Rolling Horizon Based Temporal Decomposition for the Offline Pickup and Delivery Problem with Time Windows

Youngseo Kim[1], **Danushka Edirimanna**[1], **Michael Wilbur**[2], **Philip Pugliese**[3], **Aron Laszka**[4], **Abhishek Dubey**[2], **Samitha Samaranayake**[1]

[1]Cornell University
yk796@cornell.edu,
[2]Vanderbilt University,
[3]Chattanooga Area Regional Transportation Authority,
[4]Pennsylvania State University

Abstract

The offline pickup and delivery problem with time windows (PDPTW) is a classical combinatorial optimization problem in the transportation community, which has proven to be very challenging computationally. Due to the complexity of the problem, practical problem instances can be solved only via heuristics, which trade-off solution quality for computational tractability. Among the various heuristics, a common strategy is problem decomposition, that is, the reduction of a large-scale problem into a collection of smaller sub-problems, with spatial and temporal decompositions being two natural approaches. While spatial decomposition has been successful in certain settings, effective temporal decomposition has been challenging due to the difficulty of *stitching* together the sub-problem solutions across the decomposition boundaries. In this work, we introduce a novel temporal decomposition scheme for solving a class of PDPTWs that have narrow time windows, for which it is able to provide both fast and high-quality solutions. We utilize techniques that have been popularized recently in the context of online dial-a-ride problems along with the general idea of rolling horizon optimization. To the best of our knowledge, this is the first attempt to solve offline PDPTWs using such an approach. To show the performance and scalability of our framework, we use the optimization of paratransit services as a motivating example. Due to the lack of benchmark solvers similar to ours (i.e., temporal decomposition with an online solver), we compare our results with an offline heuristic algorithm using Google OR-Tools. In smaller problem instances (with an average of 129 requests per instance), the baseline approach is as competitive as our framework. However, in larger problem instances (approximately 2,500 requests per instance), our framework is more scalable and can provide good solutions to problem instances of varying degrees of difficulty, while the baseline algorithm often fails to find a feasible solution within comparable compute times.

1 Introduction

The pickup and delivery problem with time windows (PDPTW) is a challenging optimization problem (Desaulniers, Madsen, and Ropke 2014). The PDPTW is a generalization of the vehicle routing problem with time windows (VRPTW) in which requests include both pickup and delivery time windows (Dumas, Desrosiers, and Soumis 1991). Furthermore, the PDPTW problem can be categorized as offline or online. In the offline setting, trip requests are gathered ahead of time and vehicle routes are optimized in advance for the day. In the online setting, requests are scheduled in real time as they arrive. In this work, we focus on the offline problem. Due to computational challenges, current state-of-the-art is focused on heuristic approaches, which often compromise solution quality for scalability (Desaulniers, Madsen, and Ropke 2014). Previous research, motivated by real-world applications, has addressed various practical considerations, such as utilizing multiple depots and vehicles, enforcing strict time windows, allowing selective visits to customers, or optimizing multiple objectives at once (Ho et al. 2018; Ropke, Cordeau, and Laporte 2007). These approaches aim to address specific cases of the offline PDPTW and therefore often lack flexibility.

For solving the VRPTW, decomposition is a common strategy—dividing the original large-scale problem into a number of smaller sub-problems with respect to time or space. In spatial decomposition, requests are clustered by location. Then, decomposed problems are solved independently and solutions are merged together. There is significant literature on spatial decomposition based on the cluster-first and route-second principle (Ouyang 2007; Desaulniers et al. 2002). In temporal decomposition, the time axis is split into multiple time intervals, and the original problem is strictly divided into small problems corresponding to these intervals. While some papers incorporate temporal information on top of a spatial decomposition (Bent and Van Hentenryck 2010; Qi et al. 2012; Tu et al. 2015), we are not aware of successful approaches with pure temporal decomposition technique. Hence, we introduce a novel temporal decomposition. Naïve spatial or temporal decomposition can significantly degrade the solution as routes are isolated to individual spatial partitions or time bins. Also, it is non-trivial to stitch routes together that are independently obtained for each time interval in a post processing step (Zheng and Zhang 2019). Our temporal decomposition overcomes this issue by creating overlapping time bins and solving the problem via a rolling horizon approach. We note that a similar approach with spatial decomposition is much harder in pickup and delivery problems as each request is defined spatially in a four dimensional space (the product of two dimensional coordinates for the origin and destination nodes), while decomposing time only involves a single dimension.

	Depots	Trips	Vehicles	Fleet	Vehicle capacity	Time windows	Ride time	Route duration	Selective visits	Obj.[a]	Static /Dyn.[b]
(Masmoudi et al. 2018)	S	S	M	HE	✓	✓		✓		S	Static
(Sayarshad and Gao 2018)	S	M	M	HO	✓				✓	M	Dyn.
(Tellez et al. 2018)	M	S	M	HE	✓	✓	✓			S	Static
(Luo, Liu, and Lim 2019)	S	M	M	HE	✓	✓	✓		✓	M	Static
(Bongiovanni, Kaspi, and Geroliminis 2019)	M	M	M	HE	✓	✓	✓	✓		S	Static
(Liang et al. 2020)	M	S	M	HO	✓	✓	✓	✓	✓	S	Dyn.
(Malheiros et al. 2021)	M	S	M	HE	✓	✓	✓	✓		S	Static
(Rist and Forbes 2021)	S	S	M	HO	✓	✓	✓	✓		S	Static
Ours	M	M	M	HE	✓	✓	✓	✓	✓	M	Dyn.

Table 1: Recent literature for PDPTW variants in large scale network (published after 2018)
[a] Abbreviation for objective, [b] Abbreviation for dynamic, S is an abbreviation for single, M is an abbreviation for multiple, HE is an abbreviation for heterogeneous, HO is an abbreviation for homogeneous.

We consider a different approach for temporal decomposition based on rolling horizon optimization, where instead of dividing the problem into non-overlapping time intervals we iteratively solve the problem over a sequence of overlapping intervals. More precisely, we pick a time window size T_w and a step size t_s, and create a sequence of sub-problems corresponding to time windows $\{(0, T_w), (t_s, t_s + T_w), (2t_s, 2t_s + T_w), \dots\}$. This approach eliminates the problem of boundary stitching and achieves smooth temporal transitions since part of the solution from one time interval can be updated in the next iteration. The time window (T_w) and step size (t_s) are hyperparameters that control the desired trade-off between computational efficiency and solution quality. This approach for temporal decomposition is also flexible enough to incorporate practical problem considerations, since adding complexity to a smaller problem instance (i.e., the sub-problem) is more computationally tractable than doing so with the full problem instance.

The drawback of this approach is the need to solve a large number of sub-problems (T/t_s instead of T/T_w) of size T_w. Thus, obtaining fast computation times requires a fast PDPTW solver. This leads us to consider recent approaches utilized in the online PDPTW literature and in particular an approach that works extremely well when the time windows are narrow (Alonso-Mora et al. 2017), which is a common characteristic in many passenger centric applications. To the best of our knowledge, this is the first attempt to deploy an online algorithm in conjunction with rolling horizon optimization for solving an offline PDPTW. We empirically show the performance and scalability of the rolling horizon framework through experiments.

The remainder of this article is structured as follows. In Section 2, we review variants of PDPTW and their practical considerations, and heuristics that have been developed for PDPTW. In Section 3, we explain the mathematical definition of the rolling horizon framework and the online solver that it is built upon. In Sections 4-5, we show the performance and scalability of our framework through experiments on paratransit scheduling problems utilizing two sources of data. Finally, in Section 6, we provide concluding remarks and discuss possible future directions.

2 Literature Review

2.1 PDPTW Variants for Real-World Application

Ropke, Cordeau, and Laporte (2007) provide a comprehensive survey of PDPTW solvers developed up to 2007. Ho et al. (2018) conducted a comprehensive literature review for PDPTW variants published from 2007 to 2018 (see Table 5-8 in their paper (Ho et al. 2018)) and we provide an extended table for papers published after 2018 in Table 1. Columns are all the typical features that have been considered in PDPTW variants, and a more detailed explanation of the features is explained in online appendix (Kim et al. 2022).

Ho et al. (2018) pointed out the research gaps and encouraged the development of techniques that can be adapted to solve the many variants of the PDPTW. Our approach can or can be easily modified to consider all the typical features of the PDPTW variants shown in Table 1. For example, even though it is outside the scope of this work, our approach can be easily adapted to handle dynamic updates of an existing solution in real time. While reviewing papers published after 2018, we could not find any approaches that can efficiently accommodate as many practical considerations as our approach.

The flexibility of our framework stems from two types of decompositions. First, the online VRP algorithm (Alonso-Mora et al. 2017) that our approach is built upon decomposes the high-capacity vehicle-passenger matching problem into a routing problem and an assignment problem, and develops heuristics for effectively computing feasible routes when time windows are relatively tight. Second, the temporal decomposition allows for dividing the original problem into a sequence of more tractable sub-problems. The computational gains achieved via these decompositions allow for solving more complex problem variants.

2.2 Solution Methods for PDPTW

Previous literature has proposed different solution approaches to solve PDPTW and its variants. The vast majority of exact algorithms are developed using techniques such as branch-and-cut (Cordeau 2006; Ropke, Cordeau, and Laporte 2007), branch-and-price (Garaix et al. 2010), and branch-and-price-and-cut (Qu and Bard 2015). Exact methods are unable to solve large instances within a reasonable time due to the intrinsic hardness of PDPTWs. (Ropke and Cordeau 2009) obtained the exact solution up to 500 customers in some benchmark instances, but many of the benchmark instances remain unsolved in optimality (Ho et al. 2018). Furthermore, the largest solvable instance size gets smaller for more complex variants of the problem.

Spatial and temporal decomposition is one common strategy to tackle large scale problems. Bent and Van Hentenryck (2010) suggested an iterative and adaptive decomposition scheme that defines sub-problems based on the spatiotemporal features of the existing solution, solves them independently, and inserts them into the existing solution. Qi et al. (2012) decided sub-problems using a clustering method based on spatiotemporal distance, and merged them after solving independently. Tu et al. (2015) defined a spatial-temporal distance and used it to speed up a local search heuristic. However, all of these techniques explicitly depend on the problem being a VRPTW, and cannot be applied to situations involving pickup and delivery.

While there are no attempts to introduce temporal decomposition in PDPTW, various heuristics and metaheuristics have been developed for solving real-world scale instances. These methods include insertion based heuristics (Luo and Schonfeld 2007; Häme 2011), tabu search (Kirchler and Calvo 2013; Detti, Papalini, and de Lara 2017), simulated annealing (Braekers, Caris, and Janssens 2014), variable neighborhood search (Parragh, Doerner, and Hartl 2010), large neighborhood search (Ropke and Pisinger 2006), and genetic algorithms (Jorgensen, Larsen, and Bergvinsdottir 2007). In general, it is hard to compare performance among these algorithms due to the lack of uniformity in the specific problem variants they consider and the use of different benchmark instances of different sizes.

3 Methodology

3.1 Problem Input and Output

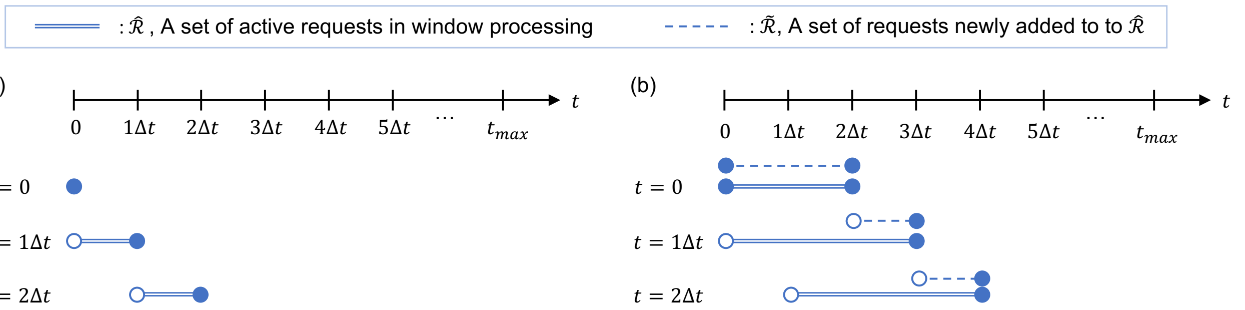
We assume a set of vehicles $\mathcal{V} = \{v_1, \ldots, v_m\}$ with a fixed vehicle-specific capacity and a set of requests $\mathcal{R} = \{r_1, \ldots, r_n\}$, where each request r_k contains a pickup and drop-off locations, and desired pickup time $\hat{t}_k^{pickup}$. We calculate $\hat{t}_k^{dropoff}$, the earliest possible drop-off time, by adding the estimated travel time between pickup and drop-off locations to the desired pickup time. Each problem instance is also defined by the following set of parameters: maximum allowed delay time D_{max} and waiting time W_{max} of customers. The time horizon of the problem t_{max} is divided into smaller intervals with length T_w, which we refer to as the sliding window size. The rolling horizon optimization also defines a step size t_s, which in our approach is equal to the *batch size* of the online optimization algorithm (see below). Correspondingly, we pick a sliding window size that is a multiple of the batch size, which we can interpret as a look-ahead window from the point of view of the online optimization problem. Selecting an appropriate step size and sliding window size is part of our investigation. The output of the model is a set of vehicle routes $\mathbb{R} = \{R_1, R_2, \ldots, R_m\}$, where each route is an ordered set of pickup and drop-off locations and estimated time to visit. We obtain vehicle routes that maximize service rate while minimizing vehicle miles traveled as a secondary objective.

3.2 Computational Approach

In the rolling horizon framework, sub-problems are solved in sequential order along the time axis, with the solution to each sub-problem determining the best passenger-vehicle matching for the corresponding time interval. The solution of the first t_s minutes is fixed, but the remainder $(T_w - t_s)$ can be reoptimized in the next iteration. We use the term batch to refer to the set of requests belonging to a given sub-problem, i.e, the requests whose desired pickup time is within the sliding window corresponding to that sub-problem. The process for selecting the requests corresponding to each batch is called window processing, which is illustrated in Figure 1. The online approach with $T_w = t_s$ is a myopic strategy because each batch only includes requests that have already entered the system. In contrast, our rolling horizon framework can consider batches that extended into the future (known exactly in our case because this is an offline problem and all the demands are known). After solving the optimization problem for a given sub-problem, the routing schedule is finalized for all the requests that will not appear in the next sub-problem. In the next sub-problem, we solve another optimization problem with a new time interval which is shifted forward one step (i.e., by t_s).

3.3 Problem Formulation and Algorithm

The entire algorithm follows an iterative process. In each sub-problem, an optimization problem considers an active request set $\hat{\mathcal{R}}$ $(\subset\mathcal{R})$. New requests obtained from the window processing are added to the active requests set. Requests that have been picked up by any vehicle are removed from the active request set at the end of each iteration. To track requests that have already been picked up, we maintain the set $\mathcal{P}_{v,t}$ which is the set of passengers that have boarded onto vehicle $v \in \mathcal{V}$ prior to time t. Then, we solve the matching problem (RTV-ILP) between the active request set $\hat{\mathcal{R}}$ and the vehicle set $\mathcal{V}$. As a result, we obtain the order and the scheduled time to pickup and drop-off customers for each vehicle. Then, the discrete event vehicle simulator receives the vehicle location from the prior sub-problem, executes scheduled routes from the optimizer until the current simulation time, updates travel times, and then finalizes the vehicle locations for the next sub-problem. From the vehicle simulation, we get a set of vehicle routes until the current time. The finalized routes in the current iteration cannot be changed in the later iterations. We describe the process in more detail in the following sections. Pseudo code for the entire algorithm and



: $\hat{\mathcal{R}}$, A set of active requests in window processing
: $\tilde{\mathcal{R}}$, A set of requests newly added to to $\hat{\mathcal{R}}$
(a)
(b)
t
0
1Δt
2Δt
3Δt
4Δt
5Δt
...
t_{max}
$t = 0$
$t = 1\Delta t$
$t = 2\Delta t$

Figure 1: Window processing in (a) online (b) rolling horizon framework (when sliding window size is $3\Delta t$ and step size is Δt)

window processing is provided in the online appendix (Kim et al. 2022).

Window processing. The procedure for selecting a set of requests to be considered based on their desired pickup time, $\tilde{\mathcal{R}}$. As discussed earlier, the sliding window size T_w and step size t_s are part of the investigation. In general, increased sliding window size leads the better solution quality but exponentially increases computational time. Step size t_s can be the overlapped window size $(T_w - t_s)$, which will also increase solution quality. Decreasing the step size linearly increases computational time as we need to solve around T/t_s sub-problems.

RTV-ILP. The integer linear program (ILP) for assigning requests to trips and trips to vehicles. The RTV-ILP framework can solve fairly large problem instances, which accommodates larger sub-problems in the temporal decomposition. The illustration of the RT-V structure can be found in the online appendix (Kim et al. 2022). We refer readers to (Alonso-Mora et al. 2017) for more detail. Requests $\mathcal{R}$ are aggregated into trips $\mathcal{T}$ based on service constraints. The RT-V graph contains all feasible trip-vehicle pairings, $\mathcal{E}_{TV}$. The existence of an edge between a trip $T_i \in \mathcal{T}$ and a vehicle $v_j \in \mathcal{V}$ indicates that it is feasible for vehicle j to serve trip i. The feasibility of a trip is determined by whether all the requests that belong to the trip can be served by a vehicle while satisfying the constraints in Equations 1 and 2. The constraints ensure that i) the waiting time is not greater than the maximum waiting time W_{max}, and ii) the delay time is not greater than the maximum delay time D_{max}. Recall that t_k^{pickup} denotes the actual pickup time and $\hat{t}_k^{pickup}$ denotes the desired pickup time. Also, $t_k^{dropoff}$ denotes the actual dropoff time and $\hat{t}_k^{dropoff}$ denotes the earliest possible dropoff time.

$$t_k^{pickup} - \hat{t}_k^{pickup} \leq W_{max}, \forall k \tag{1}$$

$$t_k^{dropoff} - \hat{t}_k^{dropoff} \leq D_{max}, \forall k \tag{2}$$

Note that in the case of high-capacity vehicles, enumerating all possible combinations of pickups and deliveries is still computationally expensive. Therefore, we use an insertion-based heuristic when dealing with more than 4 passengers (Alonso-Mora et al. 2017; Wilbur et al. 2022). This process can be further improved with heuristics that appropriately trade-off speed and accuracy.

After building the RT-V graph, we need to solve an ILP to obtain the optimal matching.

$$\underset{\epsilon_{ij},\chi_k}{\text{argmin}} \quad \sum_{\{ij:\epsilon_{ij}\in\mathcal{E}_{TV}\}} c_{ij}\epsilon_{ij} + \sum_{k\in\mathcal{R}} c_k\chi_k \tag{3}$$

$$\text{s.t.} \quad \sum_{\{i:T_i\in\mathcal{T}\}} \epsilon_{ij} \leq 1 \qquad ,\forall v_j \in \mathcal{V} \tag{4}$$

$$\sum_{\{i:T_i\in\mathcal{T}\}} \sum_{\{j:V_j\in\mathcal{V}\}} \epsilon_{ij} + \chi_k = 1 \qquad ,\forall r_k \in \mathcal{R} \tag{5}$$

Decision variables ϵ_{ij} and χ_k are binary. If vehicle j is assigned to trip T_i, $\epsilon_{ij} = 1$; otherwise, $\epsilon_{ij} = 0$. If the request r_k cannot be served by any vehicle, $\chi_k = 1$; otherwise, $\chi_k = 0$. In the objective function, there is cost c_{ij} which is the total vehicle miles traveled by vehicle v_j when serving trip T_i. The other cost c_k is a large constant to penalize unserved requests, such that the service rate is maximized. Thus, the objective function prioritizes maximizing the service rate and then minimizing the vehicle miles traveled as a secondary objective. Equation 4 guarantees that a vehicle is assigned to at most one trip. Equation 5 imposes that each request should be either served by a vehicle or ignored.

An RT-V graph is built at each sub-problem and the best passenger-vehicle matching can be updated until the passenger is picked up or their maximum waiting time is exceeded. In other words, passengers can be swapped to another vehicle prior to pickup. However, additional constraints enforce that a previously matched passenger cannot be ignored.

4 Experimental Design

We showcase the performance of our framework using paratransit scheduling as a motivating application. Paratransit is a service for passengers who are unable to use fixed-route transit, which is provided by public transit agencies as mandated by the Americans with Disabilities Act. Due to the large costs associated with operating paratransit services, transit agencies are constantly looking to improve service efficiency and provide a high level of service at a lower cost. Customers book paratransit trips either by phone or via an app, and can make a reservation as early as two weeks in advance or within a few days of travel. Customers are given a confirmation of travel with an estimated pickup and dropoff time shortly after the booking request. The service is required to provide a tight pickup time and drop-off window, typically around 30 minutes. Additionally, paratransit consists of high capacity vehicles which adds to the computational complexity of the problem. Some paratransit services allow customers to request trips in real time, but service is not guaranteed for these trips. Accordingly, the paratransit problem is an offline optimization problem because most service providers require reservations to be made by the end of the previous day, with a few exceptions allowing reservations to be placed during the service day.

We obtain data from two different sources for experiments. First, we use paratransit trip requests provided by a public transit agency. Second, we use New York City taxi (City 2016) data to investigate the scalability of our approach with a larger dataset. The parameters that we use for the experiments can be found in the online appendix (Kim et al. 2022). We provide two baselines. The first baseline is a fully online solver while the second baseline is an offline heuristic solver.

Real-world paratransit data We used 6 months of paratransit trip requests between January 1, 2021 and June 30, 2021. The data is provided by our partner agency, the *Chattanooga Area Regional Transportation Authority (CARTA)*, which is a mid-sized public transit agency in the United States. As this dataset is directly from a transit agency, we use it to show the performance of our approach on real-world data. Each trip request contains pickup and dropoff locations and the requested pickup time. For privacy considerations, the location information is anonymized as follows. The service area is discretized into a grid of one square mile tiles. For each request, the service location is shifted to a random location within the corresponding cell. Thus, all the demands from a given cell are randomly redistributed within that cell. The requests are also temporally aggregated by the agency to 15 minutes. Thus, we set the step size to 15 minutes, which is the step size that we can use. We randomly selected 30 weekdays for the experiments. There were an average of 172 trip requests per instance in this dataset.

New York City taxi data Recall that a primary advantage of our approach is its ability to scale to a large number of requests. To test this, we acquired 31 days of taxi trip requests (January 1, 2016 to January 30, 2016) in New York City (City 2016). To better represent paratransit trips, we deleted short-distance trips which are less than 5 km. We randomly sampled 1% and 20% of the NYC taxi data to generate two new paratransit datasets. We sampled 1% of the data to have a similar size of data to the real-world paratransit dataset, having an average of 129 trips per instance. The 20% dataset had an average of 2,587 requested trips per instance and was used to investigate the scalability of our approach. We refer to the 1% sampled data as Scenario 1 and the 20% sampled data as Scenario 2. A detailed description of the data statistic is provided in Appendix 7.3. As New York City taxi data is larger than the paratransit data, we set the step size to 5 minutes.

4.1 Metrics

For each day, we calculate three metrics to evaluate the performance of our approach compared to the baselines. Service rate is the number of requests served divided by the total number of requests in a day. The compute time per request is the total time the solver takes to run for a day divided by the number of requests on that day. The average delay time is the average time difference between the actual dropoff time and the earliest possible dropoff time that is calculated by adding the shortest travel time to the requested pickup time.

4.2 Baselines

Our approach provides an efficient, tune-able approach that provides a better trade-off between computational efficiency and solution quality. To investigate this trade-off, we implement a fully online, myopic solution and a fully offline heuristic solver.

Online Solver We use our own approach with a sliding window size T_w to be equal to the step size t_s to represent the online solver. By having an overlapped window size $(T_w - t_s)$ of zero, our solver is simplified to a purely online approach as it does not utilize future knowledge. We refer to this baseline as RH0 comparing with our approach RH1, RH2, RH3, which have overlapped windows whose size is a multiple of step size of 1, 2, and 3, respectively.

Offline Solver We implemented an offline PDPTW benchmark solver in Google OR-Tools, a well established, modern, and publicly available VRP solver developed by Google (Perron and Furnon 2022). We use guided local search (GLS) which is known as the best performing setting for the OR-Tools PDPTW solver as the baseline heuristic. The general GLS approach was first proposed by Kilby, Prosser, and Shaw (1999) as an efficient anytime heuristic, that aims to iteratively improve the solution for a fixed set of time. GLS requires an initial solution to improve upon. To find the initial solution we used a parallelized cheapest insertion approach which iteratively builds a solution by inserting the cheapest node at its cheapest position without violating the PDPTW constraints; cost is defined by the objective function (Perron and Furnon 2022). We optimize for service rate by setting a sufficiently large penalty for trips that are not serviced. The secondary objective is to minimize passenger travel time. The baseline includes constraints on pickups and dropoffs, as well as time windows (as in our approach).

In addition, in the online appendix (Kim et al. 2022) we also compare our algorithm with the LKH-3 solver that uses a modified Lin-Kernighan-Helsgaun heuristic. This is a state-of-the-art solver for constrained traveling salesman and vehicle routing problems.

4.3 Reproducibility

All the codes and datasets we used for experiments are available online. Software code for the rolling horizon framework is available in the following repository: https://github.com/MAS-Research/RollingHorizon.git. Code for the baseline offline heuristic using Google OR-Tools is available in the following: https://github.com/MAS-Research/RollingHorizon_baseline_ORTools.

5 Results

5.1 Evaluation on Real-World Paratransit Dataset

Figure 2 provides service rates for the 31 day experiment. Overall, our framework (RH1, RH2, RH3) outperforms the online approach (RH0). In an experiment with 6 fleets, RH3 brings an increase in service rate from 77.7% to 85.6% compared to RH0, which corresponds to a 10.1% increase. We observe that the improvement of the service rate reduces as the sliding window size increases. This is because future information has a more significant impact on current decisions when the events are imminent. However, computational time exponentially increases as we increase the sliding window size (The result can be found in the online appendix (Kim et al. 2022).). In practice, operators can decide the appropriate sliding window size and step size to trade off service rate and compute time.

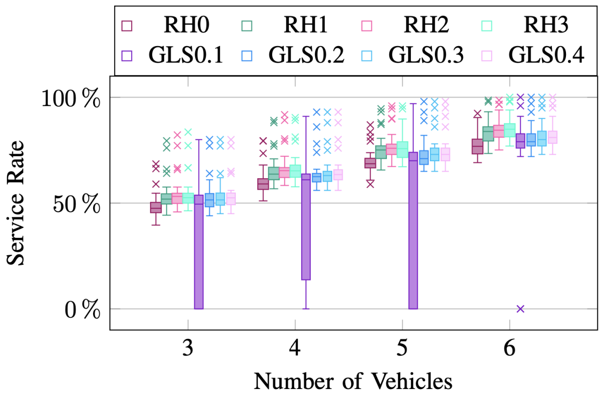


Figure 2: Service rate, Chattanooga paratransit dataset

5.2 Evaluation of NYC Datasets

In Figure 3, the left column shows the results of scenario 1 (with an average of 129 requests per instance) and the right column shows the results of scenario 2 (with an average of 2,587 requests per instance). In scenario 1, the performance of the rolling horizon framework is as good as that of GLS. Recall that GLS is an anytime algorithm from which we obtain the best among the solutions that have been found within a time limit. We obtain the best solution of GLS after the given time limit of 0.1, 0.2, 0.3, and 0.4 seconds per request. GLS fails to find a feasible solution in some instances with 0.1, 0.2 seconds time limit, and starts to provide results to all instances with 0.3 seconds time limit. More specifically, with a 0.1 second time limit, GLS cannot get a feasible solution in 25, 22, 21, 20 instances among 31 instances with 3, 4, 5, 6 vehicles, respectively. With a 0.2 second time limit, GLS cannot get a feasible solution in 4, 1, 1 instances among 31 instances with 3, 4, 5 vehicles, respectively.

In our framework, the mean of service rates increases as the sliding window size increases (RH0, RH2, RH4, RH6, in that order), and it has a saturation point in all experiment settings with varying fleet sizes. The rolling horizon approach achieves saturation points when the time limit becomes 0.3 seconds per request, which is the time limit that GLS starts performing well. The saturated service rates are close to or even exceed the service rates of GLS. This indicates that the rolling horizon framework obtains good enough solution quality which is comparable to that of GLS within similar compute times. Also, the average delay time from the rolling horizon framework is smaller than that of GLS, which would lead to better user experiences.

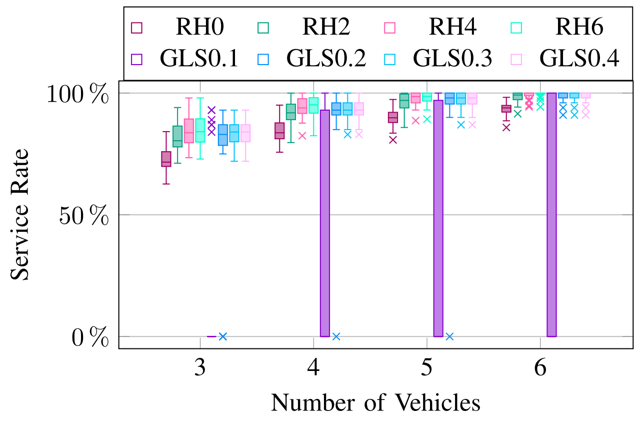
Scenario 2 highlights the scalability of our framework. Recall that the number of requests in scenario 2 is 20 times larger than that of scenario 1. Our framework achieves reasonable service levels and average delay times. Compare to RH0 which shows 75.6% of the average service rate and 98.4% of the maximum service rate, RH2 shows improvement with 79.0% of the average service rate and 100% of the maximum service rate. Even RH2 whose compute times are the largest in our framework can solve any instance within 1 second per request. On the contrary, GLS cannot get any feasible solution in 29 among 31 instances within the time limit of 1 and 3 seconds per request. In a time limit of 5 seconds per request, GLS performs better but still cannot get any feasible solution in 16 instances. The time limit of 5 seconds per request corresponds to around 3.5 hours time limit for the entire instance, which is already quite long considering that operators need to obtain a schedule for the next day within a couple of hours.

6 Conclusions

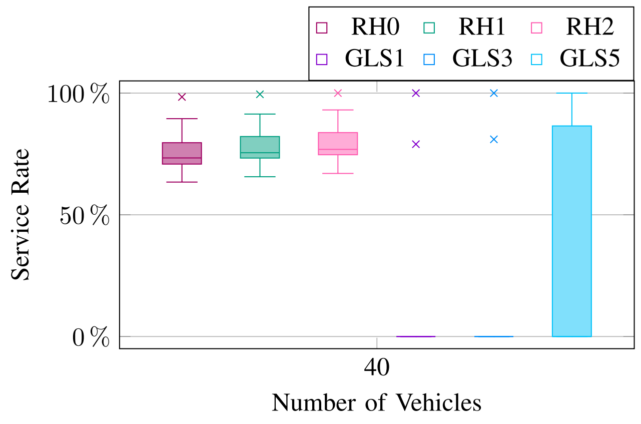
The pickup and delivery problem with time windows (PDPTW) is a challenging operational problem, and several generalizations based on practical considerations make the problem even more complicated. In this paper, we introduce a new temporal decomposition scheme to solve the PDPTW at scale. Our approach uses a state-of-the-art online algorithm within a rolling horizon framework to solve a sequence of smaller sub-problems that collectively cover the entire original problem. This strategy avoids the primary pitfall of more naïve temporal decompositions, namely the challenge of stitching together sub-problems. The computational gains made through this decomposition provide us the flexibility to add additional features (corresponding to practical needs) even though this introduces extra complexity.

We choose the paratransit scheduling problem to showcase the performance of our rolling horizon framework in different scales of networks with different demand profiles. In the real-world size instances, the rolling horizon framework achieves as high service rates as the benchmark offline solver within comparable computational times. Moreover, our framework can be scaled up to around 2,500 requests while the benchmark solver often fails to find any feasible solution within the 5 seconds time limit per request. Due to its computational efficiency, our approach can be used to continuously add new requests to the system after a service plan is made, giving paratransit operators more flexibility in

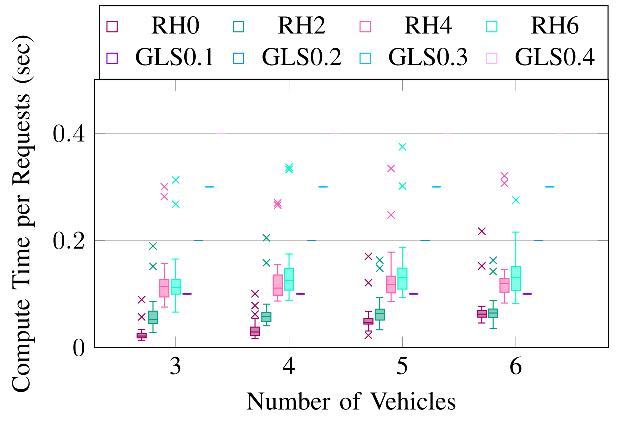
Published in the proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI 2023).



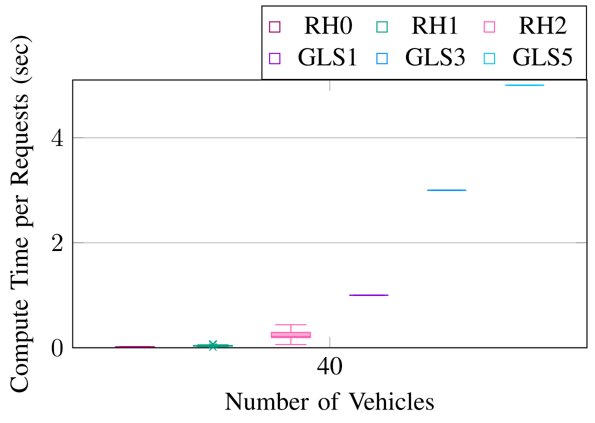
Service rate, NYC dataset - Scenario 1.



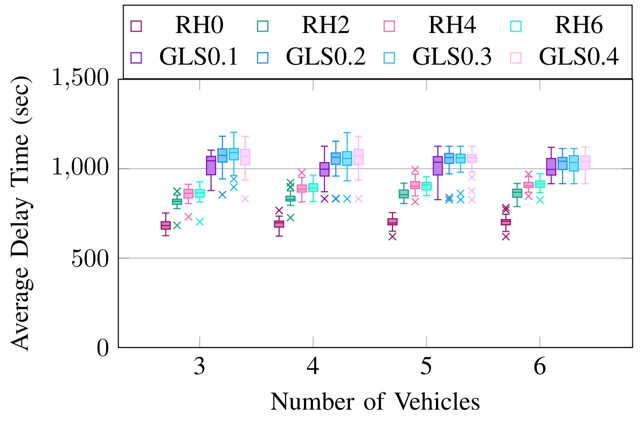
Service rate, NYC dataset - Scenario 2.



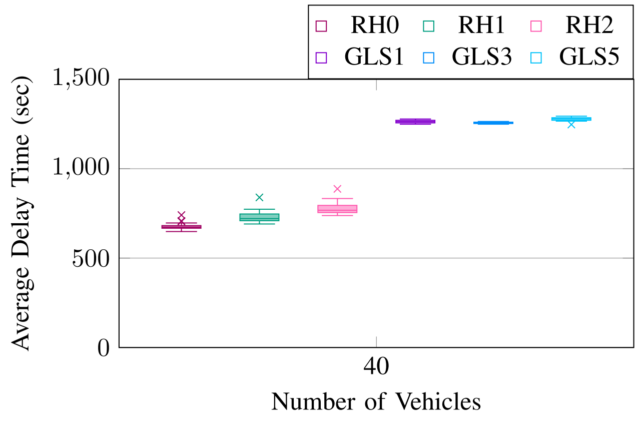
Compute Time, NYC dataset - Scenario 1.



Compute Time, NYC dataset - Scenario 2.



Delay Time, NYC dataset - Scenario 1.



Delay Time, NYC dataset - Scenario 2.

Figure 3: Comparison of the rolling horizon framework with the baseline approach. RHX denotes the rolling horizon solutions with X indicating the rolling horizon factor. GLSY denotes the guided local search solutions with the Y indicating time limit.

adding last minute requests. Our framework can be also used to obtain a reasonably good initial feasible solution quickly, making it a good candidate to be combined with other local search heuristics that can subsequently improve the solution as time permits.

Acknowledgments

This research is partially sponsored by the National Science Foundation under grants CNS-1952011 and CIS-2144127, and the Department of Energy under Award Numbers DE-EE0009212 and DE-EE0008464. We thank Matthew Zalesak for his assistance with implementing the proposed framework.

References

Alonso-Mora, J.; Samaranayake, S.; Wallar, A.; Frazzoli, E.; and Rus, D. 2017. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 114(3): 462–467.

Bent, R.; and Van Hentenryck, P. 2010. Spatial, temporal, and hybrid decompositions for large-scale vehicle routing with time windows. In *International Conference on Principles and Practice of Constraint Programming*, 99–113. Springer.

Bongiovanni, C.; Kaspi, M.; and Geroliminis, N. 2019. The electric autonomous dial-a-ride problem. *Transportation Research Part B: Methodological*, 122: 436–456.

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

City, N. Y. 2016. TLC Trip Record Data. https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page. [Online; accessed 26-Oct-2022].

Cordeau, J.-F. 2006. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54(3): 573–586.

Desaulniers, G.; Desrosiers, J.; Erdmann, A.; Solomon, M. M.; and Soumis, F. 2002. VRP with Pickup and Delivery. *The vehicle routing problem*, 9: 225–242.

Desaulniers, G.; Madsen, O. B.; and Ropke, S. 2014. *Chapter 5: The Vehicle Routing Problem with Time Windows*, chapter 5, 119–159.

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

Dumas, Y.; Desrosiers, J.; and Soumis, F. 1991. The pickup and delivery problem with time windows. *European journal of operational research*, 54(1): 7–22.

Garaix, T.; Artigues, C.; Feillet, D.; and Josselin, D. 2010. Vehicle routing problems with alternative paths: An application to on-demand transportation. *European Journal of Operational Research*, 204(1): 62–75.

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

Ho, S. C.; Szeto, W. Y.; 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.

Jorgensen, R. M.; Larsen, J.; and Bergvinsdottir, K. B. 2007. Solving the dial-a-ride problem using genetic algorithms. *Journal of the operational research society*, 58(10): 1321–1331.

Kilby, P.; Prosser, P.; and Shaw, P. 1999. Guided local search for the vehicle routing problem with time windows. In *Metaheuristics*, 473–486. Springer.

Kim, Y.; Edirimanna, D.; Wilbur, M.; Pugliese, P.; Laszka, A.; Dubey, A.; and Samaranayake, S. 2022. Offline Pickup and Delivery Problem with Time Windows via Rolling Horizon Trip-Vehicle - Online appendix. arXiv:https://doi.org/10.48550/arXiv.2303.03475.

Kirchler, D.; and Calvo, R. W. 2013. A granular tabu search algorithm for the dial-a-ride problem. *Transportation Research Part B: Methodological*, 56: 120–135.

Liang, X.; de Almeida Correia, G. H.; An, K.; and van Arem, B. 2020. Automated taxis' dial-a-ride problem with ride-sharing considering congestion-based dynamic travel times. *Transportation Research Part C: Emerging Technologies*, 112: 260–281.

Luo, Y.; and Schonfeld, P. 2007. A rejected-reinsertion heuristic for the static dial-a-ride problem. *Transportation Research Part B: Methodological*, 41(7): 736–755.

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

Malheiros, I.; Ramalho, R.; Passeti, B.; Bulhões, T.; and Subramanian, A. 2021. A hybrid algorithm for the multi-depot heterogeneous dial-a-ride problem. *Computers & Operations Research*, 129: 105196.

Masmoudi, M. A.; Hosny, M.; Demir, E.; Genikomsakis, K. N.; and Cheikhrouhou, N. 2018. The dial-a-ride problem with electric vehicles and battery swapping stations. *Transportation research part E: logistics and transportation review*, 118: 392–420.

Ouyang, Y. 2007. Design of vehicle routing zones for large-scale distribution systems. *Transportation Research Part B: Methodological*, 41(10): 1079–1093.

Parragh, S. N.; Doerner, K. F.; and Hartl, R. F. 2010. Variable neighborhood search for the dial-a-ride problem. *Computers & Operations Research*, 37(6): 1129–1138.

Perron, L.; and Furnon, V. 2022. OR-Tools v9.4. https://developers.google.com/optimization/routing/vrp. [Online; accessed 26-Oct-2022].

Qi, M.; Lin, W.-H.; Li, N.; and Miao, L. 2012. A spatiotemporal partitioning approach for large-scale vehicle routing problems with time windows. *Transportation Research Part E: Logistics and Transportation Review*, 48(1): 248–257.

Qu, Y.; and Bard, J. F. 2015. A branch-and-price-and-cut algorithm for heterogeneous pickup and delivery problems with configurable vehicle capacity. *Transportation Science*, 49(2): 254–270.

Rist, Y.; and Forbes, M. A. 2021. A new formulation for the dial-a-ride problem. *Transportation Science*, 55(5): 1113–1135.

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

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

Ropke, S.; and Pisinger, D. 2006. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4): 455–472.

Sayarshad, H. R.; and Gao, H. O. 2018. A scalable non-myopic dynamic dial-a-ride and pricing problem for competitive on-demand mobility systems. *Transportation Research Part C: Emerging Technologies*, 91: 192–208.

Tellez, O.; Vercraene, S.; Lehuédé, F.; Péton, O.; and Monteiro, T. 2018. The fleet size and mix dial-a-ride problem with reconfigurable vehicle capacity. *Transportation Research Part C: Emerging Technologies*, 91: 99–123.

Tu, W.; Li, Q.; Fang, Z.; and Zhou, B. 2015. A novel spatial-temporal Voronoi diagram-based heuristic approach for large-scale vehicle routing optimization with time constraints. *ISPRS International Journal of Geo-Information*, 4(4): 2019–2044.

Wilbur, M.; Kadir, S. U.; Kim, Y.; Pettet, G.; Mukhopadhyay, A.; Pugliese, P.; Samaranayake, S.; Laszka, A.; and Dubey, A. 2022. An online approach to solve the dynamic vehicle routing problem with stochastic trip requests for paratransit services. *arXiv preprint arXiv:2203.15127*.

Zheng, J.; and Zhang, Y. 2019. A fuzzy receding horizon control strategy for dynamic vehicle routing problem. *IEEE Access*, 7: 151239–151251.

7 Appendix

7.1 A List of Practical Considerations

1. Multi vehicle utilizing multiple fleets.
2. Multi depots multiple starting and ending locations for the vehicle fleet.
3. Multi trips allowing a vehicle to return to a depot multiple times in a single day.
4. Heterogeneous fleet various combinations of equipment for different types of passengers.
5. Vehicle capacity limiting the maximum number of passengers.
6. Time window user-specified earliest and latest time bound for pick-up and drop-off.[1]
7. Ride time limiting the maximum time difference between the scheduled pickup and dropoff time.
8. Route duration limiting the maximum time difference between the times of leaving from and returning to a depot.
9. Selective service allowing selective pickups and deciding which requests to accommodate.
10. Multiple objectives objective functions consisting of multiple measures.
11. Dynamic allowing dynamic modification of existing plans in response to new information.

7.2 RT-V Graph

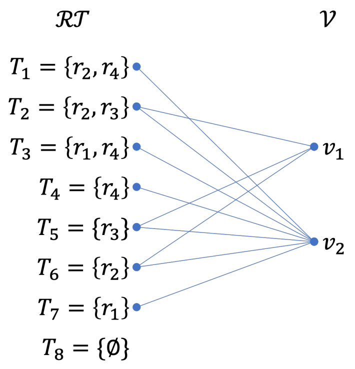


Figure 4: RT-V graph for an example instance (reconstructed from the schematic figure in Alonso-Mora et al.'s work)

Figure 4 illustrates the structure of RT-V graph. Requests $r_k \in \mathcal{R}$ are aggregated into trips $\mathcal{T}$ if it is feasible to serve those requests by one vehicle, in other words, if those requests are shareable. Edges $\mathcal{E}_{TV}$ between a trip $T_i \in \mathcal{T}$ and a vehicle $v_j \in \mathcal{V}$ represent all possible candidate pairs between trip and vehicle meaning that it is feasible for the vehicle to serve all requests in the trip T_i.

[1]Although our framework does not explicitly have the concept of the time window, inputs of our model can translate to time windows. Desired pickup time/earliest drop-off time corresponds to the start of the time window. Then, the end of the time window can be set by adding the maximum waiting time/detour time. The definition of the inputs can be found in Section 3.1.

7.3 Data Statistics

An overview of the two datasets is provided in Table 2. The real-world paratransit dataset contains 30 instances. Each instance represents different operation days. There are an average of 172 requests per instance with a standard deviation of 33. NYC taxi data contains 31 instances. Scenario 1 contains 129 requests per instance with a standard deviation of 29. Scenario 2 contains 2587 requests per instance with a standard deviation of 570.

Table 2: Overview of Datasets

	Real-world Dataset	**NYC Scenario 1**	**NYC Scenario 2**
Number of Days	30	31	31
Mean Request per instance	172	129	2587
Std. Dev. Requests per instance	33	29	570

7.4 System Parameters

We identified parameter settings based on discussions with the public transit agency, which are shown in Table 3. The number of vehicles varies from 3 to 7 and each vehicle can accommodate up to 8 passengers at a time. Maximum waiting time is defined as the time difference between the actual pickup time and the desired pickup time which is set to 30 minutes. The maximum delay time is defined as the time difference between the actual dropoff time and the earliest possible dropoff time and is also set to 30 minutes. In practice, our partner agency set the same goal for the service levels. On average, our partner agency plans for 5-10 minutes to load/unload a passenger, thus we conservatively set the dwell time to 10 minutes. We set the step size to 15 minutes, which is the minimum interval that we can use because time information in the original data was aggregated by 15 minutes.

Table 3: Parameter settings for real-world paratransit dataset

Parameter	**Values**
Fleet size	3, 4, 5, 6, 7
Vehicle capacity	8
Maximum waiting time	30 (min)
Maximum delay time	30 (min)
Dwell time	10 (min)
Step size	15 (min)

The parameter settings for the NYC taxi investigation are provided in Table 4. For scenario 1 we vary the fleet size from 3 to 6 vehicles while for scenario 2 we investigate fleet sizes of 30, 40, 50, and 60. The vehicle capacity is again set to 8. Due to the higher frequency of trip requests, we set the step size to 5 minutes instead of the 15 minutes interval so as to deal with the NYC data which is larger than the previous dataset. Likewise, we can make step sizes even smaller to deal with larger demand. One caveat is that the decreasing length of the interval may lead decisions in each batch to become more myopic and deteriorate the solution quality as we utilize restricted information.

Table 4: Parameter settings for New York City dataset

Parameter	**Scenario 1**	**Scenario 2**
Data	1% sampled	20% sampled
Fleet size (M)	3, 4, 5, 6	30, 40, 50, 60
Step size	5 (min)	5 (min)
RH factor	0,1,2,3	0,1,2,3

7.5 Results of Chattanooga Paratransit Dataset

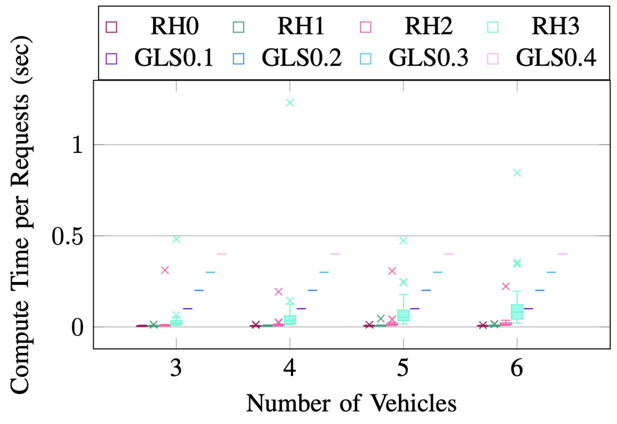


Figure 5: Compute time, Chattanooga paratransit dataset

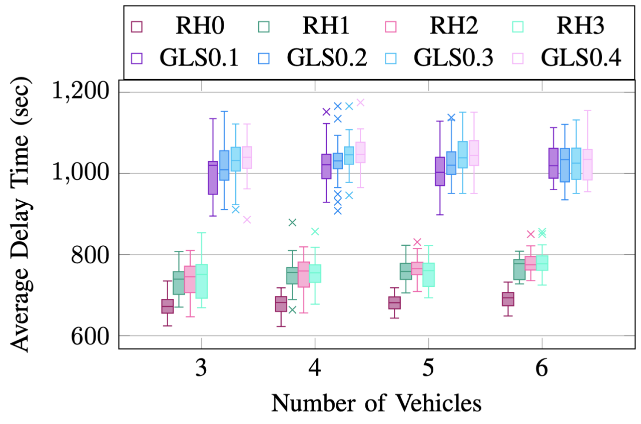


Figure 6: Delay time, Chattanooga paratransit dataset

Overall, the trend of the results from the Chattanooga paratransit dataset shows the same as the results from New York City taxi dataset.

7.6 Pseudo Code

Algorithm 1 shows the overview of our offline PDPTW algorithm. Starting from the initial simulation time, we iteratively call the functions **WindowProcessing**, **RTV-ILP**, and **SimulateVehicle** until the end of the simulation time.

Algorithm 1: OfflineScheduling$(\mathcal{R}, \mathcal{V})$

```
$t \leftarrow 0$
$\hat{\mathcal{R}} \leftarrow \emptyset$
while $t < t_{max}$ do
    $\hat{\mathcal{R}} \leftarrow \hat{\mathcal{R}} \cup$ WindowProcessing$(t)$
    $Assignments \leftarrow$ RTV-ILP$(\hat{\mathcal{R}}, \mathcal{V})$
    $\mathcal{P}_{v \in \mathcal{V}, t}, \mathbb{R}_t \leftarrow$ SimulateVehicle$(Assignments, t)$
    for $v \in \mathcal{V}$ do
        $\hat{\mathcal{R}} \leftarrow \hat{\mathcal{R}} \setminus \mathcal{P}_{v,t}$
    end
    $t \leftarrow t + \Delta t$
end
Result: $\mathbb{R}$
```

Algorithm 2 shows a pseudo code for window processing. Rolling horizon factor c^{RH} is introduced to select an eligible set of requests $\tilde{\mathcal{R}}$. The procedure for selecting a set of requests to be considered based on their desired pickup time **PickupTime**(r_k). The window processing makes $\tilde{\mathcal{R}}$ to contain requests within the sliding window size T_w .

Algorithm 2: WindowProcessing(t)

```
$\tilde{\mathcal{R}} \leftarrow \emptyset$
if $t == 0$ then
    for $r_k \in R$ do
        if PickupTime$(r_k) \leq c^{RH} \times t_s$ then
            $\tilde{\mathcal{R}} \leftarrow \tilde{\mathcal{R}} \cup \{r_k\}$
        end
    end
end
else
    for $r_k \in R$ do
        if PickupTime$(r_k) \leq (t + c^{RH} \times t_s)$
         $\wedge$PickupTime$(r_k) > (t + c^{RH} - 1) \times t_s$
         then
            $\tilde{\mathcal{R}} \leftarrow \tilde{\mathcal{R}} \cup \{r_k\}$
        end
    end
end
Result: $\tilde{\mathcal{R}}$
```

7.7 Comparison to LKH3

We compare our solver to an extension of Lin-Kernighan-Helsgaun (LKH3). LKH3 is the state-of-the-art solver to solve TSP and its variants. Among 222 PDPTW benchmark instances, only 3 are less than the best-known solution (BKS), 180 are equal to BKS, and 139 are better than BKS. For more details, readers are encouraged to visit http://webhotel4.ruc.dk/~keld/research/LKH-3/.

The benchmark instances are artificially generated for PDPTW having varying lengths of time windows and ser-

Instance	LKH3		Rolling horizon Framework						Gap (%)
	VMT[a]	Compute time[b]	Compute time[b]		Service rate (%)		VMT[a]		
			$T_w = 5$	$T_w = 10$	$T_w = 5$	$T_w = 10$	$T_w = 5$	$T_w = 10$	
lc101	**997**	**12.05**	0.42	**0.44**	100	**100**	1095	**1127**	13.00
lc105	**1011**	**15.92**	0.50	**0.50**	98	**100**	1116	**1140**	12.73
lc106	**1032**	**22.81**	0.49	**0.50**	100	**100**	1172	**1163**	12.69
lc107	**1021**	**18.51**	0.49	**0.60**	98.03	**100**	1165	**1085**	6.23
lc108	**1030**	**18.81**	0.50	**0.52**	96.15	**100**	1209	**1120**	8.74
lc201	**1779**	**50.85**	0.39	**0.40**	100	**100**	1981	**2021**	13.57

Table 5: Comparison of LKH3 and Rolling Horizon Framework (t_s is set to 5 min; T_w is set to 5 and 10 min).
[a]Abbreviation for vehicle miles traveled; [b]The unit of the compute time is second.

vice times, which may not be the case in the dial-a-ride problem. We modified the PDPTW instances to better represent our problem setting. The entire time horizon varies in different instances and has no unit. We find a time window that corresponds to 30 minutes when we consider the entire horizon as 12 hours. Dwell time for pick up and drop off customers is also set to the value corresponding to 5 minutes in 12 hours. In order to have a fair comparison, we selected six adjusted instances (lc 101, 105, 106, 107, 108, and 201) that LKH3 can find routes to serve all customers without violating any demand and time window constraint.

Unlike our solver imposing hard constraints for demands and time windows, LKH3 has soft constraints. LKH3 defines a penalty to measure how much the constraints are violated. The primary objective is to minimize the penalty and the secondary objective is to minimize cost, so if a solution route has a positive penalty, it means the solver could not find such a route that does not violate any constraint. For the instances that LKH3 shows a 100% service rate with a positive value of penalty, our solver drops some customers because of infeasibility. For those instances, a fair comparison between our solver and LKH3 is hard. Hence, we will focus on the solutions with no penalty.

Table 5 compares the performance of our framework to LKH3. The total vehicle miles traveled (VMT) and computation time are used as metrics. The VMT of the rolling horizon framework is higher than the VMT of LKH3 with the optimality gaps varying from 6.23% to 13.57%. However, the rolling horizon framework is much faster than LKH3. While LKH3 takes 12.05 sec to 50.85 sec, the rolling horizon framework takes less than 0.6 seconds for all instances. In other words, the rolling horizon framework is around 20 to 80 times faster than the benchmark solver.