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 [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 [Sivagnanam *et al.*, 2022].

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 full formal definition in Appendix B [Sivagnanam *et al.*, 2022].

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 $L_i^{\textit{pickup}}$, a drop-off location $L_i^{\textit{dropoff}}$, 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 $w_i^{\textit{start}}$ and latest $w_i^{\textit{end}}$ pickup time. The input also includes constants, such as the maximum allowed duration $D^{\textit{maxroute}}$ 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 $L_i^{\textit{pickup}}$ and drop-offs $L_i^{\textit{dropoff}}$ (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 $L_i^{\textit{pickup}}$ is included in exactly one route R_j, the corresponding dropoff $L_i^{\textit{dropoff}}$ 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 $\textit{VRP}^*(\boldsymbol{T}, \boldsymbol{w})$ denote the total cost of an optimal solution for problem instance $(\boldsymbol{T}, \boldsymbol{w})$. That is, $\textit{VRP}^*(\boldsymbol{T}, \boldsymbol{w}) = \min_{\boldsymbol{R} \in \mathcal{R}(\boldsymbol{T}, \boldsymbol{w})} \mathcal{C}(\boldsymbol{R})$. Since there is a vast literature on solving offline VRPs, we assume that an offline VRP solver (i.e., heuristic or approximation algorithm for $\textit{VRP}^*$) 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 $\textit{VRP}^*(\boldsymbol{T}, \boldsymbol{w})$ 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 $W_i^{\textit{start}}$ and latest $W_i^{\textit{end}}$ pickup time for request T_i. The input also includes the probability distribution $\mathcal{D}$, the maximum duration of tight pickup windows $D^{\textit{window}}$ (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 $D^{\textit{window}}$ long (i.e., $w_i^{\textit{end}} - w_i^{\textit{start}} \leq D^{\textit{window}}$) and falls within the broad window (i.e., $W_i^{\textit{start}} \leq w_i^{\textit{start}} \leq w_i^{\textit{end}} \leq W_i^{\textit{end}}$).

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):

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

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., 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 [Sivagnanam *et al.*, 2022].

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 $D^{maxroute} = 10$ hours, and maximum tight pickup window duration $D^{window} = 30$ 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 $\alpha^{SimAnn+Greedy}$. Since these are based on standard techniques, we describe them in Appendix D [Sivagnanam *et al.*, 2022]. 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 and the Google OR-Tools Vehicle Routing framework. 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 from the Keras library.

4.2 Results

We provide supplementary numerical results in Appendix E [Sivagnanam *et al.*, 2022].

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

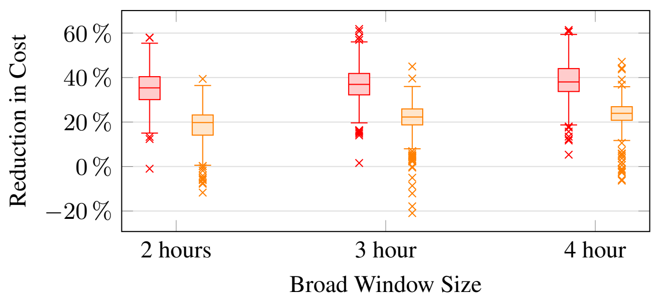

Figure 1: Reduction in total cost due to using our approach for selecting tight pickup windows (policy μ^* supported by anytime algorithm $\alpha^{SimAnn+Greedy}$), compared to using naïve pickup windows with VROOM (■) and Google OR-Tools (■) as offline VRP solvers.

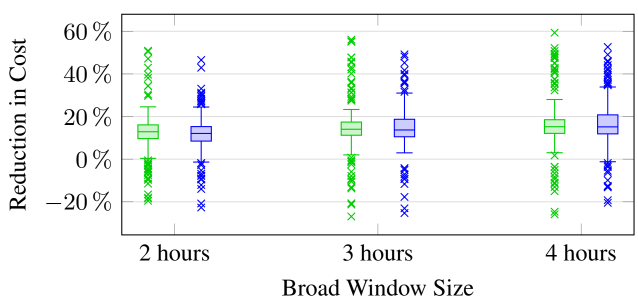

Figure 2: Reduction in total cost due to using our complete approach (policy μ^* supported by anytime algorithm $\alpha^{SimAnn+Greedy}$), compared to using a policy μ^* without anytime support (■) and using naïve pickup windows with algorithm $\alpha^{SimAnn+Greedy}$ as the offline VRP solver (■).

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

[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.

[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.

[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.

[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.

[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.

[Sivagnanam *et al.*, 2022] Amutheezan Sivagnanam, Salah Uddin Kadir, Ayan Mukhopadhyay, Philip Pugliese, Abhishek Dubey, Samitha Samaranayake, and Aron Laszka. Offline vehicle routing problem with online bookings. *arXiv preprint arXiv:2204.11992*, 2022.

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

[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.