Offline Vehicle Routing Problem with Online Bookings: A Novel Problem Formulation with Applications to Paratransit

Amutheezan Sivagnanam[1] , **Salah Uddin Kadir**[1] , **Ayan Mukhopadhyay**[2] , **Philip Pugliese**[3] , **Abhishek Dubey**[2] , **Samitha Samaranayake**[4] and **Aron Laszka**[1]

[1]University of Houston
[2]Vanderbilt University
[3]Chattanooga Area Regional Transportation Authority
[4]Cornell University

Abstract

Vehicle routing problems (VRPs) can be divided into two major categories: offline VRPs, which consider a given set of trip requests to be served, and online VRPs, which consider requests as they arrive in real-time. Based on discussions with public transit agencies, we identify a real-world problem that is not addressed by existing formulations: booking trips with flexible pickup windows (e.g., 3 hours) in advance (e.g., the day before) and confirming tight pickup windows (e.g., 30 minutes) at the time of booking. Such a service model is often required in paratransit service settings, where passengers typically book trips for the next day over the phone. To address this gap between offline and online problems, we introduce a novel formulation, the *offline vehicle routing problem with online bookings*. This problem is very challenging computationally since it faces the complexity of considering large sets of requests—similar to offline VRPs—but must abide by strict constraints on running time—similar to online VRPs. To solve this problem, we propose a novel computational approach, which combines an anytime algorithm with a learning-based policy for real-time decisions. Based on a paratransit dataset obtained from the public transit agency of Chattanooga, TN, we demonstrate that our novel formulation and computational approach lead to significantly better outcomes in this setting than existing algorithms.

1 Introduction

Vehicle routing problems (VRPs) can be divided into two major categories. Offline VRPs consider a set of requests at once and optimize their assignment to planned vehicle routes [Golden *et al.*, 2008; Laporte, 1992]. Online VRPs, on the other hand, process requests as they arrive in real-time—either one-by-one or in small batches—and optimize their assignment to vehicle routes that may already be in progress [Toth and Vigo, 2002; Pillac *et al.*, 2013]. While online VRPs typically optimize fewer requests at a time, they are subject to stricter constraints on running time due to the online nature of the problems.

A socially beneficial application of VRPs is optimizing *paratransit* services [Lave and Mathias, 2000], which are curb-to-curb transportation services provided by public transit agencies for passengers who are unable to use fixed-route transit (e.g., passengers with disabilities). These services are crucial for providing transit accessibility to disadvantaged populations. Paratransit trips are typically booked at least one day in advance, which enables transit agencies to optimize routes as an offline VRP: before each day, an agency can optimize paratransit routes for that day based on all the requested pickup and drop-off locations and pickup time windows.

However, based on discussions with public transit agencies, we identified a problem that is not addressed by existing VRP formulations. When passengers book trips over the phone, they often request *broad pickup windows* (e.g., going for groceries in the afternoon, sometime between 2pm and 5pm). While passengers may have no preference between pickup times within these broad windows, they do strongly prefer to know in advance when they will be picked up. So, transit agencies must confirm a *tight pickup window* (e.g., 30 minute interval within the broad window) at the time of booking. The reason for this is very practical: vehicles may arrive at any time within the confirmed windows, and passengers need to be ready to be picked up. This presents an interesting online optimization problem: *how to select tight pickup windows at the time of booking, based on information available at the time, assuming that vehicle routes will be optimized as an offline VRP once all the trips have been booked?*

We formulate this as the *offline vehicle routing problem with online bookings*. We assume that trip requests with broad pickup windows are received one-by-one, and for each request, a tight pickup window must be selected in a matter of seconds. At the end of the booking process, vehicle routes are optimized as an offline VRP based on the selected tight windows. The objective of optimizing the tight pickup windows is to minimize the cost of the resulting offline VRP. Note that this booking problem *can be defined with respect to a wide range of offline VRP formulations* (that consider pickup windows), so our framework could be applied to a range of real-world problems where tight pickup windows must be chosen during booking (e.g., scheduling the delivery of refrigerated goods or dial-a-ride services). In this paper, we consider an offline VRP formulation that models paratransit services.

This problem is very challenging computationally since it faces the complexity of considering large sets of trips—

similar to offline VRPs—but must abide by strict limits on running time—similar to online VRPs. To address this challenge, we propose a novel computational approach that combines an *anytime algorithm* with a *reinforcement-learning based policy*. We demonstrate that our novel formulation and computational approach lead to significantly better outcomes in the paratransit-booking setting than existing algorithms using real-world data from a public transit agency.

2 Model and Problem Formulation

We formulate the *offline VRP with online bookings* by first introducing an *offline vehicle routing problem*, which models the optimization of allocating trip requests to vehicle routes once all the trip requests have been booked and their tight pickup windows have been confirmed. Building on this offline problem, we then formulate the *online booking problem*, which models the optimization of tight pickup windows in real time, assuming that vehicle routes will be optimized afterwards. Table 1 in Appendix A provides a list of symbols.

2.1 Vehicle Routing Problem with Time Windows

Offline VRPs with time windows is a family of classical combinatorial problems. Here, we introduce the offline VRP formulation that we employ in our experiments, which we developed to model paratransit services. However, it is important to note that the online bookings problem could be defined with respect to a wide range of offline VRP formulations with pickup time windows, and our proposed solution approach can incorporate existing offline VRP solvers for these problems. Since our offline VRP is a minor variation of classical formulations, here we provide only a concise summary of this problem, which is sufficient for formulating the novel online bookings problem. Due to lack of space, we provide a detailed formal definition in Appendix B.

Problem Input The input of the offline VRP problem is an ordered set of *trip requests* $\boldsymbol{T} = \langle T_1, T_2, \ldots, T_n \rangle$, where each trip request T_i contains a pickup location L_i^{pickup}, a drop-off location $L_i^{dropoff}$, and the number of passengers to be transported P_i; and a corresponding ordered set of *tight pickup time windows* $\boldsymbol{w} = \langle w_1, w_2, \ldots, w_n \rangle$, where each time window w_i is defined by an earliest w_i^{start} and latest w_i^{end} pickup time. The input also includes constants, such as the maximum allowed duration $D^{maxroute}$ of a vehicle route, the passenger capacity V of the vehicles, and so on (see Appendix B). For ease of presentation, we will not list these constants explicitly and represent a VRP instance simply as $(\boldsymbol{T}, \boldsymbol{w})$, assuming that the constants are provided implicitly.

Solution and Objective A solution to the offline VRP problem is a set of *vehicle routes* $\boldsymbol{R} = \{R_1, R_2, \ldots, R_m\}$, where each route is an ordered set of pickups L_i^{pickup} and drop-offs $L_i^{dropoff}$ (note that trips may be combined in a route, i.e., pickups and drop-offs of different trips may be interleaved). A set of vehicle routes $\boldsymbol{R}$ is a feasible solution if each pickup L_i^{pickup} is included in exactly one route R_j, the corresponding dropoff $L_i^{dropoff}$ is also included in R_j, and every route satisfies time constraints (passengers are picked up within the pickup time windows, travel times between locations are respected, etc.) and vehicle capacity constraints (see Appendix B). We let $\mathcal{R}(\boldsymbol{T}, \boldsymbol{w})$ denote the set of feasible solutions for a VRP instance $(\boldsymbol{T}, \boldsymbol{w})$.

The cost of a solution depends on the number of vehicle routes and the duration of each route (see Appendix B). By letting $\mathcal{C}(\boldsymbol{R})$ denote the cost of a solution $\boldsymbol{R}$, we can express the offline VRP problem as $\operatorname{argmin}_{\boldsymbol{R} \in \mathcal{R}(\boldsymbol{T}, \boldsymbol{w})} \mathcal{C}(\boldsymbol{R})$. Finally, we let $VRP^*(\boldsymbol{T}, \boldsymbol{w})$ denote the total cost of an optimal solution for problem instance $(\boldsymbol{T}, \boldsymbol{w})$. That is, $VRP^*(\boldsymbol{T}, \boldsymbol{w}) = \min_{\boldsymbol{R} \in \mathcal{R}(\boldsymbol{T}, \boldsymbol{w})} \mathcal{C}(\boldsymbol{R})$. Since there is a vast literature on solving offline VRPs, we assume that an offline VRP solver (i.e., heuristic or approximation algorithm for VRP^*) is given, and focus on the online bookings problem in this paper.

2.2 Online Bookings Problem

Building on the offline VRP formulation, we now introduce the online bookings problem. In this real-time decision problem, trip requests $T_1, T_2, \ldots$ are received one-by-one, and each trip request T_i is accompanied by a *broad pickup window* W_i. Our goal is to select a tight pickup window $w_i \subset W_i$ for each trip request T_i in real-time (i.e., in a few seconds after the request is received), so that once we have received all the trip requests $\boldsymbol{T}$ and selected all the pickup windows $\boldsymbol{w}$, the total cost $VRP^*(\boldsymbol{T}, \boldsymbol{w})$ of the resulting offline VRP is minimized. To model uncertainty and expectations about future requests, we assume that the sets of trip requests and broad pickup windows $(\boldsymbol{T}, \boldsymbol{W})$ are drawn at random from a known probability distribution $\mathcal{D}$ (note that the number of requests is variable). So, each decision is based on previously received requests and expectation of future ones (i.e., distribution $\mathcal{D}$).

Problem Input Formally, the input for the ith decision is the ordered set of trip requests (including the ith request) $\langle T_1, \ldots, T_i \rangle$, the ordered set of previously selected tight pickup windows (up to the $(i-1)$th request) $\langle w_1, \ldots, w_{i-1} \rangle$, and a broad pickup window W_i, which specifies the earliest W_i^{start} and latest W_i^{end} pickup time for request T_i. The input also includes the probability distribution $\mathcal{D}$, the maximum duration of tight pickup windows D^{window} (e.g., 30 minutes), and any additional inputs that are required by the offline VRP (e.g., vehicle capacity, maximum route duration); for ease of presentation, we will not list these additional inputs explicitly.

Decision Space and Objective The output of the ith decision is a tight pickup window w_i that is at most D^{window} long (i.e., $w_i^{end} - w_i^{start} \leq D^{window}$) and falls within the broad window (i.e., $W_i^{start} \leq w_i^{start} \leq w_i^{end} \leq W_i^{end}$).

Whether a decision w_i is optimal depends not only on the received requests and on our expectation of future requests, but also on how we will respond to those future requests. Thus, instead of trying to formulate the online booking problem as optimizing each decision w_i, we formulate it as optimizing a decision-making policy μ, which maps each input $(\langle T_1, \ldots, T_i \rangle, \langle w_1, \ldots, w_{i-1} \rangle, W_i)$ to a tight pickup window w_i. Formally, our goal is to find an *optimal decision policy* μ^*, which minimizes the expected cost of the resulting offline VRP instance $(\boldsymbol{T}, \boldsymbol{w})$:

$$\operatorname{argmin}_{\mu} \mathbb{E}_{(\boldsymbol{T},\boldsymbol{W})\sim\mathcal{D}}\left[\mathit{VRP}^*\left(\boldsymbol{T}, \boldsymbol{w}\right) \Big|_{w_i = \mu(\langle T_1,\ldots,T_i\rangle, \langle w_1,\ldots,w_{i-1}\rangle, W_i)} \right].$$

Since the decision problem is subject to real-time constraints (i.e., when someone calls over the phone to book a paratransit trip, the transit agency must respond during the phone call, within seconds), we must be able to evaluate the policy μ^* in a matter of seconds for any input.

3 Solution Approach

3.1 Anytime Algorithm

The online booking problem is computationally challenging since it incorporates the offline VRP into its objective, which is a computationally-hard combinatorial optimization problem ([Lenstra and Kan, 1981]). Indeed, existing approaches for solving offline VRPs are not well suited for real-time applications (i.e., finding solutions within a matter of seconds).

To address this challenge, we propose *performing computation between consecutive decisions*. While there is limited time for each real-time decision, there is significantly more time between consecutive decisions (i.e., from when a tight window is selected to when the next request is received). We can take advantage of this extra time by continuously working on a vehicle-routing solution, which can then be used as supporting input in the next real-time decision.

Unfortunately, the amount of time between consecutive requests is not known in advance since calls arrive at random times (note that the arrival time of a request is different from its broad pickup window). Thus, we propose to employ an *anytime VRP algorithm*, which we can start after each real-time decision and stop when the next request arrives.

Anytime-supported Online Bookings Problem

Based on the above ideas, we reformulate our online decision problem as the *anytime-supported online bookings problem.*

Policy Input and Decision Space The input for the ith decision is the same as before, but now also includes a feasible VRP solution $\boldsymbol{R}^{(i-1)}$ (i.e., a set of routes), provided by the $(i-1)$th execution of the anytime algorithm, which we specify below. For ease of exposition, we define $\boldsymbol{R}^{(0)} = \emptyset$ for the very first request T_1. The output of the ith decision is also the same as before, but now also includes a feasible VRP solution $\hat{\boldsymbol{R}}^{(i)}$ (i.e., $\hat{\boldsymbol{R}}^{(i)} \in \mathcal{R}(\langle T_1, \ldots, T_i\rangle, \langle w_1, \ldots, w_i\rangle)$, provided as supporting input for the ith execution of the anytime algorithm. Note that we could omit the VRP solution $\hat{\boldsymbol{R}}^{(i)}$ from the output of the online decision and let the anytime algorithm assign the new request to a route. However, we found that providing a feasible solution as a starting point for the anytime algorithm is very beneficial in practice since the selection of the pickup window must consider anyway how the request will "fit" into the routes. Also note that finding a feasible VRP solution does not introduce a computational challenge since we can let the decision policy assign each new request to a new route and leave existing routes unchanged (achieving feasibility, but leaving all of the VRP optimization to the anytime algorithm); we of course train our policy to provide better solutions.

Anytime Algorithm Input and Output The input for the ith execution of the anytime algorithm consists of the trip requests $\langle T_1, \ldots, T_i\rangle$, the tight pickup windows $\langle w_1, \ldots, w_i\rangle$, and a feasible VRP solution $\hat{\boldsymbol{R}}^{(i)}$, provided by the ith decision of the policy. The output of the ith execution of the anytime algorithm is an improved feasible VRP solution $\boldsymbol{R}^{(i)}$, provided for the $(i+1)$th decision of the policy.

The objective of the anytime algorithm α is to find a minimum-cost feasible solution for the VRP instance $(\langle T_1, \ldots, T_i\rangle, \langle w_1, \ldots, w_i\rangle)$, using the solution $\hat{\boldsymbol{R}}^{(i)}$ as a *warm start* (e.g., as initial solution for simulated annealing):

$$\boldsymbol{R}^{(i)} = \alpha\big(\langle T_1, \ldots, T_i\rangle, \langle w_1, \ldots, w_i\rangle, \hat{\boldsymbol{R}}^{(i)}\big) \approx \operatorname{argmin}_{\boldsymbol{R} \in \mathcal{R}(\langle T_1,\ldots,T_i\rangle, \langle w_1,\ldots,w_i\rangle)} \mathcal{C}(\boldsymbol{R}).$$

Note that the objective of the anytime algorithm does not consider future requests, only ones that have been received. In our experiments, we found that we can attain very good performance by letting the decision policy handle expectations about future requests, and restricting the anytime algorithm to optimizing for requests that have been received.

Optimal Decision Policy Finally, we can reformulate our goal for the online bookings problem as finding an optimal decision policy μ^* for selecting tight pickup windows, supported by the anytime algorithm α:

$$\operatorname{argmin}_{\mu} \mathbb{E}_{(\boldsymbol{T},\boldsymbol{W})\sim\mathcal{D}}\left[\mathit{VRP}^*\left(\boldsymbol{T}, \boldsymbol{w}\right) \Big|_{\left(w_i, \hat{\boldsymbol{R}}^{(i)}\right) = \mu\left(\langle T_1,\ldots,T_i\rangle, \langle w_1,\ldots,w_{i-1}\rangle, W_i, \boldsymbol{R}^{(i-1)}\right),\ \boldsymbol{R}^{(i)} = \alpha\left(\ldots\right)} \right].$$

Note that the anytime algorithm α can be implemented using an existing offline VRP solver—as long as it is anytime.

3.2 Decision Policy

We can view online bookings as a Markov decision process (MDP): a decision input $(\langle T_1, \ldots, T_i\rangle, \langle w_1, \ldots, w_{i-1}\rangle, W_i, \boldsymbol{R}^{(i-1)})$ is a *state of the environment*, a decision output $\big(w_i, \hat{\boldsymbol{R}}^{(i)}\big)$ is an *action*, and running the anytime algorithm until a new request arrives at random is the *state transition* (reaching a terminal state when no more requests arrive for the day). To formulate an MDP, we also have to define the *immediate cost* (i.e., immediate negative reward) that we incur for taking an action. The online bookings problem quantifies costs at the end of the day—after the last decision—based on the total cost of the resulting offline VRP instance $(\boldsymbol{T}, \boldsymbol{w})$. Thus, the immediate cost c_i incurred for the ith decision is

$$c_i = \begin{cases} 0 & \text{if } i < |\boldsymbol{T}| \\ \mathit{VRP}^*(\boldsymbol{T}, \boldsymbol{w}) & \text{if } i = |\boldsymbol{T}|. \end{cases}$$

By formulating the online bookings problem as an MDP, we enable the application of *reinforcement learning* (RL) to find an optimal decision policy μ^*. The advantage of RL is that once a policy μ^* has been trained, the computational cost of execution is low, which is crucial for real-time decisions.

To find an optimal policy, an RL algorithm gathers experiences by repeatedly interacting with the environment in a number of training episodes, recording the experienced states, actions, and immediate costs. In our case, experiences can be gathered by running simulations of the online bookings process, where an input $(\boldsymbol{T}, \boldsymbol{w})$ is drawn at random from distribution $\mathcal{D}$ for each episode. From these experiences, an RL algorithm can learn an optimal policy that minimizes the expected cumulative cost. Many popular RL algorithms, such as deep Q-learning (DQN) and its variants, learn a policy by learning an *action-value function*, which estimates the expected cumulative cost when taking a given action in a given state (e.g., using the recursive Bellman equation to consider future costs). Once the action-value function has been learned, the optimal policy is to simply choose an action that minimizes the action-value function in the current state.

Cost Design RL approaches face two significant challenges in our environment. First, the total cost of the offline VRP instance depends as much on the decision inputs (e.g., on the sheer number of trip requests) as it does on the decisions of the policy. Since the decision inputs are random and vary significantly (e.g., there are significant differences between the number of trip requests each day), experiences will be extremely noisy and difficult to learn from. Second, simulating the environment is very expensive computationally since each state transition requires running the anytime algorithm for a significant amount of time (e.g., 5 minutes of running time to obtain a single experience). This greatly exacerbates the problem of noisy experiences since a low number of noisy experience can lead to very inaccurate action-value functions.

To address these challenges, we replace the original immediate cost c_i of the MDP with a *shaped cost* $\tilde{c}_i$, which assigns a cost to each individual decision:

$$\begin{aligned}\tilde{c}_i = &\mathit{VRP}^*(\boldsymbol{T}, \quad \langle w_1, \ldots, w_{i-1}, w_i, W_{i+1}, \ldots, W_{|\boldsymbol{T}|}\rangle) \\ &- \mathit{VRP}^*(\boldsymbol{T}, \langle w_1, \ldots, w_{i-1}, W_i, W_{i+1}, \ldots, W_{|\boldsymbol{T}|}\rangle).\end{aligned}$$

The rationale behind the above formulation is to capture the impact of narrowing down the broad window W_i to a tight window w_i in the ith decision. This shaped cost $\tilde{c}_i$ formulation has two advantages. First, notice that

$$\sum_{i=1}^{|\boldsymbol{T}|} \tilde{c}_i = \left(\sum_{i=1}^{|\boldsymbol{T}|} c_i \right) - \mathit{VRP}^*(\boldsymbol{T}, \boldsymbol{W}).$$

In other words, the difference between the original cumulative cost $\sum c_i$ and the shaped cumulative cost $\sum \tilde{c}_i$ is removing a part of the cost that does not depend on the decisions (i.e., removing the cost $\mathit{VRP}^*(\boldsymbol{T}, \boldsymbol{W})$ of a "baseline" VRP instance $(\boldsymbol{T}, \boldsymbol{W})$, where we could schedule pickups for any time within the broad windows $\boldsymbol{W}$). So, shaped costs $\tilde{c}_i$ capture only the impact of the decisions, thereby reducing noise.

Second, since the shaped cost $\tilde{c}_i$ captures the impact of a decision considering all future requests (i.e., considering the complete ordered sets $\boldsymbol{T}$ and $\boldsymbol{W}$), we can use it to quantify the value of a decision without taking future costs into account. In other words, the expected impact of a decision on the total cost $\mathit{VRP}^*(\boldsymbol{T}, \boldsymbol{w})$ is captured by the immediate shaped cost $\tilde{c}_i$ since this cost $\tilde{c}_i$ is the increase in the cost of the offline VRP with all the trip requests $\boldsymbol{T}$. Hence, we can use experiences to learn a value function that estimates the expected immediate shaped cost $\tilde{c}_i$ of a given action in a given state; once the value function has been learned, our policy is to simply choose an action that minimizes the value (i.e., cost) in the current state. This significantly reduces the complexity of learning and, thus, the number of experiences required.

Note that during training, we can estimate shaped cost $\tilde{c}_i$ since we simulate the environment, so we can generate all the trip requests $(\boldsymbol{T}, \boldsymbol{W})$ before feeding them to the policy one-by-one. Once the value function has been learned, calculating the shaped cost $\tilde{c}_i$ is no longer necessary since the policy is to choose an action that minimizes the learned value (i.e., cost) function in the current state. Finally, note that calculating VRP^* is computationally hard, so we can use a heuristic VRP solver during training to estimate shaped cost $\tilde{c}_i$.

Value Function Since our value function considers only the immediate shaped cost $\tilde{c}_i$, we can apply a simplified version of the popular DQN algorithm. Before applying DQN, we have to discretize the action space, that is, constrain each decision to a discrete set of choices (e.g., pickup times must be multiples of 15 minutes). Such discretization is natural for transit agencies that prefer "round" pickup times.

Our goal is to learn the value function Q:

$$Q\big(\underbrace{\langle T_1, \ldots, T_i\rangle, \langle w_1, \ldots, w_{i-1}\rangle, W_i, \boldsymbol{R}^{(i-1)}}_{\mathit{state}}, \underbrace{w_i, \hat{\boldsymbol{R}}^{(i)}}_{\mathit{action}}\big) \approx \tilde{c}_i.$$

Once we have learned the value function Q, our policy μ^* is to iterate over the actions and select one that minimizes the cost in the current state:

$$\mu^*(\mathit{state}) = \operatorname{argmin}_{w_i, \hat{\boldsymbol{R}}^{(i)}} Q(\mathit{state}, w_i, \hat{\boldsymbol{R}}^{(i)}).$$

To enable learning, we represent Q as a neural network, which we initialize with random weights. During training, we execute our policy in simulated environments, where the inputs $(\boldsymbol{T}, \boldsymbol{w})$ are drawn at random from distribution $\mathcal{D}$. We collect experiences, that is, tuples of state, action, and immediate cost, and we use these experiences to train the neural network. As is standard in RL, we also include random actions in the training to balance exploration and exploitation.

Features Learning the value function Q poses one last challenge due to the size and complexity of the state space (i.e., space of all possible decision inputs, including possible sets of requests $\langle T_1, \ldots, T_i\rangle$ and feasible sets of runs $\boldsymbol{R}^{(i-1)}$). While similar state spaces have been considered in prior work (e.g., [Joe and Lau, 2020; Yu *et al.*, 2019]), the challenge in our problem is exacerbated by the prohibitively high computational cost of the environment (one state transition may require running an anytime algorithm for 5 minutes), which limits the number of experiences that we can collect.

To reduce the number of experiences required for training the value function, we map the large and complex space of states and actions to a low- and fixed-dimensional space of *feature vectors*, and we replace the input of the value function Q with a feature vector. Specifically, we map each state-action pair to a vector of features, which consider how well the action fits the current state (i.e., how well it fits the previously chosen tight windows $\langle w_1, \ldots, w_{i-1}\rangle$ and vehicle routes $\boldsymbol{R}^{(i-1)}$) and our expectation of future trip requests (i.e., distribution $\mathcal{D}$).

Features that consider how well the action $(w_i, \hat{\boldsymbol{R}}^{(i)})$ fits the previously chosen tight windows $\langle w_1, \ldots, w_{i-1}\rangle$ and vehicle routes $\boldsymbol{R}^{(i-1)}$ include (1) the increase in the duration of routes due to taking the action (compared between $\hat{\boldsymbol{R}}^{(i)}$ and $\boldsymbol{R}^{(i-1)}$), (2) the increase in the driving distance of routes, and (3) the "tightness" of the route schedule $\hat{\boldsymbol{R}}^{(i)}$, that is, how much time slack is left in the route before and after serving trip request T_i. Features that consider the distribution $\mathcal{D}$ include the expected number of trips requests whose pickup locations are nearby the pickup location L_i^{pickup} of request T_i and/or whose drop-off locations are nearby the drop-off location L_i^{dropoff} and/or whose broad pickup windows are around the same time as window w_i, as well as the expected number of future trip requests (i.e., expectation of $|\boldsymbol{T}| - i$). For any state and action, these features can be calculated at a relatively low computational cost based on historical data, which enables real-time application. Due to lack of space, we provide a formal description of these features in Appendix C.

Training Process Before we begin training, we initialize the value function Q, which is represented by a neural network, with random weights. We then train the policy μ (i.e., value function Q) over a number of training episodes, where each episode is a simulation of the online booking process with a random input $(\boldsymbol{T}, \boldsymbol{W})$ drawn from the probability distribution $\mathcal{D}$, which we estimate based on historical data. To simulate the online booking process, we process the trip requests $\boldsymbol{T}$ one-by-one, first applying the decision policy μ and then the anytime algorithm α for each request T_i. To apply the policy μ, we calculate the feature vector for every action $(w_i, \hat{\boldsymbol{R}}^{(i)})$, evaluate the value function Q over these feature vectors, and select the action that minimizes the value (i.e., cost). Next, we run the anytime algorithm α, which provides supporting input for the next policy decision. We terminate the algorithm after a random amount of running time, which models the random inter-arrival time of requests (based on historical data). Then, we repeat with next request T_{i+1}; or with the next episode if $i = |\boldsymbol{T}|$.

After each simulated decision, we collect an experience (i.e., a tuple of the feature vector and the shaped cost c_i) by calculating the shaped cost $\tilde{c}_i$ using $\boldsymbol{T}$, $\boldsymbol{W}$, $\langle w_1, \ldots, w_i\rangle$, and a heuristic for *VRP**. We use these experiences to train the value function Q (i.e., the neural network). In the beginning, the policy μ chooses actions at random since function Q is initialized randomly; but as we train function Q using more and more experiences, the policy improves and converges to an optimum μ^* (given feature-vector inputs and objective c_i). To balance exploration and exploitation and to avoid converging to a local optimum, we occasionally take random actions during training, as is standard in RL.

4 Evaluation

4.1 Dataset and Experimental Setup

Paratransit Data To evaluate our proposed approach, we obtained real-world paratransit data from CARTA, the public transit agency of Chattanooga, TN, a mid-sized U.S. city. This dataset spans 180 days of paratransit service, with an average of 140 trips per day (minimum of 7 and maximum of 234 trips). Each trip has an associated pickup and drop-off location (specified as latitude-longitude pairs), the number of passengers, and the scheduled pickup time. The anonymized dataset and our software implementation are available at https://github.com/smarttransit-ai/ijcai22

Based on input from the agency, we instantiate our model with vehicle capacity $V = 9$, maximum route duration $D^{\mathit{maxroute}} = 10$ hours, and maximum tight pickup window duration $D^{\mathit{window}} = 30$ minutes. Since the agency did not record the requested broad windows (current over-the-phone booking process is manual), we assume each broad window W_i to be 3 hours long (which is a practical value for the service) and centered around the scheduled pickup time.

Experimental Setup We calculate travel times between the locations using road-network data from OpenStreetMaps. The road network contains 10,788 nodes (i.e., intersection) and 28,100 edges (i.e., roads). For calculating feature vectors, we consider two locations to be nearby if they have the same ZIP code. We assume that the transit agency operating paratransit services must serve all the passenger requests according to the Americans with Disabilities Act.

We implemented our framework in Python 3.8. To provide anytime algorithms, we implemented a heuristic greedy α^{Greedy} and a meta-heuristic simulated annealing algorithm α^{SimAnn}, which we use in tandem as our anytime VRP solver $\alpha^{\mathit{SimAnn+Greedy}}$. Since these are based on standard techniques, we describe them in Appendix D. During both training and evaluation, we let the anytime algorithm run for 5 minutes on average. In our experiments, we consider two offline VRP solvers: VROOM [VRoom Project, 2021] and the Google OR-Tools Vehicle Routing framework [Google, 2021]. We do not impose a running time limit on either VROOM or Google OR-Tools. To represent the value function Q, we use a neural network with one input layer, one hidden layer (64 neurons, ReLU activation), and one output layer (linear activation). To train the network, we use the Adam optimizer [Kingma and Ba, 2014] from the Keras library.

4.2 Results

We provide supplementary numerical results in Appendix E.

Running Time We run all algorithms on an Intel Xeon E5-2680 28-core CPU with 128GB of RAM. The running time of the trained decision-making policy μ^*, including the calculation of the feature vector, is 0.25 seconds on average and 2 seconds in the worst case. This is *sufficiently low for our problem setting*, where we typically have a couple of seconds to make an online decision. The running time of one episode of training is 1 day on average and 2 days in the worst case. Note that this running time cannot be significantly lowered (other than simulating multiple episodes in parallel) because the training environment has to simulate the real bookings process, where the anytime algorithm is running for the entire day. As for the offline VRP solvers, the greedy algorithm α^{Greedy} can assign all trip requests $\boldsymbol{R}$ for a day in 15 seconds with tight pickup windows $\boldsymbol{w}$ and in 3

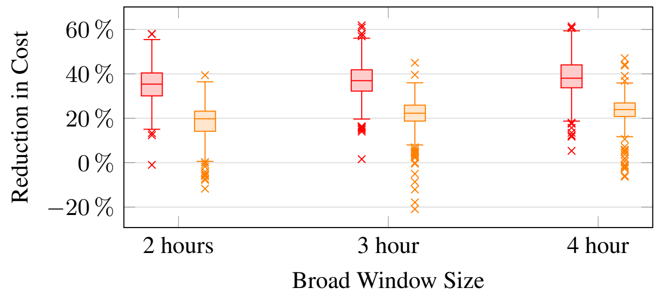

Figure 1: Reduction in total cost due to using our approach for selecting tight pickup windows (policy μ^* supported by anytime algorithm $\alpha^{SimAnn+Greedy}$), compared to using naïve pickup windows with VROOM (🟥) and Google OR-Tools (🟧) as offline VRP solvers.

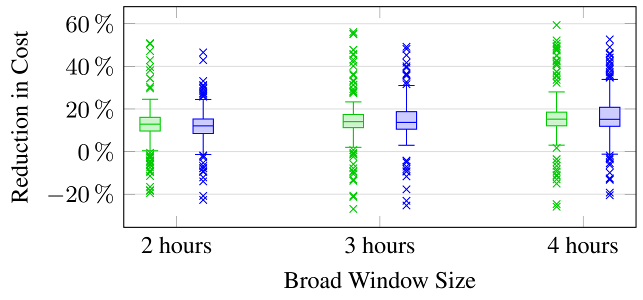

Figure 2: Reduction in total cost due to using our complete approach (policy μ^* supported by anytime algorithm $\alpha^{SimAnn+Greedy}$), compared to using a policy μ^* without anytime support (🟩) and using naïve pickup windows with algorithm $\alpha^{SimAnn+Greedy}$ as the offline VRP solver (🟦).

minutes with broad pickup windows $\boldsymbol{W}$; the simulated annealing algorithm α^{SimAnn} performs around 30 iterations per second; and VROOM and Google OR-Tools take around 3.3 minutes and 1 minute on average, respectively, to solve an offline VRP instance.

Proposed Approach vs. Naïve Pickup Windows Next, we demonstrate the effectiveness of our proposed approach by showing that optimizing tight pickup windows can lead to significant reductions in cost. Since the online bookings problem is novel to the best of our knowledge, existing VRP solvers do not address the online selection of tight pickup windows. Therefore, to provide baselines for comparison, we consider existing offline VRP solvers with "naïvely selected" pickup windows, which we define as selecting the middle interval of broad pickup windows as tight windows.

Figure 1 shows the reduction in the total cost of vehicle routes due to using our proposed approach, compared to using VROOM and Google OR-Tools with naïvely selected pickup windows. For each comparison, we evaluate the algorithms on 180 days of paratransit data, and plot the distributions. We observe a significant reduction in costs compared to both baseline solvers. Further, we find that our approach is robust to variations in the duration of broad pickup windows since it maintains a significant advantage when broad windows are 2- or 4-hours long, even though the decision policy μ^* was trained only on 3-hour broad windows.

Advantage of Combining Policy with Anytime Algorithm While Figure 1 demonstrates the effectiveness of our proposed approach, it does not prove that every element of our approach is necessary. One may wonder if a simpler approach would work equally well. To demonstrate that both the anytime algorithm and the learning-based decision policy are crucial, we compare our complete approach to (1) using the decision policy μ^* without anytime support and (2) using the anytime algorithms $\alpha^{SimAnn+Greedy}$ as offline VRP solvers with naïve pickup windows (i.e., without decision policy).

Figure 2 shows the reduction in the total cost of vehicle routes due to using our complete approach compared to incomplete variants (1) and (2). We observe that there is a significant reduction in cost compared to both, which demonstrates that both the learning-based decision policy μ^* and the anytime algorithms $\alpha^{SimAnn+Greedy}$ are crucial.

5 Related Work

Some prior works focus on solving the dial-a-ride problem, an online VRP [Berbeglia *et al.*, 2012; Liu *et al.*, 2015; Parragh *et al.*, 2015; Wilbur *et al.*, 2022].

Mo *et al.* [2018] focus on advance booking in an offline VRP. We also consider advance bookings (i.e., day before the travel). De Filippo *et al.* [2021] consider enhancing the solution quality of offline VRP by using online algorithms that can optimize the solution obtained from offline VRP algorithms. Prior works such as [Lowalekar *et al.*, 2019; Shen *et al.*, 2019; Alonso-Mora *et al.*, 2017; Ota *et al.*, 2016; Simonetto *et al.*, 2019; Yu *et al.*, 2019; Joe and Lau, 2020] consider real-time demand. Among them, Simonetto *et al.* [2019] and Alonso-Mora *et al.* [2017] consider real-time positioning of vehicles. Gupta *et al.* [2010] and Wen *et al.* [2018] consider both real-time vehicle scheduling and advance booking. Simonetto *et al.* [2019] consider a system where the agency uses idle vehicles by relaxing the time-related constraints, rather than rejecting user requests.

Nguyen *et al.* [2019] consider a hierarchical approach by prioritizing requests. In our paratransit service setting, we treat all service requests with the same priority. Simonetto *et al.* [2019] assign one request to one vehicle from a given batch of requests for faster real-time assignment. Goodson *et al.* [2017] consider a lookahead strategy by using rollout algorithms. Joe and Lau [2020] consider a route-based MDP.

6 Conclusion

Optimizing pickup windows during day-ahead trip booking can be crucial for offline VRPs (e.g., paratransit service applications). In this paper, we propose a novel problem formulation to capture offline VRPs with online bookings. We also introduce a novel computational approach that combines a learning-based policy with an anytime algorithm. Based on experiments with real-world paratransit data from CARTA, the public transit agency of Chattanooga, TN, we observe a significant reduction in costs due to selecting pickup windows using our decision policy instead of naïve selection. Further, our experiments also show a reduction of 14 - 18% in costs due to using our policy in tandem with an anytime algorithm instead of using the policy by itself.

Acknowledgements

This material is based upon work sponsored by the National Science Foundation under Grant CNS-1952011 and by the Department of Energy under Award DE-EE0009212. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or the Department of Energy.

References

[Agatz *et al.*, 2011] Niels Agatz, Alan L Erera, Martin WP Savelsbergh, and Xing Wang. Dynamic ride-sharing: A simulation study in metro atlanta. *Procedia-Social and Behavioral Sciences*, 17:532–550, 2011.

[Alonso-Mora *et al.*, 2017] Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 114(3):462–467, 2017.

[Berbeglia *et al.*, 2012] Gerardo Berbeglia, Jean-François Cordeau, and Gilbert Laporte. A hybrid tabu search and constraint programming algorithm for the dynamic dial-a-ride problem. *INFORMS Journal on Computing*, 24(3):343–355, 2012.

[De Filippo *et al.*, 2021] Allegra De Filippo, Michele Lombardi, and Michela Milano. Integrated offline and online decision making under uncertainty. *Journal of Artificial Intelligence Research*, 70:77–117, 2021.

[Golden *et al.*, 2008] Bruce L Golden, Subramanian Raghavan, and Edward A Wasil, editors. *The vehicle routing problem: latest advances and new challenges*. 2008.

[Goodson *et al.*, 2017] Justin C Goodson, Barrett W Thomas, and Jeffrey W Ohlmann. A rollout algorithm framework for heuristic solutions to finite-horizon stochastic dynamic programs. *European Journal of Operational Research*, 258(1):216–229, 2017.

[Google, 2021] Google. Google OR-Tools routing framework. https://developers.google.com/optimization/routing, 2021.

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

[Gupta *et al.*, 2010] Diwakar Gupta, Hao-Wei Chen, Lisa A Miller, and Fajarrani Surya. Improving the efficiency of demand-responsive paratransit services. *Transp. Research Part A: Policy and Practice*, 44(4):201–217, 2010.

[Joe and Lau, 2020] Waldy Joe and Hoong Chuin Lau. Deep reinforcement learning approach to solve dynamic vehicle routing problem with stochastic customers. In *30th International Conference on Automated Planning and Scheduling (ICAPS)*, volume 30, pages 394–402, 2020.

[Kingma and Ba, 2014] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[Laporte, 1992] Gilbert Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Oper. Res.*, 59(3):345–358, 1992.

[Lave and Mathias, 2000] Roy Lave and Rosemary Mathias. State of the art of paratransit. *Transportation in the New Millennium*, 478:1–7, 2000.

[Lenstra and Kan, 1981] Jan K Lenstra and AHG Rinnooy Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227, 1981.

[Liu *et al.*, 2015] Mengyang Liu, Zhixing Luo, and Andrew Lim. A branch-and-cut algorithm for a realistic dial-a-ride problem. *Transportation Research Part B: Methodological*, 81:267–288, 2015.

[Lowalekar *et al.*, 2019] Meghna Lowalekar, Pradeep Varakantham, and Patrick Jaillet. ZAC: A zone path construction approach for effective real-time ridesharing. In *29th International Conference on Automated Planning and Scheduling*, volume 29, pages 528–538, 2019.

[Mo *et al.*, 2018] Daniel Y Mo, Yue Wang, YCE Lee, and Mitchell M Tseng. Mass customizing paratransit services with a ridesharing option. *IEEE Transactions on Engineering Management*, 67(1):234–245, 2018.

[Nguyen *et al.*, 2019] Quoc Chinh Nguyen, Liuqin Yang, Zhuolun Li, and Wenqing Chen. Hierarchical vehicle routing for online delivery platform. In *22nd IEEE Intelligent Transportation Systems Conf.*, pages 1709–1714, 2019.

[Ota *et al.*, 2016] Masayo Ota, Huy Vo, Claudio Silva, and Juliana Freire. Stars: Simulating taxi ride sharing at scale. *IEEE Transactions on Big Data*, 3(3):349–361, 2016.

[Parragh *et al.*, 2015] Sophie N Parragh, Jorge Pinho de Sousa, and Bernardo Almada-Lobo. The dial-a-ride problem with split requests and profits. *Transportation Science*, 49(2):311–334, 2015.

[Pillac *et al.*, 2013] Victor Pillac, Michel Gendreau, Christelle Guéret, and Andrés L Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1–11, 2013.

[Qian *et al.*, 2017] Xinwu Qian, Wenbo Zhang, Satish V Ukkusuri, and Chao Yang. Optimal assignment and incentive design in the taxi group ride problem. *Transportation Research Part B: Methodological*, 103:208–226, 2017.

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

[Ropke and Cordeau, 2009] Stefan Ropke and Jean-François Cordeau. Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science*, 43(3):267–286, 2009.

[Ropke and Pisinger, 2006] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.

[Ropke *et al.*, 2007] Stefan Ropke, Jean-François Cordeau, and Gilbert Laporte. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks: An International Journal*, 49(4):258–272, 2007.

[Salazar *et al.*, 2018] Mauro Salazar, Federico Rossi, Maximilian Schiffer, Christopher H Onder, and Marco Pavone. On the interaction between autonomous mobility-on-demand and public transportation systems. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2262–2269. IEEE, 2018.

[Shen *et al.*, 2019] Bilong Shen, Bo Cao, Ying Zhao, Haojia Zuo, Weimin Zheng, and Yan Huang. Roo: Route planning algorithm for ride sharing systems on large-scale road networks. In *6th IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 1–8, 2019.

[Simonetto *et al.*, 2019] Andrea Simonetto, Julien Monteil, and Claudio Gambella. Real-time city-scale ridesharing via linear assignment problems. *Transportation Research Part C: Emerging Technologies*, 101:208–232, 2019.

[Toth and Vigo, 2002] Paolo Toth and Daniele Vigo. *The vehicle routing problem*. SIAM, 2002.

[Turmo *et al.*, 2018] Vincent Turmo, Mahour Rahimi, Eric J Gonzales, and Price Armstrong. Evaluating potential demand and operational effects of coordinated Americans with disabilities act paratransit and taxi service. *Transportation Research Record*, 2672(8):686–697, 2018.

[VRoom Project, 2021] VRoom Project. VRoom: Vehicle routing open-source optimization machine. http://vroom-project.org/, 2021.

[Wen *et al.*, 2018] Jian Wen, Yu Xin Chen, Neema Nassir, and Jinhua Zhao. Transit-oriented autonomous vehicle operation with integrated demand-supply interaction. *Transportation Research Part C: Emerging Technologies*, 97:216–234, 2018.

[Wilbur *et al.*, 2022] Michael Wilbur, Salah U Kadir, Youngseo Kim, Geoffrey Pettet, Ayan Mukhopadhyay, Philip Pugliese, Samitha Samaranayake, Aron Laszka, and Abhishek Dubey. An online approach to solve the dynamic vehicle routing problem with stochastic trip requests for paratransit services. In *13th ACM/IEEE International Conf. on Cyber-Physical Systems*, 2022.

[Yu *et al.*, 2019] James JQ Yu, Wen Yu, and Jiatao Gu. Online vehicle routing with neural combinatorial optimization and deep reinforcement learning. *IEEE Trans. on Intelligent Transp. Systems*, 20(10):3806–3817, 2019.

Published in the proceedings of the 31st International Joint Conference on Artificial Intelligence (IJCAI 2022).

A Notation

Table 1: List of Symbols

Symbol	Description
Offline Vehicle Routing Problem (VRP)	
$\boldsymbol{L}$	set of locations
$L^{depot} \in \boldsymbol{L}$	location of the vehicle depot (i.e., garage)
$\boldsymbol{T}$	ordered set of trip requests ($\boldsymbol{T} = \langle T_1, T_2, \ldots \rangle$)
$L_i^{pickup} \in \boldsymbol{L}$	pickup location of trip request $T_i \in \boldsymbol{T}$
$L_i^{dropoff} \in \boldsymbol{L}$	drop-off location of trip request $T_i \in \boldsymbol{T}$
P_i	passenger occupancy of trip request $T_i \in \boldsymbol{T}$ (i.e., number of passengers to be transported)
$\boldsymbol{w}$	ordered set of pickup time windows ($\boldsymbol{w} = \langle w_1, w_2, \ldots \rangle$; $\lvert\boldsymbol{T}\rvert = \lvert\boldsymbol{w}\rvert$)
w_i^{start}	start time of pickup window $w_i \in \boldsymbol{w}$ (i.e., earliest pickup time)
w_i^{end}	end time of pickup window $w_i \in \boldsymbol{w}$ (i.e., latest pickup time)
D^{dwell}	dwell time for pickup and dropoff
$D^{maxroute}$	maximum duration of a vehicle route
D_{l_1,l_2}^{travel}	time to drive from location $l_1 \in \boldsymbol{L}$ to location $l_2 \in \boldsymbol{L}$
V	vehicle passenger capacity (i.e., maximum number of passengers on a vehicle at a time)
$C^{nroutes}$	cost factor for the number of routes in the objective function
Offline VRP Solution	
$\boldsymbol{R}$	set of vehicle routes ($\boldsymbol{R} = \langle R_1, R_2, \ldots \rangle$)
R_i^{start}	start time of route $R_i \in \boldsymbol{R}$ (i.e., time when vehicle leaves the depot)
R_i^{end}	end time of route $R_i \in \boldsymbol{R}$ (i.e., time when vehicle returns to the depot)
$\mathcal{R}(\boldsymbol{T}, \boldsymbol{w})$	set of feasible solution for the VRP instance $(\boldsymbol{T}, \boldsymbol{w})$
$\mathcal{C}(\boldsymbol{R})$	total cost of the VRP solution $\boldsymbol{R}$
$VRP^*(\boldsymbol{T}, \boldsymbol{w})$	total cost of an optimal solution for the VRP instance $(\boldsymbol{T}, \boldsymbol{w})$ (i.e., $VRP^*(\boldsymbol{T}, \boldsymbol{w}) = \min_{\boldsymbol{R} \in \mathcal{R}(\boldsymbol{T}, \boldsymbol{w})} \mathcal{C}(\boldsymbol{R})$)
Online Bookings Problem	
$\boldsymbol{W}$	ordered set of broad pickup time windows ($\boldsymbol{W} = \langle W_1, W_2, \ldots \rangle$; $\lvert\boldsymbol{W}\rvert = \lvert\boldsymbol{T}\rvert$)
W_i^{start}	start time of window $W_i \in \boldsymbol{W}$ (i.e., earliest pickup time)
W_i^{end}	end time of window $W_i \in \boldsymbol{W}$ (i.e., latest pickup time)
$\mathcal{D}$	probability distribution of $(\boldsymbol{T}, \boldsymbol{W})$
D^{window}	maximum duration of a tight pickup window
μ^*	optimal decision policy for the online bookings problem
Solution Approach	
α	anytime algorithm for solving an offline VRP with supporting input $\hat{\boldsymbol{R}}^{(i)}$
$\boldsymbol{R}^{(i)}$	feasible solution (i.e., set of routes) output by the ith execution of the anytime algorithm α
$\hat{\boldsymbol{R}}^{(i)}$	feasible solution (i.e., set of routes) output by the ith execution of the decision policy μ
c_i	immediate cost incurred after the ith decision (MDP formulation)
$\tilde{c}_i$	shaped immediate cost incurred after the ith decision
Q	value function for predicting cost $\tilde{c}_i$ for a state-action pair (i.e., for a decision input and decision)

B Offline Vehicle Routing Problem with Time Windows

In Section 2.1, due to lack of space, we provided a brief description of the *offline vehicle routing problem with time windows*, which was sufficient for formulating the online bookings problem. Here, we provide a complete and formal description of the offline VRP problem. Please note that the online bookings problem can be defined with respect to a range of offline VRP variants that consider pickup time windows, and our proposed solution approach could incorporate any offline VRP solver for these problems.

Notation

Throughout the description of the model, we use $\mathbb{R}$ to denote the set of real numbers, $\mathbb{N}$ to denote the set of natural numbers, $\boldsymbol{L}$ to denote the set of locations, and $D^{travel}_{l_1,l_2}$ to denote the travel time between locations $l_1 \in \boldsymbol{L}$ and $l_2 \in \boldsymbol{L}$. We assume that points in time as well as time durations are represented by real numbers.

Input

The input of the offline VRP problem is

- an ordered set of *trip requests* $\boldsymbol{T} = \langle T_1, T_2, \ldots, T_n \rangle$, where each trip request T_i contains a pickup location $L^{pickup}_i \in \boldsymbol{L}$, a drop-off location $L^{dropoff}_i \in \boldsymbol{L}$, and the number of passengers to be transported P_i;
- a corresponding ordered set of *tight pickup time windows* $\boldsymbol{w} = \langle w_1, w_2, \ldots, w_n \rangle$, where each time window w_i is defined by an earliest $w^{start}_i \in \mathbb{R}$ and latest $w^{end}_i \in \mathbb{R}$ pickup time;
- the maximum allowed duration $D^{maxroute} \in \mathbb{R}$ of a vehicle route (e.g., maximum length of a driver's shift);
- the passenger capacity $V \in \mathbb{N}$ of a vehicle (i.e., maximum number of passengers on board at a time);
- the dwell time $D^{dwell} \in \mathbb{R}$ at the pickup and drop-off locations;
- the location $L^{depot} \in \boldsymbol{L}$ of the vehicle depot (i.e., garage);
- the cost factor $C^{nroutes}$ for the number of routes in the objective function.

For ease of presentation, we represent a VRP instance as $(\boldsymbol{T}, \boldsymbol{w})$, assuming that the constants are provided implicitly.

Solution Representation

A solution to the offline VRP problem is a set of *vehicle routes* $\boldsymbol{R} = \{R_1, R_2, \ldots, R_m\}$. Informally, each vehicle route is a list of locations (pickup or drop-off) with associated arrival times (i.e., when the vehicle arrives at the location to pick up or drop off passengers). Formally, a vehicle route R_i is an ordered set of tuples $\langle l, t \rangle \in \boldsymbol{L} \times \mathbb{R}$, where l is either a pickup L^{pickup}_j or drop-off location $L^{dropoff}_j$, and t is the time when the vehicle on route R_i arrives at location l. Note that in Section 2.1, we represented a vehicle route as an ordered set of pickup and drop-off locations for ease of exposition. While we could use that representation here, it will actually be easier to use this representation to provide a complete formal definition of the offline VRP problem.

Constraints

A set of vehicle routes $\boldsymbol{R}$ is a feasible solution to the offline VRP problem $(\boldsymbol{T}, \boldsymbol{w})$ if it satisfies the following set of constraints.

First, each trip $T_i \in \boldsymbol{T}$ is picked up by at most one vehicle route $R_j \in \boldsymbol{R}$:

$$\forall T_i \in \boldsymbol{T}, R_j \in \boldsymbol{R}, t_j \in \mathbb{R}, R_k \in \boldsymbol{R}, t_k \in \mathbb{R} :$$
$$\langle L^{pickup}_i, t_j \rangle \in R_j \wedge \langle L^{pickup}_i, t_k \rangle \in R_k \Rightarrow j = k \wedge t_j = t_k$$

In other words, if trip T_i is picked up by route R_j at t_j, then no other route R_k can pick up this trip (and neither can this route R_j at any other time t_k). Similarly, each trip $T_i \in \boldsymbol{T}$ is dropped off by at most one route $R_j \in \boldsymbol{R}$:

$$\forall T_i \in \boldsymbol{T}, R_j \in \boldsymbol{R}, t_j \in \mathbb{R}, R_k \in \boldsymbol{R}, t_k \in \mathbb{R} :$$
$$\langle L^{dropoff}_i, t_j \rangle \in R_j \wedge \langle L^{dropoff}_i, t_k \rangle \in R_k \Rightarrow j = k \wedge t_j = t_k$$

Second, each trip $T_i \in \boldsymbol{T}$ is served by at least one route $R_j \in \boldsymbol{R}$ such that the passengers are picked up by the vehicle within the time window w_i and dropped off on time, by $w^{end}_i + D^{travel}_{L^{pickup}_i, L^{dropoff}_i}$ at latest:

$$\begin{aligned}\forall T_i \in \boldsymbol{T} :\quad & \exists R_j \in \boldsymbol{R}, t^{pickup} \in \mathbb{R}, t^{dropoff} \in \mathbb{R} \wedge \Big(\\ & \quad \langle L^{pickup}_i, t^{pickup} \rangle, \langle L^{dropoff}_i, t^{dropoff} \rangle \in R_j \\ & \quad \wedge\, w^{start}_i \le t^{pickup} \le w^{end}_i \\ & \quad \wedge\, t^{pickup} < t^{dropoff} \\ & \quad \wedge\, t^{dropoff} \le w^{end}_i + D^{travel}_{L^{pickup}_i, L^{dropoff}_i} \Big)\end{aligned}$$

In other words, for each trip request $T_i \in \boldsymbol{T}$, there exists a vehicle run $R_j \in \boldsymbol{R}$ that picks up the passengers at some time $t^{pickup} \in \mathbb{R}$ and drops them off at some time $t^{dropoff} \in \mathbb{R}$ (second and third lines), the pickup time t^{pickup} is within the pickup window w_i (fourth line), the pick time t^{pickup} is earlier thn the drop-off time $t^{dropoff}$ (fifth line), and the drop-off time $t^{dropoff}$ is no later than $w^{end}_i + D^{travel}_{L^{pickup}_i, L^{dropoff}_i}$ (sixth line). Note that the last clause ensures that passengers arrive at their dropoff location $L^{dropoff}_i$ no later than if they were picked up at the latest possible time w^{end}_i and drove to the dropoff location without any detours. Our formulation and algorithms could very easily be extended to consider detour times (defined with respect to pickup times). In our experiments, we consider the above formulation since it captures the requirements of the transit agency.

Third, every vehicle route $R_j \in \boldsymbol{R}$ satisfies travel-time and dwell-time constraints:

$$\begin{aligned}\forall R_j \in \boldsymbol{R}, l_1 \in \boldsymbol{L}, l_2 \in \boldsymbol{L}, t_1 \in \mathbb{R}, t_2 \in \mathbb{R} :& \\ \langle l_1, t_1 \rangle, \langle l_2, t_2 \rangle \in R_j \wedge t_1 < t_2 & \\ \Rightarrow t_1 + D^{dwell} + D^{travel}_{l_1,l_2} \le t_2 &\end{aligned}$$

In other words, if vehicle route $R_j \in \boldsymbol{R}$ arrives at location $l_1 \in \boldsymbol{L}$ at time $t_1 \in \mathbb{R}$ and later arrives at location $l_2 \in \boldsymbol{L}$ at time $t_2 \in \mathbb{R}$, then the time difference between t_1 and t_2

must be at least D^{dwell} (time required to pick up or drop off passengers at location l_1) plus $D^{travel}_{l_1,l_2}$ (time required to drive from location l_1 to l_2). Note that if the constraint is satisfied for consecutive locations, then it is also satisfied for non-consecutive ones due to the triangle inequality, so the above constraint could be reformulated to consider only consecutive locations. We use the above formulation for the sake of simplicity.

Fourth, the duration of each vehicle route $R_j \in \boldsymbol{R}$ is at most $D^{maxroute}$, where the duration of a route is the time between leaving the garage (i.e., vehicle depot) and returning to it. We introduce this constraint based on input from the transit agency, which stores all vehicles at a garage overnight, and has strict constraints on the duration of the routes due to the drivers' labour contracts. To express this, we define the start time R^{start}_j of route R_j as the time when the vehicle needs to leave the garage to serve its first trip:

$$R^{start}_j = \min_{\langle l,t\rangle \in R_j} t - D^{travel}_{L^{depot},l}$$

Similarly, we define the end time R^{end}_j of route R_j as the time when the vehicle can arrive at the garage after serving its last trip:

$$R^{end}_j = \max_{\langle l,t\rangle \in R_j} t + D^{dwell} + D^{travel}_{l,L^{depot}}$$

Then, we can formulate the constraint on the duration of vehicle routes as follows:

$$\forall R_j \in \boldsymbol{R}: \quad R^{end}_j \leq R^{start}_j + D^{maxroute}$$

Finally, the number of passengers on board a vehicle does not exceed the passenger capacity V of a vehicle at any time. When vehicle route $R_j \in \boldsymbol{R}$ picks up passengers at a pickup location L^{pickup}_i, the occupancy of the vehicle serving route R_j is incremented by the number of passengers P_i. Similarly, when route $R_j \in \boldsymbol{R}$ drops off passengers at a drop-off location $L^{dropoff}_i$, the occupancy of the vehicle serving route R_j is decreased by the number of passengers P_i. For each route $R_j \in \boldsymbol{R}$, the occupancy of the vehicle serving the route at any time $t \in \mathbb{R}$ is less than or equal to the passenger capacity V:

$$\forall R_j \in \boldsymbol{R}, t \in \mathbb{R}:$$
$$\sum_{T_i \in \boldsymbol{T}, t' \in \mathbb{R}:\ \langle L^{pickup}_i, t'\rangle \in R_j \wedge t' < t} P_i - \sum_{T_i \in \boldsymbol{T}, t' \in \mathbb{R}:\ \langle L^{dropoff}_i, t'\rangle \in R_j \wedge t' < t} P_i \leq V$$

Note that the first summation adds up all the passengers who have been picked up before (at some time $t' < t$), while the second summation adds up all the passengers who have been dropped up before (at some time $t' < t$). Hence, their difference is the number of passengers on board the vehicle at time t.

Objective

We define the objective of the offline VRP as minimizing the total cost of the vehicle routes $\boldsymbol{R}$, which depends on the duration and number of vehicle routes. Formally, we define the total cost $\mathcal{C}(\boldsymbol{R})$ of a set of vehicle routes $\boldsymbol{R}$ as follows:

$$\mathcal{C}(\boldsymbol{R}) = \sum_{R_j \in \boldsymbol{R}} (R^{end}_j - R^{start}_j) + C^{nroutes} \cdot |\boldsymbol{R}| \qquad (1)$$

where $C^{nroutes}$ is a cost factor that captures the constant "overhead" costs associated with each vehicle route. In practice, an 8-hour vehicle route does not cost eight times as much as a 1-hour vehicle route since there are constant costs associated with the route (e.g., preparing a vehicle for the drive, bringing in a driver, or markup for outsourcing). In fact, very short routes (e.g., 20 minutes) may be prohibitively uneconomical in practice. The second term of the offline VRP objective enables capturing this.

C Features

In this section, we provide a detailed description of the features that the value function Q takes as input, which we briefly introduced in Section 3.2. Please recall that the values of these features are defined for a particular state (i.e., decision input) $(\langle T_1, \ldots, T_i \rangle, \langle w_1, \ldots, w_{i-1} \rangle, W_i, \boldsymbol{R}^{(i-1)})$ and particular action (i.e., decision) $(w_i, \hat{\boldsymbol{R}}^{(i)})$.

First, we describe features based on the probability distribution $\mathcal{D}$, which capture our expectations of future requests.

Busyness ($\mathcal{BN}$) These features consider how busy certain locations and certain times of the day are, in terms of the number of expected trip requests around those locations and times of day. Specifically, they consider the expected number of daily trip requests whose pickup locations L_j^{pickup} are in the same geographical area as the pickup location L_i^{pickup} of request T_i (e.g., in the same ZIP-code area in the U.S.) and/or whose drop-off locations $L_j^{dropoff}$ are in the same geographical area as the drop-off location $L_i^{dropoff}$ and/or whose broad pickup windows start around the same time W_j^{start} as the narrow window w_i^{start} (e.g., within the same 1-hour interval). In other words, these features consider the expected number of trip requests received for a day that satisfy some of the following criteria:

- same geographical area as the pickup location L_i^{pickup},
- same geographical area as the drop-off location $L_i^{dropoff}$,
- similar time as the tight pickup windows w_i.

Based on the above three criteria, we define four variants of the *busyness feature*:

- $\mathcal{BN}(\mathcal{D}, L_i^{pickup}, L_i^{dropoff}, w_i)$: expected number of daily trip requests that travel from the geographical area of L_i^{pickup} to the geographical area of $L_i^{dropoff}$ and whose broad pickup windows start around the same time as w_i.
- $\mathcal{BN}(\mathcal{D}, L_i^{pickup}, w_i)$: expected number of daily trip requests that travel from the geographical area of L_i^{pickup} and whose broad pickup windows start around the same time as w_i.
- $\mathcal{BN}(\mathcal{D}, L_i^{dropoff}, w_i)$: expected number of daily trip requests that travel to the geographical area of $L_i^{dropoff}$ and whose broad pickup windows start around the same time as w_i.
- $\mathcal{BN}(\mathcal{D}, w_i)$: expected number of daily trip requests whose broad pickup windows start around the same time as w_i.

All of the above features can be estimated based on historical data (i.e., using an empirical distribution for $\mathcal{D}$) for any given state-action pair. Note that the distribution of requests $\mathcal{D}$ may vary significantly between days (e.g., weekends are typically less busy than weekdays); hence, we can use a different $\mathcal{D}$ depending on the day of week.

Expected Requests ($\mathcal{ER}$) While the above features consider the expected number of requests received in a whole day, it is also helpful to consider how many *more* requests we expect to receive for the day. To capture this, we introduce the *expected requests* $\mathcal{ER}(\mathcal{D}, i)$ feature, which is the expected value of $|\boldsymbol{T}| - i$ when making the ith decision. Note that in practice, this feature can also depend on the day of week since the distribution $\mathcal{D}$ varies among the days. Further, since requests do not arrive at the same rate throughout the day (i.e., during some hours of the day, the agency receives many more calls to book trips than during other hours), we also consider the time of day when the booking call is received to estimate $\mathcal{ER}$ (based again on historical data, in this case the rate at which booking calls are received throughout the day).

State Features

Finally, besides considering future requests, we must also consider trip requests that we have already been received. To this end, we introduce three features that capture how well a decision $(w_i, \hat{\boldsymbol{R}}^{(i)})$ fits the previously selected tight windows $\langle w_1, \ldots, w_{i-1} \rangle$ and vehicle routes $\boldsymbol{R}^{(i-1)}$:

- *Time increase* $\mathcal{TI}(\hat{\boldsymbol{R}}^{(i)}, \boldsymbol{R}^{(i-1)})$: increase in the duration of the routes, compared between $\hat{\boldsymbol{R}}^{(i)}$ and $\boldsymbol{R}^{(i-1)}$:

$$\mathcal{DI}(\boldsymbol{R}^{(i-1)}, \hat{\boldsymbol{R}}^{(i)}) = \sum_{R_j \in \hat{\boldsymbol{R}}^{(i)}} \left(R_j^{end} - R_j^{start}\right) - \sum_{R_j \in \boldsymbol{R}^{(i-1)}} \left(R_j^{end} - R_j^{start}\right)$$

- *Distance increase* $\mathcal{DI}(\hat{\boldsymbol{R}}^{(i)}, \boldsymbol{R}^{(i-1)})$: increase in the driving distances of the routes, compared between $\hat{\boldsymbol{R}}^{(i)}$ and $\boldsymbol{R}^{(i-1)}$ (i.e., same as $\mathcal{TI}$, but considering distance driven instead of time spent).
- *Tightness of schedule* $\mathcal{TS}(\hat{\boldsymbol{R}}^{(i)})$: tightness of the route schedule captures how well trip T_i fits into route R_j, where R_j is the route serving trip T_i in solution $\hat{\boldsymbol{R}}^{(i)}$. The rationale behind this feature is to express how "fragmented" a route schedule is, i.e., how much waiting time there is between consecutive trips, which is not long enough to allow serving another trip, but long enough to significantly increase route duration. To capture this, we consider the amount of waiting time x between trip T_i and the preceding trip on route R_j (i.e., waiting time before trip T_i on route R_j), and the amount of waiting time y between trip T_i and the following trip (i.e., waiting time after serving trip T_i), where the waiting time between two consecutive trips is defined as the amount of time between dropping off all passengers from the previous trip and needing to leave for the next trip. Note that when consecutive trips are interleaved, waiting time is defined to be zero. Then, we can formulate the *tightness of schedule* feature as $\mathcal{TS}(\hat{\boldsymbol{R}}^{(i)}) = \frac{|x-y|}{|x+y|}$. By maximizing this feature, we ensure that time gaps in the route schedule are either minimized (to avoid waiting) or maximized (so that another trip can be served in the gap).

D Simulated-Annealing and Greedy Algorithms

We first provide a high-level overview of the simulated-annealing and greedy algorithms in Appendix D.1 and then describe them in detail in Appendix D.2 and Appendix D.3, respectively.

D.1 Overview of Algorithms

Simulated Annealing

First, we introduce a simulated annealing algorithm α^{SimAnn}, which improves upon a given feasible solution $\hat{\boldsymbol{R}}^{(i)}$ using an iterative random search. We provide a detailed description of this algorithm in Appendix D. In each iteration, the algorithm tries to improve upon the current solution $\boldsymbol{R}$ by generating a random neighbor $\boldsymbol{R}'$ of the current solution $\boldsymbol{R}$ using two operations. The first one, called *Swap*, randomly chooses two vehicle routes $R_x, R_y \in \boldsymbol{R}$ that overlap in time, and tries to swap a pair of randomly chosen trip request T_i, T_j (where $L_i^{pickup} \in R_x$, $L_j^{pickup} \in R_y$) between the two routes. The second one, called *SplitAndMerge*, also chooses two overlapping routes $R_x, R_y \in \boldsymbol{R}$ at random, but then splits each route into two halves (earlier trips in the first half, later ones in the second) and tries to merge the first half of R_x with the second half of R_y and vice versa. In each iteration, the algorithm repeatedly applies these operations to the current solution $\boldsymbol{R}$ to obtain a feasible random neighbor $\boldsymbol{R}'$. Whether this random neighbor replaces the current solution ($\boldsymbol{R} \leftarrow \boldsymbol{R}'$) or if it is discarded is decided at random, with a probability that depends on $\mathcal{C}(\boldsymbol{R}) - \mathcal{C}(\boldsymbol{R}')$. When terminated, the algorithm returns the best feasible solution that it has encountered during the search as the VRP solution $\boldsymbol{R}^{(i)}$.

Greedy Algorithm

To enhance the practical performance of our approach, we also introduce a greedy algorithm α^{Greedy} for solving the paratransit VRP formulation. While this is not an anytime algorithm, its running time is low enough so that we can successfully execute it between most consecutive requests. The solution output by the greedy algorithm α^{Greedy} can then be fed into the simulated annealing α^{SimAnn} (if it is better than the current solution). This algorithm also follows an iterative approach: starting with an empty solution $\boldsymbol{R} = \emptyset$, it adds a new routes to the solution one-by-one. For each route, it starts with an empty set of requests $R = \emptyset$, and tries to assign unserved requests to this route one-by-one, always choosing one that minimizes a heuristic cost function, until there are no feasible assignments or the minimum cost exceeds a threshold. We provide a detailed description of this algorithm in Appendix D.

D.2 Simulated Annealing

The simulated-annealing algorithm follows an iterative process. In each iteration, the algorithm obtains a random neighboring solution $\boldsymbol{R}'$ of the current feasible solution $\boldsymbol{R}$ using **RandomNeighbor**. If the total cost ($\mathcal{C}$) of $\boldsymbol{R}'$ is lower than the total cost of $\boldsymbol{R}$, then the algorithm always accepts $\boldsymbol{R}'$ as the new current solution. Otherwise, the algorithm computes the probability *AcceptProbability* of accepting $\boldsymbol{R}'$ based on the cost difference between $\boldsymbol{R}'$ and $\boldsymbol{R}$ a decreasing temperature value H^t, and then accepts $\boldsymbol{R}'$ at random. Initially, the temperature H^t and probability values are very high (i.e., we initialize $H^0 = H^{start}$ with a very high value), which helps the local search to avoid "getting stuck" in local optima; over time, the temperature and probability values decrease (i.e., H^t decreases as t increases), enabling the search to converge to an optimum. The algorithm terminates after its total running time ($t^{current} - t^{start}$) exceeds the configured maximum running time t^{run}, and returns the best solution found up to that point.

Algorithm 1: Simulated Annealing($\boldsymbol{R}, t_{run}, p_{start}, p_{end}, p_{alter}, L^{depot}, D^{dwell}, D^{maxroute}, V$)

```
$Solutions \leftarrow \{\boldsymbol{R}\}$
$H^{start} \leftarrow \frac{-1}{\ln p_{start}}$
$H^{end} \leftarrow \frac{-1}{\ln p_{end}}$
$H^{rate} \leftarrow \left(\frac{H^{end}}{H^{start}}\right)^{\frac{1}{t_{run}-1}}$
$H^t \leftarrow H^{start}$
$\delta_{avg} \leftarrow 0$
$t^{start} \leftarrow$ GetCurrentTime()
$t^{current} \leftarrow t^{start}$
while $t^{current} - t^{start} \le t_{run}$ do
    $\boldsymbol{R}' \leftarrow$ RandomNeighbor($\boldsymbol{R}, p_{alter}$)
    $\delta_e \leftarrow \mathcal{C}(\boldsymbol{R}') - \mathcal{C}(\boldsymbol{R})$
    if $t^{current} - t^{start} = 1$ then
        $\delta_{avg} \leftarrow \delta_e$
    end
    $AcceptProbability \leftarrow \exp\left(\frac{-\delta_e}{\delta_{avg} \cdot H^t}\right)$
    if $\mathcal{C}(\boldsymbol{R}') < \mathcal{C}(\boldsymbol{R})$ or $AcceptProbability >$
     UniformRandom($[0,1]$) then
        $\boldsymbol{R} \leftarrow \boldsymbol{R}'$
        $\delta_{avg} \leftarrow \delta_{avg} + \frac{\delta_e - \delta_{avg}}{|Solutions|}$
        $Solutions \leftarrow Solutions \cup \{\boldsymbol{R}\}$
    end
    $t^{current} \leftarrow$ GetCurrentTime()
    $H^t \leftarrow H^{start} \cdot H^{rate\{t^{current} - t^{start}\}}$
end
$\boldsymbol{R}^* \leftarrow \mathrm{argmin}_{\boldsymbol{R}' \in Solutions}\, \mathcal{C}(\boldsymbol{R}')$
Result: $\boldsymbol{R}^*$
```

Algorithm 2: RandomNeighbor($\boldsymbol{R}, p_{alter}, L^{depot}, D^{dwell}, D^{maxroute}, V$)

```
$NumberOfAlterations \leftarrow$ max$\{1, |\boldsymbol{R}| \cdot p_{alter}\}$
for $1, \ldots, NumberOfAlterations$ do
    $operation \leftarrow$
     UniformRandom($[Swap, SplitAndMerge]$)
    $\boldsymbol{R} \leftarrow operation(\boldsymbol{R})$
end
Result: $\boldsymbol{R}$
```

Algorithm 2 follows an iterative process: in each iteration, the algorithm randomly chooses one operation from *Swap* and *SplitAndMerge*, and modifies the input solution $\boldsymbol{R}$. The detailed descriptions of the two operations is in the next paragraphs. The running time of this algorithm is $\mathcal{O}\left(|\boldsymbol{T}|^4\right)$.

Algorithm 3: Swap$(\boldsymbol{R}, L^{depot}, D^{dwell}, D^{maxroute}, V)$

```
$R_1, R_2 \leftarrow$ UniformRandom$(\boldsymbol{R})$
$T_1 \leftarrow$ UniformRandom$(R_1)$
$T_2 \leftarrow$ UniformRandom$(R_2)$
$R_1^{'}, SolCost_1 \leftarrow$
  Feasible$(R_1, T_2, L^{depot}, D^{dwell}, D^{maxroute}, V)$
$R_2^{'}, SolCost_2 \leftarrow$
  Feasible$(R_2, T_1, L^{depot}, D^{dwell}, D^{maxroute}, V)$
if $SolCost_1 \neq \infty \wedge SolCost_2 \neq \infty$ then
    $\boldsymbol{R} \leftarrow \boldsymbol{R} \setminus \{R_1, R_2\} \cup \{R_1^{'}, R_2^{'}\}$
end
Result: $\boldsymbol{R}$
```

Algorithm 3 randomly chooses two vehicle routes $R_1, R_2 \in \boldsymbol{R}$ that overlap in time, and tries to swap a pair of randomly chosen trip request T_1, T_2 (where $L_1^{pickup} \in R_1$, $L_2^{pickup} \in R_2$) between the two routes. If the swap is feasible (i.e., satisfies time and occupancy constraints), then update $\boldsymbol{R}$ with the newly modified routes (R_1', R_2') and remove the initial routes (R_1, R_2). Finally, return the updated routes $\boldsymbol{R}$. The time complexity of this algorithm is $\mathcal{O}\left(|\boldsymbol{T}|^3\right)$.

Algorithm 4: SplitAndMerge$(\boldsymbol{R}, L^{depot}, D^{dwell}, D^{maxroute}, V)$

```
$R_1, R_2 \leftarrow$ UniformRandom$(\boldsymbol{R})$
$R_1^1, R_1^2 \leftarrow$ SplitRuns$(R_1)$
$R_2^1, R_2^2 \leftarrow$ SplitRuns$(R_2)$
success$_1$, $R_1^{'} \leftarrow$
  MergeRuns$(R_1^1, R_2^2, L^{depot}, D^{dwell}, D^{maxroute}, V)$
success$_2$, $R_2^{'} \leftarrow$
  MergeRuns$(R_2^1, R_1^2, L^{depot}, D^{dwell}, D^{maxroute}, V)$
if $success_1 \wedge success_2$ then
    $\boldsymbol{R} \leftarrow \boldsymbol{R} \setminus \{R_1, R_2\} \cup \{R_1^{'}, R_2^{'}\}$
end
Result: $\boldsymbol{R}$
```

Algorithm 4 chooses two overlapping routes $R_1, R_2 \in \boldsymbol{R}$ at random, and then splits each route into two halves (earlier trips in the first half R_i^1, later ones in the second half R_i^2). Then, the algorithm obtains the merged route R_1' by merging R_1^1 and R_2^2, and similarly obtains the merged route R_2' by merging R_2^1 and R_1^2. If both merged routes are feasible, then update $\boldsymbol{R}$ with newly modified routes (R_1', R_2') and remove the initial routes (R_1, R_2). Finally, return the updated routes $\boldsymbol{R}$. The algorithm **MergeRuns** has the time complexity of $\mathcal{O}\left(|\boldsymbol{T}|^4\right)$. Accordingly, the time complexity of this algorithm is $\mathcal{O}\left(|\boldsymbol{T}|^4\right)$.

D.3 Greedy Algorithm

Inputs

In our VRP formulation (Appendix B), we defined time windows only for pickups since the latest drop-off time is implicitly defined by the latest pickup time and the travel time for each trip request. Here, we present a greedy algorithm that can solve a more general VRP formulation, in which time windows are defined for both pickups and drop-offs. Specifically, we present a greedy algorithm that takes as input—in addition to the inputs defined in Appendix B—an ordered set of drop-off time windows $\boldsymbol{dw}$, where dw_i^{start} is the earliest time that passengers from trip T_i can be dropped off and dw_i^{end} is the latest time that they can be dropped off. Then, we can solve instances of our problem formulation from Appendix B by letting $dw_i^{start} = w_i^{start} + D^{travel}_{L_i^{pickup}, L_i^{dropoff}}$ and $dw_i^{end} = w_i^{end} + D^{travel}_{L_i^{pickup}, L_i^{dropoff}}$ for each trip request T_i.

Outputs

The output of the algorithm is a set of routes $\boldsymbol{R}$, as described in Appendix B. To facilitate the description of our algorithms, we extend this solution representation with additional fields. First, we let the term *node* refer to a pair $\langle l, w \rangle$, where $l \in \boldsymbol{L}$ is a location and $w \in \boldsymbol{w} \cup \boldsymbol{dw}$ is a time window.

We let each route $R_k \in \boldsymbol{R}$ consist of the following:

- Y_{R_k}: list of nodes served by route R_k. We let $Y_{R_k}^{first}$ denote the first node in Y_{R_k} and let $Y_{R_k}^{last}$ denote the last node in Y_{R_k}.
- D_{R_k}: list of points in time $(|D_{R_k}| = |Y_{R_k}|)$, where each element represents the time when route R_k reaches the location of the corresponding node of Y_{R_k}. We let $D_{R_k}^{first}$ denote the first time point in D_{R_k} and let $D_{R_k}^{last}$ denote the last time point in D_{R_k}.
- O_{R_k}: list of integers $(|O_{R_k}| = |Y_{R_k}|)$, where each element represents the occupancy of the vehicle (i.e., number of passengers on board) when route R_k reaches the location of the corresponding node of Y_{R_k}.

Algorithm 5 identifies the time-feasible placements of a new node $\langle l_n, w_n \rangle$ into the existing list of nodes Y_{R_i} of route R_i (i.e., positions at which at the new node $\langle l_n, w_n \rangle$ could be inserted into the list Y_{R_i} without violating any time constraints). The implementation of the algorithm considers three cases:

- First case: the algorithm checks whether the new node $\langle l_n, w_n \rangle$ can be placed before the first node of route R_i. In other words, it checks whether the vehicle route R_i can reach location $l_k \in \boldsymbol{L}$ from location $l_n \in \boldsymbol{L}$ without violating any travel-time and time-window constraints.
- Second case: the algorithm checks for each pair of consecutive nodes $Y_{R_i}[k]$ and $Y_{R_i}[k+1]$ whether it can place the new node $\langle l_n, w_n \rangle$ between nodes $Y_{R_i}[k]$ and $Y_{R_i}[k+1]$. In other words, it checks whether vehicle route R_i can reach location $l_n \in \boldsymbol{L}$ from location $l_k \in \boldsymbol{L}$ and reach location $l_{k+1} \in \boldsymbol{L}$ from location $l_n \in \boldsymbol{L}$ without violating any travel-time and time-window constraints.

Algorithm 5: GetPlacements$(R_i, l_n, w_n, \textit{index}, D^{\textit{dwell}})$

```
$Placements \leftarrow \emptyset$
for $\langle l_k, w_k \rangle \in Y_{R_i}$ do
    if $k \geq \textit{index}$ then
        if $k = 0 \wedge$ Reachable$(l_n, w_n, l_k, w_k, D^{\textit{dwell}})$ then
            $Placements \leftarrow Placements \cup \{0\}$
        end
        if $k < |Y_{R_i}| - 1$ then
            $l_{k+1}, w_{k+1} \leftarrow Y_{R_i}[k+1]$
            if Reachable$(l_k, w_k, l_n, w_n, D^{\textit{dwell}}) \wedge$ Reachable$(l_n, w_n, l_{k+1}, w_{k+1}, D^{\textit{dwell}})$ then
                $Placements \leftarrow Placements \cup \{k+1\}$
            end
        else if $k = |Y_{R_i}| - 1$ then
            if Reachable$(l_n, w_n, l_k, w_k, D^{\textit{dwell}})$ then
                $Placements \leftarrow Placements \cup \{k+1\}$
            end
        end
    end
end
Result: Placements
```

- Third case: The algorithm checks whether the new node $\langle l_n, w_n \rangle$ can be placed as the last node into the route R_i. In other words, it checks whether vehicle route R_i can reach location $l_n \in \boldsymbol{L}$ from the location $l_k \in \boldsymbol{L}$ without violating any travel-time and time-window constraints.

The running time of this algorithm is $\mathcal{O}(|\boldsymbol{T}|)$.

Algorithm 6 checks whether each request $T_j \in \boldsymbol{T}$ can be inserted into the given route R_i based on existing nodes (Y_{R_i}) in the route R_i. This algorithm considers two cases; first, when the route is empty (i.e., no nodes exist for the route), this case is trivial. The algorithm adds the request by just appending the pickup node $\langle L_j^{\textit{pickup}}, w_j \rangle$ and drop-off node $\langle L_j^{\textit{dropoff}}, dw_j \rangle$ without checking any time constraints and capacity constraints. Next, when the route is not empty, the algorithm first obtains all the possible placements for the pickup node $\langle L_j^{\textit{pickup}}, w_j \rangle$. If it can obtain at least one placement, then the algorithm tries to obtain all the possible placements for drop-off node $\langle L_j^{\textit{dropoff}}, dw_j \rangle$. If it has feasible placements for both the pickup and drop-off nodes, then the algorithm checks the feasibility of each pair of placement values using **Adjust**. If the assignment is feasible (i.e., the *SolCost* $\neq \infty$), then the algorithm computes threshold value using the function **GThreshold**. We can formally express the function **GThreshold** as follows:

$$GThreshold(\textit{ratio}, R_i') = p_{const} + p_{ratio-thres} \cdot \textit{ratio} + p_{length} \cdot ((R_i')^{\text{end}} - (R_i')^{\textit{start}}) \quad (2)$$

If the *SolCost* is less than the threshold value and less than the current minimum cost, then update the current minimum cost with the *SolCost* and update the best-updated route

Algorithm 6: Feasible$(R_i, T_j, \textit{ratio}, L^{\textit{depot}}, D^{\textit{dwell}}, D^{\textit{maxroute}}, V)$

```
$success \leftarrow False$
$(R_i^{'})^{\textit{best}} \leftarrow R_i$
$\textit{MinCost} \leftarrow \infty$
if $|Y_{R_i}| = 0$ then
    $(R_i^{'})^{\textit{best}}, \textit{MinCost} \leftarrow$
        Adjust$(R_i, 0, 0, T_j, \textit{ratio}, L^{\textit{depot}}, D^{\textit{dwell}}, V)$
else if $dw_j^{\textit{end}} + < R_i^{\textit{start}} + D^{\textit{maxroute}}$ then
    $\textit{index} \leftarrow 0$
    $Placements^{\textit{pickup}} \leftarrow$
        GetPlacements$(R_i, L_j^{\textit{pickup}}, w_j, \textit{index}, D^{\textit{dwell}})$
    $Placements^{\textit{dropoff}} \leftarrow \emptyset$
    if $|Placements^{\textit{pickup}}| > 0$ then
        $\textit{index} \leftarrow \min(Placements^{\textit{pickup}}) - 1$
        $Placements^{\textit{dropoff}} \leftarrow$
            GetPlacements$(R_i, L_j^{\textit{dropoff}}, dw_j, \textit{index}, D^{\textit{dwell}})$
    end
    if $|Placements^{\textit{pickup}}| \times |Placements^{\textit{dropoff}}| > 0$ then
        $MinCost \leftarrow \infty$
        for $p_{idx} \in Placements^{\textit{pickup}}$ do
            for $d_{idx} \in Placements^{\textit{dropoff}}$ do
                if $p_{idx} < d_{idx}$ then
                    $R_i^{'}, \textit{SolCost} \leftarrow$
                        Adjust$(R_i, p_{idx}, d_{idx}, T_j, \textit{ratio}, L^{\textit{depot}}, D^{\textit{dwell}}, V)$
                    if $\textit{SolCost} \neq \infty$ then
                        $threshold \leftarrow$
                            GThreshold$(\textit{ratio}, R_i^{'})$
                        if $\textit{SolCost} < threshold \wedge \textit{SolCost} < MinCost$ then
                            $MinCost \leftarrow \textit{SolCost}$
                            $(R_i^{'})^{\textit{best}} \leftarrow R_i^{'}$
                        end
                    end
                end
            end
        end
    end
end
Result: $(R_i^{'})^{\textit{best}}, \textit{MinCost}$
```

($(R_i')^{\textit{best}}$), and update the extra time and wait time corresponding to the minimum cost. At the end of this iterative process, the algorithm will return whether a feasible assignment exists for the request (T_j), the best-updated route ($(R_i')^{\textit{best}}$), and the cost *MinCost* corresponding corresponding to the best-updated route. The running time of this algorithm is $\mathcal{O}(|\boldsymbol{T}|^3)$.

Algorithm 7 checks whether the assignment of a new request to a given pickup placement ($p_{\textit{idx}}$) and dropoff placement ($d_{\textit{idx}}$) violates any travel-time, capacity, or route length constraints (such as maximum route duration). If the as-

```
Algorithm 7: Adjust($R_i, p_{idx}, d_{idx}, T_j, ratio, L^{depot}, D^{dwell}, D^{maxroute}, V$)
  $Y_{R_i'} \leftarrow$ AdjustNodes($R_i, p_{idx}, d_{idx}$)
  $success_d, D_{R_i'} \leftarrow$ AdjustTimes($Y_{R_i'}$)
  $success_o, O_{R_i'} \leftarrow$ AdjustCapacities($Y_{R_i'}, V$)
  $SolCost \leftarrow \infty$
  if $success_d \wedge success_o$ then
      $l^{first}, w^{first} \leftarrow Y_{R_i'}^{first}$
      $l^{last}, w^{last} \leftarrow Y_{R_i'}^{last}$
      $D^{source} \leftarrow D^{travel}_{L^{depot}, l^{first}}$
      $D^{sink} \leftarrow D^{travel}_{l^{last}, L^{depot}}$
      $D^{newduration} \leftarrow D^{last}_{R_i'} - D^{first}_{R_i'} + D^{source} + D^{sink} + D^{dwell}$
      if $D^{newduration} \leq D^{maxroute}$ then
          $D^{extra} \leftarrow D^{newduration} - R_i^{end} - R_i^{start}$
          $D^{wait} \leftarrow 0$
          if $Y_{R_i'} = Y_R + \{(L_j^{pickup}, w_j), (L_j^{dropoff}, dw_j)\}$
           then
              if $|Y_R| > 0$ then
                  $l_{prev}, w_{prev} \leftarrow Y_R^{last}$
                  $D^{wait} \leftarrow$
                    $\max(0, w_{prev}^{start} - D^{travel}_{l_{prev}, L_j^{pickup}} - D^{dwell})$
              end
          end
          $SolCost \leftarrow$ GCost($ratio, D^{extra}, D^{wait}$)
      end
  end
  Result: $R_i', SolCost$
```

```
Algorithm 8: BestAssignment($R_i, \boldsymbol{T}, ratio, L^{depot}, D^{dwell}, D^{maxroute}, V$)
  MinCost $\leftarrow \infty$
  $(R_i')^{best} \leftarrow R_i$
  $(T_j)^{best} \leftarrow None$
  for $T_j \in \boldsymbol{T}$ do
      $R_i', SolCost \leftarrow$
        Feasible($R_i, T_j, ratio, L^{depot}, D^{dwell}, D^{maxroute}, V$)
      if $SolCost \neq \infty$ then
          if $SolCost < MinCost$ then
              $MinCost \leftarrow SolCost$
              $(R_i')^{best} \leftarrow R_i$
              $(T_j)^{best} \leftarrow T_j$
          end
      end
  end
  Result: $(R_i')^{best}, (T_j)^{best}$
```

signment does not violate any constraints, then the algorithm computes the increase in the route duration (D^{extra}) for route R_i, as well as the additional wait time D^{wait} before serving the request T_j in the current route R_i if the algorithm assign request T_j to the route R_i by placing the pickup node at the pickup placement value (p_{idx}) and placing the drop-off node at the drop-off placement value (d_{idx}). Finally, the algorithm computes the cost based on the function **GCost** (see Equation (3)) with increase in the route duration (D^{extra}), additional wait time (D^{wait}), and the ratio of requests assigned so far (*ratio*). We can formally express the function **GCost** as follows:

$$GCost(ratio, D^{extra}, D^{wait}) = D^{extra} + (p_{wait} + p_{ratio-cost} \cdot ratio) \cdot D^{wait} \quad (3)$$

The algorithm returns the updated route ($(R_i')^{best}$) and the corresponding cost *SolCost*. Algorithms **AdjustTimes** and **AdjustCapacities** have a time complexity of $\mathcal{O}(|\boldsymbol{T}|)$, whereas algorithm **AdjustNodes** has a time complexity of $\mathcal{O}(1)$. Hence, the computation time of this algorithm is $\mathcal{O}(|\boldsymbol{T}|)$.

Algorithm 8 finds the request that has the lowest weighted cost when the algorithm inserts the request into the given route R_i based on existing nodes (Y_{R_i}) for the route R_i and remaining requests ($\boldsymbol{T}$). The algorithm iterates over each request $T_j \in \boldsymbol{T}$ and first checks whether the algorithm can add request T_j to route R_i. If it can, then the algorithm computes the weighted cost. At the end of this process, the algorithm returns the request ($(T_j)^{best}$) with the lowest weighted cost and the updated best route ($(R_i')^{best}$). The running time of this algorithm is $\mathcal{O}(|\boldsymbol{T}|^4)$.

```
Algorithm 9: Greedy($\boldsymbol{T}, L^{depot}, D^{dwell}, D^{maxroute}, V$)
  $i \leftarrow 0$
  $\boldsymbol{R} \leftarrow \emptyset$
  $t^{size} \leftarrow |\boldsymbol{T}|$
  while $|\boldsymbol{T}| > 0$ do
      $R_i \leftarrow$ CreateEmptyRun($i$)
      while $|\boldsymbol{T}| > 0$ do
          $ratio \leftarrow \frac{|\boldsymbol{T}|}{t^{size}}$
          $R_i', T_j \leftarrow$
            BestAssignment($R_i, \boldsymbol{T}, ratio, L^{depot}$,
            $D^{dwell}, D^{maxrun}, V$)
          if $T_j \neq None$ then
              $R_i \leftarrow R_i'$
              $\boldsymbol{T} \leftarrow \boldsymbol{T} \setminus \{T_j\}$
          else
              break
          end
      end
      $\boldsymbol{R} \leftarrow \boldsymbol{R} \cup \{R_i\}$
      $i \leftarrow i + 1$
  end
  Result: $\boldsymbol{R}$
```

Algorithm 9 assigns all the available requests $\boldsymbol{T}$. The algorithm follows an iterative process, where in each iteration, the algorithm generates a new route R_i and tries to add requests one-by-one from $\boldsymbol{T}$. When no more feasible requests are available for the route R_i, but there are more requests remaining to be assigned, then the algorithm generates the next route and repeats the process. This iterative process terminates when the algorithm has assigned all the requests to routes. The running time of this algorithm is $\mathcal{O}(|\boldsymbol{T}|^5)$.

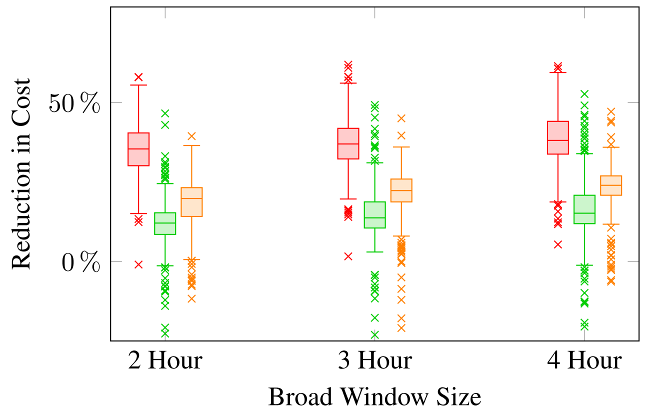

Figure 3: Reduction in total cost due to using our approach (μ^* and $\alpha^{\mathit{SimAnn+Greedy}}$) for selecting tight pickup windows, compared to using naïve window selection, with three different offline VRP solvers: VROOM (🟥), Google OR-Tools (🟧), and our greedy and simulated annealing algorithms as offline VRP solvers (🟩).

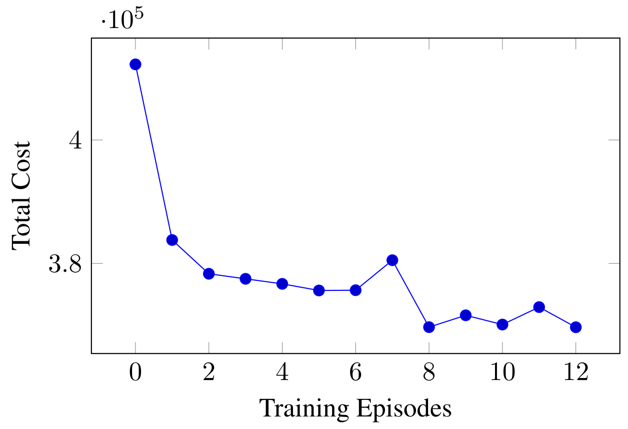

Figure 4: Evolution of the performance of the policy μ, measured as the average total cost of the resulting VRPs, throughout the training process. Each datapoint is based on 5 distinct policies (trained for the given number of episodes), which are evaluated on 5 different problem instances each.

E Supplementary Numerical Results

Hyperparameter Search

We first perform a grid search to find optimal hyperparameters for the greedy algorithm. Based on the results of this search, we set values for the wait-time $p_{wait} = 0.1$ and fraction of requests served $p_{ratio-cost} = 0.1$ parameters, which are used by the algorithm to estimate the cost of assigning a request to a route (see Algorithm 7). We also set values for the length of the route $p_{length} = 10$, the fraction of requests served $p_{ratio-thres} = 0.1$, and the constant value of the threshold $p_{const} = 0.1$ parameters, which restrict the assignment of requests to routes (see Algorithm 6).

We next perform a grid search to find optimal hyperparameters for the simulated annealing algorithm. Based on the results of this search, we set the altering rate to $p_{alter} = 0.4$ (see Algorithm 2) and the initial and final probabilities to $p_{start} = 0.9$ and $p_{end} = 0.5$, respectively (see Algorithm 1).

Proposed Approach vs. Naïve Pickup Window

Here, we present additional numerical results showing the advantage of using our proposed approach to select tight pickup windows. Figure 3 shows the reduction in the total cost of vehicle routes due to using our approach (policy μ^* supported by anytime algorithm $\alpha^{\mathit{SimAnn+Greedy}}$) to select tight pickup windows, compared to naïvely selected pickup windows. We consider three offline VRP solvers to compare our windows with the naïve ones: VROOM, Google OR-Tools, and our greedy and simulated annealing algorithms applied as offline VRP solvers. In each case, we evaluate the algorithms on 160 days of paratransit data, and plot the distributions. We observe that when we use our algorithms as offline VRP solvers, there is a significant, 14-18% reduction in cost, which is consistent with Figure 2. Again, we see that longer broad pickup windows lead to more pronounced reduction since they provide more flexibility for optimizing the tight pickup windows.

On the other hand, we see a less significant reduction in cost when using VROOM and Google OR-Tools: 4-6% and 1-2% reduction, respectively. We hypothesise that this is explained by the fact that we train our policy μ^* with our algorithms ($\alpha^{\mathit{SimAnn+Greedy}}$ as the anytime and α^{Greedy} as the shaped cost $\tilde{c}_i$ estimator); hence, our policy μ^* learns to work well with our VRP solvers. Since VROOM, Google OR-Tools, and our algorithms are all heuristic, the tight windows selected by our policy μ^* may not work well with VROOM and Google OR-Tools. We can address this by training a policy μ with VROOM and with Google OR-Tools, so that the policy μ will select tight windows that fit these VRP solvers. We will investigate this in future work.

Training Process

Finally, we present numerical results on the reinforcement-learning process to show how the performance of the policy μ improves as it is trained on more and more episodes. Since the reinforcement-learning process is non-deterministic (as the neural-network representation of Q is initialized at random, and random actions are chosen occasionally to ensure exploration), we provide robust results by initializing 5 different neural-network instances and then training these instances independently of each other over a number of episodes. After each episode, we evaluate each policy (i.e., each neural-network instance) on 5 problem instances (i.e., 5 days of paratransit data). Figure 4 shows the average performance of these 5 policies over the 5 problem instances, for training episodes 0 (i.e., untrained networks) to 12.

We observe that trained policies (more than 0 episodes) have a significant advantage over untrained policies (0 episodes), which select tight windows at random (within the broad windows) since the neural networks are initialized with random weights. Further, we see that the performance curve starts to “flatten out” around 8-12 episodes, which suggests that we may not need many more episodes to find optimal policies.

Published in the proceedings of the 31st International Joint Conference on Artificial Intelligence (IJCAI 2022).

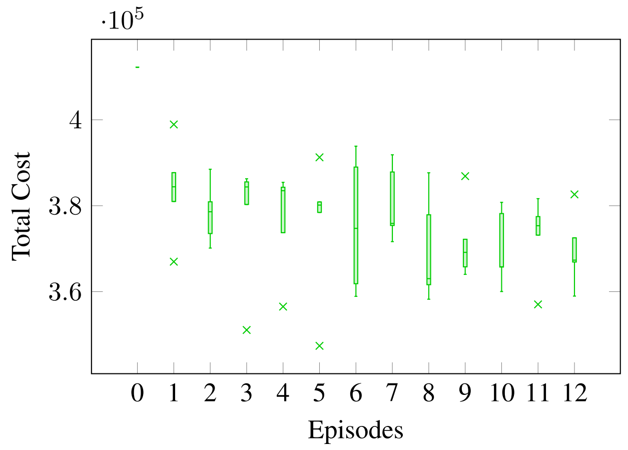

Figure 5: Evolution of the performance of the policy μ, measured as the average total cost of the resulting VRPs, throughout the training process. Each box plot is based on 5 distinct policies (trained for the given number of episodes), which are evaluated on 5 different problem instances each.

F Dataset Statistics

In this section, we study how the spatial and temporal distribution of the trips varies based on the booking time. Accordingly, we categorize the requests into the requests made in the morning (9am - 12pm), afternoon (12pm - 3pm), and evening (3pm - 5pm). Then, we look into the spatial and temporal distribution of the requests based on these three categories.

The Figure 6 shows the distribution of the pickup time of requests based on the time of booking. In the figure, we observe that customers are more likely to book requests for the next day early morning, either in the morning or afternoon of the previous day. Similarly, the customers are more likely to book requests for the next day late evening in the evening of the booking day.

The Figure 7 shows the distribution of the pickup location (ZIP code) of the requests based on the time of the booking. We shifted the coordinates to anonymize the data. The distribution of pickup locations does not vary much between the requests made in the morning and afternoon. Moreover, the distribution of pickup locations varies considerably between requests made in the evening and the rest of the day.

These observations indicate that there is a considerable difference between the pickup location and pickup time, between the requests booked in the morning, afternoon, and evening.

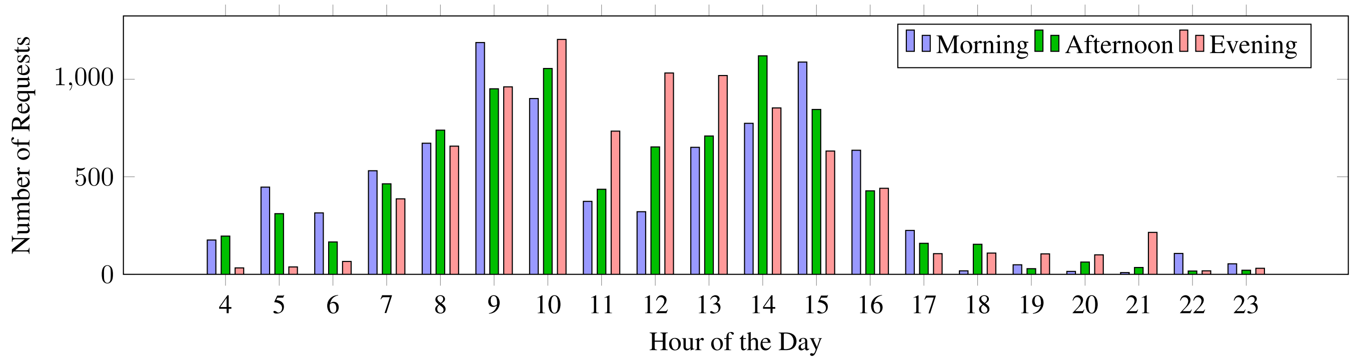

Figure 6: Distribution of the pickup time of the requests based on the booking time.

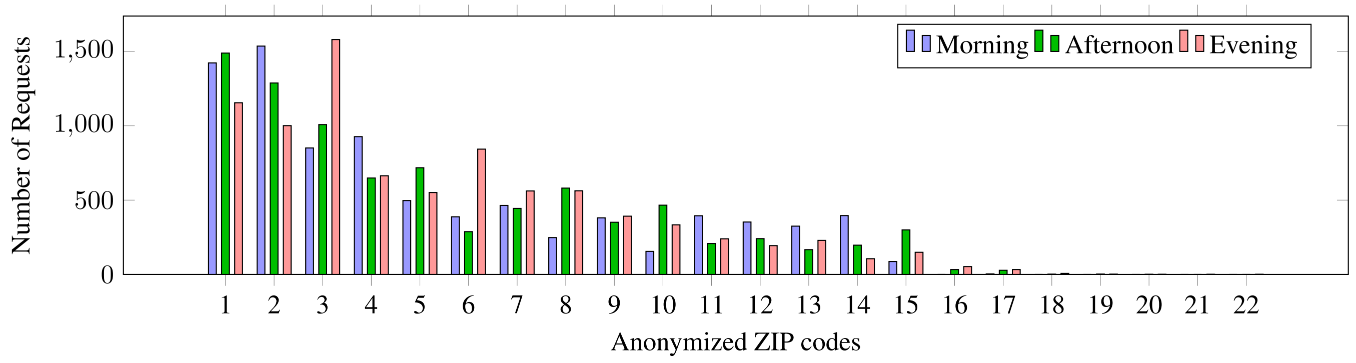

Figure 7: Distribution of the pickup location of the requests based on the booking time.

G Extended Discussion of Related Work

We can divide prior work on vehicle routing problems into two main categories: offline (or day-ahead) VRPs and online (or real-time) VRPs. We presented a discussion of relate work focusing on this distinction in Section 5. Here, we provide a further discussion of related work focusing on the objective formulation of VRPs as well as the approaches used to solve dial-a-ride problems.

VRP Objectives

Vehicle routing problems can also be categorized based on their objectives, that is, based on how they quantify the cost of a VRP solution. Agatz *et al.* [2011], Turmo *et al.* [2018], Alonso-Mora *et al.* [2017], Ota *et al.* [2016], Simonetto *et al.* [2019], and Wen *et al.* [2018] consider minimizing the total miles traveled by the vehicles. Meanwhile, Turmo *et al.* [2018], Gupta *et al.* [2010], and Simonetto *et al.* [2019] consider minimizing the total hours traveled by the vehicles, similar to our VRP formulation. In contrast, Alonso-Mora *et al.* [2017], Turmo *et al.* [2018], and Wen *et al.* [2018] focus on improving the quality of VRP solutions from the perspective of passengers by minimizing the passengers' total waiting time. Similarly, Salazar *et al.* [2018] consider reducing the total passenger travel time. In our VRP formulation, we consider minimizing the total travel time of the vehicles and the number of vehicle routes.

Dial-a-Ride Problem

Previous research efforts use approaches such as tabu search [Mo *et al.*, 2018; Berbeglia *et al.*, 2012], greedy approach [Qian *et al.*, 2017; Alonso-Mora *et al.*, 2017], and exact methods [Liu *et al.*, 2015; Parragh *et al.*, 2015; Gschwind and Irnich, 2015] to solve the scheduling in dial-a-ride problem and paratransit operations.

Pickup and Delivery Problem

In earlier research efforts, researchers used adaptive neighbourhood search [Ropke and Pisinger, 2006], and exact-methods [Ropke *et al.*, 2007; Ropke and Cordeau, 2009; Qu and Bard, 2015] to solve the pickup and delivery problem.