2 \quad \text{and} \quad b_{P_s} := \sigma \cdot |P_s|,$$

with the values of  $\delta$ ,  $\sigma$ , and  $\lambda$  calibrated to express the relative importance of the aspects.

In the practical software implementation of our algorithms there are some more constraints that need to be considered, like for example a maximal total dis-

tance a vehicle can travel. We have decided not to discuss them in order to keep the presentation cleaner.

### 3 SINGLE VEHICLE PROBLEM

In this section we are interested in finding an optimal route for a single vehicle. The vehicle has a limited passenger capacity and goes from some initial location to some final location. On the way it must serve a set of passengers, each of whom is to be transported from their pickup location to their delivery location.

It is worth noting that this problem can be solved using integer linear programming methods, or even brute force methods (at least for the instance sizes that we use to evaluate our method). We present our solution, as it easily scales to larger instances, and the presentation serves as a gentle introduction to our approach. In Section 4 we employ the same general approach to solve a more demanding problem, for which we didn’t find a solution based on standard methods.

Let us begin by specifying the problem more formally. We are given a set  $X$  of geographic **locations**. These include the pickup and delivery locations requested by the passengers, and initial and final locations of the vehicle. Locations can be represented by vertices of a strongly connected directed graph modeling the road network. However, as detailed routes are not really relevant to our solution, we will assume the graph to be the complete graph on  $X$ , with the road distance between each pair of locations  $x$  and  $y$  given as  $d_{x,y}$ .

Next, we have a set  $P$  of passengers to be transported, with pickup and delivery locations of a passenger  $p$  denoted by  $p_{\text{in}}$  and  $p_{\text{out}}$ , respectively. Each location might be requested as a pickup and/or deliv-

ery location by some numbers of passengers. In case of a point which is simultaneously a pickup and a delivery location, it always makes sense to let the passengers being delivered leave the vehicle before the ones being picked up board it. Therefore we can focus on the net occupancy change at each point  $x \in X$ , computed as

$$c_x := |\{p \in P : p_{\text{in}} = x\}| - |\{p \in P : p_{\text{out}} = x\}|.$$

The vehicle has a limited passenger capacity  $c$ , begins the journey at location  $x_0$  with no passengers on board, visits every location in  $X$  exactly once, and finishes at location  $x_n$  after all passengers have been delivered.

Now, a **solution** to the single vehicle problem is a linear ordering  $\prec$  on  $X$ , satisfying the following **validity constraints**:

- the vehicle starts and ends as specified:  $\forall x \in X : x_0 \prec x \prec x_n$ ,
- every pickup location is visited before the corresponding delivery location:  
 $\forall p \in P : p_{\text{in}} \prec p_{\text{out}}$ , and
- vehicle capacity is never exceeded:  $\forall x \in X : c \geq \sum_{y \prec x} c_y$ .

Whenever a particular ordering is clear from the context, we might refer to its consecutive locations using indices:  $x_0 \prec x_1 \prec \dots \prec x_n$ .

To specify optimization criteria, we need to measure the **cost** of a particular solution. The detailed composition of a practical cost function has been discussed in Section 2. Here, we only assume it to have the following general properties:

1. The total cost of the solution can be expressed as the sum of costs of the route segments between successively visited locations.
2. The cost of each segment might depend on the road distance  $d$  covered on this segment and on the set  $V \subseteq X$  of locations already visited, but *not* on the order in which they are visited. This segment cost function will be denoted as  $\kappa(V, d)$ .

Note that the penalties discussed in Section 2 can be easily expressed in this way, because the set of visited locations implies the actual set of passengers on board the vehicle.

Overall, we wish to optimize

$$\kappa_{\prec} := \sum_{i=0}^{n-1} \kappa(\{x_0, \dots, x_i\}, d_{x_i, x_{i+1}}),$$

over all linear orders  $\prec$  satisfying the validity constraints. Note, that while our algorithms can work with arbitrary  $\kappa(V, d)$ , for the purpose of experimental evaluation we use the simplest possible function

$\kappa(V, d) = d$  that implies optimization of the total distance traveled by the vehicle.

Being a generalization of the asymmetric Traveling Salesman Problem, this problem is computationally hard (NPO-hard), even when the cost function satisfies the triangle inequality. The best known approximation ratio for the problem without additional constraints is  $22 + \varepsilon$  (Traub and Vygen, 2020). Thus, practical solutions, including the one presented in this work, employ various heuristics.

As outlined in the Introduction, our approach consists of two phases. First, we construct an initial ordering of locations satisfying the validity constraints. Then, we optimize this solution by considering all its viable “local” reorderings.

While multiple methods following this general scheme are known in the literature, none of them take advantage of the specific nature of our problem instances, discussed in Section 1. Compared, for example, to the classic cut-and-rejoin local search algorithm by Lin (Lin, 1965), our optimization phase can efficiently consider larger reshufflings by limiting them to segments of the initial ordering.

We provide more details on both phases in the following subsections.

### 3.1 Initial Ordering

Let us begin by constructing the initial ordering of locations. We will not attempt to make it a cost-wise approximation of the optimum. Instead, we want it to be “similar” to the optimum wrt. the indices occupied by particular locations (we will formalize this notion near the end of Section 3.2).

To this aim, we exploit the assumption of all passengers sharing a general direction of travel. For each passenger  $p$ , we construct the vector  $v_p$  representing the direction from  $p_{\text{in}}$  to  $p_{\text{out}}$ . From all  $v_p$ , we compute the average<sup>1</sup> direction  $v$  which should represent the general direction of travel. Finally, we assign priorities to the locations, highest priority to  $x_0$ , lowest to  $x_n$ , and decreasing priorities to other locations in the order of their projection on the average direction  $v$ .

The resulting order given by the priorities might still violate precedence and capacity constraints. While the second phase of our approach does not formally require them to be satisfied, it only ever considers orders “similar” to the initial one and thus will fail to find a solution if none of those are valid. This issue

<sup>1</sup>The average might be weighted, with weights depending non-linearly on the distances between pickup and delivery locations.

is easily fixed by making sure that the order is valid up front.

To achieve this we use the following procedure. We construct the order by adding locations at the end one by one. In each step we consider each location which has not yet been added and which, when added at the end, would not violate any of the constraints. Among such locations we select the one with the highest priority.

The above process is easily implemented in time  $O(cn + n \log n)$ . While in general it does not guarantee to find a valid order even if one exists,<sup>2</sup> we have never observed such failure in practice.<sup>3</sup> Indeed, the orders produced are usually very close to the “purely projective”. Important discrepancies show up in areas of mixed pick-ups and deliveries, in which locations further down the route might need to be visited earlier in order to prevent capacity constraint violation.

### 3.2 Peephole Optimization

Given an initial ordering  $x_0 \prec x_1 \prec \dots \prec x_n$  of locations, we can frame the peephole optimization algorithm as dynamic programming.

Fixing a parameter  $w$ , called the **window length**, and denoting by  $I$  the set  $\{0, \dots, n\}$  of location indices, each sub-problem  $\langle k, W, j \rangle$  to be solved will be parameterized by:

- a **prefix length**  $k \in I$ ,
- a **window**  $W \subseteq \{k+1, \dots, k+w\} \cap I$ , and
- a **final index**  $j \in W \cup \{k\}$ .

The solution to the sub-problem  $\langle k, W, j \rangle$  will be the (locally) optimal order of visiting all the locations that are either in  $W$ , or no later than  $k$ -th in  $\prec$ , i.e., locations  $V_{k,W} := \{x_0, \dots, x_k\} \cup \{x_i : i \in W\}$ , with  $x_j$  coming last. In particular, the whole problem is captured as the sub-problem  $\langle n, \emptyset, n \rangle$ .

A sub-problem  $\langle k, W, j \rangle$  is deemed **viable** if it satisfies the following conditions:

- $j = 0$  implies  $W = \emptyset$  (no location can be visited before  $x_0$ ),

<sup>2</sup>For an example, consider six passengers  $A, B, C, D, E, F$  and a linear route with six points:  $A_{\text{in}}, B_{\text{in}}, A_{\text{out}} = C_{\text{in}} = D_{\text{in}}, B_{\text{out}} = E_{\text{in}} = F_{\text{in}}, C_{\text{out}} = D_{\text{out}}, E_{\text{out}} = F_{\text{out}}$ . A vehicle with capacity 2 can handle this route, but *not* once it has picked up both  $A$  and  $B$ .

<sup>3</sup>Should such unfortunate case ever occur, one can deal with it by artificially duplicating each location so that all pickup and delivery locations are distinct. This might cause some geographical locations to be visited more than once, but guarantees to find a valid order, in the worst case serving the passengers one-by-one.

- $p_{\text{out}} = j$  implies  $p_{\text{in}} \in V_{k,W}$  (passengers delivered at the last location have been picked up earlier), and
- $c \geq \sum_{y \in V_{k,W}} c_y$  (the set of visited locations meets capacity constraints).

A sub-problem  $\langle k, W, j \rangle$  is called **canonical** if either  $k = 0$  or  $k + w \in W$ . When neither of these holds then  $W \cup \{k\}$  is a correct window for prefix length  $k - 1$ ,  $V_{k,W} = V_{k-1, W \cup \{k\}}$ , and one could replace sub-problem  $\langle k, W, j \rangle$  by the equivalent sub-problem  $\langle k - 1, W \cup \{k\}, j \rangle$ . Thus, every non-canonical sub-problem can be reduced to a canonical one, while non-viable ones can either be assigned infinite cost or completely removed from the consideration.

We wish to define **locally optimal solution** for a sub-problem, inductively, as a solution that extends some locally optimal solution for a smaller sub-problem by visiting one location more and optimizing cost over all such extensions. More formally, locally optimal solution to viable canonical sub-problem  $\langle k, W, j \rangle$  is naturally defined as follows:

- $\mathbf{j} = \mathbf{0}$ . The only viable sub-problem here is  $\langle 0, \emptyset, 0 \rangle$ , with zero cost.
- $\mathbf{j} = \mathbf{k} > \mathbf{0}$ . With location  $x_k$  visited last, the prefix length of the sub-problem covering the remaining locations must have been smaller than  $k$ . But then, with  $k + w \in W$ , such situation cannot be expressed by our parameterization (its window would also need to contain  $k + w$ , exceeding its allowed size) and thus will simply *not* be considered. We thus return an infinite cost. This is a place in which our solution might not be globally optimal.
- $\mathbf{j} > \mathbf{k}, \mathbf{j} < \mathbf{k} + \mathbf{w}$ . With neither  $x_k$ , nor  $x_{k+w}$  visited last, the prefix length of the preceding sub-problem must be the same. This sub-problem might thus have been  $\langle k, W \setminus \{j\}, j' \rangle$ , for an arbitrary (legal)  $j'$ . We consider all of them, augment their solutions by the segment from  $x_{j'}$  to  $x_j$ , and choose the one with minimal cost.
- $\mathbf{j} = \mathbf{k} + \mathbf{w}$ . This case is similar to the previous one, but we allow the prefix length of the preceding sub-problem to be smaller. This sub-problem might have been  $\langle k', W', j' \rangle$  for an arbitrary (legal)  $k' \leq k, k' \geq k - w, j'$ , and  $W' = \{k' + 1, k' + 2, \dots, k\} \cup W \setminus \{j\}$ .

Note how the properties assumed of the cost function fit nicely into this setting. For each transition between successive sub-problems, their parameterizations prescribe the set of visited locations and the endpoints (and thus the road distance) of the corresponding segment, and hence its cost. With the costs

being additive, we only need to remember the cost-optimal solution to each sub-problem.

Note also, that while viability of  $\langle n, \emptyset, n \rangle$  does not by itself imply the validity of the constructed order, the algorithm is in fact correct. This is because its result is obtained from a sequence of solutions to viable sub-problems with successively larger  $V_{k,W}$ 's. In such a sequence, the second and third conditions on viability imply respectively the precedence and capacity constraints.

The number of sub-problems is  $\Theta(nw2^w)$ , leading to the running time of  $\Theta(nw^22^w)$ . One might even take  $w := n$ , making our algorithm a direct analogue of the standard dynamic programming solution to the traveling salesman problem (Bellman, 1962; Held and Karp, 1962), guaranteed to find the *global* optimum. This course of action might be viable when solving a relatively small number of instances with small  $n$ .

When the number of instances considered becomes large, foregoing global optimality by lowering the window length might be necessary to obtain practical running times. Such situations occur naturally when the passengers are to be split between multiple vehicles, an example of which will be discussed in Section 4.

To capture the situations in which our algorithm behaves optimally, let us formally capture the notion of “similar” orders:

**Definition 1.** A permutation  $\pi : I \mapsto I$  is *w-local* iff for all indices  $i, j \in I$

$$\pi(i) - \pi(j) \geq w \implies i > j.$$

We can now prove the following:

**Proposition 2.** For an initial order  $x_0 \prec \dots \prec x_n$  and a *w-local* permutation  $\pi : I \mapsto I$  such that  $\pi(\prec) := x_{\pi(0)}, \dots, x_{\pi(n)}$  is a solution (i.e., satisfies validity constraints), peephole optimization of  $\prec$  with window length  $w$  returns a result with cost no worse than  $\kappa_{\pi(\prec)}$ .

*Proof.* Let us show that for every  $r = 0, \dots, n$ , there exists a viable sub-problem  $\langle k, W, j \rangle$  with  $j = \pi(r)$ , for which  $V_{k,W} = \{x_{\pi(0)}, \dots, x_{\pi(r)}\}$ .

We proceed by induction on  $r$ . For  $r = 0$ , we have  $\pi(0) = 0$ , and sub-problem  $\langle 0, \emptyset, 0 \rangle$  satisfies the conditions. Now, assume  $\langle k, W, j \rangle$  is a viable sub-problem with  $j = \pi(r)$  and  $V_{k,W} = \{x_{\pi(0)}, \dots, x_{\pi(r)}\}$ . Among different equivalent representations of the sub-problem, we choose the one, possibly non-canonical, that maximizes  $k$ . For this representation, we have  $k+1 \notin W$ .

Let  $j' = \pi(r+1)$ . If  $j' = k+1$ , then  $\langle k+1, W, k+1 \rangle$  is a viable sub-problem satisfying the conditions for  $r+1$ . Otherwise, we have that  $k+1 =$

$\pi(r')$  for some  $r' > r+1$ . The assumption on  $\pi$  being *w-local* gives that  $j' = \pi(r+1) < \pi(r') + w \leq k+w$ , and  $\langle k, W \cup \{j'\}, j' \rangle$  is a viable sub-problem satisfying conditions for  $r+1$ .  $\square$

As an immediate consequence we get that whenever some globally optimal solution is a *w-local* permutation of the initial order, our algorithm is optimal.

### 3.3 Evaluation

The experimental data for this problem comes from the operation of the Hoper ride-sharing service. Passenger requests for 8 different travel routes, out of 17 routes depicted in Figure 1a, have been gathered for 16 months, giving 2970 test cases in total. The vehicles used on these routes were able to accommodate 8 passengers. On average, there are 12.2 points to arrange per test case. Figure 2a shows the distribution of instance sizes. Observe that the size is more often an even number (which is to be expected when pickup and delivery locations are all different). It seems that at least some of the larger instances were not really handled by a single vehicle (the best solutions for these cases require to travel more than 2000 kilometers), but we have decided to keep them as they are.

As mentioned before, for evaluation purposes, we use  $\kappa(V, d) = d$  as the segment cost, and effectively we optimize the total distance traveled by the vehicle. The heuristic solution applied to all the test cases takes 1.6 seconds of CPU time<sup>4</sup> and finds routes with the total distance of 2092549 kilometers. The algorithm with window size set to 16 takes 70 seconds to compute and gives routes with the total distance of 1790980 kilometers, saving just above 100 kilometers on average per test case. Solving the largest test case with 33 locations takes 0.88 seconds to compute. Figure 2c shows the processing time and the total travel distance for other values of the window size. Figure 2e shows the quality gain for different sizes of test cases (measured in the number of locations) for window sizes 4, 8, 12, and 16.

It is worth noticing that most, but not all, of the cases in this data set can easily be solved by a brute-force algorithm that checks all the possible orderings of the locations. In order to evaluate our solution on larger instances we have artificially constructed a new data set, where each test case is constructed from two cases corresponding to the same route in the original data set by combining the passenger requests and

<sup>4</sup>All computations were performed on a single core of an Intel Xeon Gold 6154 processor

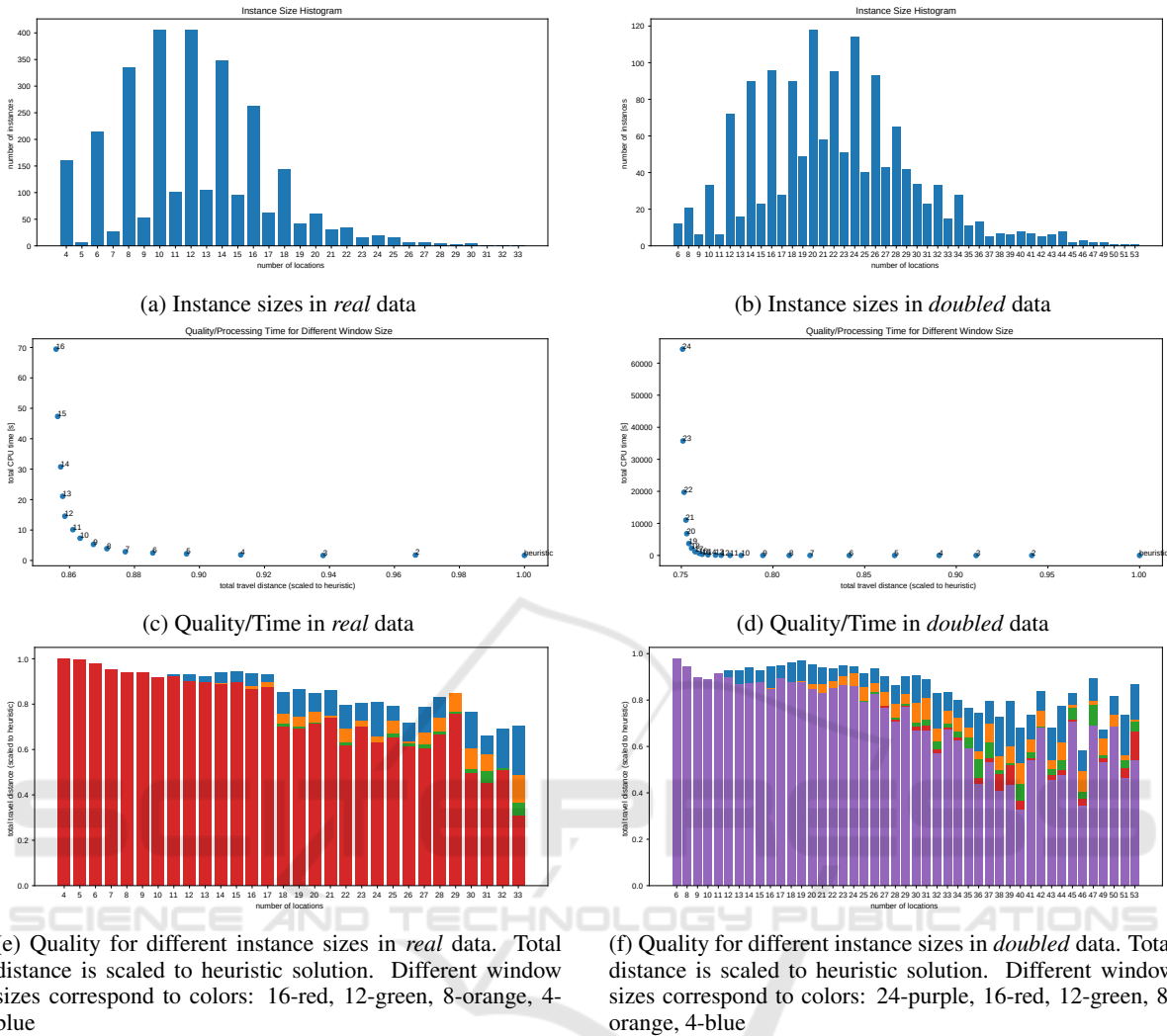


Figure 2: Experimental results for single vehicle problem.

doubling the capacity of the vehicle. This way we obtain 1842 test cases with vehicles of size 16 and 22.2 locations to visit on average. Figure 2b shows the distribution of instance sizes, and Figure 2d shows the processing time and the total travel distance for different window sizes. Figure 2f shows the quality gain for different sizes of test cases (measured in the number of locations) for window sizes 4, 8, 12, 16, and 24.

The evaluation shows that the window size 20 for the artificially constructed tests with vehicle of capacity 16 is still computable within one minute even for the test cases with over 50 locations and gives solutions that are difficult to improve by increasing the window size further.

## 4 MULTIPLE VEHICLES PROBLEM

We now turn to the more demanding problem of routing multiple vehicles. The general discussion from the Introduction still applies: the passengers in each problem instance are assumed to apply for the same route, thus requesting a similar direction of travel.

When the geographical area covered by the route is a narrow strip, the paths chosen for individual vehicles are not expected to be drastically different, and thus optimizing the assignment of passengers might not be as important as optimizing the route of each vehicle. However, with multiple vehicles available it makes sense to consider serving wider areas, with the hope of grouping passengers with similar direct path

between pickup and delivery locations into the same vehicle.

Moreover, as the travel is expected to take significant time, and thus stopovers are a near necessity for both the passengers and the drivers, it makes sense to look for solutions where passengers change vehicles during a stopover. The vehicles participating in such an exchange must be coordinated time-wise, and thus only a single exchange will be considered. This is however enough to make the grouping of (most) passengers according to their pickup locations independent from the grouping according to their delivery locations, enabling effective handling of a much greater diversity of possible requests.

With a single designated exchange location in place, the whole problem might be split into two symmetric sub-problems: routing the vehicles from their initial locations to the exchange, and routing them from the exchange to their final locations. With the two being symmetric, we will focus our attention only on the first one. Note however, that while one can reasonably expect the pickup and delivery locations of the majority of passengers to lie respectively before and after the exchange point, we still need to accommodate passengers with both pickup and delivery location visited before the exchange location.

The formal setting of our multiple vehicle problem is thus as follows. The set of locations  $X$  together with their distances  $d_{x,y}$ , the set of passengers  $P$  with their pickup and delivery locations  $p_{in}$  and  $p_{out}$ , vehicle capacity  $c$  and the segment cost function  $\kappa(V, d)$  are as specified in Section 3. However, we are now given a fleet of  $m$  vehicles<sup>5</sup> with common initial and final locations  $z_{in}, z_{out} \in X$  (the latter, being the exchange point, is also expected to be the delivery location of multiple passengers).

Also, due to the route still having a general direction, and the nature of the split, most of the locations in  $X$  are expected to lie in a common half-plane.

As stated, the problem lies in between the Capacitated Vehicle Routing Problem and Vehicle Routing Problem with Pickup and Delivery (CVRP and VRPPD; see, e.g., the survey (Toth and Vigo, 2001) for an exposition on both). The full solution needs to consist of an assignment of passengers to vehicles, and the routing of each vehicle (i.e., a valid order of pickup and delivery locations of passengers assigned to it).

<sup>5</sup>For simplicity, we consider all vehicles to be essentially identical, i.e., with the same capacity, cost function, and initial location. The algorithm presented here does generalize to a non-homogeneous vehicle fleet, but its presentation becomes much more complex, and its complexity grows significantly with fleet diversity, which makes it impractical for more than a few types of vehicles.

A priori there are no constraints on the passenger assignment. We do not even have to directly obey the capacity constraint, as a vehicle will in general serve more passengers than its capacity. In practice, we will however limit the number of “exchange” passengers (i.e., those being delivered to the exchange point) assigned to each vehicle to  $c$  to avoid visiting the exchange point more than once, and the total number of passengers assigned to each vehicle to no more than  $2c$  to limit the number of additional stops on the way.

Multiple known heuristic approaches to CVRP variants consist of two phases, following either a cluster-then-route or a route-then-cluster scenario. Here, having already presented a viable solution for the single vehicle problem which successfully takes advantage of our specific setting, we follow the cluster-then-route path. Restricting our attention to the yet unsolved part of the problem, we thus define the solution as an assignment of passengers to vehicles. Note however, that the single-vehicle algorithm will be employed not only after the assignment has been formed, but also (possibly with a smaller window size) to provide the cost estimation for many other possible assignments.

Our proposed solution to the assignment problem consists of two inner phases, analogous to those in the single-vehicle case. First, we heuristically construct an initial linear order on the passengers. Then, we apply dynamic programming following that order to obtain a “locally optimal” assignment. The two phases will be discussed in detail in the following subsections.

#### 4.1 Initial Ordering

The goal of the first phase is to construct an ordering of passengers. This time there is not even an obvious notion of an optimal order. Instead, preparing the ground for the peephole optimization, we want the passengers traveling together in an optimal solution to be placed not too far from each other, index-wise.

The basic observation that we want to exploit is that, modulo the relatively few passengers not being delivered to the exchange, crossing vehicle paths are usually sub-optimal. It then seems reasonable for individual vehicles to “cover” geographically disjoint areas, and such areas, all touching the exchange point, can be naturally ordered angularly around it.

This suggests that a good initial order might be formed by ranking passengers by the angle (direction) from their pickup location to the exchange. Unfortunately, this turns out not to be the case. We have implemented such heuristic and found it not to be satisfactory. It seems, that the reason for this is that the



angular differences in direction of travel do not translate well to operating costs. For the same angular difference, the cost of traveling between two locations heavily depends on their distances to the exchange point, which is not reflected in such an ordering. Even for locations with these distances similar, the costs may vary significantly, because the road network is usually not very uniform.

To better capture the actual distances (and thus, better approximate the actual operating costs), we propose a different heuristic. First, we introduce a measure<sup>6</sup> of **dissimilarity** between any pair of passengers  $p$  and  $q$ , intuitively capturing the additional cost of serving them with the same vehicle. Denoted by  $\delta_{p,q}$ , it is formally defined as the difference between:

- the travel distance of an optimal route serving both  $p$  and  $q$ , and
- the larger of the travel distances of optimal routes serving either only  $p$ , or only  $q$ .

To form the ordering, we first find the pair of most dissimilar passengers  $q$  and  $r$ , who will be ranked first and last, respectively. The remaining passengers are ranked based on their relative dissimilarity to  $q$  and  $r$ : passenger  $p$  receives a rank equal to  $\frac{\delta_{p,q}}{\delta_{p,q} + \delta_{p,r}}$ . Note that this strategy only makes sense because of our expectation about the pickup locations occupying a common half-plane. Otherwise, one could easily imagine the order produced to alternate between the two half-planes split by the line connecting the locations of  $q$  and  $r$ . It is hard to imagine that assignments based on such ordering can be close to the optimal solution.

## 4.2 Peephole Optimization

Given a suitable ordering of passengers, an assignment could now be obtained by splitting the ordering in  $m - 1$  places and assigning these groups to individual vehicles; this is easily done by dynamic programming, requiring time  $\Theta(n \cdot m \cdot c)$  and  $\Theta(n \cdot c)$  single-vehicle queries.

However, it turns out that a significant improvement can again be obtained by optimizing consecutive fixed-size windows. With the initial ordering  $p_0, \dots, p_{n-1}$  of passengers, the set of indices  $I := \{0, \dots, n - 1\}$ , and a fixed window size  $w$ , each sub-problem  $\langle k, W, l \rangle$  in appropriate dynamic programming is parameterized by:

- a **prefix length**  $0 \leq k \leq n$ ,

<sup>6</sup>Note, that this measure, while non-negative and symmetric, is not in general a metric.

- a **window**  $W \subseteq \{k - 1, \dots, k - w\} \cap I$ , and
- a **vehicle count**  $0 \leq l \leq m$ .

In the sub-problem  $\langle k, W, l \rangle$ , we are interested in the (locally) optimal assignment of the passengers from the set  $P_{k,W} := \{p_0, \dots, p_{k-1}\} \setminus \{p_i : i \in W\}$  to a fleet of at most  $l$  vehicles. The whole problem is captured by sub-problem  $\langle n, \emptyset, m \rangle$ .

To simplify the exposition below, let us first note that we only need to consider the sub-problems for which  $k - 1 \notin W$ . We call these sub-problems **canonical**. This is because each non-canonical sub-problem  $\langle k, W, l \rangle$  (i.e., with  $k - 1 \in W$ ) can be replaced by an equivalent sub-problem  $\langle k - 1, W \setminus \{k - 1\}, l \rangle$ .

The solutions to sub-problems are now found as follows:

- **I = 0**. The only viable canonical sub-problem in this case is  $\langle 0, \emptyset, 0 \rangle$ , with zero cost.
- **I > 0**. For a sub-problem  $\langle k, W, l \rangle$ , let us consider the set of passengers  $U$  which can be assigned to a single vehicle. Following the peephole principle, we only allow placing together passengers from a common window. Thus, we consider each  $U \subseteq \{k - 1, \dots, k - w\} \setminus W$  (in fact, with indistinguishable vehicles, one may limit this by requiring that either  $U = \emptyset$ , or  $k - 1 \in U$ , by focusing on the vehicle to which passenger  $p_{k-1}$  gets assigned). The remaining passengers must have been assigned to the remaining vehicles, leading to the (possibly non-canonical) sub-problem  $\langle k, W \cup U, l - 1 \rangle$ .

There are  $\Theta(n \cdot m \cdot 2^w)$  sub-problems to be solved, and the total time complexity of the algorithm is  $\Theta(n \cdot m \cdot 3^w)$  (as there are  $3^w$  pairs of disjoint subsets of  $\{k - 1, \dots, k - w\}$ ), not counting the time needed to answer single-vehicle queries. The latter becomes an important consideration, as there are  $\Theta(n \cdot 2^w)$  of these queries. If we employ the algorithm from Section 3 with window size  $w'$ , the time required to answer all the queries will be  $\Theta(n \cdot 2^w \cdot c \cdot w'^2 \cdot 2^{w'})$ . For  $w' + 2 \log_2 w' > (\log_2 3 - 1)w + \log_2 m - \log_2 c + \Theta(1)$  it becomes the dominating component of the time complexity.

The window size  $w$  for this algorithm needs to be at least  $c$  (the capacity of a single vehicle), as otherwise not all seats will be occupied. However, for better results, we suggest the window size of about  $2c$ . This accounts for the passengers with delivery locations before the exchange, and allows “consecutive” vehicles to exchange some passengers. As we expect many of the passengers to visit the exchange location, we don’t expect the number of locations in calls to the algorithm from Section 3 to exceed  $2c$  by much. Thus, it is reasonable to expect that setting

$w' = c$  should give a good enough approximation of optimal solutions. With these values of  $w$  and  $w'$  we get that the algorithm runs in  $\Theta(n \cdot m \cdot 9^c + n \cdot c^3 \cdot 8^c)$ .

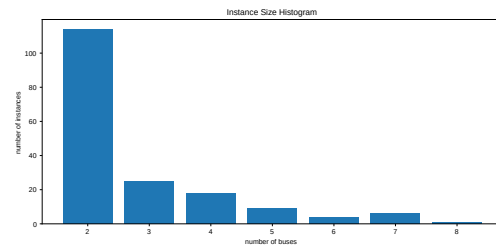
### 4.3 Evaluation

The experimental data for this problem comes from the operation of an undisclosed cross-border long-distance ride-sharing service which was provided to us by Teroplan S.A. Passenger requests for a single cross-border travel route have been gathered for 14 months. Operator of the route used a fleet of a dozen of vehicles, each able to accommodate 8 passengers. Operator used a single exchange location, which was situated close to the Polish-German border, see Figure 1b. All the vehicles were required to make a time-synchronized stop at the exchange location. Almost all of the passengers visited the exchange location and crossed the border. After adjusting data for our problem, and filtering out instances where only one vehicle was enough to serve the passenger requests, we have obtained 177 test cases. Majority of the cases are served with only two vehicles, but there is also a case where 8 vehicles are necessary. On average, there are 19.4 pickup/delivery locations to arrange per test case. Figures 3a and 3b show the distributions of instance sizes measured in the number of vehicles used to serve the requests and the number of travel points, respectively.

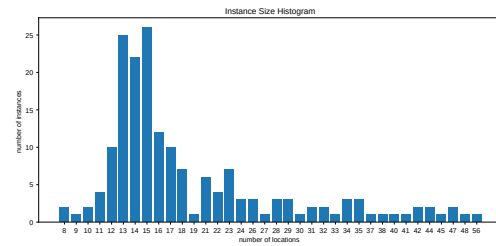
The algorithm with window size  $w = 16$  and using window size  $w' = 8$  for estimation of the single vehicle ride cost of a single vehicle ride takes 5620 seconds to compute and gives routes with the total distance of 339993 kilometers. For comparison, algorithm with window size 8 takes 15.2 seconds to compute and gives routes of total distance 346117 kilometers. Changing the value of  $w'$  (between 4 and 16) does not influence running time, or quality of solutions by much. Increasing  $w$  higher also does not bring any important increase in the quality of solutions, but the processing time quickly becomes an issue. See Figure 3c for a comparison of running time and solution quality for different values of  $w$ .

## 5 SUMMARY

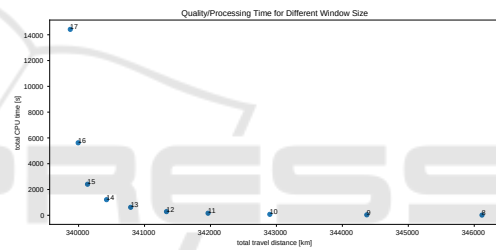
We have presented two heuristic algorithms for vehicle routing problems, with a parameter, window size, controlling their time-vs-quality trade-off. Experimental evaluation has shown that with a proper choice of window sizes the algorithms can be both effective and efficient on data sets representing actual travel requests. In fact, the approach has been developed into



(a) Vehicle counts in *multi* data



(b) Location counts in *multi* data



(c) Quality/Time in *multi* data

Figure 3: Experimental results for multiple vehicle problem.

a practical software solution which is used to route vehicles on a daily basis by Hoper, a commercial long-distance ride-sharing operator.

In both cases, the proposed heuristic exploits the fact that the requests in each problem instance share a common general direction of travel. Still, we believe that the more general approach—looking for an efficiently computable, linear ordering of objects that is close (as a permutation) to an optimal one, and subsequently processing it with peephole optimization—might also be applicable to other optimization problems.

## REFERENCES

Bellman, R. (1962). Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM*, 9(1):61–63.

Bianco, L., Mingozzi, A., and Ricciardelli, S. (1994). A set partitioning approach to the multiple depot vehicle

- scheduling problem. *Optimization Methods and Software*, 3(1-3):163–194.
- Cordeau, J.-F. (2006). A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54(3):573–586.
- Desrosiers, J., Dumas, Y., and Soumis, F. (1984). A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows. *American Journal of Mathematical and Management Sciences*, 6(3-4):301–325.
- Healy, P. and Moll, R. (1995). A new extension of local search applied to the dial-a-ride problem. *European Journal of Operational Research*, 83(1):83–104.
- Held, M. and Karp, R. M. (1962). A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210.
- Ho, S. C., Szeto, W., Kuo, Y.-H., Leung, J. M., Petering, M., and Tou, T. W. (2018). A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research Part B: Methodological*, 111:395–421.
- Kikuchi, S. (1984). Scheduling of demand-responsive transit vehicles. *Journal of Transportation Engineering*, 110(6):511–520.
- Lin, S. (1965). Computer solutions of the traveling salesman problem. *The Bell System Technical Journal*, 44(10):2245–2269.
- Molenbruch, Y., Braekers, K., and Caris, A. (2017). Typology and literature review for dial-a-ride problems. *Annals of Operations Research*, 259(1):295–325.
- Parragh, S. N., Doerner, K. F., and Hartl, R. F. (2008). A survey on pickup and delivery problems: Part II: Transportation between pickup and delivery locations. *Journal für Betriebswirtschaft*, 58:81–117.
- Psarafitis, H. N. (1980). A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14(2):130–154.
- Psarafitis, H. N. (1983). Analysis of an  $O(N^2)$  heuristic for the single vehicle many-to-many Euclidean dial-a-ride problem. *Transportation Research Part B: Methodological*, 17(2):133–145.
- Renaud, J., Boctor, F. F., and Laporte, G. (2002). Perturbation heuristics for the pickup and delivery traveling salesman problem. *Computers & Operations Research*, 29(9):1129–1141.
- Renaud, J., Boctor, F. F., and Ouenniche, J. (2000). A heuristic for the pickup and delivery traveling salesman problem. *Computers & Operations Research*, 27(9):905–916.
- Ritzinger, U., Puchinger, J., Rudloff, C., and Hartl, R. F. (2022). Comparison of anticipatory algorithms for a dial-a-ride problem. *European Journal of Operational Research*, 301(2):591–608.
- Ropke, S., Cordeau, J.-F., and Laporte, G. (2007). Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks*, 49(4):258–272.
- Savelsbergh, M. W. P. (1990). An efficient implementation of local search algorithms for constrained routing problems. *European Journal of Operational Research*, 47(1):75–85.
- Tafreshian, A., Abdolmaleki, M., Masoud, N., and Wang, H. (2021). Proactive shuttle dispatching in large-scale dynamic dial-a-ride systems. *Transportation Research Part B: Methodological*, 150:227–259.
- Toth, P. and Vigo, D., editors (2001). *The Vehicle Routing Problem*. Discrete Mathematics and Applications. SIAM.
- Traub, V. and Vygen, J. (2020). An improved approximation algorithm for ATSP. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, pages 1–13.