An Online Approach to Solve the Dynamic Vehicle Routing Problem with Stochastic Trip Requests for Paratransit Services

Michael Wilbur[1], Salah Uddin Kadir[2], Youngseo Kim[3], Geoffrey Pettet[1], Ayan Mukhopadhyay[1], Philip Pugliese[4], Samitha Samaranayake[3], Aron Laszka[2], and Abhishek Dubey[1]
[1]Vanderbilt University, Nashville TN, United States
[2]University of Houston, Houston TX, United States
[3]Cornell University, Ithaca NY, United States
[4]Chattanooga Area Regional Transportation Authority (CARTA), Chattanooga TN, United States

ABSTRACT

Many transit agencies operating paratransit and microtransit services have to respond to trip requests that arrive in real-time, which entails solving hard combinatorial and sequential decision-making problems under uncertainty. To avoid decisions that lead to significant inefficiency in the long term, vehicles should be allocated to requests by optimizing a non-myopic utility function or by batching requests together and optimizing a myopic utility function. While the former approach is typically *offline*, the latter can be performed *online*. We point out two major issues with such approaches when applied to paratransit services in practice. First, it is difficult to batch paratransit requests together as they are temporally sparse. Second, the environment in which transit agencies operate changes dynamically (e.g., traffic conditions can change over time), causing the estimates that are learned offline to become stale. To address these challenges, we propose a fully online approach to solve the dynamic vehicle routing problem (DVRP) with time windows and stochastic trip requests that is robust to changing environmental dynamics by construction. We focus on scenarios where requests are relatively sparse—our problem is motivated by applications to paratransit services. We formulate DVRP as a Markov decision process and use Monte Carlo tree search to evaluate actions for any given state. Accounting for stochastic requests while optimizing a non-myopic utility function is computationally challenging; indeed, the action space for such a problem is intractably large in practice. To tackle the large action space, we leverage the structure of the problem to design heuristics that can sample promising actions for the tree search. Our experiments using real-world data from our partner agency show that the proposed approach outperforms existing state-of-the-art approaches both in terms of performance and robustness.

KEYWORDS

Vehicle Routing Problem, Monte Carlo Tree Search, Online Planning, Smart Transit, Decision-making under Uncertainty

1 INTRODUCTION

The vehicle routing problem (VRP) is a well-known combinatorial optimization problem that seeks to assign a fleet of vehicles to routes to serve a set of customers/requests [7]. Many real-world use cases of transportation agencies are modeled by the dynamic version of the problem (DVRP) with stochastic trip requests. In such settings, some customer requests may be known at the time of planning while others are unknown, and some stochastic information may be available about potential future requests [3]. Although the dynamic and the stochastic versions have traditionally been tackled separately, Bent and Van Hentenryck [3] and Hvattum et al. [13], among others, showed that dynamic planning could use the stochastic information to improve performance. There are three broad approaches for solving DVRPs with stochastic requests. First, as requests arrive, a group of requests can be batched together, and routes can be optimized myopically for the particular batch in an *online* manner [1]. Second, the routing problem can be solved to maximize a non-myopic utility function by learning a policy in an *offline* manner that maps any given state of the problem to an action (i.e., a route plan for the vehicles) [21]. Third, a combination of offline computation and online heuristics can be used for non-myopic planning [3, 15].

A fundamental challenge in solving the DVRP non-myopically is computational tractability; indeed, real-world applications of DVRP are often intractable since they entail solving a hard combinatorial optimization problem with consideration for future requests. Prior work based on the combination of offline learning and online heuristics has addressed this bottleneck to some extent. Bent and Van Hentenryck [3] continuously generate and store a pool of promising plans. Then, at the time of execution, they use a least-commitment strategy to select a key plan from the pool. Shah et al. [29] use approximate dynamic programming and leverage a neural network-based approximation of the value function to handle the complexity from combinations of passenger requests. Joe and Lau [15] ensure near real-time response by combining online routing-based heuristics (e.g., simulated annealing) with offline approaches (e.g., value function approximation) [15]. Myopic approaches, on the other hand, focus on pooling requests together to optimally allocate a specific batch of requests to routes [1]. A major challenge in such a setting is computing solutions fast enough to assign vehicles to requests in real-time (in practice, some delay is acceptable after a request is made). Hence, myopic approaches that operate in an online manner must be *anytime* or able to compute a feasible action quickly; requests are stochastic and a decision must be computed before future requests arrive. Alonso-Mora et al. [1] scale a myopic approach to assigning routes to pooled requests for real-world use cases by using an anytime algorithm that starts from a greedy solution and then improves it through constrained optimization.

We focus on *paratransit* services [20] in this paper, which are an important real-world example of DVRPs. Paratransit service is a socially beneficial curb-to-curb transportation service provided by public transit agencies for passengers who are unable to use

fixed-route transit (e.g., passengers with disabilities). Our partner agency, the Chattanooga Area Regional Transportation Authority (CARTA), operates paratransit services in a mid-sized metropolitan area in the USA. While paratransit services resemble traditional on-demand ride-pooling services in some ways (e.g., the arrival of real-time requests and ride-sharing), our collaboration revealed some crucial differences between canonical examples of DVRPs in prior literature (e.g., ride-pooling or cargo delivery) and paratransit services. First, the frequency of requests is usually lower than services like taxis. For example, CARTA operates paratransit services in a metropolitan area with about 1.8 million people and usually serves about 200 requests per day (with 10 hours of operation each day). This constraint makes it difficult to batch requests together for myopic algorithms. Note that while the requests are temporally sparse, the decision for each request must be computed quickly. Second, paratransit services operate under the Americans with Disabilities Act (ADA), which enforces time windows as a hard constraint, unlike on-demand taxi services, thereby requiring strict adherence to such constraints. Our discussion with CARTA also revealed a potential issue with offline approaches for solving DVRPs. In practice, the environment in which CARTA operates is highly dynamic; traffic conditions in a city can change due to construction, events, or accidents, and the number of available vehicles or drivers can vary. In such cases, offline approximations can potentially lead to decisions that are far from optimal.

This paper introduces a *fully online* approach for use-cases such as paratransit services that is *anytime*, *non-myopic*, *robust* to dynamic changes in the environment, and also *scalable* to real-world applications. Designing a completely online approach to solve DVRPs in a non-myopic setting is extremely challenging — the action space for such a problem is intractably large for real-world applications. For example, for our partner agency in a midsize metropolitan city in US with five vehicles, each with a capacity of eight passengers, the action space is of the order of 10^{22}. Also, note that while requests are relatively sparse in our problem setting (one request every few minutes on average), computation for each decision needs to be fairly quick; naturally, it is infeasible to keep customers waiting for more than some exogenously defined duration.

The summary of contributions are as follows: **(1)** We design a fully online and non-myopic solver for DVRP with stochastic requests that scales to real-world problems by leveraging the structure of the problem instance. Our approach, MC-VRP (**M**onte **C**arlo tree search based solution for **v**ehicle **r**outing **p**roblem) is robust to environmental dynamics by construction. **(2)** We model the DVRP as a route-based Markov decision process (MDP) [31]. Given an arbitrary state of the MDP, we use generative models over customer requests and travel time to simulate the environment under consideration, which in turn enables us to use Monte Carlo tree search [19] to find promising actions for the state. Our approach does not require offline training, the only requirement is a generative model that can be sampled at run-time. **(3)** To tackle the intractably large action space, we leverage the structure of the problem to find promising actions. Specifically, given a set of routes and a new request, we create a weighted graph based on a budget-based heuristic whose edges represent: (a) which vehicles can serve the new request, and (b) which vehicles can swap unpicked requests from their routes to maximize utility. Then, we sample independent sets from the graph that correspond to feasible actions for the given state of the MDP.

To evaluate the proposed approach, we consider three baselines. First, we look at a greedy strategy, which assigns a new request to the vehicle that provides the highest myopic utility. Second, we look at recent work by Joe and Lau [15], which outperformed well-known state-of-the-art approaches such as the multiple scenario-based approach [3] and the approximate value iteration [34]. Third, we compare our approach with a batched myopic setting by using the work by Alonso-Mora et al. [1]. Our experimental evaluations show that MC-VRP approach outperforms the baselines in terms of performance and robustness to changing environmental conditions.

2 PROBLEM DESCRIPTION AND MODEL

Typically, in paratransit services, passengers request pick-up times in advance of the trip, even when requesting on the same day. This is in contrast of on-demand ride services that are often requested with a short lead time. For example, customers can call in the morning to request for a ride during early afternoon. However, in some cases customers request rides with a short lead time. We assume that requests are i.i.d. according to a distribution D. We denote the request at a given time t by R_t, which consists of a pick-up location, a drop-off location, a pick-up time, and a drop-off time.

In practice, it is common for paratransit services to use a pick-up window, e.g., they commit to picking passengers up at most 15 minutes before the requested pickup time and drop them off at most 15 minutes after the requested drop-off time. For a request R_t, we refer to such time points as the earliest pick-up time (denoted by e_t) and the latest drop-off time (denoted by p_t). Naturally, the latest drop-off time must be greater than the sum of the earliest pick-up time and the minimum time it takes to travel to the drop-off location from the pick-up location. We assume that requests follow such constraints. In practice, the application used for making requests or a human operator can enforce such constraints. Note that the time windows specified by the customers are treated as strict guidelines making this a pickup and delivery problem with time-windows (PDPTW) [8]. In case constraints cannot be met, the system *rejects* requests. Given this setting, the goal of the decision-maker is to maximize the total number of requests that can be served in a day while ensuring that the pick-up and drop-off constraints are met.

2.1 Route-based Markov Decision Process

Our problem consists of a set of identical vehicles, denoted by V, each with a capacity of c passengers. The set of all possible locations in the area is represented by the graph $G = (L, E)$, where L denotes the set of vertices (or locations) in the graph, and E denotes the set of edges weighted by travel time. A route plan for vehicle $v^i \in V$ at time t is denoted by θ_t^i, which is an ordered sequence of pick-up/drop-off locations that the vehicle needs to visit in its route. Therefore, a route plan θ_t^i can be represented as an ordered set $\{l^1, l^2, \ldots\}$, where each $l^i \in L$ is a vertex of graph G. At any given time t, the set of all feasible route plans is denoted by Θ_t (we define feasibility below). We assume that vehicles start operations at a depot and return to the depot at the end of the day ($depot \in L$).

To solve the problem of identifying the best routes for all vehicles, we model the dynamic vehicle routing problem (DVRP) as a

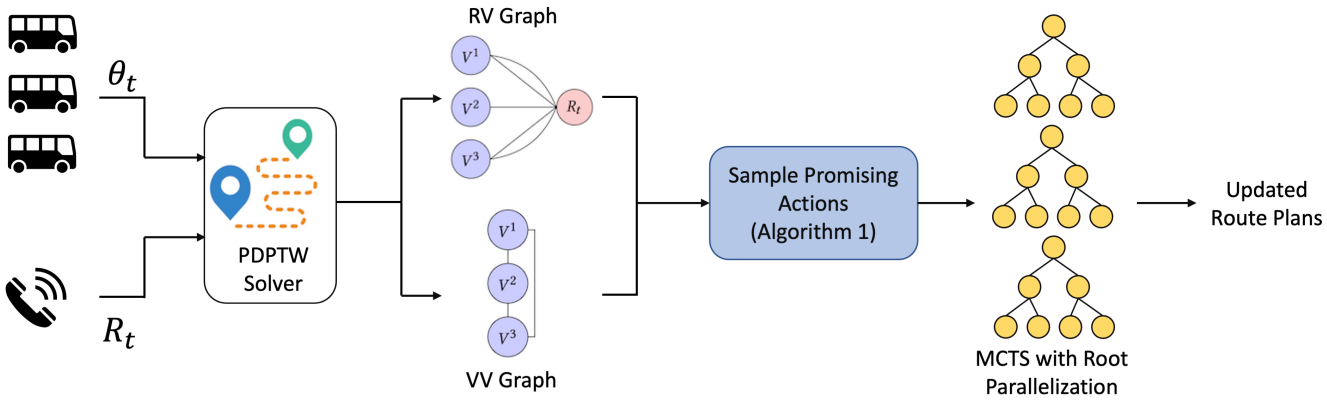

Figure 1: An overview of MC-VRP. A decision epoch corresponds with a new trip request R_t . We generate request-vehicle (RV) and vehicle-vehicle (VV) graphs by combining a heuristic based PDPTW solver with metrics to quickly estimate the utility of route plans. We then select promising actions from the graphs by sampling independent sets of high weights to be evaluated by an online approach based on MCTS.

Markov decision process (MDP) based on prior work done by Ulmer et al. [31]. An MDP is defined by a 4-tuple $\{S, A, P, \gamma\}$, where S is a set of states that capture relevant information for decision-making, A is a set of control choices or actions, P is a state-action transition function, and γ is a reward function that defines the utility of taking an action at a given state [18]. The route-based MDP formulation is beneficial in DVRP settings where actions involve assigning the current request to a vehicle and optimizing the entire route under consideration. Since our goal is to maximize the number of requests that can be served in expectation (with respect to the distribution D), optimizing the route as a whole towards this goal is a natural choice.

Decision Epoch We define decision epochs as in prior work by Joe and Lau [15]. A decision epoch occurs at the time a request is received. Between requests, the environment evolves in continuous time, e.g., the vehicles move continuously, and requests can arrive at any point in time. At each decision-epoch, the decision-maker takes an action (states and actions are defined below). The effect of the action results in a state transition, which consists of two parts — a transition from a pre-decision state to a post-decision state, and from the post-decision state to the next pre-decision state [25, 31].

State We denote the set of states by S. We use s_t to denote the pre-decision state at time t, which includes the vehicle locations, current route plans for all vehicles, and details about passengers aboard the vehicles. Note that while it suffices to keep track of the route plan for some formulations, we must track drop-off times for passengers already on board as actions may dynamically change routes. Formally, we represent the state s_t by the tuple $(R_t, R, vloc_t, \theta_t)$, where R_t is the new trip request at time epoch t, $R = \{R^1, \ldots, R^{|V|}\}$ is the set of requests assigned to each of the vehicles, $vloc_t$ is a vector of location of all the vehicles at time t, and $\theta_t = \{\theta_t^1, \ldots, \theta_t^{|V|}\}$ is the current route plan for each of the vehicles. The post-decision state is denoted by s_x^t, which denotes the effect of an action x on s_t, and includes the updated route plan [31].

We associate some additional information with each route plan. Consider a route plan θ_t^i for vehicle $v^i \in V$. For each location l^j in the route plan, we associate four pieces of information. First, let $a(\theta_t^i, l_j)$ denote the planned arrival time of the vehicle v_i at location l^j. We can calculate the arrival time from a pre-computed travel time matrix (or a router). Second, let $e(\theta_t^i, l^j)$ be the earliest time service may begin at this location. If l^j is a pick-up location, $e(\theta_t^i, l^j)$ is equal to the earliest pick-up time for the customer; otherwise, we set it to some default value. Third, $p(\theta_t^i, l^j)$ denotes the latest time at which service is desired at this location. If l^j is a dropoff location, $p(\theta_t^i, l^j)$ is set to the latest drop-off time for the customer associated with the location; otherwise, we set it to some default value. We also maintain the number of passengers on-board the vehicle at each location as $w(\theta_t^i, l^j)$. Lastly, let y^j be a binary variable set to 0 if l^j is a pickup location and set to 1 otherwise.

Actions We denote the set of all feasible actions at time t by X_t. An arbitrary action in X_t involves assigning the new trip request R_t to a vehicle $v^i \in V$ and subsequently updating the route plan of *all* the vehicles. Note that in our problem setting, X_t simply reduces to Θ_t, the set of all feasible route plans at time t. Updating the route plan can entail changing the order in which existing requests are picked up/dropped off in a vehicle's route plan or swapping requests that have not been picked up between vehicles. While allowing swapping between vehicles increases the complexity of the problem significantly, we include such actions nonetheless to maximize utility. The action space is, therefore, the set of *all* feasible route plans. A feasible route must meet three requirements: first, the pick-up location for each trip requests must appear before the corresponding drop-off location in the route plan. Second, the pick-up time for a request must not occur before its earliest pick-up time, and finally, the drop-off time must not occur after its latest possible drop-off time.

State Transitions and Rewards At decision-epoch t, the decision-maker can take an action $x^j \in X_t$ (say), which results in a transition from the pre-decision state s_t to the post-decision state s_t^x. Then, the system transitions from s_t^x to the next pre-decision state s_{t+1}, within which a new request might arrive. We assume that requests are drawn from a distribution D. We refrain from defining the exact state-action transition function since we use a simulator (or a generative model) for online planning. The reward

for an action $x^j \in X_t$, denoted by $\gamma(x^j)$, is simply the number of additional requests served by the action. Since only one request arrives at any decision epoch, the reward simplifies to 1 if the request can be accommodated and 0 otherwise. Since paratransit services require strict adherence to time windows, note that some requests can be rejected. In practice, human operators can suggest alternative choices to customers in such situations. We present a notation lookup table in the technical appendix (section A.6) for convenient reference to notation.

3 APPROACH

We provide an overview of MC-VRP in Figure 1. Our approach consists of the following three broad components: **1)** For a given state of the MDP, we sample feasible and promising actions by exploiting the structure of the problem. To compute feasibility, we use a heuristic-based solution approach to the PDPTW. To compute the potential utility of a feasible action, we introduce two heuristics, one based on passenger travel times and another based on *budget* (or slack) in route plans. **2)** We compute weighted graphs based on the feasible actions and their potential utilities. Specifically, we create vehicle-vehicle (VV) and request-vehicle (RV) graphs (we describe the graphs below). Then, we generate promising actions by sampling independent sets from the graphs (based on the weights of the sets). **3)** The sampled actions are then used by our online non-myopic planner based on Monte Carlo tree search. To build the search tree into the future, we sample future requests from a data-driven generative model. Finally, to lower computation time, we utilize pre-computed samples of requests and *root parallelization* [23] to efficiently explore the search space and recommend an action for the given state. We describe each component of our approach in detail below.

3.1 PDPTW Solver

At each decision epoch, our goal is to optimize existing vehicles' routes to accommodate a new request. First, we check whether a vehicle can accommodate the request given its current route plan. Recall that accommodating a request in our setting requires strict adherence to time windows. We use a heuristic-based solver for the pickup and delivery problem with time-windows (PDPTW). The solver enables us to check if the current request can be accommodated in a feasible route plan. The PDPTW is NP-hard [8, 10]; as a result, we use a heuristic subroutine to solve it. Using a heuristic approach is critical in our setting; as we show below, the PDPTW solver needs to be invoked for the given state of the MDP as well as for states we sample as we look into the future. While any heuristic designed for solving PDPTW can be incorporated in our framework, we use the *insertion heuristic* [1], which seeks to insert the pickup and dropoff locations of the new request within the existing route plan. We introduce some additional notation to describe the PDPTW solver. Consider a vehicle $v^i \in V$ that has an assigned route θ_t^i at time t. For a new request R_t, we use $PDPTW.feasiblePlans(\theta_t^i, R_t)$ to denote a module that returns the set of all feasible route plans for the vehicle v^i that include the new request. Also, let $PDPTW.bestFeasiblePlan(\theta_t^i, R_t) = \text{argmax}_\theta U_\omega(PDPTW.feasiblePlans(\theta_t^i, R_t))$, which denotes a function that computes the utility of each feasible route plan generated by the PDPTW solver based on a specific utility function U and a metric ω, and returns the one with the highest utility. For example, a metric could be the total travel time of the route, and the utility function, in the simplest case, can be the identity function.

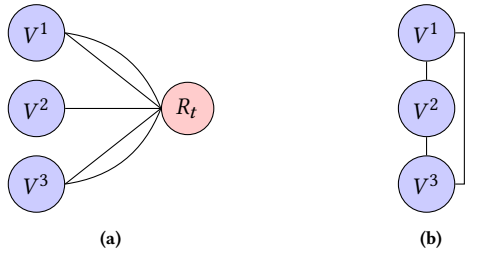

Figure 2: (a) RV-graph (G_{RV}): there is an edge between a request and a vehicle for every feasible route plan in which a vehicle can service the new request. (b) VV-graph (G_{VV}): an edge between vehicle v^i and v^j represents the swap with the highest utility between the two vehicles.

3.2 Handling Exponential Action Space

A feasible action in our problem corresponds to a set of route plans for all vehicles, given that one of them can accommodate the new request in consideration. In case no feasible action is found, the request is rejected. Additionally, we design our action space to let vehicles swap requests from their assigned routes that have not been picked up (we describe how we use the PDPTW solver to this end below). As a result, the number of possible actions for a given state is combinatorially large; on average, an arbitrary state in our MDP has 10^{22} possible actions. Such an action space is infeasible to explore in an online setting. To address this challenge, we introduce an approach that enables us to sample promising actions from the set of feasible actions. We start by introducing two heuristic metrics that can be used to gauge the long-term utility of a route plan quickly.

1.) Maximizing the budget to serve future requests: Our goal is to maximize the number of requests the vehicles serve on a given day while following the specified time constraints. Intuitively, a vehicle can accommodate future requests in an existing route plan if there is sufficient *room* (time) in the route. To capture this idea formally, we build upon prior work by Ulmer et al. [33] to extend the idea of a budget-based heuristic to DVRPs with capacity and time window constraints. Our budget-based heuristic captures the idea that maximizing the time a vehicle has no passengers on board also maximizes the slack to serve future requests. We define the budget-based utility for a route plan for vehicle $v^i \in V$ at time t as

$$\overline{b}(\theta_t^i) = t_{max} - t - \sum_{j \in \{1,\dots,|\theta_t^i|-1\}} \mathbb{1}\left(w(\theta_t^i, l^j) > 0\right)\left\{a(\theta_t^i, l^{j+1}) - a(\theta_t^i, l_j)\right\} \tag{1}$$

where t_{max} denotes the maximum time up to which the vehicle is available to serve requests (e.g., end of a day), $(w(\theta_t^i, l_j))$ denotes the number of passengers on board vehicle v^i from location l^j in its route plan to the next location, $\mathbb{1}()$ denotes the indicator function, and $a(\theta, l^k)$ denotes the time a vehicle operating under a route plan θ reaches location l^k. The summation in the equation 1 represents the total time in the route plan for which at least one passenger is on-board.

2.) Minimizing passenger travel time: An alternative to the budget-based heuristic is to minimize passenger travel time (PTT). Intuitively, by minimizing passenger travel time, we maximize the available capacity in each vehicle over the time horizon. We define the utility of the PTT-based heuristic for a route plan θ_t^i as $\overline{PTT}(\theta_t^i)$ for vehicle $v^i \in V$ at time t as shown in equation:

$$\overline{PTT}(\theta_t^i) = \sum_{j \in \{1,\ldots,|\theta_t^i|-1\}} w(\theta_t^i, l_j) * (a(\theta_t^i, l_{j+1}) - a(\theta_t^i, l_j)) \quad (2)$$

In this case $\overline{PTT}(\theta_t^i)$ is the summation of the number of passengers on board after picking up the passenger at location j, represented by $w(\theta_t^i, l_j)$, multiplied by the time to reach the next location, $j+1$. Therefore, by minimizing $\overline{PTT}(\theta_t^i)$ we maximize the number of seats available to incorporate future requests.

Having described metrics to assess the potential utility of a specific vehicle route for a given vehicle, we now introduce an approach to sample feasible route plans. We begin by describing two graphs we construct based on the new request that arrives at a decision epoch and the existing route plans of the vehicles.

RV graph: At each decision epoch, we first generate a graph that incorporates which vehicles can service the new request. Our idea is based on prior work by Alonso-Mora et al. [1]. We denote the RV graph by $G_{RV} = (L_{RV}, E_{RV})$, where L_{RV} denotes a set of vertices and E_{RV} denotes a set of edges. To create the RV graph, we add a node for each vehicle $v_i \in V$. We add an additional node to denote the request R_t. Then, for each feasible vehicle route that can accommodate the request, we add an edge between the node denoting the request and each node representing a vehicle. We denote a specific edge in E_{RV} by $e_{RV}(i, j)$, which denotes the jth feasible route plan for vehicle v^i that can accommodate the request under consideration. The feasible route plans are generated by the module $PDPTW.feasiblePlans(\theta^i, R_t)$ (for vehicle $v^i \in V$ and request R_t). We associate two pieces of information with each edge. First, for an edge $e_{RV}(i, j)$, we use $U_\omega(e_{RV}(i, j))$ to denote the utility of the edge based on a utility function U and metric ω. Also, we use $\theta(e_{RV}(i, j))$ to denote the updated route plan corresponding to the edge $e_{RV}(i, j)$. We show an example of an RV graph in figure 2.

VV-graph: While an RV graph is useful to represent which vehicles can accommodate the new request in their existing route plans, it is possible that vehicles might want to swap requests that have not been picked up to maximize utility. Note than in a non-myopic setting, the route plans are optimized at each decision-epoch to maximize the expected utility with respect to D (the request arrival distribution). However, a specific realization of R_t presents an opportunity for vehicles to re-plan and potentially swap requests. In order to represent such swapping actions, we create an undirected vehicle-vehicle (VV) graph at each decision epoch. We denote a VV graph by $G_{VV} = (L_{VV}, E_{VV})$.

To construct the graph, we add a node for each vehicle $v^i \in V$. Edges between two vehicles denote potential swaps of requests that have not been picked up. An edge $e_{VV}(i, j, k) \in E_{VV}$ denotes the potential swap of request R_k from vehicle v^i to vehicle v^j. Note that a swap action creates two new route plans, one for each vehicle. We use $\theta(e_{VV}(i, j, k), v^i)$ to denote the route plan for vehicle v^i *after* the swap (similarly, we use $\theta(e_{VV}(i, j, k), v^j)$ to denote the route plan for vehicle v^j). The utility of a swapping action, denoted by $U_\omega(e_{VV}(i, j, k))$ is denoted as the difference in utility of the updated route plans and the original route plans, i.e., the plans without the swapping action. As before, U_ω denotes utility computed according to a function U and metric ω. For example, using an identity utility function and a metric based on the budget heuristic introduced in equation 1, $U_{budget}(e_{VV}(i, j, k)) = \overline{b}(\theta(e_{VV}(i, j, k), v^j)) + \overline{b}(\theta(e_{VV}(i, j, k), v^i)) - \overline{b}(\theta_t^i) - \overline{b}(\theta_t^j)$. Note that once a request is swapped, its pickup and dropoff can be inserted at multiple places within the existing route of the vehicle that receives it. For computational tractability, we choose the best insertion point using the module $PDPTW.bestFeasiblePlan$ (introduced in section 3.1). We show an example of an VV graph in figure 2.

Generating Feasible Actions: A feasible action is an updated set of route plans for all vehicles that do not violate the time window or capacity constraints. We seek to sample a feasible route plan from the constructed graph. Our approach is motivated by prior work by Zalesak and Samaranayake [36]. Each edge in the RV graph represents an operation that creates an updated route plan for a vehicle that includes the new request. Additionally, each edge in the VV-graph represents a swapping operation which creates a new route plan for the two vehicles involved in the swap. Therefore, any selected edge in $G_{RV} \cup G_{VV}$ represents a feasible action. Additionally, multiple edges can be selected to generate a new action, *if and only if* the set of selected edges includes only one of the edges from the RV graph. The rationale for such a condition is straightforward; a new request can only be assigned to one vehicle, and consequently, only one edge from the RV graph can be selected for a particular feasible action at each decision epoch. To generate potential feasible actions quickly, we sample independent sets (a set of edges that have no vertices in common) from $G_{RV} \cup G_{VV}$ that includes one edge from G_{RV}. Such an independent set guarantees feasibility, as we show below:

Theorem 1. *Consider graphs G_{RV} and G_{VV} generated at decision epoch t. An independent set of edges from $G_{RV} \cup G_{VV}$ that includes one and only one edge from G_{RV} must be a feasible action for the current state s_t of the MDP.*

Proof. We first list the conditions for feasibility: 1) An action is feasible if it services the current request with adherence to time constraints, and 2) time constraints for existing requests are met (including potential swaps). An independent set with one edge from G_{RV} meets condition 1 by construction—all edges from G_{RV} service the current request and meet time constraints (recall that edges are checked for feasibility through the PDPTW solver). As the vehicle that services the request cannot swap requests (by the property of independence), all swaps sampled from G_{VV} are also feasible—all edges in G_{VV} are checked for feasibility through the PDPTW solver. Hence the set of independent edges must correspond to a feasible action for the given state. □

We point out that the vehicle that services the request *can* engage in swapping and still maintain feasibility; however, ensuring that such a vehicle does not participate in swapping *guarantees* feasibility. As a result, we use the heuristic of sampling independent sets (with one edge from the RV graph) that guarantee feasibility by construction. We present a more detailed description of why independent sets ensure feasibility in the technical appendix (section A.1.

Algorithm 1 Generating feasible actions

Require: $\theta_t, R_t, K_{max}, M, G_{VV}, G_{RV}$
1: $\Theta_t \leftarrow \{\}$ ▷ empty set of feasible actions
2: $\mathbb{U} \leftarrow \{\}$ ▷ empty set of utilities for each action
3: **for** $e_{RV}(i,j) \in E_{RV}$ **do**
4: $\theta \leftarrow \theta_t$
5: $\theta[i] = \theta(e_{RV}(i,j))$
6: $\Theta_t.append(\theta)$
7: $U_x \leftarrow U_\omega(e_{RV}(i,j))$
8: $\mathbb{U}.append(U_x)$
9: $G'_{VV} = G_{VV}[\{v_k \in L_{VV} : k \neq i\}]$
10: **while** $|E'_{VV}| > 0$ **do**
11: $e_{VV}(m,n,k) = \arg\max_{(m,n,k)}(U_\omega(e_{VV}(m,n,k))$
12: $\theta[m] = \theta(e_{VV}(m,n,k), v^m)$
13: $\theta[n] = \theta(e_{VV}(m,n,k), v^n)$
14: $\Theta_t.append(\theta)$
15: $U_x = U_x + U_\omega(e_{VV}(m,n,k))$
16: $\mathbb{U}.append(U_x)$
17: $G'_{VV} = G'_{VV}[\{v_k \in L_{VV} : k \neq m \ \& \ k \neq n\}]$
18: **end while**
19: **end for**
20: // Filter: X_t is the K_{max} actions with highest utility
21: $X_t = \Theta_t[argSort(U)[1 : K_{max}]]$
22: **Return:** X_t

Recall that our goal is to sample promising actions from the set of feasible actions since evaluating all feasible actions in an online setting is not tractable. We present our approach to selecting promising actions in algorithm 1. Our approach is based on sampling independent sets based on the cumulative sum of edge weights of the sets. First, we initialize an empty set of feasible actions Θ_t and an empty set of utilities $\mathbb{U}$. Then, we begin selecting an independent set by selecting an edge from G_{RV} (step 3). Next, we initialize the utility for the route plan (that the independent set corresponds to) with the utility of the chosen edge (step 7). Then, we drop the corresponding vehicle node from G_{VV} and consider swaps iteratively (step 9-10). The utility of the resulting action is calculated by adding the utility of serving the request and the swapping action (step 15). Finally, we return a subset of K_{max} actions with the highest total utility (step 21-22) to be evaluated by the tree search, where K_{max} is an exogenous parameter.

3.3 MCTS Evaluation

Non-myopic approaches to DVRP rely on hybrid offline-online solutions in which an offline component is trained on historical data and embedded in an online search [15, 32]. Offline components typically require long training periods and must be re-trained each time the environment changes, making them unsuitable for highly dynamic environments. This motivates us to use MCTS, an online probabilistic search algorithm, to evaluate the long-term utility of potential actions. MCTS is an anytime algorithm, and any changing environmental conditions that are detected can immediately be incorporated into its underlying generative models for making decisions.

MCTS represents the planning problem as a "game tree," where states are represented by nodes in the tree. The current state is treated as the root node, and actions represent edges that mark transitions from one state to another. The fundamental idea behind

Algorithm 2 MCTS evaluation

Require: X_t, S_t, E, n_{chains}
1: $eventChains = E.sample(S_t, n_{chains})$ ▷ sample chains
2: $A = MCTS(X_t, eventChains)$ ▷ action scores
3: $\bar{A} = \{\}$
4: **for** $a \in A$ **do parallel**
5: $\bar{A}.append(mean(a))$ ▷ aggregate across chains
6: **end for**
7: // Return action with highest action score
8: **Return:** $argmax_{x_i \in X}(\bar{A}[i])$

MCTS is that the search tree can be explored asymmetrically, biasing the search toward actions that appear promising. To estimate the value of an action, MCTS simulates a "rollout" to the end of the planning horizon using a *default policy*. In practice, the rollout policy only needs to be computationally cheap; a common method involves selecting actions randomly during rollout. As the tree is explored and nodes are revisited, each node's utility is estimated. As the search progresses, the estimates converge towards the true value of the node. This asymmetric tree exploration allows MCTS to search very large action spaces quickly. MCTS typically requires a few domain specific components: a generative model of the environment, the *tree policy* used to navigate the search tree, and the *default policy* used to estimate the value of a node. We describe each component below.

The role of the generative demand model (denoted by E) provides a method for sampling new requests as the tree is built into the future. We use a hierarchical modeling approach to sample trip requests based on historical data. We optimize model parameters based on maximum likelihood estimation. For the sake of brevity, we present a detailed description of the generative model in the appendix (section A.2). We use the standard Upper Confidence bound for Trees (UCT) [19] to navigate the search tree and decide which nodes to expand. When expanding a node, we use algorithm 1 to sample feasible actions for the given state. When working outside the MCTS tree to estimate the value of an action during rollout, we rely on a default policy. This is a lightweight policy which is simulated up to a time horizon and the utility of the simulation is propagated up the tree. Our default policy is a greedy assignment—for a given state, we choose the edge with the highest myopic utility from the RV-graph. To ensure tractability, we do not incorporate swapping requests between vehicles into our rollout policy; this saves time during MCTS evaluation as the VV graph is not generated during rollout.

Root parallelization: Given that the sampled paratransit requests can be both sparse and highly uncertain in time and space, sampling one chain of requests might not adequately represent future demand. To handle this uncertainty, we use *root parallelization*, which involves sampling many chains, and instantiating a separate MCTS tree for each with the current request as the root node. Crucially, each tree is explored in parallel. After execution, the score for each of the actions from the common root node is averaged across trees. Then, the action with the highest average score across the trees is returned as the selected action.

The process for evaluating and selecting an action is provided in algorithm 2. The algorithm takes X_t from the feasible actions component as well as the current state S_t, the generative model E and the

number of chains to sample n_{chains}. First, the set of *eventChains* is sampled from the generative model (line 1). Second, a tree is instantiated and MCTS is performed in parallel on each of the trees. This processed is represented by $A = MCTS(X_t, eventChains)$, where A is a two dimensional tensor of size $|X_t| \times n_{chains}$ (line 2). In this sense, each row in A represents a feasible action and the columns represent the chains. The mean of each row in A represents the action score and is stored in $\bar{A}$ (line 5). Finally, the action with the maximum action score in $\bar{A}$ is returned (line 8).

4 EXPERIMENTS AND RESULTS

4.1 Experimental Setup and Data Description

Paratransit dataset: We acquired six months of paratransit trip requests between January 1, 2021 and July 1, 2021 from CARTA. Our dataset consists of a total of 25,843 trip requests. Each request in our dataset consists of a geo-coordinate (longitude, latitude) for the pickup location and dropoff location, and also the requested pickup time.

Road network and travel time matrix: We use OpenStreetMap (OSM) [22] for the road network for the area under consideration and OSMNX [4] to generate a routing graph of the road network with travel time for edge weights. We describe our approach for generating the travel time matrix in the technical appendix (section A.3).

Data pre-processing: We processed the paratransit data and mapped each request to the road network graph. We describe preprocessing steps in the technical appendix (section A.3). The processed data spanned 129 days. We randomly selected 15 days from the dataset for the evaluation (test) set. The data from the rest of the days was used for training the generative model and hyperparameter tuning.

System parameters: Our experimental parameters are as follows: we vary the number of vehicles from 3 to 5 as we find that 5 vehicles can serve 100% of the trip requests in our setting using MC-VRP. The vehicle capacity is set to 8, which is in accordance with the capacity of typical paratransit vans. The request arrival time, defined as the time before the requested pickup time that the request is available to the system, is set to 60 minutes. In practice, such requests can also be made before 60 minutes. The time window, i.e., the amount of time before the requested pickup time that a request can be picked up is set to 15 minutes (in accordance with settings used by CARTA). Additionally, when a customer requests a trip we provide an estimated dropoff time which is the sum of the requested pickup time and the minimum travel time to the dropoff location. The late time window is 15 minutes after the estimated dropoff time which is again, in accordance with settings used by our partner agency. Additionally, we reiterate that as the time windows are hard constraints, a trip is only feasible if the passenger is picked up and dropped off between the early pickup window and late dropoff window.

Since our online approach is anytime, we vary the amount of time that is allocated to the algorithm for making a decision (referred to as runtime cutoff). Practitioners can vary this parameter to account for the maximum time they can afford to assign a request to a vehicle. Note that the passenger making the request need not wait for this duration to receive a feedback; whether a request can be serviced or not can be computed in less than a second using our approach (we only need to construct the RV graph to find at least one feasible action). We consider two variants of our proposed MC-VRP approach. ***MC-VRP (budget)*** uses our budget-based utility for scoring feasible actions in algorithm 1, while ***MC-VRP (PTT)*** uses the PTT-based utility for scoring feasible actions. The MCTS evaluation as outlined in section 3.3 remains the same for both variants.

4.2 Baselines

We evaluate the performance of MC-VRP against the following baselines. For all baselines, we use the same number of vehicles (we present results by varying the number of vehicles) and vehicle capacity as MC-VRP. We compare our approach to two myopic online approaches (greedy and MA-RTV) and one non-myopic hybrid approach (DRLSA).

- **Greedy assignment (greedy-PTT, greedy-budget):** Greedy assignment consists of first generating feasible actions according to algorithm 1 and then selecting the action with the highest utility. For comparison, we include greedy assignment using both the PTT and budget utilities. We refer to greedy assignment with the PTT utility as greedy-PTT and greedy assignment with the budget utility as greedy-budget.
- **Deep reinforcement learning-based vehicle routing [15] (DRLSA):** Joe and Lau combine deep reinforcement learning, which approximates a state-value function for routing, with a simulated-annealing based routing heuristic to solve the dynamic vehicle routing problem [35]. The state representation of DRLSA is based on the total cost of the planned routes of the vehicles. While the original approach is designed for cargo-delivery problems, we add pickup and dropoff constraints given our problem setting. Also, the problem formulation by Joe and Lau [15] seeks to minimize the sum of the travel and waiting times of all vehicles. It serves all requests and incurs a penalty cost for time-window violations. To apply DRLSA to our problem formulation, where time-windows constraints are strict, we take the route plans output by DRLSA and iteratively remove requests that violate time windows until a maximal feasible set of requests is left. Further, for a fair comparison, we run the DRLSA algorithm itself with shorter time windows but remove requests that violate the original time windows, which we found significantly improves the service rate of DRLSA in our setting. Finally, we point out that since there was no open-source implementation available for this approach, we implemented DRLSA for this paper.
- **Myopic and Anytime trip-vehicle assignment-based on RTV graphs [1] (MA-RTV):** We also compare our approach with an anytime algorithm that batches requests together and then maximizes the myopic utility for the specific batch of requests. For each batch, the algorithm creates an RTV graph by checking *shareability* between requests as well as requests and vehicles. Since the best passengers-vehicle pair is found in each batch, the approach works well in practice despite being myopic (our experiments confirm this). We refer to this baseline as *MA-RTV*. To adapt MA-RTV for our problem setting, we set the parameters as follows. First, as MA-RTV is not able to handle requests in advance, we set the request arrival time to the

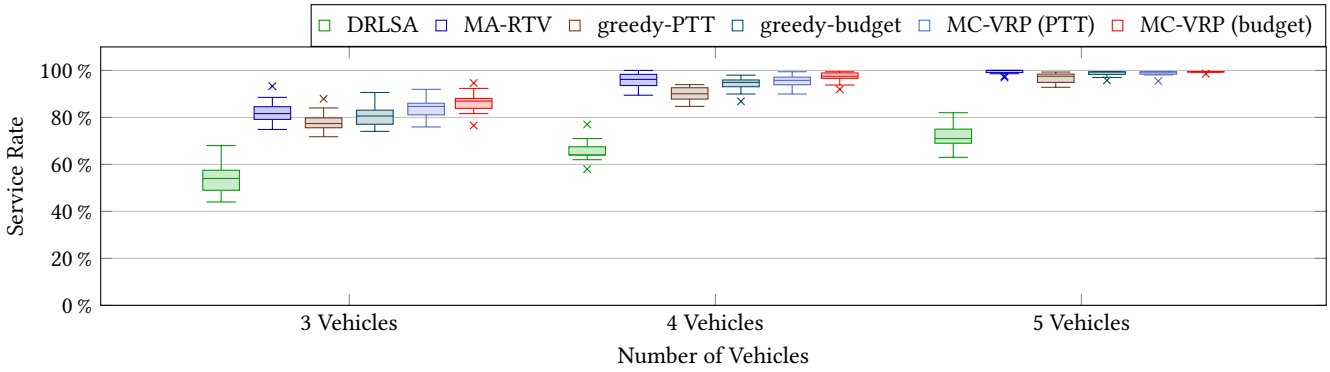

Figure 3: Service rate, defined as percentage of trips served, per day on 15 day test set for 3, 4 and 5 vehicles. 5 vehicles was enough to service all request on most days for MC-VRP (budget), MC-VRP (PTT) and MA-RTV. MC-VRP (budget) had the highest median service rate for 3 and 4 vehicles. The budget-based heuristic improves service rate for MC-VRP by 2.2% and 2.1% and greedy assignment by 3.4% and 4.9% for 3 and 4 vehicles respectively.

early time window. Second, we set the latest dropoff time to 30 minutes (15 minutes for early time window and 15 minutes for late time window) plus the shortest path between the pickup and dropoff locations. Therefore, the time window requirements are equivalent to our problem setup. Lastly, MA-RTV is a batching approach which waits for a set period of time before grouping requests together for assignment. In the paratransit setting, requests rarely arrive very close together and are expected to be handled one at a time, typically over the phone. Therefore we set the batch interval to 20 seconds, which closely mimics the observed paratransit data.

Parameter Tuning: For all baselines, we used 3 days of paratransit data for hyperparameter tuning. We present details about hyperparameter search in the technical appendix (section A.4).

Reproducibility: Our source code, implementation of the baseline approaches, and samples of our datasets are available online (https://github.com/smarttransit-ai/iccps-2022-paratransit-public). We implemented MC-VRP in the Julia programming language using the pomdps.jl framework [9]. Results presented in this paper were obtained using the Chameleon testbed supported by the National Science Foundation [16].

4.3 Results

Service Rates: Our primary objective is to maximize the number of requests serviced each day. The service rate per day for the 15 days in the evaluation (test) set is provided in figure 3 with fleet sizes of 3, 4, and 5 vehicles. We first present results in a setting where MC-VRP (budget) and MC-VRP (PTT) are allowed to run for 1000 iterations without early termination. First, we note that 5 vehicles is enough to service all of the requests on most days as MC-VRP (budget), MC-VRP (PTT), and MA-RTV all have a median service rate of 100%. We observe that MC-VRP (budget) outperforms all baselines for fleet sizes of 3 and 4 with with a median service rate of 87.0% and 97.6% respectively. MC-VRP (PTT) had a service rate of 84.7% and MA-RTV had a service rate of 81.7% for 3 vehicles. MA-RTV outperformed MC-VRP (PTT) in the case of 4 vehicles with a median service rate of 96.2% compared to 95.8%.

We also observe that the budget-based heuristic works better than the travel time-based heuristic for both Monte Carlo tree search as well as greedy assignment. Indeed, MC-VRP (budget) results in a higher service rate than MC-VRP (PTT) and greedy-budget outperforms greedy-PTT for all fleet sizes. This indicates that the budget utility, which aims to maximize time for which a vehicle has no passengers on-board, can be used to quickly compute promising actions to explore. It is important to note that while we focus on paratransit services, the budget-based heuristic can be applied for other DVRPs with capacity and time window constraints. We also observe that DRLSA had the lowest service rate across all fleet sizes. An influencing factor, as discussed in section 4.2, is that time windows are soft constraints in DRSLA which is not particularly suited to paratransit settings.

Computation Time: The computation time per request for MC-VRP and the baselines is shown in figure 4. MC-VRP (budget) takes slightly longer than MC-VRP (PTT) to compute a decision, with a median computation time of 38 seconds, 49 seconds, and 40 seconds as compared to 36 seconds, 44 seconds, and 37 seconds for 3, 4, and 5 vehicles respectively. DRLSA, MA-RTV, and greedy all had median computation times less than 3 seconds. In this context, the fact that MC-VRP is both non-myopic and fully online is reflected in the higher runtimes; while the observed runtimes are acceptable in our setting, i.e., paratransit services, the application of online and non-myopic methods remains an open question in general VRP settings. Note that if the MCTS evaluation of our approach is not used (i.e., the early stopping time approaches 0), our approach simplifies to the greedy baseline, which takes less than a second on average. This observation means that the majority of computation time in our approach is spent on MCTS.

As discussed in section 4.1, the MCTS evaluation can be run in the background between calls. Therefore, a potentially limiting factor for our approach is the rate at which requests arrive; even when running in background between requests, MC-VRP must be stopped on the arrival of a new request. Since MC-VRP is an *anytime* algorithm, it is possible to set a maximum computation time per request. In such a setting, the algorithm is stopped early (in our case, before the default number of iterations is reached), but

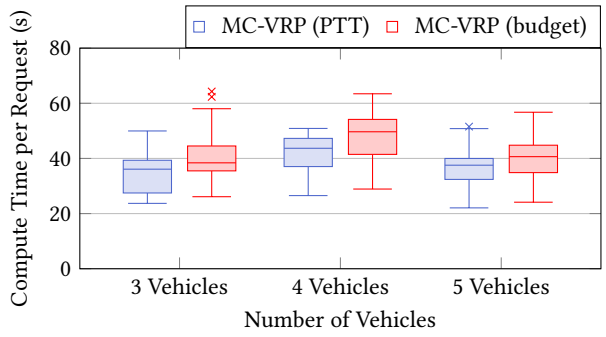

Figure 4: Computation time per request in seconds for each day in the test set for 3, 4 and 5 vehicles. Median computation time per request for MC-VRP (budget) is 38 seconds, 49 seconds and 40 seconds for 3, 4 and 5 vehicles respectively. DRLSA, MA-RTV and greedy all had median computation times less than 3 seconds per request.

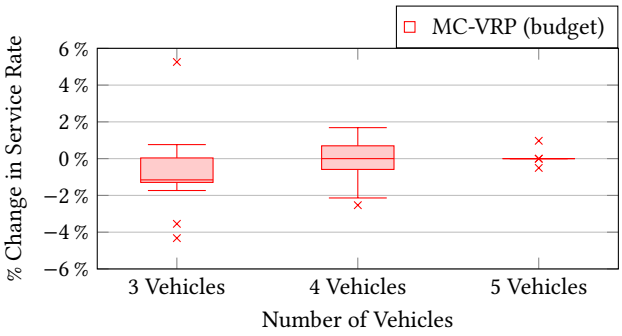

Figure 5: Runtime analysis: Percent change in service rate for MC-VRP (budget) with a cutoff of 30 seconds per request compared to no cutoff. For three vehicles there was a median of 1.2 percent decrease in service rate on test set while the computation cutoff had negligible affect for four and five vehicles.

outputs a feasible solution. In our dataset, 99% of requests arrive more than 30 seconds after the previous request and the median time between requests was approximately 5 minutes. Therefore, we evaluate the performance of MC-VRP with a 30 second cut-off time in figure 5. We observe only a negligible decrease in service rates; the median service rate decreases by 1.2% for 3 vehicles and 0% for 4 and 5 vehicles.

Robustness: To evaluate the robustness of the proposed approach with respect to changing environmental conditions, we change the travel time distribution in the city. For experiments, we assume the existence of a service that uses short term observations about an environmental variable (e.g., travel times) and provides an updated model. In practice, transit agencies can use services like Google maps for this purpose. In this case, when a user requests a ride, we can use the updated travel time matrix to give an accurate estimate of the dropoff time. We assume that all approaches have access to the updated travel time matrix at inference time, i.e. when a user requests a ride. However, approaches that rely on models trained offline do not have the ability to update their model. We generate a modified travel time matrix that resembles a congested road network. The distribution of the free-flow travel time matrix and the congested travel time matrix is shown in Figure 8. On average, the congested travel time matrix had speeds that were 30% slower

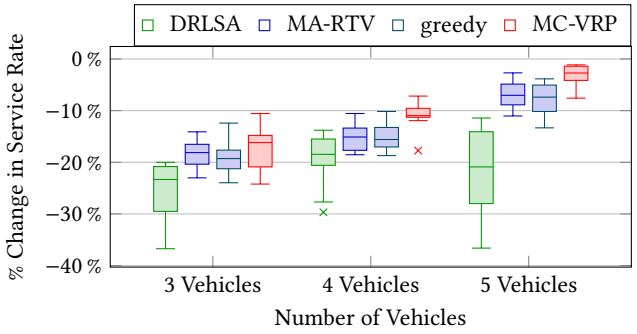

Figure 6: Robustness to roadway congestion: Percent change in service rate for DRLSA, MA-RTV, greedy-budget and MC-VRP (budget) on test set using congested travel time matrix compared to free-flow speed matrix. MC-VRP (budget) had less of a decrease in service rate for 3, 4 and 5 vehicle fleets compared to the baselines.

than the free-flow speed. We describe the process to generate the updated matrix in the technical appendix (section A.5).

We evaluated DRLSA, MA-RTV, greedy-budget and MC-VRP (budget) on the 15 day evaluation set using the congested travel time network and compared the resulting service rates to those generated without congestion. We observe that MC-VRP (budget) incurs less of a decrease in service rate compared to DRLSA, MA-RTV and greedy-budget across all sizes of vehicle fleets as shown in figure 6. As expected, DRLSA had the greatest decreases in service rate due to its reliance on a value function that was trained offline using the free-flow travel time matrix. While hybrid approaches such as DRLSA have access to the real-time travel conditions, it is difficult to re-train offline components, resulting in a higher degradation in performance. Both MA-RTV and greedy-budget outperform DRLSA when evaluated for robustness. This result is expected since MA-RTV and greedy-budget are both online approaches, it is able to adapt to the updated matrix. We also observe that MC-VRP (budget) performs better MA-RTV and greedy-budget, showing that our non-myopic online solution improved upon the myopic online approaches as well.

5 RELATED WORK

Vehicle Routing Problems (VRPs) can be broadly classified as either static or dynamic [26]. In static VRPs, all inputs are received before optimizing routes, whereas in dynamic VRPs inputs are updated concurrently with the determination of the route. The focus of this paper is on dynamic VRPs (DVRP), which can be either dynamic-deterministic, in which no stochastic information about future inputs is known, or dynamic-stochastic, in which some probabilistic information is known about the inputs that dynamically evolve [24]. These stochastic inputs can include models over travel times, demands, and customer information [27]. DVRPs can be solved to maximize myopic rewards [1] or a non-myopic utility function [15, 29]. Exact methods for solving DVRPs seek to find an optimal solution, but are often constrained to small problem instances due to computational complexity, e.g., such approaches include column-generation [5] and the set-partitioning method [6]. Metaheuristic approaches have also been applied to DVRPs, including particle swarm optimization [17], genetic algorithms [30], and tabu search [2, 11]. Decision theoretic approaches have also been

applied to DVRPs. DVRP is a sequential decision-making problem, and can be modeled as a Markov Decision Process (MDP). One approach is to use a conventional MDP structure, where actions consist of determining the next customer to serve at each decision epoch. Another approach, proposed by Ulmer et al. [31], is based on using a route-based MDP where the state-action space includes not only assigning an incoming request to a vehicle, but also optimizing the vehicles' routes.

There are three ways to solve the DVRP MDPs: offline, online, and hybrid solution methods. Offline methods pre-compute a policy that is queried while executing a plan [21, 29, 34]. Offline policies can be slow to learn, but can make decisions very quickly at execution time, and are therefore useful when there are strict time constraints on decision-making. Online solution methods perform computations during plan execution. These are generally sampling approaches, and only focus on the states of the system relevant to the current decision being made. Online methods include rollout algorithms [12, 28] and multiple scenario approach (MSA) [3]. Online approaches have typically been applied to problem settings without strict time constraints on decision-making (the time it takes to compute a decision), and in dynamic environments where policies generated offline can become stale. Hybrid solution methods attempt to combine offline and online approaches to leverage the strengths of both. For example, Ulmer et al. [32] propose an approach that embeds a value function learned using ADP into an online rollout algorithm. Joe and Lau [15] propose a similar approach, which combines Deep Reinforcement Learning to approximate a value function offline with an online simulated annealing approach.

6 CONCLUSION

We design a non-myopic online approach for DVRP for paratransit services that is robust to environmental changes by construction and scalable to real-world applications. To tackle the intractable action space, we leverage the structure of the problem to design heuristics that can sample promising actions for evaluation through MCTS. Our experimental results demonstrate superior performance and increases robustness against state-of-the-art baselines.

ACKNOWLEDGEMENTS

This material is based upon work sponsored by National Science Foundation under grant CNS-1952011 and Department of Energy under Award Number DE-EE0009212.

REFERENCES

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

[2] Andrea Attanasio, Jean-François Cordeau, Gianpaolo Ghiani, and Gilbert Laporte. 2004. Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. *Parallel Comput.* 30, 3 (2004), 377–387.

[3] Russell W Bent and Pascal Van Hentenryck. 2004. Scenario-based planning for partially dynamic vehicle routing with stochastic customers. *Operations Research* 52, 6 (2004), 977–987.

[4] Geoff Boeing. 2017. OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems* 65 (2017), 126–139.

[5] Huey-Kuo Chen, Che-Fu Hsueh, and Mei-Shiang Chang. 2006. The real-time time-dependent vehicle routing problem. *Transportation Research Part E: Logistics and Transportation Review* 42, 5 (2006), 383–408.

[6] Christian H Christiansen and Jens Lysgaard. 2007. A branch-and-price algorithm for the capacitated vehicle routing problem with stochastic demands. *Operations Research Letters* 35, 6 (2007), 773–781.

[7] George B Dantzig and John H Ramser. 1959. The truck dispatching problem. *Management Science* 6, 1 (1959), 80–91.

[8] Yvan Dumas, Jacques Desrosiers, and Francois Soumis. 1991. The pickup and delivery problem with time windows. *European Journal of Operational Research* 54, 1 (1991), 7–22.

[9] Maxim Egorov, Zachary N Sunberg, et al. 2017. POMDPs. jl: A framework for sequential decision making under uncertainty. *The Journal of Machine Learning Research* 18, 1 (2017), 831–835.

[10] Maria Gabriela S Furtado, Pedro Munari, and Reinaldo Morabito. 2017. Pickup and delivery problem with time windows: a new compact two-index formulation. *Operations Research Letters* 45, 4 (2017), 334–341.

[11] Michel Gendreau et al. 1999. Parallel tabu search for real-time vehicle routing and dispatching. *Transportation Science* 33, 4 (1999), 381–390.

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

[13] Lars M Hvattum et al. 2006. Solving a dynamic and stochastic vehicle routing problem with a sample scenario hedging heuristic. *Transportation Science* 40, 4 (2006), 421–438.

[14] INRIX. 2014. INRIX Interface Guide. (2014).

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

[16] Kate Keahey, Jason Anderson, et al. 2020. Lessons Learned from the Chameleon Testbed. In *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*.

[17] Mostepha R Khouadjia, Briseida Sarasola, Enrique Alba, Laetitia Jourdan, and El-Ghazali Talbi. 2012. A comparative study between dynamic adapted PSO and VNS for the vehicle routing problem with dynamic requests. *Applied Soft Computing* 12, 4 (2012), 1426–1439.

[18] Mykel J Kochenderfer. 2015. *Decision Making Under Uncertainty: Theory and Application*. MIT press.

[19] Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *European Conference on Machine Learning*. 282–293.

[20] Roy Lave and Rosemary Mathias. 2000. State of the Art of Paratransit. *Transportation in the New Millennium* 478 (2000), 1–7.

[21] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence V Snyder, and Martin Takáč. 2018. Reinforcement learning for solving the vehicle routing problem. *Neural Information Processing Systems* (2018).

[22] OpenStreetMap contributors. 2017. https://planet.osm.org .

[23] Geoffrey Pettet, Ayan Mukhopadhyay, Mykel J Kochenderfer, and Abhishek Dubey. 2021. Hierarchical planning for resource allocation in emergency response systems. In *International Conference on Cyber-Physical Systems*. 155–166.

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

[25] Warren B Powell. 2007. *Approximate Dynamic Programming: Solving the curses of dimensionality*. Vol. 703. John Wiley & Sons.

[26] Harilaos N Psaraftis, Min Wen, and Christos A Kontovas. 2016. Dynamic vehicle routing problems: Three decades and counting. *Networks* 67, 1 (2016), 3–31.

[27] Ulrike Ritzinger, Jakob Puchinger, and Richard F Hartl. 2016. A survey on dynamic and stochastic vehicle routing problems. *International Journal of Production Research* 54, 1 (2016), 215–231.

[28] Nicola Secomandi. 2001. A rollout policy for the vehicle routing problem with stochastic demands. *Operations Research* 49, 5 (2001), 796–802.

[29] Sanket Shah, Meghna Lowalekar, and Pradeep Varakantham. 2020. Neural approximate dynamic programming for on-demand ride-pooling. In *AAAI Conference on Artificial Intelligence*, Vol. 34. 507–515.

[30] Eiichi Taniguchi and Hiroshi Shimamoto. 2004. Intelligent transportation system based dynamic vehicle routing and scheduling with variable travel times. *Transportation Research Part C: Emerging Technologies* 12, 3-4 (2004), 235–250.

[31] Marlin W Ulmer et al. 2017. *Route-based markov decision processes for dynamic vehicle routing problems*. Technical Report.

[32] Marlin W Ulmer, Justin C Goodson, Dirk C Mattfeld, and Marco Hennig. 2019. Offline–online approximate dynamic programming for dynamic vehicle routing with stochastic requests. *Transportation Science* 53, 1 (2019), 185–202.

[33] Marlin W Ulmer, Dirk C Mattfeld, and Felix Köster. 2018. Budgeting time for dynamic vehicle routing with stochastic customer requests. *Transportation Science* 52, 1 (2018), 20–37.

[34] Marlin W Ulmer, Barrett W Thomas, and Dirk C Mattfeld. 2019. Preemptive depot returns for dynamic same-day delivery. *EURO Journal on Transportation and Logistics* 8, 4 (2019), 327–361.

[35] Peter JM Van Laarhoven and Emile HL Aarts. 1987. Simulated annealing. In *Simulated annealing: Theory and applications*. 7–15.

[36] Matthew Zalesak and Samitha Samaranayake. 2021. Assignment algorithms for on-demand high-capacity ridepooling. *Technical Report* (2021).

A TECHNICAL APPENDIX

A.1 Independent Sets and Feasible Actions

Recall theorem 1, which states that an independent set of edges from $G_{RV} \cup G_{VV}$ that includes one and only one edge from G_{RV} must be a feasible action for the current state s_t of the MDP. Notice that the only condition in which a chosen set of edges from $G_{RV} \cup G_{VV}$ might result in an infeasible action is the following: consider a situation in which the vehicle (say v^i) that is assigned the request (denoted by an edge from the RV graph) also engages in swapping and receives an additional request from a different vehicle (say v^j). While both the actions, namely the swapping action and the servicing of the new request, are checked for feasibility in isolation, vehicle v^i might violate time constraints if it seeks to service both. In theory, it is possible to enumerate over each possible set of edges in $G_{RV} \cup G_{VV}$ and check for feasibility; however, enumerating over all such sets is intractable in practice.

A.2 Generative Model

We model two processes as part of the generative demand model. First, we model the distribution of the number of requests per day as a Gaussian distribution. We learn the parameters of the distribution by maximizing the likelihood of historical paratransit data. Second, to model individual trip requests, we aggregate historical trip requests and weigh each trip request by the number of times it is observed (often, some passengers in paratransit services request trips that have the same source, destination, and time every week). To sample a sequence of trip requests for a day, we perform the following steps: 1) we sample the number of requests (say m) from the learned Gaussian distribution over trip requests. 2) We sample m requests from the weighted aggregation of trip requests.This process is repeated a number of times to generate multiple sampled *chains* offline. During inference at decision epoch t, we provide a method $E.sample(t, n)$ which samples n chains uniformly at random from E and returns the requests in each chain that occur after the current time t.

A.3 Paratransit Data Pre-Processing

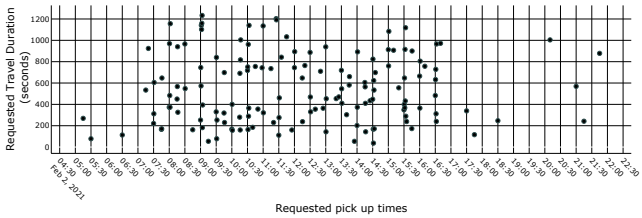

Figure 7: The temporal distribution of requests and the requested travel duration for the requests from one day.

Paratransit Data: Our dataset consists of a total of 25843 trip requests. As an example, we show the temporal distribution of one of the days and the requested travel duration for each request in figure 7. For pre-processing, each pickup and dropoff location was assigned to the nearest node in the road network graph. We used the haversine distance to calculate the distance between two geo-coordinates. We filtered out any requests that had a pickup or dropoff location that was farther than 200 meters from the nearest node in the road network; this process filtered out approximately 5% of the requests. Of the 25843 trips in the original dataset, 24543 remained after the distance filter. As paratransit services are significantly more sparse on weekends, we included only weekday trips for our analysis that spanned 129 days (94% of the trips in our data occur on weekdays).We randomly selected 15 days from the dataset for the evaluation (test) set. The remaining 114 days were use to compute 100 synthetic days worth of trip requests (i.e., chains) for the generative model using the procedure outlined in section 3.3. Three more chains were generated as a calibration set for parameter tuning.

Road Network and Travel Time Matrix: We use OpenStreetMap (OSM) [22] for the road network for the area under consideration and OSMNX [4] to generate a routing graph of the road network with travel time for edge weights. Unless otherwise noted, the experiments used the free flow speed as edge weight. Then, we calculated the shortest paths between all pairs of network to generate a matrix of travel times. The travel time matrix is generated offline and therefore, provides constant lookup time for querying travel times between arbitrary locations in the area under consideration. Additionally, we collected historical traffic data from INRIX [14] to estimate typical travel times during times of high congestion. We use congestion data for evaluating the robustness of the proposed approach.

A.4 Parameter Tuning

Table 1: Parameter settings

Parameter	Values
Number of vehicles (M)	3, 4, 5
Vehicle capacity	8
Request arrival time	60m
Time window	15m
Runtime cutoff	Inf, 30s
K_{max}	10
MCTS depth	20
MCTS iterations	1000
n_{chains}	25

MC-VRP has two parameters for tuning—the depth of the search tree and the number of feasible actions. MCTS tree depth, which is the number of future requests to consider from the generative model, was varied between $\{10, 20, 30\}$. The number of feasible actions to explore, denoted by K_{max}, effectively sets the maximum branching factor for MCTS. We varied this parameter between $\{10, 15, 20, 25, 30\}$. We performed a grid search using the calibration set and found the best parameters in terms of service rate (MCTS depth of 20 and $K_{max} = 10$).

The calibration set was also used to select the parameters for DRLSA. The neural network of the DRLSA baseline approach consists of 1 hidden layer with 64 neurons. The activation function of the hidden layer is rectified linear (ReLu) and linear for the output layer. We trained the network with a batch size of 32, the discount factor for future reward is 0.99, and the learning rate is 0.01. We found the best result with $K_{max} = 500$ for Simulated Annealing. Similar to MC-VRP, we used grid search to find the best parameters for DRSLA.

A.5 Congested Travel Time Matrix

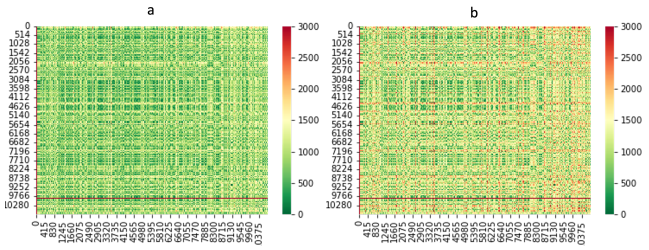

Figure 8: (a) Free-flow speed vs (b) Irregular travel times in seconds. Irregular travel times represent a congested roadway network. Each cell shows the travel time from a location x to another location y. There are total 10788x10788 combinations in the travel time matrix.

To evaluate the robustness of our approach to changes in traffic conditions we generated a second travel time matrix representing a congested road network. The original travel time matrix was calculated by finding shortest paths between all nodes in an OSM graph where the edge weights were travel time between the nodes using free flow speed. Each edge in the OSM graph had a unique OSM ID which mapped to roadway speeds in the INRIX dataset. The INRIX dataset included average roadway speeds per hour for each day in the week. For each OSM ID we took the speed at the 5th percentile and updated the edge weights accordingly. The distribution of the free-flow travel time matrix and the congested travel time matrix is shown in Figure 8. On average, the congested travel time matrix had speeds that were 30% slower than the free-flow speed.

A.6 Notation Lookup

Table 2: Symbols

Notation	Description
V	A set of vehicles each of capacity c
θ^t	The set of route plans for all the vehicles with θ_t^i denoting the plan vehicle $v_i \in V$
Θ_t	The set of all possible route plans at time t
$vloc_t$	A set consisting of locations of all vehicles at time t with $vloc_t^i$ denoting the location of $v_i \in V$
R_t	A new trip request at time t
$\mathcal{R}$	Ordered set of trip requests for a day
$[e_t, p_t]$	Service time window for request R_t
$C(\theta_i^m)$	Cost of vehicle m servicing at route plan
M	Set of vehicles in the paratransit fleet
S_i	State tuple $< T_i, vloc_i, \theta_i >$ for timestep i
E	Generative demand model
K_{max}	Maximum number of feasible actions to consider at each time epoch