An End-to-End Solution for Public Transit Stationing and Dispatch Problem

JOSE PAOLO TALUSAN, Vanderbilt University, USA
CHAEEUN HAN, Pennsylvania State University, USA
DAVID ROGERS, Vanderbilt University, USA
AYAN MUKHOPADHYAY, Vanderbilt University, USA
ARON LASZKA, Pennsylvania State University, USA
DAN FREUDBERG, WeGo Public Transit, USA
ABHISHEK DUBEY, Vanderbilt University, USA

Public bus transit systems provide critical transportation services for large sections of modern communities. On-time performance and maintaining the reliable quality of service is therefore very important. Unfortunately, disruptions caused by overcrowding, vehicular failures, and road accidents often lead to service performance degradation. Though transit agencies keep a limited number of vehicles in reserve and dispatch them to relieve the affected routes during disruptions, the procedure is often ad-hoc and has to rely on human experience and intuition to allocate resources (vehicles) to affected trips under uncertainty. In this paper, we describe a principled approach using non-myopic sequential decision procedures to solve the problem and decide (a) if it is advantageous to anticipate problems and proactively station transit buses near areas with high-likelihood of disruptions and (b) decide if and which vehicle to dispatch to a particular problem. Our approach was developed in partnership WeGo Public Transit, a public transportation agency based in Nashville, Tennessee and models the system as a semi-Markov decision problem (solved as a Monte-Carlo tree search procedure) and shows that it is possible to obtain an answer to these two coupled decision problems in a way that maximizes the overall reward (number of people served). We sample many possible futures from generative models, each is assigned to a tree and processed using root parallelization. We validate our approach with both real-world and scaled-up data from two agencies in Tennessee. Our experiments show that the proposed framework serves 2% more passengers while reducing deadhead miles by 40%. Finally, we introduce Vectura, a dashboard providing transit dispatchers a complete view of the transit system at a glance along with access to our developed tools.

CCS Concepts: • **Computing methodologies** → *Markov decision processes*; • **Applied computing** → **Transportation**; **Multi-criterion optimization and decision-making**.

Additional Key Words and Phrases: Public transit, Monte Carlo, Optimization

Authors' Contact Information: Jose Paolo Talusan, jose.paolo.talusan@vanderbilt.edu, Vanderbilt University, Nashville, Tennessee, USA; Chaeeun Han, Pennsylvania State University, University Park, USA, cfh5554@psu.edu; David Rogers, Vanderbilt University, Nashville, Tennessee, USA, david.rogers@vanderbilt.edu; Ayan Mukhopadhyay, ayan.mukhopadhyay@vanderbilt.edu, Vanderbilt University, Nashville, Tennessee, USA; Aron Laszka, Pennsylvania State University, University Park, USA, alaszka@psu.edu; Dan Freudberg, WeGo Public Transit, Nashville, USA, dan.freudberg@nashville.gov; Abhishek Dubey, abhishek.dubey@vanderbilt.edu, Vanderbilt University, Nashville, Tennessee, USA.

ACM 2378-9638/2025/11-ART43
https://doi.org/10.1145/3754454

ACM Reference Format:
Jose Paolo Talusan, Chaeeun Han, David Rogers, Ayan Mukhopadhyay, Aron Laszka, Dan Freudberg, and Abhishek Dubey. 2025. An End-to-End Solution for Public Transit Stationing and Dispatch Problem. *ACM Trans. Cyber-Phys. Syst.* 9, 4, Article 43 (November 2025), 26 pages. https://doi.org/10.1145/3754454

1 Introduction

As cities grow, so does the public demand for efficient and reliable transit systems [30]. In the wake of the COVID-19 pandemic, public transit ridership is recovering—now surpassing 70% of pre-pandemic levels [2]. However, this recovery brings renewed pressure on transit agencies, which are increasingly confronted with service disruptions caused by overcrowding, vehicle breakdowns, and traffic-related incidents. These disruptions reduce quality of service, erode public trust, and risk reversing gains in ridership. In 2022, WeGo Public Transit operated a fleet of around 100 buses, each assigned an average of 15 trips per day to serve thousands of passengers. That year, the agency recorded over 6,500 service disruptions due to weather, mechanical failures, or accidents. When disruptions occur, agencies typically rely on a small reserve of substitute vehicles, stationed either in garages or at stops across the city. The task of dispatching these limited resources is typically handled by human operators using intuition and experience—resulting in myopic decisions that do not account for system-wide implications or future uncertainties.

We refer to this as the **dynamic scheduling and dispatch problem for fixed-line transit systems**. Traditional approaches assume fixed schedules and regular intervals to ensure broad coverage. However, these assumptions break down under real-world conditions with stochastic disruptions and highly variable demand. Although prior work has proposed dynamic routing, deadheading, and stop-skipping strategies [13], most existing methods either oversimplify the decision space, ignore downstream effects, or require computational resources that limit real-time applicability. Moreover, few methods explicitly model uncertainty or provide tools that integrate seamlessly with existing dispatcher workflows. Solving this problem in practice requires reasoning over a large and highly dynamic decision space. The state space includes the real-time locations and statuses of all vehicles, scheduled and disrupted trips, and time-varying passenger demand. The action space is similarly complex, encompassing dispatch assignments, idle stationing, and vehicle reallocations, all under operational constraints. These challenges are exacerbated by the stochastic nature of disruptions and the rarity of extreme events, which limit the effectiveness of learning-based methods such as deep reinforcement learning. These methods often require extensive training data, struggle with sparse rewards, and lack robustness in high-dimensional, combinatorial environments. Given these limitations, we turn to a search-based approach: we formulate the problem as a semi-Markov decision process and solve it using Monte Carlo Tree Search (MCTS). MCTS enables efficient, non-myopic planning by simulating future scenarios and evaluating long-term outcomes, making it well-suited for this complex, uncertainty-laden setting.

This paper presents a principled, data-driven framework for dynamic dispatch under uncertainty, framed as a sequential decision-making problem. First, we introduce an online, non-myopic optimization method for dispatching substitute buses that aims to maximize overall utility—such as the number of passengers served—while operating under real-time constraints. Second, we model the problem as a semi-Markov decision process (SMDP) and solve it using Monte Carlo Tree Search (MCTS), which enables efficient, forward-looking planning over extended horizons. Third, we conduct extensive evaluations using both synthetic scenarios and real-world data from multiple transit agencies, showing that our method increases passengers served by up to 7% and reduces deadhead miles by 42% compared to a greedy baseline using alternate stationing strategies. Finally, we highlight our focus on practical operational integration by developing Vectura, a real-time

dashboard co-designed with WeGo Public Transit to support human operators in visualizing disruptions, monitoring system performance, and incorporating optimization algorithms into day-to-day workflows. By combining principled modeling, scalable algorithms, and practical tooling, our approach offers a meaningful step toward smarter, more adaptive transit operations.

2 Problem Statement

Table 1. Notation Lookup Table

Symbol	Definition
T	Set of trips i
L	Set of stops
$e(L_j^i)$	Scheduled arrival time for stop j on trip i
V	Set of all buses
V^R	Set of regular buses
V^O	Set of overload buses
S	Set of states
A	Set of all feasible actions
P	State transition function
T	Temporal distribution over transitions
R	Instantaneous reward function
γ	Discount factor for future rewards
$vloc_t$	Vehicle location
$c(V^i)$	Capacity of vehicle
$o(V_t^i)$	Occupancy of vehicle at time t
$b(L_j^i)$	Number of boarding for stop j on trip i
$w(V_t^i)$	Status of vehicle at time t
C	Explore and exploit parameter
E	Generative model

In this section, we describe the characteristics of fixed-line transit and formulate the dynamic stationing and dispatch problem. We then discuss the problem concepts, introduce the action spaces and finally model the problem as a semi-Markov decision process. We present a notation lookup table Table 1 for reference.

2.1 Fixed-line Transit and Substitute Buses

Fixed-line public transit relies on a set of vehicles following a consistent and predictable schedule of trips along a set of routes. Any events or incidents that would cause buses to deviate from their schedule can lead to increased passenger wait times, headway bunching, interfere with transfers, and eventually impact reliability and erode passenger trust. In response, the agency has a limited number of substitute buses, without any scheduled trips, that can be deployed to minimize the effect of unforeseen events.

Trip: A trip is a sequence of locations or stops that a bus will visit along its route. Each bus starts and ends its service day at a **depot**, which are central system hub. Formally, a trip i is defined by an ordered sequence of stops $T^i = \{L_0^i, L_1^i, L_2^i, \dots L_n^i\}$ where L_0^i and L_n^i are the depots if the vehicle is traversing a regular planned trip or staging areas if the vehicle was dispatched. Each stop has a predetermined scheduled arrival time along the trip.

Route + Direction: A bus route is a group of trips that a bus is expected to follow recurrently. A regular bus assigned to a particular route will travel in a certain direction until it turns around for the next trip going back. Regular buses will not diverge from assigned routes.

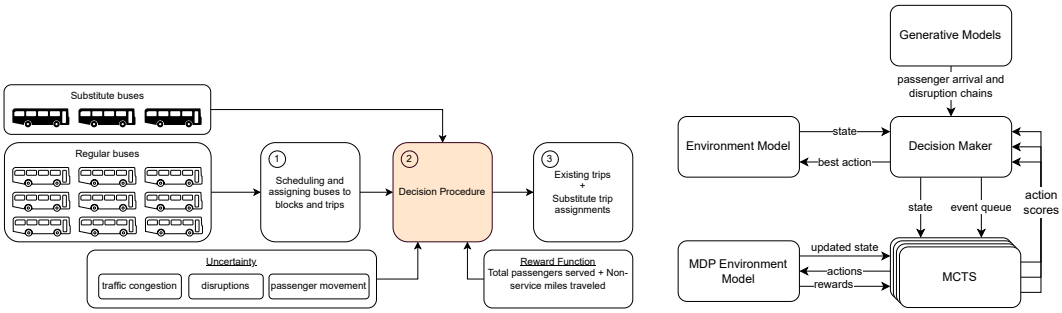
Stops: A trip i needs to pass through stops j at a scheduled time, denoted by $e(L_j^i)$. However, buses can encounter incidents along their trip that may cause delays or disruptions, preventing passengers from boarding the bus on time or preventing them from boarding at all, leading to dissatisfaction and frustration. If a bus arrives at a stop at or near full capacity, then some passengers will not be able to board leading to an **overage event**. On the other hand, if a bus breaks down or encounters an incident en route it leads to an **disruption event**.

Headways: One of the main goal for transit agencies is to maintain regular headways [10]. Headway refers to the interval of time between consecutive vehicles moving along the same route in a public transportation system. Headway is a critical measure in the operation of public transit as it directly influences the service frequency, which is the rate at which vehicles arrive at a stop. Maintaining consistent and optimal headways is essential for managing the flow of passengers, reducing wait times, and ensuring that the service is reliable. Regular headways allows for efficient use of resources, as vehicles can be evenly distributed along the route, preventing bus bunching and overcrowding on some vehicles while others remain underutilized. This balance ensures that operational costs are optimized and that each vehicle contributes effectively to the overall service. For passengers, consistent headways mean predictability and convenience. Shorter and reliable headways reduce the waiting time at stops or stations, making the service more attractive and user-friendly. This can lead to increased ridership and customer satisfaction, as passengers are more likely to use a service that they perceive as dependable and efficient.

For a regular service day, the transit agency has access to two types of buses, regular and substitute. **Regular buses** travel along their assigned routes for the entire service day. Regular buses do not diverge from their schedules and they cannot be used for other trips even when they are currently idle (between trips). In preparation for incidents, agencies may have several **substitute buses** that are kept apart from regular fixed-line services and are sent out in response to potential service disruptions such as overage events, delays, and vehicle breakdowns. Substitute buses can be *stationed* to certain stops along the city, effectively waiting until it has to be *dispatched* to either takeover a broken down regular bus or pick up passengers left behind by one. One of the main objective for using substitute buses is to maintain headways and prevent bus bunching. This effectively leads to a greater number of passengers served and less passenger wait time at the bus stops. Thus, it is important to station vehicles at locations close to routes and stops that frequently encounter issues, thereby maximizing substitute bus efficiency and decreasing response times. The efficiency of a substitute bus can be measured by the number of passengers it has served and the amount of *deadhead miles* it has traveled. Deadhead miles constitute the distance a substitute bus has to travel between where it is stationed to the dispatch location, during which they are not accepting any passengers. If there are no passenger overages or disruption incidents, then idle substitute buses can be re-stationed in preparation for future events. Ideally, there would always be a vehicle that can be dispatched to cover any affected trip at any time. However, resource constraints in both manpower and budget limit the total number of buses that can be used as substitute buses.

Therefore, a decision must be made. There are two types of actions the decision-maker, in this transit supervisors, can optimize: (1) which substitute buses to send to cover for trips (*dispatching actions*) and (2) which stops or depots to stage an idle bus in anticipation of future events (*stationing actions*). This dynamic stationing and dispatch problem is visualized in Fig. 1a, which is an extension of the traditional, fixed-line transit.

Formally, the dynamic stationing and dispatch fixed-transit problem consists of a set of vehicles denoted as V. A subset of vehicles are used to service the regular fixed-line transit schedule for the day, denoted as $V^R \subset V$ and the remaining vehicles are substitute vehicles available for stationing and dispatching $V^O = V \setminus V^R$. substitute buses start stationed at predefined stationing areas located



(a) Overview (b) Decision Procedure

Fig. 1. (a) An overview of the stationing and dispatch problem. ①Transit agencies have a set schedule for their buses as detailed in GTFS. A subset of buses is held back, to be dispatched in case of incidents, based on some ② decision procedure and some desired reward. Finally, ③ a list of assignments for all buses is generated. (b) Our proposed approach runs inside ② Decision Procedure. It creates generative models based on real-world data and outputs the optimal actions for trip assignments.

throughout the geographic region of operation. While the state space in this stationing and dispatch problem is in continuous time, it is convenient to view the system's dynamics as a set of finite decision-making states that evolve in discrete time. For example, a bus moving between stops on its trip continuously changes the state of the world but presents no scope for decision-making unless an event occurs that needs a response (such as a bus breaking down or passengers getting left behind) or the planner decides to re-station the buses. As a result, the decision-maker only needs to find optimal actions for a subset of the state space that provides the opportunity to take actions.

We define this moment of decision-making as a decision epoch. A decision epoch is triggered each time a bus arrives at a stop, has an overage or disruption event, or if a stationing event is triggered. At each decision epoch, there are three possible actions for each idle substitute vehicle: (1) They can be stationed and moved to **any** stop or depot across the region, called deadheading [13], (2) They can be dispatched to a stop on any trip. They must serve this entire trip, serving all subsequent stops, before becoming available for dispatch again, or (3) They can be made to wait at their current location.

2.2 Fixed-line Stationing & Dispatch as SMDP

To solve the problem of identifying the best assignments for substitute buses, we model the dynamic stationing and dispatching problem as a semi-Markov decision process (SMDP). We treat the model as semi-Markovian since buses must physically move from stop to stop along their trip or from staging areas when dispatched or reallocated. Travel time during these transitions are inconsistent due to latent factors such as traffic congestion, passenger behavior, or driver actions, which cause the temporal transitions between states to be non-memoryless.

An SMDP can be represented as the tuple $\{S, A, P, T, R, \gamma\}$ where S is a finite state space, A is the set of actions, P is a state transition function, T is the temporal distribution over transitions between states, R is an instantaneous reward function, and γ is the discount factor for future rewards [34].

States: We denote the set of states as S. A state at time t, denoted by s_t, consists of a tuple $(V_t, S_t, vloc_t, V_t^s)$ where V_t is the set of all vehicles, T_t is the set of all trips, $vloc_t$ is the location of all vehicles, and V_t^s is the state of each vehicle. We associate some additional information with each vehicle: $c(V^i)$ is the capacity of vehicle i, $o(V_t^i)$ is the current occupancy of vehicle i at t, and $w(V_t^i)$ is the status of the vehicle. The status of a vehicle can be *in transit* if it is actively servicing

a route, *idle* if it is sitting at a staging location ready to be dispatched, or *out-of-service* if it is no longer available. We assume that no two events occur simultaneously in our system model. If such a case arises, we can add an arbitrarily small time interval to separate the two events, creating separate states.

Actions: We denote the set of all feasible actions at time t by A^t. There are three types of possible actions for substitute buses: stationing, dispatch, and no action.

Dispatch actions selects an idle substitute vehicle and assigns it a trip. Thus, requiring them to be dispatched from their current location to a target stop along the assigned trip. A dispatch action (A_d) selects a vehicle v, trip i, and stop j along that trip given the current state of the system and can be denoted as $A_d : S \mapsto \{V^v, T^i, L^j\}$ where $V^v \in V^O$ and $L^j \in T^i$. The substitute bus must now visit all stops from L^j to L^n, where n is the last stop for trip j. The goal of dispatch actions is to directly alleviate excess demand from the system, by replacing broken vehicles or increasing capacity for crowded trips.

Stationing actions reallocates idle substitute buses by moving them between different stops around the city. The goal is to preemptively place idle vehicles closer to trips and stops that may encounter issues in the future. Stationing rebalances idle vehicles in anticipation of expected future demand. Therefore, a reallocation action can be formulated as $A_r : S \mapsto \{V^v, D^j\}$ where $V^v \in V^O$ and D^j is one of the possible stationing locations.

Otherwise, the system can simply *Do nothing*. In this case, no dispatching or reallocation actions are chosen and the system continues as is.

State Transitions: At each decision epoch t, the decision maker can take an action such as $a \in A_t$, which will transition from the pre-decision state s_t to a post-decision state s'_t. Afterward, the post-decision state s'_t transitions into the next pre-decision state s_{t+1}, within which new events will occur. During state transitions, buses travel between stops along their assigned trips, drop off passengers waiting to alight, and pick up any passengers waiting at the stop. Passengers waiting at a bus stop will get on the bus that arrives, provided that it is going along their desired route and direction, and if the bus is not yet over capacity $o(V^i_t) < c(V^i)$ Passengers who are not able to get on the bus will wait for another bus for a certain amount of time before leaving. Buses in transit can also encounter a disruption event while traveling between stops. We refrain from discussing the mathematical model and expressions for the temporal transitions and the state transition probabilities $\mathcal{P}$ since our algorithmic framework relies only on a generative model of the world and not explicit estimates of the transitions themselves.

We model the stop-to-stop travel times of buses by sampling an independent empirical distribution based on historical data. The number of boarding and alighting passengers at every stop is generated by generative models which will be discussed in the next section. Finally, disruptions are based on a statistical model trained to forecast incidents.

Rewards: The reward for an action $a^j \in A_t$, defined as $\gamma(a^j)$, is based on the total number of people that successfully get on the bus and the non-service miles traveled by substitute buses. It only considers any actions that maximize the number of people that a limited number of substitute buses can pick up while minimizing the non-service miles traveled. The decision maker's goal is to find an optimal policy that maps system states to actions to maximize long-term utility. In our case, the utility is related to reducing the overcrowding of the fixed-line transit system and mitigating the effects of vehicle breakdowns, traffic incidents, and passenger overages.

State and Action Space: The state space of our problem is inherently complex due to the combinatorial nature of the system. Remember that for a state at time, s_t consists of a tuple $(V_t, S_t, vloc_t, V^s_t)$ where V_t is the set of all vehicles, S_t is the set of all stops, $vloc_t$ is the location of all vehicles, and V^s_t is the state of each vehicle. Given discrete values for each one, the state space is $|\mathcal{S}| = |V_t| \times |S_t| \times |vloc_t| \times |V^s_t|$, reflecting the diverse combinations of bus states across the

Table 2. Detail of datasets

Data	Size	Rows	Features	Description
Automatic Passenger Count (APC)	3.3GB	17M	Date Load Arrival time Departure time	Date of trip Occupancy at stop Arrival at stop Departure from stop
Incidents	344MB	2.3M	Date Type Confidence Location	Date of trip Occupancy at stop Number of reports Coordinates
Disruptions	800KB	6534	Date Type Last stop Location	Date of trip Agency classification Last visited stop Coordinates
Road Network	31MB	7414	Network graph Geometry	OSM network Road Segment

Note: All data are from 01-01-2020 to 12-31-2022.

transit network. Encoding this high-dimensional, time-varying state space for machine learning-based approaches—such as reinforcement learning or neural network-based optimization—requires sophisticated state representation techniques to manage its combinatorial complexity, which is a research challenge in itself.

Similarly, the action space is prohibitively large, contributing to the difficulty of applying learning-based methods directly. For each of the V^O substitute buses, an action involves dispatching any substitute bus to one of L stops across the city or allocating it to one of $\hat{L}$ stops, where $\hat{L}$ is the number of possible stationing locations for standby in anticipation of disruptions. While only one bus will be either dispatched or allocated, there are $|\mathcal{A}| = (|V_t^O| \times |L_t^a|) + (|V_t^O| \times |L_t^d|)$, where V_t^O are the number of available substitute buses, L_t^a are the possible allocation locations, and L_t^d are stops that have an ongoing disruption. The need to optimize these decisions over time, while accounting for uncertainties such as real-time disruptions or demand fluctuations, further amplifies the computational burden. Designing a learning-based approach to navigate this action space efficiently would require advanced techniques, such as hierarchical decision-making or policy approximation, which are active areas of research.

3 Approach

The dynamic environment present in our problem discourages the use of typical offline-online solutions. Before being embedded in an online search, offline components require long training times and must be re-trained each time the environment changes. This motivates us to use Monte-Carlo Tree Search (MCTS). MCTS is a simulation-based search algorithm that has been widely used in game-playing scenarios. MCTS-based algorithms evaluate actions by sampling from a large number of possible scenarios. Being an anytime algorithm, MCTS can immediately incorporate any changes in the underlying environment when making decisions. We show the framework for our proposed approach in Fig. 1b. All of our code for the implementation is available at https://zenodo.org/records/10594255.

3.1 Available Data

Our solution for this problem is largely data-driven and we have been continuously collecting various datasets shown in Table 2. Automatic Passenger Counts (APC) are data gathered by electronic devices installed on public transit buses. They log the number of passengers boarding and alighting through the use of door sensors. They also collect the position of buses, as well as their arrivals and departures from stops, allowing stop-to-stop travel times to be derived. APC data is not as precise

Algorithm 1 Passenger Distribution Model Generation

Require: $e(L), features$
1: $E \leftarrow \{\}$
2: **for each** $(i, j) \in e(L)$ **do**
3: $\quad p = predict(e(L^i_j, features)$ ▷ using model from [28]
4: $\quad bin = randomBinFromProbability(p)$
5: $\quad e(\hat{L}^i_j) = randomInt(bin_{low}, bin_{hi})$
6: $\quad E(e(L^i_j)) \leftarrow e(\hat{L}^i_j))$
7: **end for**
8: **return** E

as data gathered by smart cards [22]. Thus, they require cleaning and augmentation before they can be used for training forecasting models [28].

We also collect both traffic incidents and bus failures, gathered by Waze and the transit agency respectively, to further improve our simulation of the real-world environment. However, these data do not neatly fit with APC data. We have to create a buffer in both space and time when joining them with bus APC data. Thus, they are simply approximations of the real world. Finally, the need to collect all of this data is further necessitated by the fact that APC does not collect origin-destination (OD) data. Due to the way they collect data, they do not have specific information regarding which stop a particular passenger got on and at which stop they got off.

3.2 Generative Models

Modeling the transition space and representing the stochasticity in our environment necessitates the ability to sample travel times, passenger arrival times, and potential disruptions.

Passenger arrivals: Due to limitations in the Automated Passenger Counter (APC) data, we are unable to directly observe passenger origin-destination (OD) pairs, counts and waiting times at individual stops. The dataset only provides current bus occupancy at each stop, without indicating whether passengers who attempted to board were successful or how long they waited.

Given the absence of more granular, passenger-level information in the APC data, we make two assumptions: First, that origins and destinations are independent. Second, that passengers have a predefined patience threshold for waiting for a bus before they leave the stop. Thus, passengers who are unable to board due to overcrowding either wait for the next available bus or are recorded as missed boardings, depending on capacity dynamics. We believe these are reasonable in the context of our study, since our primary objective is to maximize **aggregate ridership**—i.e., the total number of passengers served—rather than reconstruct individual travel paths. By focusing on matching aggregate boarding and alighting rates along routes, our simulation can still capture key system-level phenomena such as load profiles, boarding events, skipped passengers, and general ridership trends.

To estimate boarding and alighting counts, we developed a generative model that forecasts passenger occupancy across stops within a single trip (Algorithm 1). For each stop, the model predicts a probability distribution over occupancy bins; we then sample a bin based on these probabilities (line 4) and uniformly sample an occupancy level within that bin (line 5). This sampled occupancy is used to compute stop-level boardings and alightings, conditioned on contextual features such as weather, time of day, day of the week, and special holidays. To maintain consistency, only boardings are allowed at the beginning of trips, and all passengers are required to alight at

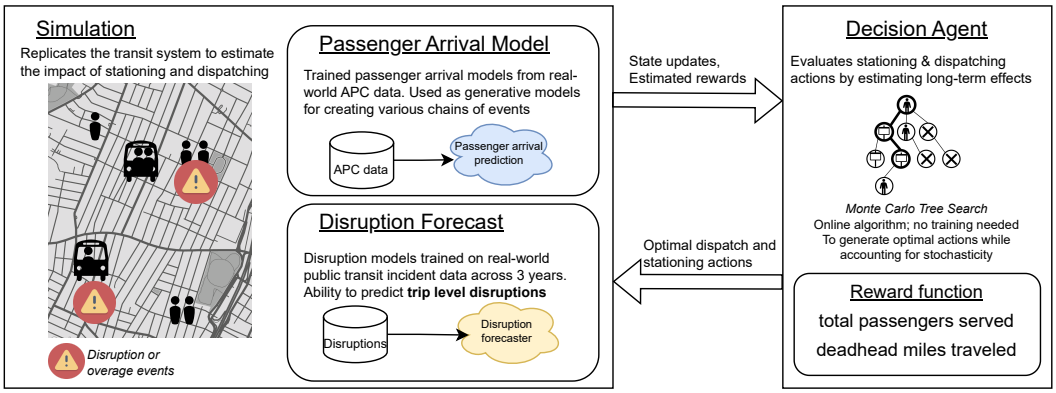


Fig. 2. Simulator diagram depicting the main components, passenger arrival model, disruption forecast, and decision agent.

the final stop. Stochasticity is introduced by generating multiple event chains for each service day and by sampling passenger arrival times from a uniform distribution up to the time of bus arrival.

Disruption forecasting: We approach the problem of forecasting the likelihood of disruptions occurring on a given trip as a supervised learning task [17]. We train an XGBoost model to predict the binary outcome of the presence of disruptions given a vector of features that include route, ridership, service-time, calendar variables, and weather data. XGBoost is favored for its robust performance and ability to capture complex nonlinear relationships [8]. It also provides a feature important score, offering valuable insights into the factors significantly influencing disruption likelihood. However, raw output probabilities do not accurately reflect the true likelihood of events. To address this, we apply post hoc calibration techniques, such as Isotonic Regression, to adjust the predicted probabilities, ensuring they match real-world disruption occurrences. This involves generating a simple probability distribution across all stops on the trip using empirical disruption data, allowing us to pinpoint the most probable stop where a disruption might occur.

Validation: The generative models used in this study are derived from two of our prior works, which include thorough validation against real-world data. In [16], we developed both trip-level and stop-level occupancy prediction models using XGBoost and LSTM architectures, respectively, trained on 17 million observations of Automated Passenger Count (APC), weather, and traffic data. These models were evaluated using root mean square error (RMSE) and binned classification accuracy, demonstrating up to 40% improvement in predictive accuracy over historical baselines and strong robustness across different time windows. While the disruption models are validated through p-value permutation tests that underscore its robustness and usefulness in forecasting disruptions in real world scenarios.

3.3 Event-Driven Simulator

The simulator recreates an entire service day using all the prior datasets and models. There are three main components: *buses, passenger arrivals, and disruptions*, shown in Fig. 2. All regular buses are assumed to start the day at the first stop for their first trip. Meanwhile, substitute buses start the day at the main garage. Any movement by substitute buses where they are not assigned any trips adds up to their deadhead miles. Regular buses have no deadhead miles.

This is an event-driven simulator, events are based on the bus' arrival at stops. Upon arriving at a stop, an active bus assigned to a trip will pick up passengers currently waiting at the stop. Since

there are several stops shared by different routes and directions, passengers will only board buses that are headed in the same route and direction that they are going to. Buses only have a limited capacity for passengers. An *overage event* is triggered when passengers cannot board the bus due to overcapacity. They then stay at the bus stop and wait for some determined amount of time before eventually leaving. They will wait for the next bus that will pass by after some headway duration. Passengers can only board at the start of the trip and are required to alight at the end of the trip unless the current trip is the first part of a multi-stage trip where passengers can choose to remain. At any time during the journey, a regular bus may encounter a *disruption event* that causes it to break down and be unable to continue on any of its assigned trips. At this moment, all passengers on the broken bus are assumed to transfer back to the last passed stop. These passengers will wait for the next bus and leave after a set amount of time.

During any overage or disruption events, if there are available substitute buses, the decision-maker can opt to dispatch one to service the stranded passengers. If a substitute bus is sent to cover for a broken-down bus, it travels from its current location to the last passed stop, accruing deadhead miles in the process. Once it reaches the stop, it takes over all the trips assigned to the previous bus and then fetches any passenger waiting at the stop (both stranded and newly arrived ones). This substitute bus is then locked to serve the remaining trips as the original broken bus. The original bus is considered broken for the entirety of the simulation. Once this substitute bus is finished with the assigned trips, it remains at the last stop, waiting to either be stationed or dispatched again. However, if a substitute bus is sent to recover stranded passengers due to overage events, it travels from its current location to the stop where the passengers are. From there. it will only visit all subsequent stops the original bus was assigned to for that given trip that caused the overage event. After reaching the terminal stop of this single trip, it then waits to be stationed or dispatched.

3.4 Decision Algorithm

In the simulator, optimal decisions are taken by an MCTS algorithm. Here stationing and dispatch problems are represented as game trees, where each node in the tree represents a state and each edge is an action that transitions from one state to the next. The current state of the environment is set as the tree's root node and the tree is incrementally expanded and asymmetrically explored. The asymmetric exploration of MCTS allows it to favor more promising nodes while other nodes still have a non-zero probability of being selected, this allows for relatively fast exploration of large action spaces. The value of an action is estimated by simulating a "rollout" to the end of a planning horizon while selecting actions based on a default policy.

In the simplest case, the default or rollout policy is computationally cheap such as selecting actions in a uniformly random manner. The tree is grown incrementally while adding and evaluating nodes and their utility until some computational budget has been reached. The algorithm then terminates and returns the best action for the current state. MCTS is a heuristic algorithm [6] that requires very little domain-specific knowledge to get acceptable performance. For our stationing and allocation problem, we provide an environmental model, a tree policy for navigating the search tree, and the default policy for producing a value estimate.

For every decision epoch, an asymmetric search tree is generated by MCTS to obtain the best action for any state. We describe below the flow of our MCTS implementation and the domain-specific components required at each step. The service day starts when buses start their first trip. The event chains constructed by the generative model (Sec. 3.2) provide the number of passengers waiting at the first stops for these initial trips. We use an empirical model to sample the travel times between stops based on historical data. As a bus travels between stops, we sample from

a distribution the probability of a trip getting disrupted either due to mechanical issues, traffic accidents, or passenger-related incidents.

Constraints on Decisions: To ensure tractability and to limit the number of decision epochs, due to the large amount of times buses will arrive at stops, we impose a strict interval between decision epochs. A bus arriving at a stop will only be considered for decision epoch for dispatch actions if it has been N minutes since the last time the particular bus has been considered as a decision epoch. Independent of this, a decision epoch for stationing actions will occur every M minutes from the start to the end of the service day.

Constraints on Actions: Upon reaching a decision epoch, there are only a limited amount of actions that are considered. For stationing decisions, then any idle substitute buses are considered for reallocation to any of the possible stationing locations. For dispatch decisions triggered by an overage event, we consider dispatching to all the stops that have been visited by the current bus on their assigned trip. This includes stops that have stranded passengers as well as the current stop the bus is currently on. We also assign a constraint that trips can only be serviced by one substitute bus. At each decision epoch, only a single substitute bus will be considered for any possible action. For decision epochs where there are no substitute buses available, no action will be taken. Finally, any stops that have been last visited by a recently broken-down bus will be considered in the action space. We use the standard Upper Confidence Bound for Trees (UCT) [19] to navigate the search tree and decide which nodes to expand:

$$\text{UCT} = \hat{X}_j + C_p \sqrt{\frac{\ln n}{n_j}} \tag{1}$$

where n is the number of times the current parent node has been visited, n_j is the number of times child j has been visited and $C_p > 0$ is a constant. Ties among child nodes are broken randomly. $\hat{X}_j$ is the estimated utility of state at node j.

When working outside the MCTS tree to estimate the value of an action during rollout, we rely on a default policy. This lightweight policy is simulated up to a time horizon, with resulting utility being propagated up the tree. The default policy only considers dispatching and it always selects the nearest idle substitute bus for dispatching.

Rewards: The overall reward is a function of the total served passengers and total deadhead miles for all substitute buses. A trip may be serviced by both a regular and substitute bus. Values are normalized by the total number of passengers (served plus those left behind) and total distance traveled by regular buses. Thus, the reward is calculated as:

$$\gamma = \alpha \sum_{i \in T} \sum_{j \in L} b(L_j^i) + \beta \sum_{v \in \mathcal{V}^o} d(v) \tag{2}$$

where $b(L_j^i)$ is the number of passengers who boarded the bus at stop j on trip i and $d(v)$ is the deadhead miles traveled by bus v.

Sampling Chain Generation: Running MCTS on a single chain of events would be unable to reflect the uncertainty that can occur in the day-to-day transit operations. We account for this stochasticity by introducing many passenger count event chains either sampled from the generative models or by adding noise to real-world data. We use *root parallelization* [7] to handle the uncertainty. In root parallelization, a large number of these chains are used to generate multiple trees in parallel at every decision epoch. The best action is decided by the node which has the highest average score across all trees. The process for evaluating and selecting an action is provided in Algorithm 2. Given the current state at time t and the generative model E, the algorithm samples n event chains. A tree is then instantiated and MCTS is executed in parallel (line 2). This returns

Algorithm 2 MCTS evaluation

Require: A_t, S_t, E, n_{chains}
1: $eventChains = E.sample(S_t, n_{chains})$ ▷ sample chains
2: $A = MCTS(A_t, eventChains)$
3: $\hat{A} \leftarrow \{\}$
4: **for** $a \in A$ **do** ▷ done in parallel
5: $\hat{A}.append(mean(a))$ ▷ aggregate across chains
6: **end for**
7: **return** $\underset{x_i \in X}{\text{argmax}}(\hat{A}[i])$

a matrix A of feasible actions as rows and event chains as columns. We average each row to get the action score over multiple event chains. Finally, the action with the maximum score in $\hat{A}$ is returned (line 7).

4 Experiments and Results

We evaluate the performance of the proposed framework's effectiveness on public transit data obtained from a major metropolitan area in the United States. We use real-world data collected and provided by our partner agency.

4.1 Experimental Setup and Data Description

APC Dataset: We used real-world APC gathered from both WeGo's entire fleet of buses over two years between January 2020 and April 2022 and another public transit agency, the Chattanooga Area Regional Transportation Authority (CARTA), located in Chattanooga, TN. We obtained 5 months of data from CARTA, from January 1, 2022 to May 1, 2022. The datasets consists of bus arrival and departure times at every stop along their route, passenger occupancy, and information on whether a particular bus was a regular service bus or an overloaded bus.

Travel time and distance matrix: We generated an empirical distribution of travel times between consecutive pairs of stops present in the dataset. We then used OpenStreetMap and OSMnx to generate distance and travel times between all pairs of stops within the target area. Traffic incidents are factored in using generative models.

Generative models: We used the entire APC dataset along with other spatiotemporal features such as weather, traffic, and holidays, to train predictive models for stop-level passenger occupancy. We used the models to generate probabilities for different ranges of possible waiting passengers at the stop. We then selected uniformly at random, the number of boarding passengers from within that range. We generated 40 distinct passenger arrival chains. We used 20 for tuning the hyperparameters and 20 for validating the framework.

Scaling up: we chose instead to synthetically increase the scale of our existing datasets. First, we identify how many regular and overload vehicles we are adding to the transit date input, V^{R+} and V^{O+}, respectively. We use this information to then select *uniformly at random* a number of groups of trips (also known as blocks) equivalent to the number V^{R+}. We then identify the minimum headway for this particular block and insert one vehicle such that the new headway would be halved. Thus, a route that serves a particular stop every one hour, would now send a bus every thirty minutes. For populating the passenger distributions on these new trips, we sampled the existing generative models [16] given the new temporal features. We scale the vehicle disruptions accordingly by querying our disruption model [17] based on the stops and route direction assigned to the new buses. However, by reducing the headway times we are potentially changing the problem qualitatively.

Table 3. Parameter Table

Parameter	Value	Description	Remark
Planning Horizon	1 hour	Distance in the future MCTS will simulate to	problem configuration
Decision Epoch Interval	0.25 hours	Time between forced decision-making epochs for agents	problem configuration
Passenger Wait Time	0.5 hours	Time before a passenger leaves the stop	problem configuration
Discount Factor	0.99997	Discount factor for future rewards	problem configuration
Number of substitute buses	5	Number of agents considered for decision-making	problem configuration
Event chains	20	Accounting for stochasticity	problem configuration
No. of MCTS Simulations	200	No. of MCTS iterations	hyperparameter
C	1000	Controls exploit & explore characteristics	hyperparameter

Consider that passengers who had to alight from a disrupted bus will be picked up sooner by the following scheduled regular buses, which reduces the impact of dispatching a substitute bus to pick them up. This will eventually result in metrics proportional to the previous experiments. Thus, we decided to artificially inflate the number of passengers per stop. Finally, the run time and per iteration time increased as consequence of the increased scale. The disruptions and overcrowding which triggers the MCTS increased, resulting in more search queries. Within every iteration of search the increased number of substitute buses, necessitated an increase in the number of iterations.

System configuration: First, we constrain the number of stationing stops to 25, instead of every available stop in the environment. This decreases the action space making search tractable while keeping it close to the real world by distributing it evenly across the city. The 25 stops are selected based on discussions with the transit agency. They have flagged these stops as both the current and potential locations where issues often occur. We also select the interval to be 15 minutes between decision epochs. We assume that there are 5 available overload buses to allocate, which is the average number of buses the agency attempts to keep in reserve. Second, there is no limit to the number of disruptions that can happen each day, based on the event chains, there are typically 2-4 disruptions per day. Third, we assume that passengers will arrive at the stop within a 10-minute window before the bus is scheduled to arrive and will wait at the stop for 30 minutes before leaving. A bus that comes early at the stop will wait until the scheduled departure time, while a bus that arrives late will pick up passengers, if there are any, and leave immediately. Finally, we assume that all buses start and end their service at the garage. We ran experiments on the Chameleon testbed using Intel Xeon Platinum 8380 machines, with 160 cores running at 2.30GHz with 251GB RAM [18].

Baseline Greedy Approaches: We evaluate the performance of our approach against a baseline greedy approach. We used the same number of vehicles, vehicle capacities, incidents, and passenger event chains. According to the agency, they will usually dispatch substitute buses, in a greedy manner if they are available (taking into account domain expertise). Thus, for our greedy baseline, if a bus leaves passengers due to overcrowding, a substitute bus will be dispatched immediately, if the number of people left amounts to 5% of the last bus' capacity. Also, a substitute bus will be sent immediately, given enough resources, to cover for a broken down bus. In the greedy approaches, there will be no stationing aside from the initial assignment at the start of the service day. There are

four possible initial stationing configurations: *GARAGE*, *AGENCY*, *SEARCH*, *BAYESIAN*. GARAGE which represents a no stationing plan where all substitute buses start from the garage parking lot and AGENCY which represents the current stationing used by the transit office based on historical data and operator expertise are simple baselines while SEARCH and BAYESIAN are solutions generated by random local search and Bayesian Optimization respectively.

Random Local Search: SEARCH is the result of performing random local search with simulated annealing [31] to find the initial stationing configuration that would serve the most passengers while traveling the least amount of deadhead miles. All substitute buses are initially stationed in the garage. At each iteration, a neighboring solution is generated by making a replacing the stationing location of a single bus. We assume that there is at most a single substitute bus at each possible stationing stop. Both the current and neighboring solutions are tested on multiple unseen event chains for a given transit date. Each transit date has the potential for disruptions and overcrowding drawn from the same incident and passenger count distributions as the main event chain. These results are aggregated, normalized and used to determine the configuration's performance. If the new solution is better, it is accepted as the current solution. If it is worse, it may still be accepted with a probability that decreases exponentially with the difference in objective function values and inversely with the temperature. The best solution after a certain number of iterations is then selected as the initial stationing configuration for experimental evaluation.

Bayesian Optimization: Bayesian Optimization uses a probabilistic surrogate model, in this case a Gaussian Process (GP) and it iteratively refines the bus assignment configurations. The GP model uses a custom kernel to measure the similarity between different bus assignment configurations, enabling the model to predict the objective function values and associated uncertainties based on the observed data. We compute the shortest path, weighted by travel time, for all pairs of bus stops across the entire city. We use this travel time matrix to get the average travel time for any configuration of stationing assignments for any set of substitute buses to all other stops. We then compute the mean distance to each stop for a chosen configuration and uses this vector as our kernel.

$$\mathbf{R} = \begin{bmatrix} r_{00} & r_{01} & \cdots & r_{0j} \\ r_{10} & r_{11} & \cdots & r_{1j} \\ \vdots & \vdots & \ddots & \vdots \\ r_{k0} & r_{k1} & \cdots & r_{kj} \end{bmatrix} \tag{3}$$

$$kern\mathbf{R} = \frac{1}{j}\sum_{w=1}^{j} \mathbf{R}_{k,j} = [\overline{r_0}, \overline{r_1} \cdots \overline{r_k}] \tag{4}$$

where r is the travel time between stop k to stop j and $\overline{r_k}$ is the average travel time for all substitute buses to stop k.

Parameter tuning: We used one week of data for tuning C, the parameter for adjusting exploration over exploitation, and IT, the number of MCTS simulations per decision epoch. This was done both in the interest of time and since public transit follows a weekly schedule resulting in a consistent pattern. We used 20 of the 40 event chains for training. Each parameter affects the performance of the approach over the greedy baseline shown in Fig. 3. The initial value of C was selected by identifying the mean of $\hat{X}_j$ and iteratively decreasing it. We select C which has the highest improvement over baseline with an acceptable balance of exploration and exploitation. We then select the highest number of MCTS simulations that would still allow us to complete a decision epoch in real time. We show in Fig. 5a how adjusting this parameter affects the time per decision epoch. The final parameters and configurations used are listed in Table 3.

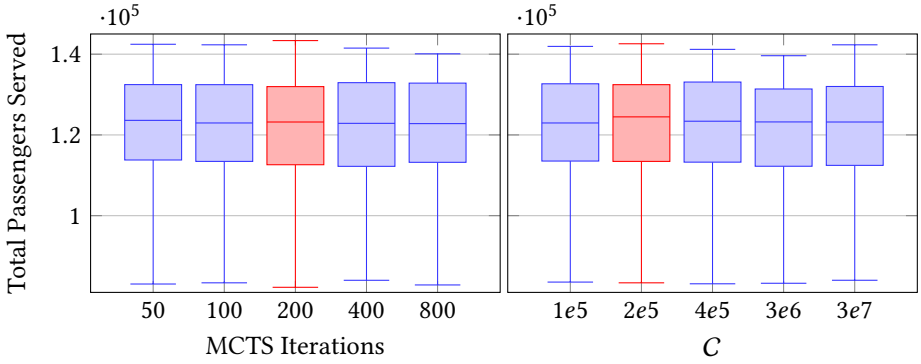


Fig. 3. Parameter search for C and number of MCTS simulations. Selected parameters are colored red, selected due to the highest average passengers served and run time.

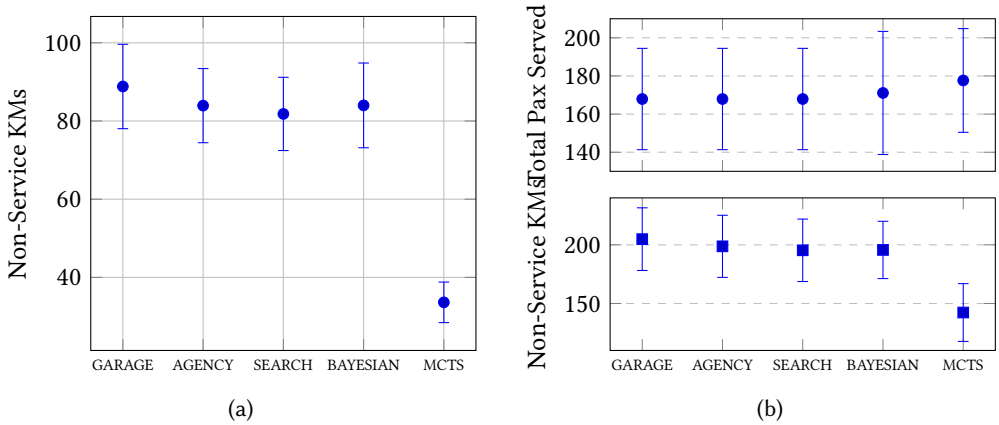


Fig. 4. Performance comparisons. (a) Comparing deadhead miles of the approach against the different stationing baseline on the real-world data. MCTS_5 is the same approach except decisions are done at 5-minute intervals compared to 15-minute intervals. (b) Comparing the performance of our approach to baselines when both the real world environment and MDP environment are sampled from generative models.

4.2 Experimental Scenarios

We ran experiments on two weeks of data unseen during hyperparameter tuning and compared the baseline policies with our proposed approach. We ran two sets of experiments, shown in Table 4. First, we used a generative model to represent the real-world environment and used event chains sampled from the generative model as the MCTS environment. Next, we validated our approach with an environment based on actual empirical data and chains based on real-world passenger counts with Gaussian noise applied to it and the models sampled from the generative model. The passenger arrival chains sampled from the generative models have a much larger passenger count, due to how the generative model was trained.

4.3 Results

The results for total passengers served are shown in Fig. 4b. The first observation is that by using the proposed framework, we improve the total number of passengers served by 10 passengers on

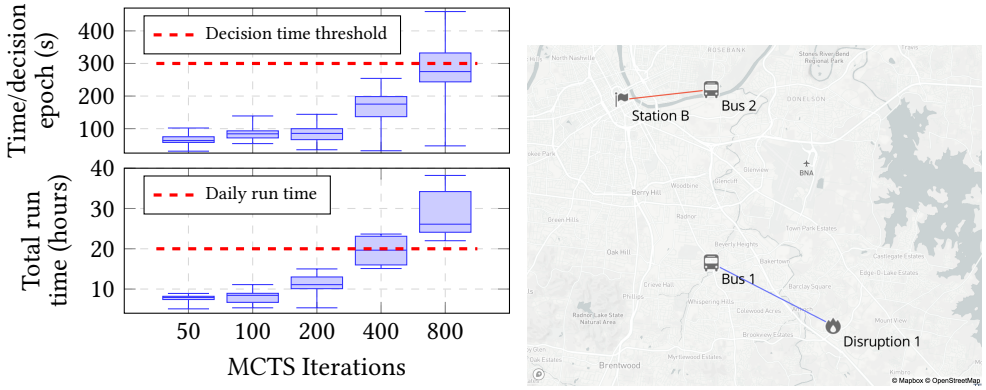
Table 4. Experimental Scenarios

Scenario	Env.	Chains	Traffic	Breakdown
Full generative	Generative	Generative	True	True
Noisy Real	Real-world	Noisy real	True	True
Generative Model	Real-world	Generative	True	True

average. This number can increase further when there are more instances of overage events and fewer disruption events. Since a bus that has been assigned to take over a broken bus is essentially unable to be dispatched to overage events, it reduces the window for optimization.

We also observe a significant reduction in total deadhead miles traveled by the overload buses when we perform dispatch using the proposed approach. We save on average around 50 kilometers per overload bus. This reduction in deadhead miles seen in Fig. 4b, is not unexpected. This can be attributed to the fact that our approach does not dispatch buses when the estimated reward is minimal, which leaves it available for future events with a larger reward.

The ability to make non-myopic decisions for when and where to allocate and dispatch overflow buses results in our approach performing better than the baseline. Consider a major overage at some time in the future. The baseline will continuously dispatch buses for minor overages before this moment, effectively exhausting its resources and leaving it unable to serve the major incident. These scenarios are effectively handled by our framework.



(a) (b)

Fig. 5. (a) Measuring processing time per decision epoch and overall run time for an entire service day. (b) Dispatch and Stationing UI on the dashboard. At this time interval, Bus 2 is to be stationed at Station B and Bus 1 should be dispatched to a disruption.

Real-world effectiveness: The end goal of our work is to be integrated into a multi-purpose tool that public transit agencies, for any major metropolitan city, can use to improve their system. Thus, we evaluate the effectiveness of our framework in a real-world environment. We use empirical data as the real-world environment and sample chains from generative models for the simulation environment. Since the passenger overages and disruption events are very sparse in the real-world data, both the baselines and our MCTS approach can serve the maximum feasible amount of passengers that can be served. However, Fig. 4a shows that our approach can outperform all baselines in terms of total deadhead miles traveled by substitute buses. This is to be expected,

the strength of our approach lies in how well we can estimate and simulate the conditions of the real world using generative models. This savings in distance traveled translates to savings in manpower and resources for the agency. An initial UI implementation of the stationing and dispatching algorithm is shown in Figure 5b. At any given time, transit dispatchers can see which buses are selected by the algorithm for either stationing or dispatching. Since the algorithm is an anytime and online solution, dispatchers will see this information in real-time.

Computation Time: Improving upon their stationing and dispatch methodology is only realistic if our online approach can produce results within the time that they must make their decisions. Fig. 5a shows how adjusting the number of MCTS simulations affects both processing time decision epoch and overall run time. The figure above shows that keeping the number of MCTS simulations below 400 will ensure that decision epochs finish at or below the five-minute interval that MTA uses as their decision intervals for their decision-making. The figure below shows the total run time in real-time for our approach to completely simulate the entire service day and provide optimal actions at each decision epoch. In our experiments, the entire service day lasts on average around 20 hours (04:00 am to 24:00 am). Thus, running 200 MCTS simulations per chain lets us finish all simulations and decision-making in 20 hours.

However, it should be noted that the total run times and computation costs are also subject to several other parameters shown in Table 3. Increasing the number of event chains may make the framework adapt to randomness in the environment, but also increase computation. Increasing the number of simulations may allow the MCTS to converge to a more optimal solution while increasing run times. Our approach is largely bounded by CPU, with minimal memory and disk requirements during run time. The desire to be able to perform in real-time for actual deployment is one of the reasons we selected certain hyperparameters such as the number of MCTS simulations and chains. Based on our results, 20 chains is an adequate lower bound for the number of event chains. Each chain takes one CPU core and is limited by the CPU clock speeds. Also, 200 MCTS simulations provide acceptable results while keeping the decision epoch below the decision-making time suggested by the partner agency. For larger cities, with different transit requirements, our approach may be able to perform better by varying different hyperparameters.

Table 5. Performance comparison across different demand multipliers (CARTA).

Agency	Multiplier	MCTS Served	Baseline Served	Total Passengers	Total Runtime (seconds)
WeGo	1	11,265	11,262	12,711	22,598
WeGo	1.5	18,843	18,696	21,284	46,265
WeGo	2	21,863	21,623	25,422	69,272
CARTA	1	4,981	4,981	5,848	4,249
CARTA	2	9,922	8,870	10,604	17,360
CARTA	2.5	10,589	9,540	11,703	64,730
CARTA	3	14,519	13,234	15,906	85,124

Scaled Inputs: The results in Table 5 show that when we increase the scale by increasing the number of trips, vehicles, and passengers, our approach still outperforms the baselines. The benefits of using our approach vary with different agencies with their own unique passenger distributions. When applied to another agency, CARTA, our approach can still outperform baselines. However, benefits only start showing up when we increase the scale. This means that for cities with minimal disruptions or low ridership, our approach performs similarly to the baseline.

However, an increased scale introduces significant computational costs. Several approaches can be done to improve it. First, the current implementation is in Python, which, while flexible for development and experimentation, is not as computationally efficient as lower-level languages like C++. Second, we can explore implementation improvements such as parallelization, early

pruning strategies, progressive widening, or hybrid heuristic-MCTS policies to further control computational costs. Finally, for future works, we can explore the possibility of using learning-based approaches to solve the stationing and dispatch problem. However, considering the non-trivial requirements of such an approach, it will require substantial effort to train due to the large problem space.

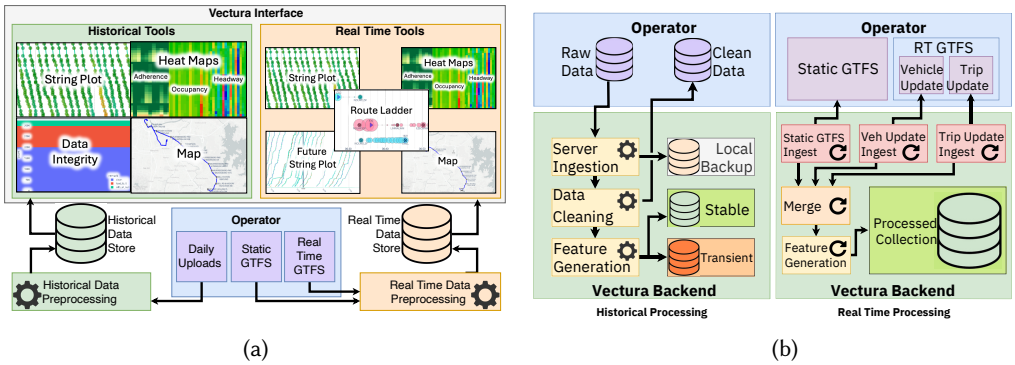


Fig. 6. (a) System Overview of Vectura, including the two main components: Historical and Real Time Analysis Tools. Data is collected from the transit agency and processed to generate the plots and for use in the algorithms. (b) Processing pipelines for both the historical and real time analytics and visualization of Vectura. Incoming data are from transit agencies.

5 Vectura: Real-world Dashboard

Vectura is built in close collaboration with transit dispatchers. This is designed to augment the existing tools used and not a replacement. This is an engineering effort to expose our algorithmic contribution and is tailored to facilitate the transition from theoretical analysis to everyday application as smoothly as possible. The main components of Vectura are the historical and real time tools shown in Figure 6a. The processing pipelines for both the historical and real time data is shown in Figure 6b. All GTFS data are generated by the transit agencies, with real time GTFS (GTFS RT) being generated every 10 seconds while Static GTFS are updated only when the transit agency needs to update their transit network. Vectura captures live data streams, processes it in real time, and extracts information on the current transit headways.

5.1 Historical Data Processing

The bus operations and passenger count (APC) data feed provides the bulk of the information used in historical visualizations. This data is inherently temporal and spatial, reflecting the movement of buses across routes and the fluctuation of passenger numbers over time. Specifically, the APC data, trip information, and vehicle information. Vectura employs a robust data pipeline that processes new information nightly shown in Fig. 6b. The pipeline is designed to ensure that the most current and relevant data is available for analysis. It incorporates data standardization, sorting, and cleansing steps to maintain repeatability, integrity, and accuracy. The pipeline is equipped with feature generation capabilities that convert raw data into standardized formats that are more representative of desired metrics.

Ingestion: An important part of the processing step is the separation of stable and transient data. Stable data is unlikely to be updated and therefore, does not need to be reprocessed daily. Transient data is likely to be updated as more information is exported from the busses so it must be

reprocessed daily to ensure that the most up to date information is presented to the user. Through discussions with operators, a threshold of 30 days was determined for this separation.

Data Cleaning: Once the data is ingested and sorted into stable and transient databases, The server starts the data cleaning process prior to generating the visualization. Real world data is never perfect. It is often full of noise, inaccuracies and caveats. The bus data is no exception. In order to provide accurate information about operation, this must be accounted for. To address this, we implemented a data cleaning pipeline [16] that labels data that may be inaccurate so it can be handled downstream.

5.2 Real Time Processing

The real time data feeds come from several data sources that intend to provide an equivalent data feed to the historical APC data, but with a low latency to ensure up to date information. These sources include GTFS static, GTFS RT vehicle updates, and GTFS RT trip updates. GTFS static contains data that outlines the transits fixed elements, such as route schedules and stop locations. GTFS RT vehicle updates deliver live positional information of buses, including load factor, latitude, longitude, bearing, and speed. GTFS RT trip updates supplement this by providing predicted arrival times for all stops in each active trip. Together, these real-time feeds offer a holistic view of the current state of transit operations, allowing for immediate response to operational challenges.

Ingestion: The process for the real-time data is designed to maintain data freshness and reliability. The GTFS static data is ingested daily, reflecting its relatively stable nature and the anticipation of schedule changes at least 24 hours in advance. The GTFS RT vehicle and trip updates are ingested every 20 seconds. This cadence ensures that the Vectura system captures the most up-to-date information without overloading the system. As stop times from the trip update feed are predictions, special care was done to determine the accuracy of this data when compared to the APC data. The GTFS RT specification does not mandate the removal of past stops from the feed. However, analysis revealed that these lingering stops, would have continuously updated stop times, which would deviate from APC data. To address this, a very specific insert operation is performed, only inserting stops where the stop time is closer to the current time than what is in the system already. To optimize downstream query performance, the system is designed to remove vehicle data entries once they have been merged in the next step. Trip update entries that have been merged are also removed, provided they are no longer present in the feed. This data management strategy ensures the database remains streamlined and responsive, supporting the real time analytical capabilities of Vectura.

Merging: The merging process within the real time data pipeline is the operation that synthesizes the various data streams to present a unified and actionable dataset. Scheduled to run every 40 seconds on a dedicated thread, this process integrates the real time vehicle information information from GTFS RT vehicle updates with the trip information from GTFS RT trip updates, and aligns them with the corresponding static GTFS data. During the merge, the system checks for congruence between the real time updates and the static schedule. Any discrepancies, such as stops or trips without matching entries in the static GTFS dataset, are logged. These anomalies are compiled into a CSV report and sent via email to transit operators on a daily basis. This reporting mechanism ensures that operators are promptly informed of any mismatches, allowing for swift resolution to the static or RT GTFS feed. For the vehicle updates, which can not be simply merged with a particular stop, a temporal buffer is used. This technique associates the vehicle's data with the temporally closest stop time, provided that the stop time falls within a 30-second window. This approach ensures that the load information is attributed to the closest point in the trip, maintaining the integrity of the merged dataset.

Feature Generation: The final step is feature generation, where merged data is processed to extract meaningful metrics that can be readily used for analysis. This processes runs every 60 seconds in a dedicated thread.

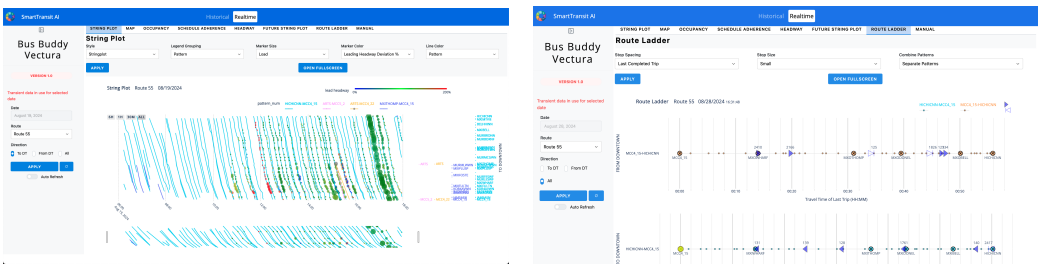
5.3 Feature Generation

Vectura augments raw transit data by generating a suite of features crucial for in-depth analysis and visualization within its tools. These features, which are not present in the original datasets, are key to enabling Vectura's interactive visualizations and giving transit operators a clearer view of service performance. One important feature is the calculation of *headways*, which are derived by examining sequential stop times on the same route and direction, allowing for comparisons across service patterns that share stops. Each stop time includes both a *leading headway*—the time since the previous stop—and a *trailing headway*—the time until the next one—providing a clearer picture of how vehicles are spaced along the route. To assess how closely service follows the intended schedule, *headway deviation* is computed as the ratio of actual to scheduled intervals. A value of one indicates perfect spacing, zero means vehicles arrived simultaneously, and higher values signal increasingly irregular gaps, offering a straightforward metric for evaluating performance. *Schedule adherence* is also measured by comparing actual and scheduled arrival times, where negative values indicate delays and positive ones indicate early arrivals. Additionally, Vectura calculates the cumulative *distance traveled between stops*, providing a spatial understanding of vehicle movement and stop proximity. To tie everything together, extra features are generated to promote consistency across visualizations, ensuring the user experience remains clear, intuitive, and reliable.

5.4 Latency

By aggregating the intervals at which each step in the data pipeline operates, the maximum age of the data presented to the user is at most 2 minutes. Despite this, the system employs a strategy that effectively minimizes the perceived latency for end-users. By processing the predictive information from the GTFS RT feeds and segmenting it according to the exact time of the query, the system creates the illusion of instantaneous data updates.

This approach leverages the short-term predictions provided by GTFS RT to bridge the gap between actual data refreshes. While the actions suggested for the most recent 135-second window are based on forecasts, the proximity in time ensures that these predictions are highly reliable. Consequently, users experience a system that appears to deliver real-time updates, thereby enhancing decision-making with timely and accurate information.



(a) (b)

Fig. 7. (a) Vectura dashboard's string plot showing dispatchers headway issues such as gapping or bunching for any route. (b) Route ladder shows the position of any bus along its trip, visualizing the temporal relationships of buses along the same route.

5.5 Data Visualization

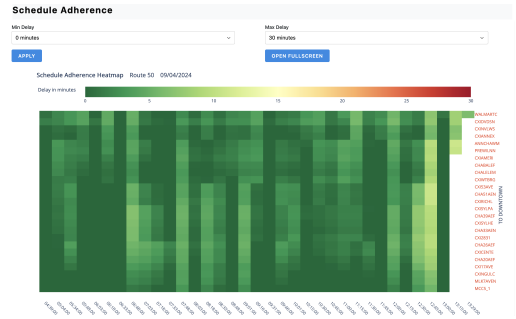
The two main plots in Vectura, are String Plots, Figure 7a, and Route Ladders, Figure 7b. These two visualization tools are developed closely with WeGo and are a result of identifying the gaps from their current tools and software.

String Plots: Available for both historical and real-time data, the string plot. It provides detailed per-stop analysis with the ability to tailor the view to specific operational needs. Stops are arranged vertically, spaced according to cumulative travel distance, with time plotted on the horizontal axis. Stop times are denoted by markers, which reveal detailed information upon hovering, and the lines connecting these markers represent individual trips. Users can adjust marker size to represent static or dynamic values such as vehicle load or headway deviations. Similarly, marker colors can be set to reflect various dynamic values or to identify specific patterns or vehicles. The string plot's flexible configuration extends to line styles, grouping options, and even the choice between plot presentations. Grouping options, such as pattern or vehicle, facilitate data sorting and enhance the plot's interactivity by allowing users to filter the displayed data through the legend. Hovering on each point in along the string plots reveal more information for each particular stop. These include the occupancy of the bus at that stop, scheduled and actual headway deviations at the instance, and schedule adherence or delay of the bus at that stop. When buses are following the scheduled headways, lines will appear almost equidistant, shown around 9:00 AM in Figure 7a. When buses are gapped or bunched, shown around 02:00 PM in Figure 7a, lines are uneven. This allows dispatchers to see across time, how a group of buses behave across any route. Finally, real time string plots are able to make use of predictive algorithms for occupancy and delays [4, 16] to inform transit dispatchers of future demand.

Route Ladder: This is an innovative tool designed to aid operators in the intricate task of managing headway in real time. It translates the physical location of vehicles into a time-based representation, where travel time is plotted along the horizontal axis. This axis can be adjusted to reflect either historical or scheduled travel times, allowing for a dynamic representation that matches the current operational conditions, providing operators with a clear picture of expected travel times. Incorporating data from GTFS-RT trip updates, we can accurately determine the spacing between stops. However, pinpointing a vehicle's precise location on the route ladder between widely spaced stops posed a challenge. To address this, we use the real time GPS coordinates of buses, projecting them onto a digital representation of the route (a line string). This projection helps us approximate the vehicle's location between stops, enabling an informed placement on the route ladder in relation to the travel time diagram. The accuracy of this placement is validated by ensuring the GPS coordinates are within an acceptable distance from the line string and that the vehicle's projected position aligns with its expected location based on trip updates. Symbols on the ladder change to indicate any potential inaccuracies in the vehicle's projected location. Additionally, the route ladder offers a configurable stop size feature, which can be set to a fixed size or can vary dynamically based on the wait time at each stop.

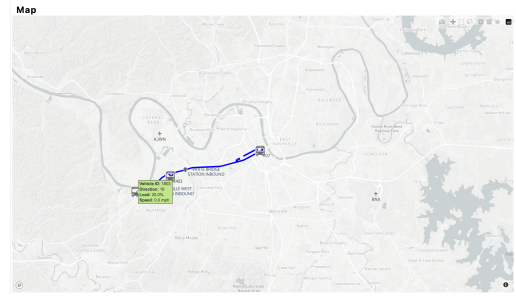
One of the most challenging aspects for operators is managing headway across shared segments of routes with multiple patterns. The route ladder addresses this by merging these shared segments and displaying buses from all patterns on the common line when they traverse this part of the route. As buses diverge onto their respective patterns, they are represented on separate lines. This feature not only simplifies headway management on complex, multi-pattern routes but also provides operators with the flexibility to toggle between combined and separate pattern views. The "Combine Patterns" option on the route ladder allows for an integrated display of all buses on shared sections, while also offering the ability to visualize individual patterns with their respective vehicles

(a) Occupancy Heatmap



(b) Schedule Adherence

(c) Headway Heatmap



(d) Real Time Bus Locations and Headway Information

Fig. 8. Other visualizations offered on Vectura. Aside from the real time bus locations, all other plots are available for both historical and real time data. Similar to the string plots, these plots provide information for a specific aspect of different routes, at a glance.

when needed. The route ladder is a useful tool for visualizing temporal relationships between buses and between a bus and its schedule.

Metric Heatmaps: Vectura also includes several heatmaps for different metrics such as occupancy (Figure 8a), schedule adherence (Figure 8b), and headways (Figure 8c). All of Vectura's heatmaps allow users to set minimum and maximum thresholds, providing a tailored view of the data. For example, setting a minimum delay threshold will color-code stop times within the specified range, enabling operators to focus on significant deviations or expand. These plots offer similar information from the string plots, although in a more distilled format. This allows transit dispatchers a clearer view of when or where issues are across the transit system.

Real Time Bus Locations: Finally, Vectura includes dynamic maps (Figure 8d) that provides geographical context to the data, allowing for spatial analysis of transit operations. The same real time map can be used to inform transit dispatchers of the best buses and stops for stationing and inform them of current disruptions or incidents where substitute buses can be dispatched to, shown in Figure 5b.

6 Related Work

Scheduling for fixed-line bus systems can be divided into static and dynamic scheduling. Static scheduling is the process of designing the fixed routes beforehand and is an offline problem. This includes the network route design, setting timetables, and scheduling vehicles to service routes.

Fixed-Line Transit Planning: In this context, static scheduling for fixed-line transit is a version of the classic line planning problem [25], [5], [15]. The design of a static fixed-line bus system aims to meet various, often competing, objectives including coverage, meeting demand, serviceability requirements, and cost. On the other hand, dynamic scheduling aims to adapt in real-time to address stochasticity in demand and uncertainty in the system. Current dynamic scheduling for fixed-line services is largely devoted to studying bus holding strategies aimed at improving bus service from the passenger's perspective. This involves designing a control system that manages the arrival times of vehicles along a route to minimize headway, passenger waiting time, or other serviceability metrics. The control strategy includes: holding control [12] [36] [32], bus speed control [11] [32], stop skipping [32] and boarding limits. Liu et al extend the notion of control strategies to focus on dynamic dispatching [21]. Their work focuses on dispatching vehicles for a single bus line. They monitor current demand, forecasted demand, and headway information along the line and then schedule when the next vehicle will leave the depot. Vehicles are not allowed to be dispatched to any stop or any trip, only at the first stop along the trip in question. Unlike our work, Liu et al do not consider reallocation strategies. Reallocation strategies have proven highly effective in rideshare and ride pooling applications [1, 20, 29]. The success of reallocation strategies in rideshare and ride pooling motivates our use of reallocation and stationing in the context of dynamic fixed-line transit.

Semi-Markov Decision Processes (SMDP): Another approach is to treat scheduling as a sequential decision making problem, model it as a Markov Decision Process (MDP), and solve it using online solution methods. MDPs are widely used in the study of planning and learning in stochastic environments [3, 27]. They provide a simple formulation of the planning problem and the general goals formulated as reward signals. Effective learning and planning methods for MDPs have been proven to be effective in a significant number of applications [14, 37]. However, traditional MDPs operate on just one time scale, with each action at a given time t influencing the subsequent state and reward at time $t+1$. MDPs lack the concept of actions that extend over varying time frames. The integration of stationing and dispatch into the problem makes state-transitions non-memoryless, so the dynamics of the underlying continuous time stochastic process are said to be semi-Markovian. In many cases, SMDPs provide more realistic models than discrete time MDPs since they take into account that time may evolve continuously.

Monte Carlo Tree Search: Even when framing the dispatch and stationing problem as an SMDP, the search space for an entire transit system is vast and highly intractable. Unlike classical search algorithms, MCTS adopts a more probabilistic and heuristic-driven method to explore the search space. It balances exploration and exploitation using random simulations, which allows it to find satisfactory solutions without the need for exhaustive search [6]. Recent work has used MCTS to efficiently evaluate and search potential transit routes based on multiple objectives [33]. MCTS has proven effective for planning in a variety of domains [38]. Planning with MCTS typically takes one of two forms. In the first setting, a value or state-action function is learned offline through reinforcement learning and can be queried at inference time to help guide the search process [26]. The second setting is purely online. In purely online MCTS generative models of the environment are used at inference time to evaluate future actions in the context of expected demand [9, 23, 24]. Pure online approaches have the benefit of not relying on value functions learned offline that can become stale in rapidly changing environments such as urban systems [35]. The algorithm can be halted at any time and still provide a viable solution based on the current state of the search tree. Thus, it is adaptable to situations with tight computational budgets or strict timing constraints. Therefore, this work focuses on the second approach where we use MCTS with passenger distribution and traffic incident generative models based on historical data.

7 Conclusion

We design a non-myopic, always online approach for optimizing stationing and dispatch of reserve or overflow buses for fixed-line transit that is scalable to real-world applications. We introduce several approximations and assumptions to allow our approach to limit the exponential action space present in the real world. We demonstrated that our approach improves upon the baseline by 2% on average, and as much as 7% on total passengers served. An additional benefit is that our approach reduces the overall deadhead miles traveled by overflow buses by 42% on average, as they are stationed and dispatched across the city. We also show that our approach, along with the generative models, can adapt to both real-world and synthetically scaled up environments based on multiple transit agencies, in some instances outperforming the baselines when more passengers are considered. We show that with careful tuning of parameters, we can execute decision epochs within the cut-off period for their decision-making. Finally, we introduce Vectura, a fully functional online dashboard developed with WeGo Public Transit in Nashville, TN, and designed specifically for transit dispatchers. Vectura bridges the gap between algorithmic research and real world deployment, allowing dispatchers access to state-of-the-art algorithms and clear data visualization tools to monitor the transit system.

8 Future Work

One caveat is that generative models is that they may potentially fail on out-of-distribution (OOD) data. This must be considered, especially for public transit applications, where reliable passenger and ridership predictions directly impact operational decisions under uncertainty. As part of our future work, we propose the use of uncertainty-aware generative models (e.g., Bayesian Variational Autoencoders or Normalizing Flows) that provide both samples and calibrated uncertainty estimates. Incorporating epistemic uncertainty enables the detection of high-uncertainty regions where model predictions may be unreliable. Additionally, when OOD inputs are detected, we propose fallback strategies such as reverting to conservative or myopic baseline policies. This would allow the system to gracefully degrade performance rather than experiencing catastrophic failure, providing robustness during unexpected situations and enabling recovery once the system returns to familiar pattern.

Acknowledgments

This material is based upon work sponsored by the National Science Foundation under Award Number 1952011 and the Federal Transit Administration COVID-19 Research Grant under Federal Award Identification Number TN-2021-015-00. Results in this paper were obtained with the Chameleon Testbed supported by the NSF.

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] American Public Transportation Association. 2022. Public Transportation Ridership Report. https://www.apta.com/wp-content/uploads/2022-Q2-Ridership-APTA.pdf

[3] Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* 72, 1 (1995), 81–138. doi:10.1016/0004-3702(94)00011-O

[4] Ammar Bin Zulqarnain, Samir Gupta, Jose Paolo Talusan, Dan Freudberg, Philip Pugliese, Ayan Mukhopadhyay, and Abhishek Dubey. 2023. Addressing APC Data Sparsity in Predicting Occupancy and Delay of Transit Buses: A Multitask Learning Approach . In *2023 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE Computer Society, Los Alamitos, CA, USA, 17–24.

[5] Ralf Borndörfer, Martin Grötschel, and Marc E Pfetsch. 2007. A column-generation approach to line planning in public transport. *Transportation Science* 41, 1 (2007), 123–132.

[6] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games* 4, 1 (March 2012), 1–43. doi:10.1109/TCIAIG.2012.2186810

[7] Guillaume M. J. B. Chaslot, Mark H. M. Winands, and H. Jaap van den Herik. 2008. Parallel Monte-Carlo Tree Search. In *Computers and Games*, H. Jaap van den Herik, Xinhe Xu, Zongmin Ma, and Mark H. M. Winands (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 60–71.

[8] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 785–794. doi:10.1145/2939672.2939785

[9] Daniel Claes, Frans Oliehoek, Hendrik Baier, and Karl Tuyls. 2017. Decentralised Online Planning for Multi-Robot Warehouse Commissioning. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems* (São Paulo, Brazil) *(AAMAS '17)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 492–500.

[10] Carlos F Daganzo. 2009. A headway-based approach to eliminate bus bunching: Systematic analysis and comparisons. *Transportation Research Part B: Methodological* 43, 10 (2009), 913–921.

[11] Carlos F Daganzo and Josh Pilachowski. 2011. Reducing bunching with bus-to-bus cooperation. *Transportation Research Part B: Methodological* 45, 1 (2011), 267–277.

[12] Xu Jun Eberlein, Nigel HM Wilson, and David Bernstein. 2001. The holding problem with real-time information available. *Transportation Science* 35, 1 (2001), 1–105.

[13] Liping Fu, Qing Liu, and Paul Calamai. 2003. Real-Time Optimization Model for Dynamic Scheduling of Transit Operations. *Transportation Research Record: Journal of the Transportation Research Board* 1857, 1 (Jan. 2003), 48–55. doi:10.3141/1857-06

[14] Nikolaos Geroliminis and Alexander Skabardonis. 2005. Prediction of arrival profiles and queue lengths along signalized arterials by using a Markov decision process. *Transportation Research Record* 1934, 1 (2005), 116–124.

[15] Konstantinos Gkiotsalitis and Oded Cats. 2021. Public transport planning adaption under the COVID-19 pandemic crisis: literature review of research needs and directions. *Transport Reviews* 41, 3 (2021), 374–392.

[16] Samir Gupta, Agrima Khanna, Jose Paolo Talusan, Anwar Said, Dan Freudberg, Ayan Mukhopadhyay, and Abhishek Dubey. 2024. A Graph Neural Network Framework for Imbalanced Bus Ridership Forecasting . In *2024 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE Computer Society, Los Alamitos, CA, USA, 14–21. doi:10.1109/SMARTCOMP61445.2024.00024

[17] Chaeeun Han, Jose Paolo Talusan, Dan Freudberg, Ayan Mukhopadhyay, Abhishek Dubey, and Aron Laszka. 2024. Forecasting and Mitigating Disruptions in Public Bus Transit Services. In *Proceedings of the 23rd Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2024, Auckland, New Zealand* (Auckland, New Zealand) *(AAMAS '24)*. 798–806.

[18] Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S. Gunawi, Cody Hammock, Joe Mambretti, Alexander Barnes, François Halbah, Alex Rocha, and Joe Stubbs. 2020. Lessons Learned from the Chameleon Testbed. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, online, 219–233. https://www.usenix.org/conference/atc20/presentation/keahey

[19] Levente Kocsis and Csaba Szepesvári. 2006. Bandit Based Monte-Carlo Planning. In *Machine Learning: ECML 2006*, Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 282–293.

[20] Kaixiang Lin, Renyu Zhao, Zhe Xu, and Jiayu Zhou. 2018. Efficient Large-Scale Fleet Management via Multi-Agent Deep Reinforcement Learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (London, United Kingdom) *(KDD '18)*. Association for Computing Machinery, New York, NY, USA, 1774–1783. doi:10.1145/3219819.3219993

[21] Yingxin Liu, Xinggang Luo, Xu Wei, Yang Yu, and Jiafu Tang. 2021. Robust Optimization Model for Single Line Dynamic Bus Dispatching. *Sustainability* 14, 1 (2021), 73.

[22] Xiaolei Ma, Yao-Jan Wu, Yinhai Wang, Feng Chen, and Jianfeng Liu. 2013. Mining smart card data for transit riders' travel patterns. *Transportation Research Part C: Emerging Technologies* 36 (2013), 1–12. doi:10.1016/j.trc.2013.07.010

[23] Ayan Mukhopadhyay, Geoffrey Pettet, Chinmaya Samal, Abhishek Dubey, and Yevgeniy Vorobeychik. 2019. An Online Decision-Theoretic Pipeline for Responder Dispatch. *CoRR* abs/1902.08274 (2019), 1–10. arXiv:1902.08274 http://arxiv.org/abs/1902.08274

[24] Geoffrey Pettet, Ayan Mukhopadhyay, Mykel J. Kochenderfer, and Abhishek Dubey. 2022. Hierarchical Planning for Dynamic Resource Allocation in Smart and Connected Communities. *ACM Trans. Cyber-Phys. Syst.* 6, 4, Article 32

(nov 2022), 26 pages. doi:10.1145/3502869

[25] Anita Schöbel. 2012. Line planning in public transportation: models and methods. *OR spectrum* 34, 3 (2012), 491–510.

[26] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484–489.

[27] Richard S Sutton, Doina Precup, and Satinder Singh. 1998. Between MDPs and Semi-MDPs: Learning, Planning, and Representing Knowledge at Multiple Temporal Scales. *Artificial intelligence* 112, 1-2 (1998), 181–211.

[28] Jose Paolo Talusan, Ayan Mukhopadhyay, Dan Freudberg, and Abhishek Dubey. 2022. On Designing Day Ahead and Same Day Ridership Level Prediction Models for City-Scale Transit Networks Using Noisy APC Data. In *2022 IEEE International Conference on Big Data (Big Data)*. IEEE Computer Society, Los Alamitos, CA, USA, 5598–5606. doi:10.1109/BigData55660.2022.10020390

[29] Xiaocheng Tang, Fan Zhang, Zhiwei Qin, Yansheng Wang, Dingyuan Shi, Bingchen Song, Yongxin Tong, Hongtu Zhu, and Jieping Ye. 2021. Value Function is All You Need: A Unified Learning Framework for Ride Hailing Platforms. arXiv:2105.08791 [cs.LG] https://arxiv.org/abs/2105.08791

[30] Federal Highway Administration U.S. Department of Transportation. 2003. Status of the nation's highways, bridges, and transit: 2002 conditions and performance report to congress. https://www.apta.com/wp-content/uploads/2022-Q2-Ridership-APTA.pdf

[31] Peter J. M. van Laarhoven and Emile H. L. Aarts. 1987. *Simulated annealing*. Springer Netherlands, Dordrecht, 7–15.

[32] Wensi Wang, Fang Zong, and Baozhen Yao. 2020. A Proactive Real-Time Control Strategy Based on Data-Driven Transit Demand Prediction. *IEEE Transactions on Intelligent Transportation Systems* 22, 4 (2020), 2404–2416.

[33] Di Weng, Ran Chen, Jianhui Zhang, Jie Bao, Yu Zheng, and Yingcai Wu. 2020. Pareto-optimal transit route planning with multi-objective monte-carlo tree search. *IEEE Transactions on Intelligent Transportation Systems* 22, 2 (2020), 1185–1195.

[34] D. J. White. 1993. A Survey of Applications of Markov Decision Processes. *The Journal of the Operational Research Society* 44, 11 (1993), 1073–1096.

[35] M. Wilbur, S. Kadir, Y. Kim, G. Pettet, A. Mukhopadhyay, P. Pugliese, S. Samaranayake, A. Laszka, and A. Dubey. 2022. An Online Approach to Solve the Dynamic Vehicle Routing Problem with Stochastic Trip Requests for Paratransit Services. In *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE Computer Society, Los Alamitos, CA, USA, 147–158.

[36] Weitiao Wu, Ronghui Liu, and Wenzhou Jin. 2016. Designing robust schedule coordination scheme for transit networks with safety control margins. *Transportation Research Part B: Methodological* 93 (2016), 495–519.

[37] Xinlian Yu, Song Gao, Xianbiao Hu, and Hyoshin Park. 2019. A Markov decision process approach to vacant taxi routing with e-hailing. *Transportation Research Part B: Methodological* 121 (2019), 114–134.

[38] Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. 2022. Monte Carlo Tree Search: a review of recent modifications and applications. *Artificial Intelligence Review* 56, 3 (July 2022), 2497–2562.

Received 4 September 2024; revised 1 May 2025; accepted 2 July 2025