A State Aggregation Approach for Stochastic Multi-Period Last-Mile Ride-Sharing Problems

Lucas Agussurja Shih-Fen Cheng
Hoong Chuin Lau
School of Information Systems
Singapore Management University

April 1, 2018

Abstract

The arrangement of last-mile services is playing an increasingly important role in making public transport more accessible. We study the use of ridesharing in satisfying last-mile demands, with the assumption that demands are uncertain and come in batches. The most important contribution of our paper is a two-level MDP framework that is capable of generating a vehicle-dispatching policy for the aforementioned service. We introduce state summarization, representative states, and sample-based cost estimation as major approximation techniques in making our approach scalable. We show that our approach converges and solution quality improves as sample size increases. We also apply our approach to a series of case studies derived from a real-world public transport dataset in Singapore. By examining three distinctive demand profiles, we show that our approach performs best when the distribution is less uniform and the planning area is large. We also demonstrate that a parallel implementation can further improve the performance of our solution approach.

1 Introduction

The world has seen an unprecedented rate of urbanization over the past few decades. In 2014, 54% of the world population already lived in cities, and by 2050, two-thirds will be urban. In absolute terms, this amounts to more than 6 billion people globally living in urban environments. Such a high rate of urbanization has put great pressure on all aspects of urban planning, in particular, the transportation system. From examples around the globe, we can see that it is critical to invest in and maintain a public transportation system with sufficient capacity to avoid extreme congestion and improve the environmental sustainability of a densely populated city (Cohen 2006).

Although public transportation provides the most efficient way to move a large number of passengers in a city, commuters usually need to find their own ways to fulfill their last-mile (LM) travel needs, i.e., moving from alighting stations to their final destinations. Such LM travels are usually quite short (one or two miles at most); however, if not addressed properly, some travelers might choose to give up public transport and make use of private cars, further aggravating urban congestion.

A wide variety of solutions have been proposed to satisfy LM travel needs, for example, bike sharing, ridesharing, short-distance trams, or feeder buses, each with different strengths and weaknesses, and not all are applicable in a given physical setting. In this paper, we are interested in satisfying LM travel demands through ridesharing using non-dedicated commercial vehicles such as taxis. Such a setup has the benefits of being dynamic, responsive, and cost-efficient, as vehicles are called in based on demand. Ridesharing has long been studied and experimented with as one way to make personalized transport more efficient, e.g., see the review by Chan and Shaheen (2012). We assume that a sizable fleet of commercial vehicles (taxis) already exists (which is the case in most Asian cities and some Western cities), and a significant fraction of these vehicles are operating vacant for the intended service hours.

An illustrative example of the ride-sharing-based LM service is found in Figure 1, with the station as a square in the center and the destinations of LM demands arriving in different time periods denoted by different shapes. Valid LM demands will be grouped together to be served by a fixed fleet, with the assumption that a committed vehicle will serve an assigned passenger group in order and then return immediately to the station after dropping off the last passenger. We will explain our LM planning framework in more detail in Section 3.

As pointed out in Wang and Odoni (2016), there are many operational challenges that need to be addressed, including but not limited to: a) demand estimation based on the traveler's demographic and personal preference and b) supply planning, which involves system parameters such as fleet size, vehicle dispatching strategy, and desired level of service. The focus of this paper is the online multi-period vehicle dispatching strategy; i.e., given both standby and uncertain future demands, we decide which subsets of LM demands to satisfy in the current period. The online and multi-period aspects of the problem are particularly important in today's environment, as travelers are increasingly used to using Internet-enabled smart devices to request for personalized transport services (see Dailey et al. (1999) and Calvo et al. (2004) for some early examples). In such a paradigm, travelers would submit their demands for service shortly before their arrivals, and this leaves a very short time (typically just a few minutes) to generate the dispatching plan. What complicates the problem even further is the fact that the problem is multi-period, implying that the generated dispatching plan should satisfy not only realized demands but also future demands that are to arrive shortly (e.g., in the next 10 to 20 minutes). With a fixed fleet size, it is often impossible to serve all demands, and thus the decision about which requests to serve in each period becomes important.

To address the challenges arising from the online and multi-period requirements, we separate the request selection problem from the routing problem and use a two-level planning framework to make real-time clustering and routing decisions. This is commonly employed in the literature to solve the delivery dispatching problem (see Section 2), of which our problem is a variant of. The upper-level problem is formulated to focus on selecting the requests to be served in a given period from the set of known requests. The selected requests are then clustered and routed in the lower-level problem. The upper-level problem, which is the focus of this work, is formulated as a Markov decision process (MDP). As with most real-world applications of MDP, our formulation suffers from the high-dimensional state space that makes exact solution intractable. On top of that, it also suffers from a large action space. In this paper, we propose a value-iteration-based approximation algorithm to address this problem. The main idea of our approach is to select a set of representative states such that any state that is not in the set is close to one that is. This closeness is measured using a domain-specific distance metric. The upper bound on this distance becomes the approximation parameter of the algorithm. Value iteration is then performed on this set instead of on the whole space, and the resulting policy is used as the solution to the original MDP. Beside this main idea, we employ two other approximation methods that are well-known in the literature, namely, state space discretization and sample average approximation (see Kearns et al. (2002) for example). Although we consider the routing element explicitly, where we employ the Savings algorithm of Clarke and Wright (1964), the lower-level problem is not the focus of this work.

In summary, we make the following major contributions:

- On the model development, we propose a two-level planning framework to satisfy LM demands by making real-time clustering and routing decisions. By having two planning levels, we are able to separate the request selection problem[1], which is computationally expensive, from the routing problem. Compared to existing approaches in the literature, our approach is different in that we incorporate realistic routing as opposed to function-based approximation in evaluating dispatching policies. This addition is important as most LM demands are short-distance and inaccurate estimations on routing choices could have significant impact on policy evaluations.

- On the algorithm development, the upper-level planning problem is modeled as a MDP

[1]Depending on the context, we sometimes also call this problem the vehicle dispatch problem.

and solved by a value-iteration-based approximation algorithm. As the MDP in its original form is intractable, we employ the following three techniques to speed up the solution process: a) representative states, b) state space discretization, and c) sample average approximation. Through computational experiments, we show that we are now able to solve previously intractable problems.

2 Literature Review

The two areas of work that are closely related to ours are on the delivery dispatching problem (DDP) and the dynamic vehicle routing problem (VRP). In the following, we provide a survey of both areas as well as the works on dealing with large-scale Markov decision processes (MDPs).

In DDP (Minkoff 1993), we are given a set of customers who are distributed geographically. Each customer consumes a certain amount of goods in each period, where the amount consumed is stochastic. And each customer may also hold up to a certain amount of inventory (unconsumed goods). In each period, given the inventory levels, the planner has to decide the amount of goods to be dispatched from a central distribution point to each customer, taking into consideration the transportation cost, the inventory-holding cost, and the unfulfilled consumption cost. Deliveries are assumed to be completed in one time period. One promising approach in solving DDP is to model it as an MDP and solve it using various approximate dynamic programming (ADP) techniques. For example, in recent works by van Heeswijk et al. (2015) and Rivera and Mes (2015), DDP variants with finite horizon, arbitrary completion times, and delivery time windows are studied. Compared to this line of work, our formulation can be seen as an extension, as we explicitly consider the vehicle routing element, instead of using close-form approximations. A similar effort has been made by van Heeswijk et al. (2017) to explicitly incorporate the routing problem in the DDP formulation. Compared to this latest work, our formulation is different in that we adopt infinite planning horizon and utilize an ADP approach that does not depend on the basis function.

MDPs are discrete-time stochastic processes used to model sequential decision-making in environments where outcomes are determined by both the decision-maker and random elements. Most methods for finding optimal policies in MDPs rely on the computation of value functions, defined on all possible states. The number of states in a typical application of an MDP, however, is computationally unmanageable. Much research has therefore focused on approximation methods for overcoming this curse of dimensionality. The two most common methods are the use of basis functions and state aggregation. In the basis function approximation approach, the value function is expressed as a linear combination of some selected basis functions. The coefficients of these functions are then tuned using methods such as simulated system trajectories or linear programming, to yield good approximations. In the state aggregation approach, the state space is aggregated into successively coarser representations, and the value function is approximated at different levels of aggregation. The choices of basis functions and the aggregated state space are non-trivial and very much dependent on problem-specific structures.

In terms of generality, there are two main directions of research. The first one seeks to find a general solution framework for most problem types. The methods include automatic state aggregation (Bertsekas and Castanon 1989, Givan et al. 2003, Ferns et al. 2004, Jong and Stone 2005, Virin et al. 2007) and automatic basis function construction (Keller et al. 2006). The second one seeks to solve specific problems by exploiting problem structures. In the specific area of transportation, efforts have been made to find good basis functions for different, related problems such as ambulance redeployment (Maxwell et al. 2010), inventory routing (Adelman 2004), and dynamic fleet management (Topaloglu and Powell 2006). Examples of the use of state aggregation include George et al. (2008), Simão et al. (2009), and Ulmer et al. (2018). The approach presented in this paper can be considered as a variant of the state aggregation method as applied to the dynamic ride-sharing problem.

In the routing literature, the ride-sharing problem is usually formulated as a variant of the dial-a-ride problem (DARP), which is itself a generalization of several problems, such as VRP with time windows (Solomon 1987) and fleet size and mix VRP (Liu and Shen 1999). In DARP, each request consists of a pickup location, a destination, and time windows for both pickup and drop-off times. The planner is then required to design the routes for a set of identical vehicles, starting from the same depot to serve the requests, subject to a set of constraints, such that the total route cost is minimized. A comprehensive survey on DARP can be found in Cordeau and Laporte (2007). And, a ride-sharing variant of the problem can be found in Baldacci et al. (2004). In the dynamic version, the planner has to sequentially construct and execute plans as requests are revealed over time. The better plans are those robust enough to accommodate new requests without incurring too much computational time or additional route cost. One of the well-known methods to tackle this problem is the family of rollout algorithms (Bertsekas et al. 1997). Rollout algorithms typically start with a base policy and try to improve it by performing a one-step policy iteration, where future scenarios are "played out" to evaluate the future cost of a given action. The policy is updated if a better action is found for a given state. Examples of the use of rollout algorithms in stochastic vehicle routing include Secomandi (2001), Novoa and Storer (2009), Goodson et al. (2013), and Ulmer et al. (2015). For an overview of the challenges in dealing with the dynamic VRP, see Agatz et al. (2012), Berbeglia et al. (2010), and Pillac et al. (2013).

In relation to our work, the lower-level problem can be viewed as a special case of DARP, where all requests have the same pickup location, i.e., the hub, and their pickup windows correspond to the passenger's arrival times at the hub. Although solving the problem optimally is difficult for large-size instances, good heuristics exist for the problem, which include the Savings algorithm of Clarke and Wright (1964) and its variants and insertion heuristics (Vigo 1996, Salhi and Nagy 1999, Campbell and Savelsbergh 2004).

3 Two-Level Planning Framework

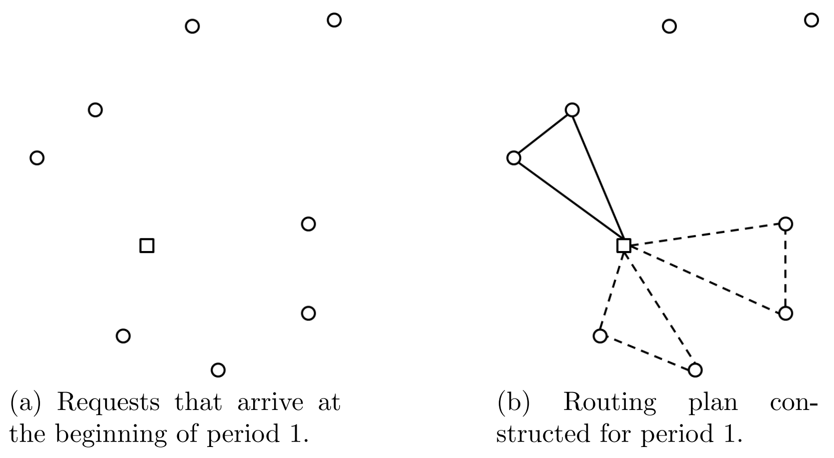
In this section, we describe the two-level planning framework. The idea is to divide the planning process into two levels: the policy-based upper level and the lower level that handles the actual routing. At the upper level, a policy is used to decide, at each period, the set of requests to be served. We formulate this request selection problem as an infinite-horizon MDP. The main advantage of this formulation is the ability to compute the upper-level policies offline. This is important because, given the context of our application, a costly finite-horizon look ahead in real time would not be feasible. However, the infinite-horizon formulation does bear the limitation of not able to handle time-dependent arrivals. To handle potentially time-dependent arrivals, we can either expand the state space (to be introduced next) to include time-related information (e.g., day of the week and moment of the day); or for practicality, solve the stationary policy for each (day of the week, moment of the day) tuple.

At the lower level, given the set of selected requests and the state of the vehicles, a routing plan is to be constructed. This is formulated as a variant of the DARP, where all requests start at the same location and some vehicles may not be available immediately. We follow closely the formulation given in Cordeau and Laporte (2007).

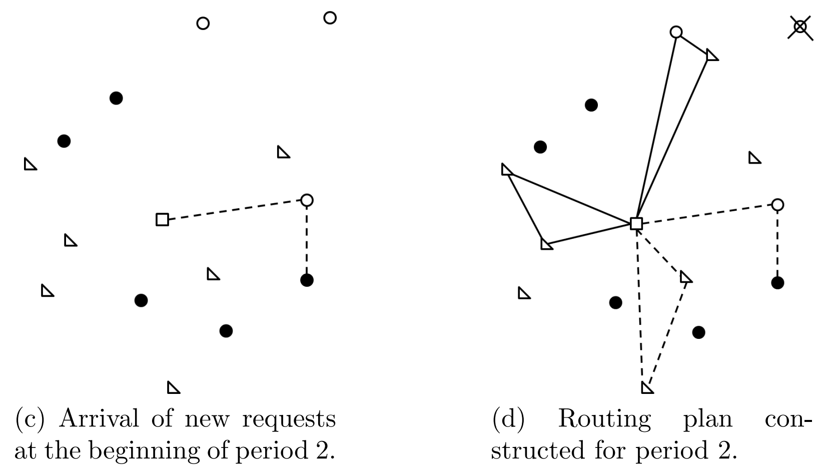
In Section 3.1, we describe the problem using a small example, and in Section 3.2, we give the MDP formulation of the upper-level problem. For completeness, we provide the exact formulation of the deterministic version of the problem in the Appendix.

3.1 Motivating Example

Figure 1 illustrates the planning process for a small, motivating example of our problem. At the beginning of the first period, as shown in Figure 1a, eight requests are received and logged into the system. The square indicates the hub, and the circles indicate the destinations of the requests. There are two vehicles, each with a capacity of two. The goal is to construct


(a) Requests that arrive at the beginning of period 1.

(b) Routing plan constructed for period 1.


(c) Arrival of new requests at the beginning of period 2.

(d) Routing plan constructed for period 2.

Figure 1: An illustration of the planning process with a small example.

a routing plan given the available information. An example of the routing plan is shown in Figure 1b. The solid lines show the route of the first vehicle, and the dashed lines show the route of the second vehicle. As indicated by the dashed lines, a vehicle might make multiple trips to/from the hub. Notice that this plan is incomplete in that two requests are left unscheduled. Due to the limited number of vehicles and the short time period, it is often impossible to serve all existing requests and have all the vehicles back at the hub before the beginning of the next period. Hence, we need to leave some requests unscheduled until the next period. In this example, the planner decides to leave two requests unscheduled until the arrival of the next batch of requests. The plan is then executed once it has been constructed.

Figure 1c illustrates the state at the beginning of the second period, where new requests are shown as triangles and served requests are shown as filled circles. At this point, the existing plan has not been executed completely, as one vehicle is still on its way to deliver a request before heading back to the hub, as indicated by the dashed lines. The other vehicle is back at the hub. The planner is now required to construct a new routing plan, taking into account the new requests and the state of the vehicles. In theory, it might be advantageous to modify the existing plan. The vehicle on the road, for example, can be asked to return to the hub to pick up another request before delivering the passenger on board. In practice, however, such a modification may not be acceptable to the passenger(s) on board. To avoid adding unnecessary complications to both modeling and implementation, modifications of existing plans are not allowed in our paper. The new plan has to be constructed by extending the existing one. Figure 1d shows an example of the new plan, where three requests are left unscheduled. Another constraint of the problem is that a request that is not scheduled for two consecutive periods is dropped from the system, incurring a penalty cost. (The demand at the top-right corner in Figure 1d is one such dropped demand and is marked by a cross.) This constraint ensures that passengers will be informed in a timely fashion if arranging a shared ride is not feasible.

The example above illustrates the challenges in solving the multi-period ride-sharing problem in the last-mile context. Since requests are not known in advance but realized incrementally from one period to the next, the actual routing plan has to be constructed in real time, partially and sequentially at each period as requests become known. Note that since the execution of the constructed plan may span multiple future periods, and since plans are not allowed to be modified, the decisions as to which requests are to be left unscheduled in anticipation of future demands are critical–which is the focus of this work. Given the nature of the problem (being stochastic and with a short planning time), neither classical static approaches nor reactive methods with expensive look-aheads are suitable.

3.2 MDP Formulation

An MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{C}, \mathcal{P}, \gamma \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}(s)$ is the set of possible actions in state $s \in \mathcal{S}$, $\mathcal{C}(s,a)$ is the cost of performing action $a \in \mathcal{A}(s)$ in the state $s \in \mathcal{S}$, $\mathcal{P}(s'|s,a)$ for $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}(s)$ is the probability of transitioning to state s' if action a is taken in the state s, and $\gamma \in (0,1)$ is the discount used to compute present values of future costs. The solution to an MDP is a policy (state-to-action mapping) π^* that minimizes the expected discounted sum of costs over an infinite horizon. This is equivalent to solving the following set of Bellman's equations:

$$V^*(s) = \min_{a \in \mathcal{A}(s)} \left\{ \mathcal{C}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) V^*(s') \right\},$$

for every $s \in \mathcal{S}$, where the vector $V^*(s)$ is known as the value function. The rest of this section describes the upper-level planning problem in terms of these components.

State Space. A state of the system consists of the unscheduled requests and the state of the vehicles. Let $\mathcal{L}$ be the set of all possible destinations of a request and K be the set of vehicles. A state $s \in \mathcal{S}$ is defined as a vector (ℓ, ℓ', v) where $\ell = (\ell_1, \ell_2, \dots)$, $\ell_i \in \mathcal{L}$, denotes the destinations of unscheduled requests from this period, $\ell' = (\ell'_1, \ell'_2, \dots)$ denotes the destinations of unscheduled requests from the previous period, and $v = (v_1, \dots, v_{|K|})$ denotes the state of the vehicles. Since a request is dropped if it is not scheduled within two periods, it is sufficient to store unscheduled requests for two periods. For each vehicle $k \in K$, v_k is the time when it will be available at the hub. A vehicle might not be available immediately because the previously constructed plan might still be in execution. Here we assume that the time between periods, vehicles' speeds and passenger service times (pickup and drop-off) are constants, and therefore v can be computed trivially from the existing plan. Since partially constructed plans are not allowed to be modified in subsequent periods, it is not necessary, for the purpose of policy computation, to store the plans in the states.

The plan's information is implicit in the state of the vehicle.

Action Space and Cost Function. Given the state $s = (\ell, \ell', v)$, an action $a \in \mathcal{A}(s) = 2^{\ell \cup \ell'}$ is a subset of the unscheduled requests $\ell \cup \ell'$ for which the current plan is to be extended. $\mathcal{C}(s, a)$ is the cost of selecting a in the state s and consists of two components. The first component is the cost associated with dropped requests, given by $\Delta|\ell' - a|$, where Δ is the penalty for dropping a request and $|\ell' - a|$ is the number of dropped requests (i.e., all demand in ℓ' not served by a). The second component is the aggregated journey time of the selected requests (which contains both the waiting time and the riding time). To obtain the second cost component, we need to solve the operational level problem. This is where the decisions of both levels are connected.

Given selected requests a and vehicle state v (when vehicles will become available), an instance of the operational problem can be constructed and solved to generate the actual routing plan. Let $\mathcal{C}_{\text{op}}(a, v)$ be the riding time resulting from the actual routing plan, and let T be the time between two consecutive periods. The complete cost $\mathcal{C}(s, a)$ can then be expressed as

$$\mathcal{C}(s, a) = \Delta|\ell' - a| + T|\ell - a| + \mathcal{C}_{\text{op}}(a, v),$$

where the term $T|\ell - a|$ accounts for the waiting time of one time period for all requests not served in the current period.

From the perspective of computing an optimal policy, the above formulation of the cost function requires a consistent operational (routing) algorithm; i.e, for the same input, it has to return the same value. However, this is not a strict requirement, as we can always replace the term $\mathcal{C}_{\text{op}}(a, v)$ with the expectation of the value returned by an inconsistent algorithm. This is particularly necessary if instances of (a, v) are large and a heuristic algorithm, most likely with randomization, is required to solve the operational problem within reasonable time. Computing the exact expectation, however, will likely be intractable, in which case, methods like sample average approximation can be employed.

Transition Dynamics. The transition from one state to another can be described in two steps: the deterministic step and the stochastic step. Consider the state $s = (\ell, \ell', v)$ and the action taken $a \subseteq \ell \cup \ell'$. The last two elements of a possible next state s' are determined, i.e., $s' = (\cdot, \ell - a, v')$, where $\ell - a$ are the requests left unscheduled until the next period, and v' is the state of the vehicles obtained after running the operational planner with input (a, v). Again, here, v' is deterministically determined because the operational planner used is consistent. The first component of s' is stochastic, and it depends on the distributions associated with the number of arriving requests and their destinations in each period. Let $\mathbf{L}(m)$ be the probability of requests m arriving in any period; the transition probability $\mathcal{P}$ can therefore be expressed as

$$\mathcal{P}((m, m', u)|(\ell, \ell', v), a) = \begin{cases} \mathbf{L}(m) & \text{if } m' = \ell - a \text{ and } u = v', \\ 0 & \text{otherwise.} \end{cases}$$

We assume that arrivals are independent from period to period. It is unnecessary, therefore, to encode the history of arrivals into the state definition to maintain the Markov property.

Optimality Equation. One note regarding the use of the discount factor γ: Although minimizing the expected undiscounted sum of costs would probably be closer to the objective of the real-world problem, we choose to solve the discounted version due to the computational consideration that such formulation would more easily lead to the convergence to a stationary policy. This is commonly seen in the dynamic programming and the MDP literature, e.g., see Singh and Bertsekas (1997) and Maxwell et al. (2010). In other words, the discount factor can be viewed as an approximation parameter that provides the tradeoff between the rate of convergence and the accuracy of the model.

4 Solution Approach

The complete MDP formulation results in an uncountably infinite state space, since both v (the times when vehicles return to the hub) and $\mathcal{L}$ (the set of potential destinations) are continuous. It is inevitable, therefore, that we employ approximation methods in computing the required policy. In this section, we present our approach, which is based on the value iteration algorithm with the following layers of approximation:

1. *Discretization and Summarization of States*: The state space is discretized by dividing the planning region into zones and the planning horizon into intervals. Furthermore, we do not distinguish between requests that are going to the same zone. Hence, rather than storing the destination of each request, the number of requests going to each zone is aggregated and stored as a state variable. The same applies to the vehicles returning to the hub. Instead of the exact return time of each individual vehicle, we store the number of vehicles that will be returning in each time interval. This makes the state space finite and greatly reduces the dimension of the state space. Interestingly, the byproduct of this is that all states now have equal lengths, which allows us to measure the distance between two states using standard distance metrics.

2. *Representative States*: Although the discretized MDP has a much smaller state space, it is still too large to be solved exactly for even moderate-size instances. We circumvent this problem by further reducing the state space. Instead of all the states, we keep a set of "representative" states for which the value iteration is performed. The property of this set is such that any state in the original state space should be within a certain distance from the "closest" representative state. The distance between two states is defined as the maximum difference along a particular state dimension (i.e., the max norm).

3. *Sample Average Approximation* (*SAA*): Unlike the previous two techniques, which aim at reducing the size of the state space, the SAA approach is used to reduce the com-

putational efforts required for evaluating the expectation terms in the value function, in particular, the immediate cost of a discretized state-action pair (by sampling its possible realizations and solving them using the operational planner) and its future cost (by sampling possible future states).

Essentially, our approach maps the original MDP into a smaller discretized MDP, solves the smaller MDP using approximate value iteration, and uses the resulting policy as the solution to the original MDP. The following subsections describe these approximations in greater detail.

4.1 State Space Discretization and the SAA Approach

A state $\bar{s} \in \bar{\mathcal{S}}$ in the discretized MDP, $\langle \bar{\mathcal{S}}, \bar{\mathcal{A}}, \bar{\mathcal{C}}, \bar{\mathcal{P}}, \gamma \rangle$, is a vector (y, y', ν) where $y = (y_1, \ldots, y_N)$ is the number of existing requests in each zone that arrive in the current period, N being the number of zones. $y' = (y'_1, \ldots, y'_N)$ is similar to y but for requests that arrived in the previous period. The state of vehicles $\nu = (\nu_0, \nu_\sigma, \nu_\infty)$ describes the number of vehicles that will be returning in different time intervals. We divide our planning horizon into three time intervals: ν_0 is the number of vehicles that are available immediately, ν_σ is the number that are available between time 0 and $\sigma > 0$, and ν_∞ is the number that are available after σ. An action $\bar{a} = (\alpha_1, \ldots, \alpha_N, \beta_1, \ldots, \beta_N)$ in the state $\bar{s} \in \bar{\mathcal{S}}$ is a vector such that, for $z \in \{0, \ldots, N\}$, $0 \leq \alpha_z \leq y_z$ and $0 \leq \beta_z \leq y'_z$. It denotes the number of existing requests from both y and y' for which the partial plan is to be extended. $\bar{\mathcal{C}}(\bar{s}, \bar{a})$, the cost of selection $\bar{a} \in \bar{\mathcal{A}}(\bar{s})$ in the state $\bar{s} \in \bar{\mathcal{S}}$, is defined as follows:

$$\bar{\mathcal{C}}(\bar{s}, \bar{a}) = \Delta \sum_{z=1}^{N} (y'_z - \beta_z) + T \sum_{z=1}^{N} (y_z - \alpha_z) + \mathbf{E}\left[\mathcal{C}_{\text{op}}(a, v)\right].$$

The first and the second components are defined similarly as the corresponding cost components in the original MDP. The third component is the expected total travel time of selected requests. Here, the pair (a, v) refers to the continuous values in the original MDP before discretization and summarization. As there is potentially an infinite number of realizations of (a, v) given $(\bar{s}, \bar{a})$, we would treat (a, v) as random variables and compute the expectation by sampling uniformly over all possible instantiations of (a, v) given $\bar{s}$ and $\bar{a}$. The computation of this expectation is the first instance where we apply the SAA approach.

The second instance of using the SAA approach is in the computation of future values of state-action pairs. Similar to the cost function, the transition dynamics in the discretized MDP relies on the instantiation of the states and actions in the original MDP. To obtain a possible next state from $(\bar{s}, \bar{a})$, we first sample a state s and an action $a \in \mathcal{A}(s)$ in the original state space that maps to $\bar{s}$ and $\bar{a}$. We then sample a possible next state s' in the original MDP from (s, a) and map that back to a state $\bar{s}'$ in the discretized state space. Note that the mapping from the original to discretized state space is a surjection (many to one), and therefore, the last step is deterministic. To use the policy computed for the discretized MDP in the original MDP (where the state space is not discretized), we need a mapping from a state in the original to the discretized MDP and from an action in the discretized MDP to the original. The former is straightforward, and for the latter, given the number of requests to schedule in a zone, we pick uniformly from the existing requests in that zone.

4.2 Representative States

Instead of maintaining all possible states, we keep a set of representative states and try to approximate the value function on this set. For a state that is not in the set, its value is treated to be the same as the closest state in the set. Here, we define the distance between two states as the maximum difference between corresponding components of the states, i.e., for two states (y, y', ν) and (q, q', ν'),

$$\begin{aligned}\textsc{Distance}((y, y', \nu), (q, q', \nu')) = \max_{m \in \{1,\ldots,N\}} \{&|y_m - q_m|, |y'_m - q'_m|, \\ &\delta|\nu_0 - \nu'_0|, \delta|\nu_\sigma - \nu'_\sigma|, \delta|\nu_\infty - \nu'_\infty|\},\end{aligned}$$

where δ is the normalizing factor that is set to one in this work. Essentially, what we want to measure is first, the difference in the distributions of requests with respect to different zones, and second, the difference in the states of vehicles with respect to their return times. Since all states have identical length, the maximum difference is one logical choice. Given the upper bound $\epsilon \in \{1, 2, \dots\}$, we can then define a representative set θ_ϵ as a subset of $\bar{\mathcal{S}}$ such that for all $\bar{s} \in \bar{\mathcal{S}}$, there exists $\bar{s}' \in \theta_\epsilon$ where $\textsc{Distance}(\bar{s}, \bar{s}') \leq \epsilon$. One such set, which is used in this work, is the following:

$$\theta_\epsilon = \left\{ (y, y', \nu) \;\middle|\; \begin{array}{ll} y_m, y'_m = \tau(2\epsilon + 1), & \sum_m y_m \leq M, \sum_m y'_m \leq M, \\ \nu_0, \nu_\sigma, \nu_\infty = \tau(2\epsilon + 1), & \nu_0 + \nu_\sigma + \nu_\infty \leq |K|, \tau \in \{0, 1, \dots\} \end{array} \right\},$$

where each element of a representative state is a multiple of $2\epsilon + 1$. θ_ϵ can be viewed conceptually as the points in a multidimensional grid. Each state in the space is mapped to the nearest point in the grid. Note that like the mapping from $\mathcal{S}$ to $\bar{\mathcal{S}}$, the mapping from $\bar{\mathcal{S}}$ to θ_ϵ is a surjection, where for every state in $\bar{\mathcal{S}}$, there is exactly one state in θ_ϵ to which it is the nearest. (We use lexicographic order to break the tie if necessary.) As an example, consider the case where $\epsilon = 1$, the maximum number of requests per period is 10, the number of zones is 2, and the number of vehicles is 3, then the set θ_ϵ is defined by $\{(0,0), (0,3), (0,6), (0,9), (3,0), (3,3), (3,6), (6,0), (6,3), (9,0)\}^2 \times \{(0,0,0), (3,0,0), (0,3,0), (0,0,3)\}$. The evaluation of different values of ϵ and their effects on the performance is explored in Section 5.

4.3 Approximate Dynamic Programming

The method to compute the policy for the discretized MDP is based on value iteration and is shown in Algorithms 1 and 2. In the beginning, the set θ of representative states is initialized with random samples of possible starting states (Algorithm 1:3–7), where $\mathcal{Y}(\cdot)$ is the distribution of arrivals in the discretized MDP, which can be computed trivially from $\mathbf{L}(\cdot)$, the distribution in the original. During the iteration phase, if a state is sampled and

Algorithm 1 ApproximateValueIteration

```
1: Parameters: $maxStateSample$, $maxIteration$
2: $\theta \leftarrow \emptyset, V \leftarrow \emptyset, V' \leftarrow \emptyset, \pi \leftarrow \emptyset$
3: for $i = 0$ to $maxStateSample$ do                          ▷ Initial representative states
4:     $y \leftarrow$ sample from $\mathcal{Y}(\cdot)$, $\nu \leftarrow (|K|, 0, 0)$, $\bar{s} \leftarrow (y, \emptyset, \nu)$
5:     $V(\bar{s}) \leftarrow 0$
6:     $\theta \leftarrow \theta \cup \{\bar{s}\}$
7: end for
8: for $i = 1$ to $maxIteration$ do                                  ▷ Value update step
9:     for all $\bar{s} \in \theta$ do
10:        for all $\bar{a} \in \bar{\mathcal{A}}(\bar{s})$ do
11:            $V'(\bar{s}|\bar{a}) \leftarrow$ Evaluate$(\bar{s}, \bar{a}, \theta, V)$         ▷ State-action evaluation function
12:        end for
13:        $V'(\bar{s}) \leftarrow \min_{\bar{a}} V'(\bar{s}, \bar{a})$
14:        $\pi_i(\bar{s}) \leftarrow \arg\min_{\bar{a}} V'(\bar{s}, \bar{a})$
15:    end for
16:    $V \leftarrow V'$                                              ▷ Value function update
17: end for
18: return $\pi = (\pi_1, \ldots, \pi_{maxIteration})$          ▷ Generated policies from different iterations
```

is far from the set, then it is added to the set and its value is initialized to zero (Algorithm 2:12–17). A state $\bar{s}'$ is far from the set θ if Distance$(\bar{s}, \bar{s}') > \epsilon$ for all $\bar{s} \in \theta$, with ϵ being the approximation parameter. A larger ϵ will result in a smaller θ, which requires fewer iterations to reach its stable size but will produce a poorer estimate of the value function. On the other hand, a smaller ϵ requires more iterations to stabilize but results in a better estimate. It is important, therefore, to find the right values that balance both in different problem settings. As mentioned previously, sample average is used in computing the expected future costs (Algorithm 2:9,21) and the expected values of discretized state-action pairs using instantiations of possible pairs in the original MDP (Algorithm 2:2,24). The call to the function Execute (Algorithm 2:8) denotes the execution of the operational routing algorithm. One final note here is the exhaustive enumeration of all possible actions (Algorithm 1:10), which is very costly. Intuitively, we know that certain actions are almost always dominated by others (a trivial example being the empty selection); we explore the possibility of using smaller subsets of actions in Section 4.5.

Algorithm 2 Evaluate $((y, y', \nu), \bar{a}, \theta, V)$

```
1:  Parameters: maxInstanceSample, maxStateSample, $\epsilon$
2:  for $i = 1$ to maxInstanceSample do                                        ▷ Instances generation
3:      $\ell \leftarrow$ sample from $\mathbf{L}(\cdot)$ w.r.t. $y$
4:      $\ell' \leftarrow$ sample from $\mathbf{L}(\cdot)$ w.r.t. $y'$
5:      $v \leftarrow$ sample uniformly w.r.t. $\nu$
6:      $s_i \leftarrow (\ell, \ell', v)$                                      ▷ An instance in the original state space
7:      $a_i \leftarrow$ choose uniformly from $\ell \cup \ell'$ w.r.t. $\bar{a}$      ▷ A sample action
8:      $\bar{\mathcal{C}}_{\text{op}}(s_i, a_i), \nu' \leftarrow$ Execute$(s_i, a_i)$        ▷ Operational planner execution
9:      for $j = i$ to maxStateSample do                                       ▷ Sample next states
10:         $z \leftarrow$ sample from $\mathcal{Y}(\cdot)$
11:         $\bar{s}'_j \leftarrow (z, y - \bar{a}, \nu')$                      ▷ A sample next state
12:         if $\exists \bar{s} \in \theta$ such that Distance$(\bar{s}, \bar{s}'_j) \leq \epsilon$ then   ▷ A representative state exists
13:             $\bar{\mathcal{C}}_{\text{fut}}(s_i, a_i | \bar{s}'_j) \leftarrow V(\bar{s})$
14:         else                                                               ▷ Otherwise sample state is added to the set
15:             $\theta \leftarrow \theta \cup \{\bar{s}'_j\}$
16:             $V(\bar{s}'_j) \leftarrow 0$
17:             $\bar{\mathcal{C}}_{\text{fut}}(s_i, a_i | \bar{s}'_j) \leftarrow 0$
18:         end if
19:     end for
20:     $\bar{\mathcal{C}}(s_i, a_i) = \Delta|\ell' - a_i| + T|a_i - \ell| + \bar{\mathcal{C}}_{\text{op}}(s_i, a_i)$     ▷ Immediate cost
21:     $\bar{\mathcal{C}}_{\text{fut}}(s_i, a_i) \leftarrow \gamma \sum_j \bar{\mathcal{C}}_{\text{fut}}(s_i, a_i | \bar{s}'_j) / maxStateSample$     ▷ Discounted future cost
22:     $\bar{V}(s_i, a_i) \leftarrow \bar{\mathcal{C}}(s_i, a_i) + \bar{\mathcal{C}}_{\text{fut}}(s_i, a_i)$     ▷ Approximate value of state-action instance
23: end for
24: return $\sum_i \bar{V}(s_i, a_i) / maxInstanceSample$                      ▷ Average value over instances
```

4.4 Convergence and Runtime Analysis

In this section, we present the analytical results on the convergence and runtime performance of our approach. We start the analysis by considering exact value iteration updates on the set of representative states θ_ϵ (with full enumeration instead of using sampling). Proposition 1 shows that this operation is a contraction. This result relies on the use of the discount factor γ and the fact that the mapping from actual states to θ_ϵ is a surjection. As a consequence, when the value iteration update is applied repeatedly to an arbitrary starting state, it converges to a fixed point (Corollary 1).

Proposition 1 *Let $\mathcal{V}$ be the range space of the value function (i.e., the set of all possible values of V), where $V \in \mathcal{V}$ is a value function defined on θ_ϵ, the set of representative states.*

And, let $\mathcal{T}$ be the exact value update operator (as defined in Algorithms 1 and 2 but without sampling) on $\mathcal{V}$. The operator $\mathcal{T}$ on $\mathcal{V}$ is a contraction mapping if the mapping φ from $\mathcal{S}$ to θ_ϵ is a surjection and the discount factor γ is less than 1.

Proof. Let $U, V \in \mathcal{V}$, and for a representative state $\bar{s} \in \theta_\epsilon$, let

$$\bar{a}^*(U, \bar{s}) \in \arg\min_{\bar{a} \in \bar{A}(\bar{s})} \left\{ \bar{C}(\bar{s}, \bar{a}) + \gamma \sum_{s' \in \mathcal{S}} P(s'|\bar{s}, \bar{a}) U(\varphi(s')) \right\},$$

and $\bar{a}^*(V, \bar{s})$ be similarly defined. Assuming, w.l.o.g., that $\mathcal{T}V(\bar{s}) > \mathcal{T}U(\bar{s})$, we have

$$\begin{aligned}
&\mathcal{T}V(\bar{s}) - \mathcal{T}U(\bar{s}) \\
&\quad = \bar{C}(\bar{s}, \bar{a}^*(V, \bar{s})) + \gamma \sum_{s' \in \mathcal{S}} P(s'|\bar{s}, \bar{a}^*(V, \bar{s})) V(\varphi(s')) \\
&\qquad - \left(\bar{C}(\bar{s}, \bar{a}^*(U, \bar{s})) + \gamma \sum_{s' \in \mathcal{S}} P(s'|\bar{s}, \bar{a}^*(U, \bar{s})) U(\varphi(s')) \right) && (1) \\
&\quad \leq \bar{C}(\bar{s}, \bar{a}^*(U, \bar{s})) + \gamma \sum_{s' \in \mathcal{S}} P(s'|\bar{s}, \bar{a}^*(U, \bar{s})) V(\varphi(s')) \\
&\qquad - \left(\bar{C}(\bar{s}, \bar{a}^*(U, \bar{s})) + \gamma \sum_{s' \in \mathcal{S}} P(s'|\bar{s}, \bar{a}^*(U, \bar{s})) U(\varphi(s')) \right) && (2) \\
&\quad = \gamma \sum_{s' \in \mathcal{S}} P(s'|\bar{s}, \bar{a}^*(U, \bar{s})) \left[V(\varphi(s')) - U(\varphi(s')) \right] \\
&\quad \leq \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a^*(U, \bar{s})) \|V - U\| && (3) \\
&\quad = \gamma \|V - U\|,
\end{aligned}$$

and thus $\|\mathcal{T}V - \mathcal{T}U\| = \max_{\bar{s} \in \theta_\epsilon} |\mathcal{T}V(\bar{s}) - \mathcal{T}V(\bar{s})| \leq \gamma \|V - U\|$. The definition of the operator $\mathcal{T}$ leads to (1). The inequality (2) is obtained by substituting in a non-optimal action to the first line, resulting in a higher cost. The action chosen is the same as the one used in the second line. Step (3) is where the surjection property of φ is used. Since both $\varphi(s')$ map to the same state in θ_ϵ, the inequality is obtained by substituting in the maximum difference. Given that $\gamma < 1$, the operator $\mathcal{T}$ is a contraction mapping. □

Corollary 1 *Let $\mathcal{T}^\eta$ be η iterations of exact value update. There exists a unique $V^* \in \mathcal{V}$ such that $\mathcal{T}V^* = V^*$ and $\lim_{\eta \to \infty} \mathcal{T}^\eta V_0 = V^*$ for any $V_0 \in \mathcal{V}$.*

Proof. Follows from the fixed point theorem due to $\mathcal{T}$ being a contraction mapping. $\square$

Next, we derive the bound on the error resulting from using the SAA approach. Proposition 2 gives the error bound for a single step of the algorithm, while Proposition 3 provides the bound for an arbitrary number of steps η. We start by defining the following auxiliary variables. Given $\bar{s} \in \theta_\epsilon$ and $\bar{a} \in \bar{A}(\bar{s})$, we define $U^*(\bar{s}, \bar{a})$ and $\overline{U}(\bar{s}, \bar{a})$ to be the following:

$$U^*(\bar{s}, \bar{a}) = \mathbf{E}\left[\mathcal{C}(s, a)\right] + \gamma \mathbf{E}\left[V^*(\varphi(s'))\right],$$

$$\overline{U}(\bar{s}, \bar{a}) = \frac{1}{\lambda_1} \sum_{i=1}^{\lambda_1} \mathcal{C}(s_i, a_i) + \gamma \frac{1}{\lambda_1 \lambda_2} \sum_{i=1}^{\lambda_1 \lambda_2} V^*(\varphi(s_i')),$$

where $V^* \in \mathcal{V}$ is as given by Corollary 1, λ_1 denotes the number of state-action samples to be evaluated, and λ_2 denotes the number of samples used to approximate future cost. The notation s in the first definition denotes a possible realization of a state in the original MDP given $\bar{s}$, and the expectation is taken uniformly over all possible realizations. The notation s' denotes a possible next state given $(\bar{s}, \bar{a})$, and the expectation is first taken uniformly over all possible realizations of $(\bar{s}, \bar{a})$ in the original MDP and then taken according to the transition governed by $\mathcal{P}$. The second variable is an empirical mean, which is essentially a sample average approximation of the first variable, with sample sizes λ_1 and λ_2 as seen in Algorithms 1 and 2. For brevity, we denote the first term of both functions as $U_1^*(\bar{s}, \bar{a})$ and $\overline{U}_1(\bar{s}, \bar{a})$ and the second term as $\gamma U_2^*(\bar{s}, \bar{a})$ and $\gamma \overline{U}_2(\bar{s}, \bar{a})$. These two functions can thus be rewritten as $U^*(\bar{s}, \bar{a}) = U_1^*(\bar{s}, \bar{a}) + \gamma U_2^*(\bar{s}, \bar{a})$ and $\overline{U}(\bar{s}, \bar{a}) = \overline{U}_1(\bar{s}, \bar{a}) + \gamma \overline{U}_2(\bar{s}, \bar{a})$.

Proposition 2 *For any $\bar{s} \in \theta_\epsilon$ and $\bar{a} \in \bar{A}(\bar{s})$, we have*

$$\Pr\left(\left|U_1^*(\bar{s}, \bar{a}) - \overline{U}_1(\bar{s}, \bar{a})\right| \le \sigma_1\right) \ge 1 - 2e^{-2\sigma_1^2 \lambda_1 / \mathcal{C}_{\max}^2}, \text{ and}$$

$$\Pr\left(\left|U_2^*(\bar{s}, \bar{a}) - \overline{U}_2(\bar{s}, \bar{a})\right| \le \sigma_2\right) \ge 1 - 2e^{-2\sigma_2^2 \lambda_1 \lambda_2 / \mathcal{V}_{\max}^2},$$

for any positive σ_1 and σ_2.

Proof. By direct application of the Hoeffding's inequality (Hoeffding 1963). $\square$

To derive the stepwise error bound, we first define the following random variable denoting the progress of the algorithm:

$$\begin{aligned} \overline{U}^{\eta}(\bar{s},\bar{a}) &= \overline{U}_1^{\eta}(\bar{s},\bar{a}) + \gamma \overline{U}_2^{\eta}(\bar{s},\bar{a}) \\ &= \frac{1}{\lambda_1}\sum_{i=1}^{\lambda_1} \mathcal{C}(s_i,a_i) + \gamma \frac{1}{\lambda_1\lambda_2}\sum_{i=1}^{\lambda_1\lambda_2} V^{\eta-1}(\varphi(s_i')). \end{aligned}$$

Proposition 3 *Let η be the current iteration of the algorithm. For any $\bar{s} \in \theta_\epsilon$, $\bar{a} \in \bar{A}(\bar{s})$ and positive σ_1, σ_2 we have:*

$$\Pr(\left|\overline{U}^{\eta}(\bar{s},\bar{a}) - U^*(\bar{s},\bar{a})\right| \le err_\eta) \ge 1 - 2\eta\mathcal{H},$$

where $err_\eta = \gamma^\eta V_{\max} + \gamma^\eta\sigma_2 + \sum_{i=1}^{\eta-1}\gamma^i(\sigma_1+\sigma_2) + \sigma_1$ and $\mathcal{H} = e^{-2\sigma_1^2\lambda_1/\mathcal{C}_{\max}^2} + e^{-2\sigma_2^2\lambda_1\lambda_2/\mathcal{V}_{\max}^2}$.

Proof. If $|U_1(\bar{s},\bar{a}) - U_1^*(\bar{s},\bar{a})| \le \sigma_1$ and $|U_2(\bar{s},\bar{a}) - U_2^*(\bar{s},\bar{a})| \le \sigma_2$, then:

$$\begin{aligned} &\left|\overline{U}^{\eta}(\bar{s},\bar{a}) - U^*(\bar{s},\bar{a})\right| \\ &\qquad \le \left|\overline{U}^{\eta}(\bar{s},\bar{a}) - U(\bar{s},\bar{a})\right| + |U(\bar{s},\bar{a}) - U^*(\bar{s},\bar{a})| \\ &\qquad \le \gamma\left|\overline{U}_2^{\eta}(\bar{s},\bar{a}) - U_2(\bar{s},\bar{a})\right| + |U_1(\bar{s},\bar{a}) - U_1^*(\bar{s},\bar{a})| + \gamma\,|U_2(\bar{s},\bar{a}) - U_2^*(\bar{s},\bar{a})| \\ &\qquad \le \gamma\left|\overline{U}^{\eta-1}(\bar{s},\bar{a}) - U^*(\bar{s},\bar{a})\right| + \sigma_1 + \gamma\sigma_2, \end{aligned}$$

where the last step proceeds from the fact that $V^{\eta-1}(\bar{s}) = \min_{\bar{a}}\left\{\overline{U}^{\eta-1}(\bar{s},\bar{a})\right\}$. Let err_η denotes the bound at iteration η, we thus have the following recursive relation:

$$err_\eta = \gamma err_{\eta-1} + \sigma_1 + \gamma\sigma_2,$$

with the base case $err_0 = V_{\max}$. Solving the recursive equations for a particular η yields

$$err_\eta = \gamma^\eta V_{\max} + \gamma^\eta \sigma_2 + \sum_{i=1}^{\eta-1} \gamma^i(\sigma_1 + \sigma_2) + \sigma_1.$$

In other words, $\left|\overline{U}^\eta(\bar{s}, \bar{a}) - U^*(\bar{s}, \bar{a})\right| \leq err_\eta$ is true if both $|U_1(\bar{s}, \bar{a}) - U_1^*(\bar{s}, \bar{a})| \leq \sigma_1$ and $|U_2(\bar{s}, \bar{a}) - U_2^*(\bar{s}, \bar{a})| \leq \sigma_2$ are satisfied at each iteration. Applying Proposition 2, the probability that the latter false is bounded by $2e^{-2\sigma_1^2\lambda_1/\mathcal{C}_{\max}^2}$ and $2e^{-2\sigma_2^2\lambda_1\lambda_2/\mathcal{V}_{\max}^2}$, respectively, for one iteration. The probability that they are true for all the iterations is therefore at least $1 - 2\eta\left(e^{-2\sigma_1^2\lambda_1/\mathcal{C}_{\max}^2} + e^{-2\sigma_2^2\lambda_1\lambda_2/\mathcal{V}_{\max}^2}\right)$. $\square$

As a consequence, the following establishes that the algorithm converges in probability to the fixed point V^* of Corollary 1. And finally, the runtime complexity of the algorithm is proven in Proposition 4.

Corollary 2 *The approximate value iteration (Algorithm 1) converges in probability with a large enough sample size (λ_1 and λ_2), that is, given the fixed point V^* of Corollary 1,*

$$\lim_{\eta\to\infty} \Pr\left(\|\mathcal{T}_{\lambda_1,\lambda_2}^\eta V_0 - V^*\| > \sigma\right) = 0,$$

for any $V_0 \in \mathcal{V}$, $\sigma > 0$ and large enough λ_1 and λ_2.

Proof. Following Proposition 3, at the limit, we have:

$$\begin{aligned}\lim_{\eta\to\infty} err_\eta &= \lim_{\eta\to\infty} \gamma^\eta V_{\max} + \gamma^\eta \sigma_2 + \sum_{i=1}^{\eta-1} \gamma^i(\sigma_1 + \sigma_2) + \sigma_1 \\ &= \frac{\sigma_1 + \sigma_2}{1-\gamma} + \sigma_1,\end{aligned}$$

and thus if we choose λ_1 and λ_2 such that $\mathcal{H} < 1$, for any $\bar{s} \in \theta_\epsilon$ and $\bar{a} \in \bar{A}(\bar{s})$,

$$\lim_{\eta\to\infty} \Pr\left(\left|\overline{U}^\eta(\bar{s}, \bar{a}) - U^*(\bar{s}, \bar{a})\right| \leq \frac{\sigma_1 + \sigma_2}{1-\gamma} + \sigma_1\right) \geq 1 - 2\lim_{\eta\to\infty} \mathcal{H}^\eta = 1.$$

□

Proposition 4 *Let M be the maximum number of requests arriving in a period, N the number of zones, and K the set of vehicles. The approximate value iteration (Algorithm 1) runs in $O\left(\eta|K|^2(M/\epsilon)^{4N+2}\log M\right)$ time, where η is the number of iterations and $\epsilon > 0$ the approximation factor.*

Proof. The exact size of the discretized state space $\bar{\mathcal{S}}$ is the product of the number of non-negative integer solutions to $0 \leq \sum_{z=1}^{N} y_z \leq M$, $0 \leq \sum_{z=1}^{N} y'_z \leq M$ and $\nu_0 + \nu_\sigma + \nu_\infty = |K|$, given by the expression:

$$|\bar{\mathcal{S}}| = \left(\frac{|K|^2 + 3|K| + 2}{2}\right)\left[\sum_{m=0}^{M}\binom{N+m-1}{m}\right]^2,$$

which is asymptotically $O\left(|K|^2 M^{2N}\right)$. Given $\epsilon > 0$, the possible values for each entry of a state is reduced by the factor $1/(2\epsilon+1)$; therefore, the size of the representative set θ_ϵ is $O\left(|K/\epsilon|^2(M/\epsilon)^{2N}\right)$. The number of possible actions (selections) given a state is $O(M/\epsilon)^{2N}$. And, in each of the η iterations, for every state-action pair, the Savings algorithm is called a constant number of times, according to the sample size. Since the maximum number of selected requests is $2M$, each call takes $O(M^2 \log M)$ time using the heap implementation of the algorithm[2]. Evaluation of future cost can be done in constant time if the value function is implemented using indexing directly from the states (since a state is basically an array of integers). The total time taken, therefore, is the product of these components. □

Note that the runtime becomes computationally intractable as the number of zones increases, even though it is polynomial in the size of the problem if we keep N small and constant. From this analysis, we can identify two factors that constitute the runtime bottleneck. The first is the number of zones, and the second is the evaluations of all state-action pairs, both of which contribute significantly to the exponent of M/e. We address the first

[2]The Savings algorithm of Clark and Wright for VRP, when implemented using a heap, runs in $O(n^2 \log n)$ time, where n is the number of destinations. In our case, in each period, the maximum number of selected requests (destinations) is 2M. Hence, each call to the algorithm takes $O(M^2 \log M)$ time.

issue by keeping the number of zones small. As a consequence, however, the way zones are formed becomes crucial. In this work, this is done using the standard k-means clustering on past data of last-mile trips (the historical last-mile trips we utilized are extracted from a real-world public transport dataset in Singapore; more details can be found in Section 5.2). The details of the clustering method and the properties of the resulting zones are discussed in the experimental section. As for the second issue, one way to address it is to further decompose the action space and consider the actions of each vehicle independently. This decomposition approach is discussed in the next section.

4.5 Action Space Decomposition

Algorithm 3 ApproximateValueIteration-DC

```
1: **Parameters:** $maxStateSample$, $maxIteration$
2: $\theta \leftarrow \emptyset, V \leftarrow \emptyset, V' \leftarrow \emptyset, \pi \leftarrow \emptyset$
3: for $i = 0$ to $maxStateSample$ do                          ▷ Initial representative states
4:     $y \leftarrow$ sample from $\mathcal{Y}(\cdot)$, $\nu \leftarrow (|K|, 0, 0)$, $\bar{s} \leftarrow (y, \emptyset, \nu)$
5:     $V(\bar{s}) \leftarrow 0$
6:     $\theta \leftarrow \theta \cup \{\bar{s}\}$
7: end for
8: for $i = 1$ to $maxIteration$ do                                  ▷ Value update step
9:     for all $\bar{s} \in \theta$ do
10:        for all vehicle $k \in K$ do
11:            for all $\bar{a}_k \in \bar{\mathcal{A}}_k(\bar{s})$ do
12:                $f(\bar{a}_k|\bar{s}) \leftarrow$ Evaluate-DC$(\bar{s}, \bar{a}_k)$        ▷ Individual action evaluation
13:            end for
14:            $\bar{a}^*_k(\bar{s}) \leftarrow \arg\min_{\bar{a}_k} f(\bar{a}_k|\bar{s})$                ▷ Individual best action
15:        end for
16:        $\pi_i(\bar{s}) \leftarrow$ Compose$(\bar{a}^*_1, \ldots, \bar{a}^*_{|K|})$               ▷ Constructing joint action
17:        $V'(\bar{s}) \leftarrow$ Evaluate$(\bar{s}, \pi_i(\bar{s}), \theta, V)$
18:    end for
19:    $V \leftarrow V'$                                                  ▷ Value function update
20: end for
21: return $\pi = (\pi_1, \ldots, \pi_{maxIteration})$      ▷ Generated policies from different iterations
```

The method presented so far works well for problem instances of moderate size (considering the number of requests in each period, the number of zones, and the number of vehicles),

Algorithm 4 EVALUATE-DC $((y, y', \nu), \bar{a}_k)$

```
1:  Parameters: maxInstanceSample
2:  for $i = 1$ to maxInstanceSample do                          ▷ Instances generation
3:      $\ell \leftarrow$ sample from $\mathbf{L}(\cdot)$ w.r.t. $y$
4:      $\ell' \leftarrow$ sample from $\mathbf{L}(\cdot)$ w.r.t. $y'$
5:      $v_k \leftarrow$ sample uniformly w.r.t. $\nu$                    ▷ The available time of vehicle $k$
6:      $s_i \leftarrow (\ell, \ell', v_k)$                         ▷ An instance in the original state space
7:      $a_k^i \leftarrow$ choose uniformly from $\ell \cup \ell'$ w.r.t. $\bar{a}_k$        ▷ A sample action of vehicle $k$
8:      $\bar{\mathcal{C}}_{\text{op}}(s_i, a_k^i) \leftarrow$ EXECUTE$(s_i, a_k^i)$              ▷ Operational planner execution
9:      $\bar{\mathcal{C}}(s_i, a_k^i) = \Delta|\ell' - a_k^i| + T|a_k^i - \ell| + \bar{\mathcal{C}}_{\text{op}}(s_i, a_k^i)$      ▷ Cost of action
10: end for
11: return $\sum_i \bar{\mathcal{C}}(s_i, a_k^i)/maxInstanceSample$              ▷ Average value over instances
```

which is adequate given the context of our application. For larger instances, however, the offline computation of policies can be prohibitively long. For these cases, we propose an action space decomposition approach, given in Algorithms 3 and 4. Instead of evaluating every possible joint action in each iteration, we consider each vehicle's actions independently (Algorithm 3:10–15). Since vehicles have limited capacities, the size of the individual action spaces combined is much smaller than the joint action space. The best action for each vehicle is evaluated by sampling and solving the operational problem for a single vehicle (Algorithm 4). They are then combined to form a joint action, by adding the number of selected requests in each zone for each period, up to the maximum number of available requests (Algorithm 3:16). The value update for the state is then performed using Algorithm 2 with the constructed joint action. The evaluation of this decomposition is given in Section 5.

5 Computational Results

In this section, we present computational results related to the scalability and performance of our approach. In Section 5.1, using synthetic data, we show how runtime is affected by the algorithmic parameters. We also describe a parallel implementation of the algorithm here. In Section 5.2, we present the comparison of our approach against baseline policies using simulations, where parameters are derived from a real-world public transport dataset in Singapore.

5.1 Scalability and Parallelization

The first set of results measure the changes in runtime with respect to the approximation factor (ϵ), the number of VRP instances used to approximate the current cost of a selection (λ_1), and the number of samples of next states used to approximate the future cost (λ_2). These results apply to the dynamic version of our approach (where the representative set are constructed over the iterations). The runtime of the static version is given analytically in Section 4.4.

The problem setting and assumptions are as follows: The space of all possible destinations is the 2D Cartesian plane with coordinates ranging from -1 to 1 for both the x- and y-axes. This space is divided into four zones, which correspond to the four quadrants of the plane. The coordinate of the hub is $(0, 0)$. Possible numbers of requests in a period are 8, 9, and 10, with equal probability. The destination of a request is uniformly distributed over the plane and thus has a 0.25 probability of being in any zone. The distance between two points is the Euclidean distance. The number of vehicles is two, their capacity four, and their speed one. The time to pick up and drop off passengers (service time) is set to zero. This is true for all subsequent experiments. The time period is 5. The results of the experiments are shown in Figures 2 – 5, where each point in the graphs is the average of 10 instances (each instance is a randomly generated demand profile).

Figure 2 shows the runtime plotted against the number of iterations for ϵ, ranging from 1 to 4. Here, we see that the runtime increases significantly as ϵ decreases. This is due to the significant increase in the number of representative states to be maintained, as shown in Figure 3. In practice, it is therefore important to choose a suitable ϵ that balances between accuracy and computational time.

Despite the difference in the size of maintained representative states, it is interesting to note that in all ϵ cases, our proposed approach explores only a small fraction of the state

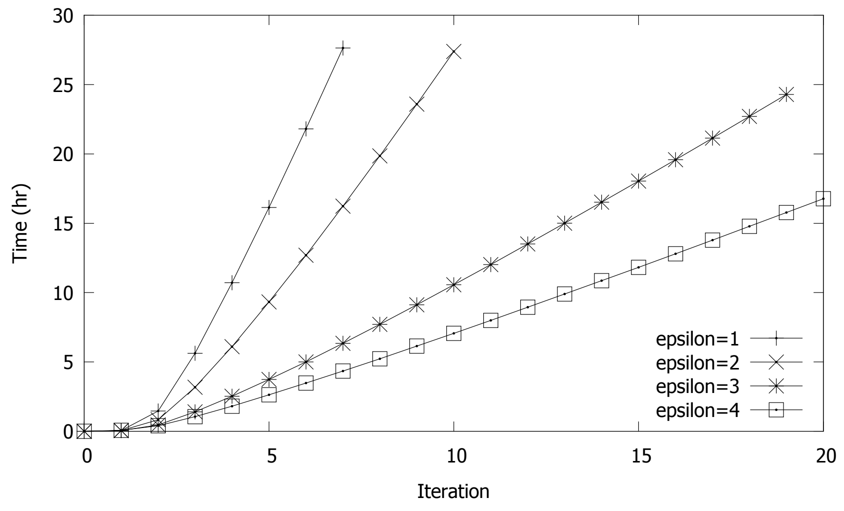


Figure 2: Runtime for different ϵ.

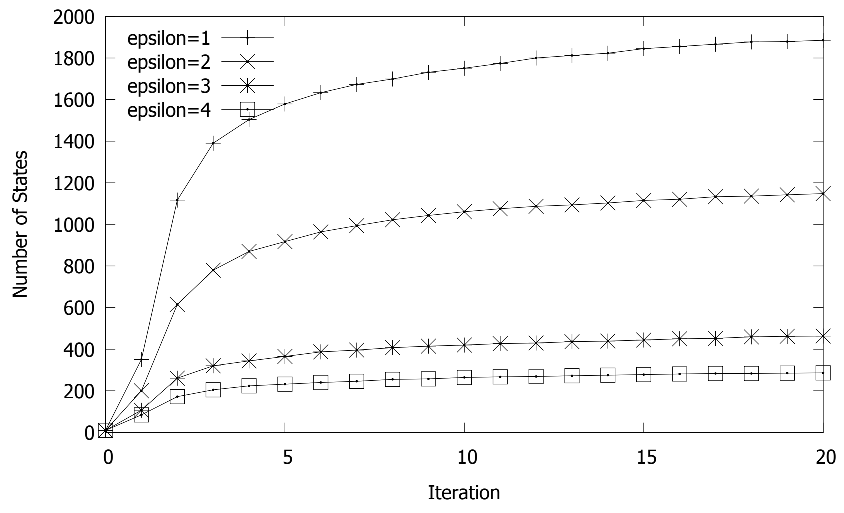


Figure 3: Number of representative states at different iterations for different ϵ.

space: at iteration 20, the fraction of the state space explored is around 0.1%, 0.06%, 0.02%, and 0.01% respectively for ϵ values of 1, 2, 3, and 4. This indicates, therefore, that our approach (and, in general, representative-state-based approximate value iteration) is best applied to problems where large numbers of states can be grouped together naturally by some form of similarity measure, which we believe is the case here. Lastly, note that the sets reach their stable sizes within few iterations, at around iteration 10.

Figure 4 shows the runtime for different values of λ_1 (the number of state-action samples

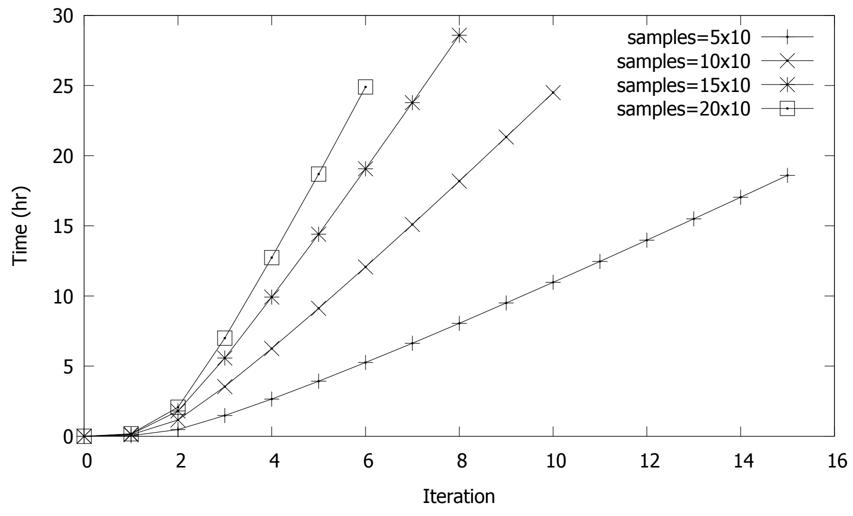


Figure 4: Runtime for different values of λ_1.

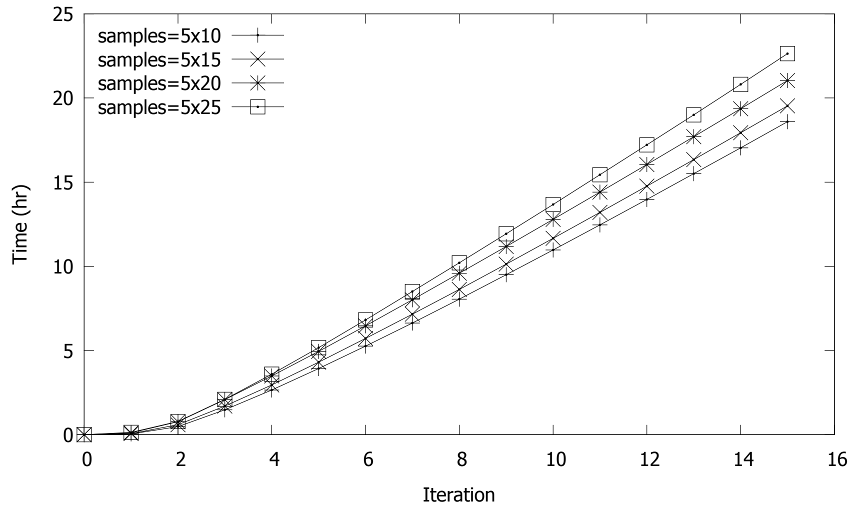


Figure 5: Runtime for different values of λ_2.

to be evaluated). An increase in λ_1 means that the number of VRP instances to be solved is increased for each evaluation of a selection. With the current implementation, given a state, the algorithm evaluates all possible selections exhaustively, and that is why runtime increases significantly as λ_1 increases. As mentioned previously, this is one of the weaknesses that we seek to improve in future works. Figure 5 shows the same for λ_2 (the number of samples to approximate future cost), and here the increase is not as significant, and we can afford to use a larger number of samples. Intuitively, a larger λ_2 also causes the set of maintained representative states to reach its stable size faster.

One way to address the scalability issue is through parallelization, and fortunately, value iteration-based algorithms lend themselves nicely to parallelization. Here, we describe briefly the parallelization of the algorithm using the MPI master-slave architecture.

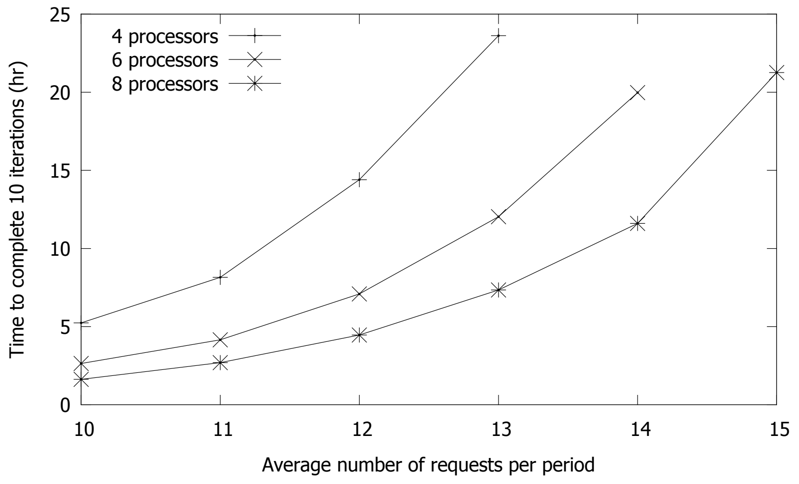


Figure 6: Parallel runtime for $\epsilon = 1$.

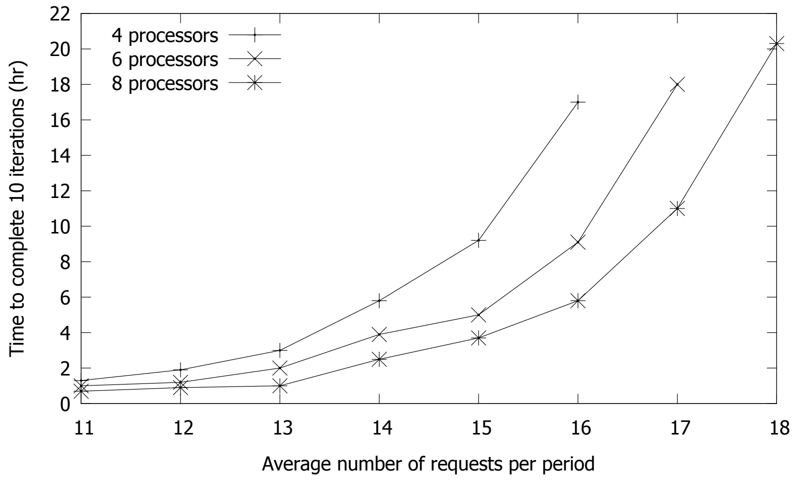


Figure 7: Parallel runtime for $\epsilon = 2$.

In this implementation, the master maintains the representative states and their values, while the slaves perform the update step. At the beginning of each iteration, the master broadcasts the representative states and their current values to all the slaves. Each slave then selects an equal number of states and updates them, computing the next value for each. At this point, if a sampled next state is far from the current set, its map in the set θ_ϵ is then added to a temporary local set. At the end of the iteration, each slave sends the updated values and the temporary set to the master. The master then consolidates the results, updating the values of the current set and adding new states if necessary. All processes are then synchronized before proceeding to the next iteration. We test our implementation on a Linux HPC cluster with eight Intel Xeon 8-core@3.16GHz and 256GB of RAM with the following setting. Similar to the previous setup, the problem is defined on a 2×2 Cartesian plane with the hub at the center. The plane is divided into six zones. The number of vehicles and their capacity is 4, the time period is 4, and the number of samples is 5 (λ_1) $\times$ 100 (λ_2). We measure the time to complete 10 iterations against the average number of requests for two different ϵ. Having m as the average number of requests means that the possible numbers are $m-1$, m, and $m+1$ with equal probability. The destination of a request is uniformly distributed on the plane. Figures 6 and 7 show the results for ϵ equals 1 and 2, respectively. They show that an increase in the number of processors by two reduces the runtime by roughly one-third. As shown, with parallelization, we manage to compute policies for moderate-size problems with small ϵ. For larger instances, like the ones in the next section, we have to compensate by increasing the value of ϵ correspondingly.

Note that due to the use of "Sample Average Approximation", the value function is approximated and thus the value iteration might not be monotonic. Therefore, if we want to identify the best policy obtainable from the approximated value iteration process, we have to numerically evaluate the policy obtained at the end of each iteration. To avoid these computational overheads, we simply use the policy obtained in the last iteration. We have empirically tested this approach, and confirm that, although the process is not monotonic as expected, the final value is numerically close to the best value.

5.2 Performance Comparison

To measure the performance of our approach, we perform two sets of experiments. The first set is designed to establish the gap between the ADP approach and the optimum (estimated using a lower bound). The second set is designed to evaluate the performance of the ADP approach on realistically sized problem instances against two baselines.

In the first set, using small synthetic instances, we compare our approach against a lower bound on the "optimum". To obtain the exact optimum, one would need to solve our MDP exactly, which is unfortunately not possible even for very small-scale instances. To provide a comparison, we derive a lower bound on the optimum instead. The lower bound is obtained by solving the last-mile ride-sharing problem deterministically, i.e., assuming that all demands are known a priori. The formulation is described in detail in the Appendix.

The synthetic instances are generated on a 2×2 Cartesian plane with the hub at the center. The number of arrivals in each period is uniformly distributed between 10-15, the time between two periods is 3, and the number of periods is 5. The traveling time between two points is set to be the Euclidean distance, and the capacity of vehicles is set to 2. For the ADP, the zones are the 4 quadrants of the plane. The results are shown in Table 1. Two different distributions of the destinations are considered. The four numbers in the first column correspond to the probabilities of a destination being in quadrants 1-4 respectively. We also vary the number of vehicles, and for each setting (row), 10 demand instances are generated and the average results are shown.

The lower bound on the optimum for each instance is obtained by solving the deterministic MIQCP model proposed in the Appendix. The MIQCP solver (implemented in CPLEX 12.7) is run until 10% optimality gap is obtained. As shown here, the ADP approach performs relatively well with respect to the lower bound, with better performance on more skewed demand distributions. Also shown here is the average runtime of the deterministic solver (to reach 10% optimality gap) and the *policy computation time* for the ADP approach. We would like to emphasize that although the ADP policy computation time seems long, it

Table 1: Performance comparison with the optimal solutions.

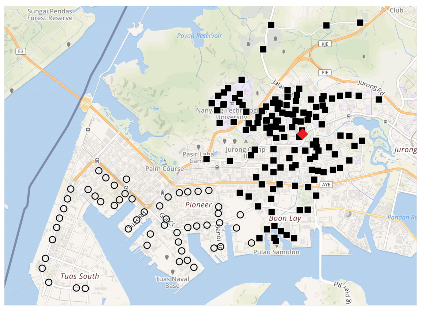
distribution of destinations	**#veh**	**MIQCP**	**avg runtime**	**ADP**	**policy comp time**	**avg optimality**
(0.25, 0.25, 0.25, 0.25)	5	4.12	3.5h	5.15	8.1h	75%
	6	3.35	4.2h	4.05	8.5h	79%
	7	2.83	5.1h	3.26	9.1h	85%
(0.5, 0.1, 0.3, 0.1)	5	4.11	3.7h	4.93	7.9h	80%
	6	3.66	3.9h	4.25	8.2h	84%
	7	2.79	5.5h	3.1	9.6h	89%

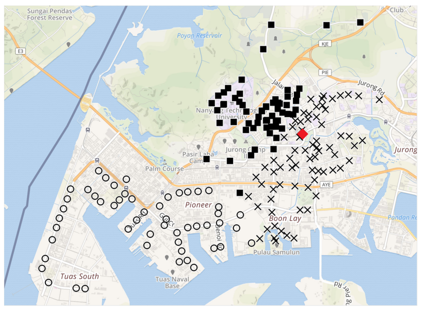
only needs to be executed once, and the runtime for actually applying the policy in practice plus solving the associated routing problem is close to instantaneous.

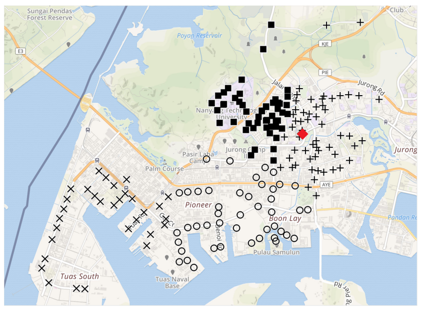
While the first set of the experiments establishes the performance gap between the ADP approach and the optimum, they are too small. The second set of the experiments is thus designed to evaluate the performance of the ADP approach on more realistically sized problem instances. As the MIQCP formulation is not scalable to these instances, we compare the performance of our ADP approach against two baseline policies: the complete selection policy and the one-step look-ahead policy. The complete selection policy is designed to serve as many arriving requests as possible in each period by solving a VRP without considering future requests. As demands from each period are served completely (fleet capacity is assumed to be large enough), demands from different periods will not have a chance to be grouped together. The complete selection policy will be viewed as the comparison baseline. The one-step look-ahead policy is obtained by looking at both current requests and requests that might arrive in the next period. The "look ahead" part is achieved by sampling several arrival scenarios in the next period, and for each sampled instance, a VRP containing both current and sampled requests is solved to obtain the policy. A current request will be selected if for more than 50% of all sampled instances, it is not grouped with any sampled request. The argument for this is, if in the majority of samples, a request is not grouped (in the routing decision) with any sampled future request, then it's less likely for this request to incur any saving by delaying it the next period.

We perform these comparisons under the steady-state assumption; i.e., there will be sufficient vehicles to serve all incoming requests, and the expected waiting/journey time stays finite. Computationally speaking, this implies that we never drop any request. To achieve this, when solving the MDP, we set the penalty for dropping a request (Δ) to infinity. When solving for the one-step look-ahead policy, additional constraints are inserted so that a request will always be selected if it is from the previous period.

The comparison is conducted using simulations at three representative train stations in Singapore: Boon Lay, Ang Mo Kio, and Marymount, where three distinct demand profiles are represented. (We will discuss them separately later.) For each station, we derive the following corresponding statistical distributions. The demand profile for each station is derived from a real-world public transport dataset, which contains comprehensive tap-in and tap-out records for all trains and buses across Singapore over one month. A last-mile trip is detected from the dataset as a bus ride that is preceded by a train ride within a short amount of time (less than 20 minutes of transit time). Since we cannot observe a passenger's real final destination, the alighting bus station is used as the proxy for the destination. By clustering all these last-mile destinations from a chosen train station using the k-means approach, the planning zones are determined (with k given as a parameter). The probability distribution over all possible destinations is computed by normalizing the observed last-mile demands from a typical work week (five days). These probability distributions are used for both our algorithms (when demands need to be sampled) and the simulations (for the evaluation purpose). The evaluation simulations are performed on the real road network, with the vehicle speed fixed at 45 km/h and the time interval set at 5 minutes. For each simulation setting, we measure the average journey time (waiting + riding times in minutes) of passengers for 50 periods. The results are summarized in Figures 8 – 10. The policy comparison for the Boon Lay station is shown in Figure 8. This station has a high volume of concentrated last-mile demands. For example, as shown in Figure 8(c), when clustered into four zones, more than 50% of all destinations are concentrated in a zone close to the


(a) 2 zones: square (0.87), circle (0.13)


(b) 3 zones: cross (0.32), circle (0.13), square (0.55)


(c) 4 zones: plus (0.27), square (0.54), circle (0.11), cross (0.08)

* The Boon Lay station is denoted as diamond.

#requests	veh cap	#veh	CS	LA	ADP					
					2 zones		3 zones		4 zones	
					$\epsilon = 2$	$\epsilon = 3$	$\epsilon = 2$	$\epsilon = 3$	$\epsilon = 2$	$\epsilon = 3$
21 - 27	4	5	21.45	19.91	16.01	16.67	14.82	16.9	14.04	15.09
		6	15.12	13.91	11.85	12.21	9.68	11.9	9.12	11.29
		7	10.7	9.01	7.92	8.26	6.87	9.46	6.38	7.26
	8	2	17.82	16.39	14.95	15.47	12.03	14.72	10.99	13.2
		3	15.3	14.57	11.47	12.05	10.44	13.12	9.49	10.45
		4	9.71	7.55	8.56	8.81	7.86	6.38	7.85	7.45
avg. improvement (against CS)				**11%**	**21%**	**18%**	**31%**	**20%**	**35%**	**28%**
					ADP-Decomp					
					2 zones		**3 zones**		**4 zones**	
					$\epsilon = 2$	$\epsilon = 3$	$\epsilon = 2$	$\epsilon = 3$	$\epsilon = 2$	$\epsilon = 3$
42 - 54	4	10	22.6	20.11	18.8	19.15	18.01	18.92	16.9	17.7
avg. improvement (against CS)				**11%**	**17%**	**15%**	**20%**	**16%**	**25%**	**22%**

* CS: complete selection policy, LA: look-ahead policy

Figure 8: Policy comparison for the Boon Lay station.

station marked by the square icons. The number of possible requests per period is uniformly distributed from 21 to 27. This range is reasonable, as we expect buses to remain the main mode of transport and only a small percentage of existing demands to eventually adopt ridesharing.

The results show that when clustered into four zones and with $\epsilon = 2$, the ADP policy manages to outperform the complete selection policy (CS) by around 35% and the one-step look-ahead policy (LA) by around 26%. Within the ADP policy, a reduction of ϵ by one improves its performance by around 10%. This comes at the cost of a longer running time. It is important, therefore, to push the number of zones to be as large as possible while keeping ϵ as low as possible. Similar performances can be seen for larger instances using ADP with action space decomposition (ADP-Decomp), albeit with smaller overall improvements. To test whether the differences in the performance means are statistically significant, we perform independent t-test for competing approaches and obtain the following relationships:

$$CS > LA > \text{2-zone } ADP > \text{3-zone } ADP > \text{4-zone } ADP.$$

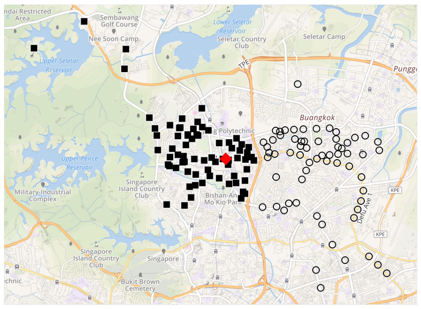
Note that all ADP variants are with $\epsilon = 2$. All relationships are significant with p-values close to 0.

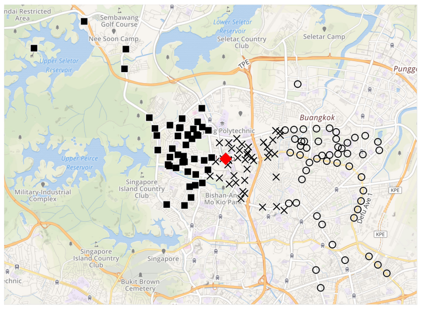
The second set of results for the Ang Mo Kio station is shown in Figure 9. The demand volume is similar to that of the Boon Lay station, but the destinations are distributed more uniformly around the station. We keep all parameter settings the same as before. Similar to the results for the Boon Lay station, the ADP approach consistently outperforms the two baseline policies. However, the advantage of the ADP approach is not as significant as in the Boon Lay case. Intuitively speaking, this demonstrates that the ADP approach is best suited for instances where demands are distributed less uniformly. (With uniform demand distribution, it is less useful to look beyond the current period.) As in the case of Boon Lay station, we also perform independent t-test for competing approaches and obtain the following relationship:

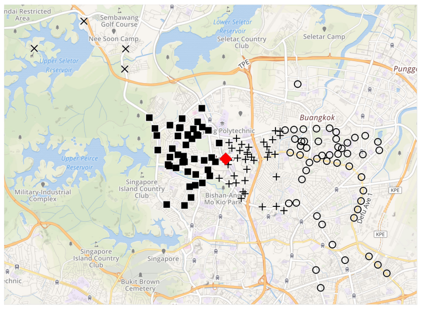
$$CS > LA > \text{3-zone } ADP \text{ or } \text{4-zone } ADP.$$

The ADP variants are with $\epsilon = 2$. All relationships are significant with p-values close to 0. For the Ang Mo Kio station, both 3-zone and 4-zone setups would lead to significantly better performance than LA; however, 4-zone ADP is not always better than 3-zone.

The last set of results for the Marymount station is illustrated in Figure 10. The service area is significantly smaller, with a much lower demand level. In this case, the ADP approach


(a) 2 zones: circle (0.41), square (0.59)


(b) 3 zones: circle (0.32), cross (0.39), square (0.29)


(c) 4 zones: plus (0.38), circle (0.32), square (0.28), cross (0.02)

* The Ang Mo Kio station is denoted as diamond.

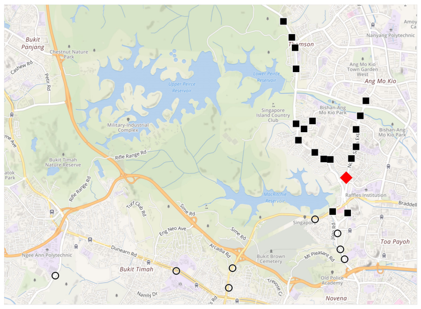
#requests	veh cap	#veh	CS	LA	ADP					
					2 zones		3 zones		4 zones	
					$\epsilon = 2$	$\epsilon = 3$	$\epsilon = 2$	$\epsilon = 3$	$\epsilon = 2$	$\epsilon = 3$
21 - 27	4	5	20.37	17.24	17.64	17.23	14.76	17.95	15.04	16.09
		6	13.11	12.27	11.15	11.15	8.97	11.81	8.76	9.21
		7	8.9	8.18	8.47	7.28	7.15	8.44	6.96	6.02
	8	2	16.81	14.75	14.49	14.71	12.89	13.8	11.57	13.38
		3	12.3	10.41	9.87	10.01	9.94	10.42	9.72	10.09
		4	8.52	7.28	6.03	8.15	6.08	5.47	5.05	6.6
avg. improvement (against CS)				**12%**	**16%**	**14%**	**25%**	**16%**	**29%**	**24%**
					ADP-Decomp					
					2 zones		**3 zones**		**4 zones**	
					$\epsilon = 2$	$\epsilon = 3$	$\epsilon = 2$	$\epsilon = 3$	$\epsilon = 2$	$\epsilon = 3$
42 - 54	4	10	22.5	19.71	19.14	19.63	18.05	19.28	17.33	18.56
avg. improvement (against CS)				**12%**	**15%**	**13%**	**20%**	**14%**	**23%**	**18%**

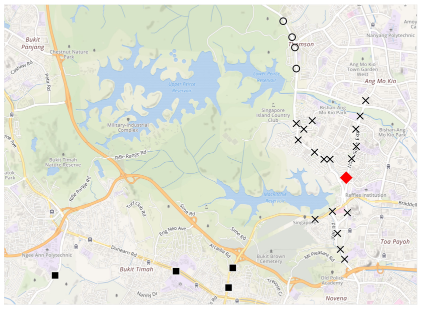
* CS: complete selection policy, LA: look-ahead policy

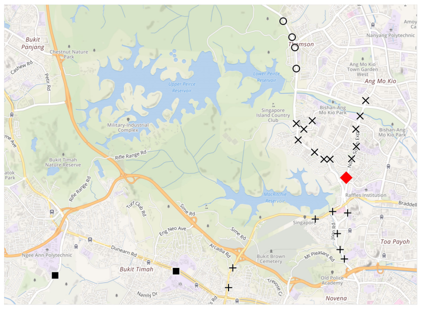
Figure 9: Policy comparison for the Ang Mo Kio station.

does not perform as well as in the previous two cases, and at 2 zones and $\epsilon = 2$, the one-step look-ahead policy actually outperforms the ADP policy, although not significantly. This is because the area covered is small and most destinations are concentrated near the station. As such, most trips are short and can be served immediately without multi-period planning.

To summarize our experience from these experiments, our ADP approach performs best with large and spatially irregular demand distributions over a large planning area.


(a) 2 zones: square (0.87), circle (0.13)


(b) 3 zones: circle (0.16), square (0.13), cross (0.71)


(c) 4 zones: circle (0.16), square (0.1), cross (0.51), plus (0.23)

* The Marymount station is denoted as diamond.

#requests	veh cap	#veh	CS	LA	ADP					
					2 zones		3 zones		4 zones	
					$\epsilon = 1$	$\epsilon = 2$	$\epsilon = 1$	$\epsilon = 2$	$\epsilon = 1$	$\epsilon = 2$
9 - 15	4	2	16.35	16.29	14.96	15.11	14.98	15.75	13.94	13.99
		3	10.2	8.68	9.06	9.56	8.57	8.87	9.99	9.59
		4	7.23	7.5	6.8	7.33	6.77	5.82	5.61	7.23
	8	1	18.21	16.9	16.94	17.37	16.11	17.41	17.57	17.56
		2	9.81	8.7	9.58	9.03	9.54	9.87	9.59	7.75
avg. improvement (against CS)				**6%**	**7%**	**5%**	**9%**	**8%**	**9%**	**9%**
					ADP-Decomp					
					2 zones		**3 zones**		**4 zones**	
					$\epsilon = 2$	$\epsilon = 3$	$\epsilon = 2$	$\epsilon = 3$	$\epsilon = 2$	$\epsilon = 3$
42 - 54	4	10	17.41	16.06	14.87	15.93	14.3	15.26	13.9	14.53
avg. improvement (against CS)				**8%**	**15%**	**9%**	**18%**	**12%**	**20%**	**17%**

* CS: complete selection policy, LA: look-ahead policy

Figure 10: Policy comparison for the Marymount station.

6 Conclusions

In this paper, we propose a two-level planning framework for dispatching vehicles for a ride-sharing-based last-mile service with stochastic multi-period demands. The upper level, which is modeled as an MDP, focuses on request selections, while the lower level is invoked when the actual operational costs need to be computed. When used in practice, the pre-computed selection policy can help the planner decide which requests to serve and when, while the actual vehicle assignments and routes can be computed in real time using any VRP algorithm.

The major methodological contribution of the paper lies in the generation of the upper-level selection policy. Due to the intractable nature of our problem, we incorporate the state summarization, representative states, and sample-based cost estimation as major approximation techniques in our solution framework. The level of approximation can be easily controlled by the degree of discretization (both spatially in the number of zones and temporally in the length of the time period), granularity of the representative states, and the sample sizes. We show that our approach converges in the limit, and the quality of our solution, measured by the error bound, improves as the sample size increases.

To demonstrate the value of our approach, we investigate a series of case studies derived from a real-world public transport dataset in Singapore. By studying three distinctive demand profiles, we show that our approach performs best when the entropy of the demand distribution is low (i.e., when the distribution is less uniform) and the planning area is large. For close-to-uniform demand distributions and small planning areas, a naive strategy of serving as many requests as possible in each period performs reasonably well.

7 Acknowledgment

This research was funded in part by the Singapore National Research Foundation (NRF) through the Singapore-MIT Alliance for Research and Technology (SMART) Center for Future Urban Mobility (FM) [10.13039/501100001474] and its Corp Lab at University scheme. The authors would also like to acknowledge the valuable inputs and guidance from Professor Amedeo Odoni, without whom this work would not have been possible.

References

D. Adelman. A price-directed approach to stochastic inventory/routing. *Operations Research*, 52 (4):499–514, 2004.

N. Agatz, A. Erera, M. Savelsbergh, and X. Wang. Optimization for dynamic ride-sharing: A review. *European Journal of Operational Research*, 223(2):295–303, 2012.

R. Baldacci, V. Maniezzo, and A. Mingozzi. An exact method for the car pooling problem based on Lagrangean column generation. *Operations Research*, 52(3):422–439, 2004.

G. Berbeglia, J.-F. Cordeau, and G. Laporte. Dynamic pickup and delivery problems. *European Journal of Operational Research*, 202(1):8–15, 2010.

D. P. Bertsekas and D. A. Castanon. Adaptive aggregation methods for infinite horizon dynamic programming. *IEEE Transactions on Automatic Control*, 34(6):589–598, 1989.

D. P. Bertsekas, J. N. Tsitsiklis, and C. Wu. Rollout algorithms for combinatorial optimization. *Journal of Heuristics*, 3(3):245–262, 1997.

R. W. Calvo, F. de Luigi, P. Haastrup, and V. Maniezzo. A distributed geographic information system for the daily car pooling problem. *Computers & Operations Research*, 31(13):2263–2278, 2004.

A. M. Campbell and M. Savelsbergh. Efficient insertion heuristics for vehicle routing and scheduling problems. *Transportation Science*, 38(3):369–378, 2004.

N. D. Chan and S. A. Shaheen. Ridesharing in North America: Past, present, and future. *Transport Reviews*, 32(1):93–112, 2012.

G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.

B. Cohen. Urbanization in developing countries: Current trends, future projections, and key challenges for sustainability. *Technology in Society*, 28(1):63–80, 2006.

J.-F. Cordeau and G. Laporte. The dial-a-ride problem: models and algorithms. *Annals of Operations Research*, 153(1):29–46, 2007.

D. Dailey, D. Loseff, and D. Meyers. Seattle smart traveler: dynamic ridematching on the World Wide Web. *Transportation Research Part C: Emerging Technologies*, 7(1):17–32, 1999.

N. Ferns, P. Panangaden, and D. Precup. Metrics for finite Markov decision processes. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, pages 162–169, 2004.

A. George, W. B. Powell, and S. R. Kulkarni. Value function approximation using multiple aggregation for multiattribute resource management. *Journal of Machine Learning Research*, 9 (Oct):2079–2111, 2008.

R. Givan, T. Dean, and M. Greig. Equivalence notion and model minimization in Markov decision processes. *Artificial Intelligence*, 147(1-2):163–223, 2003.

J. C. Goodson, J. W. Ohlmann, and B. W. Thomas. Rollout policies for dynamic solutions to the multivehicle routing problem with stochastic demand and duration limits. *Operations Research*, 61(1):138–154, 2013.

W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

N. K. Jong and P. Stone. State abstraction discovery from irrelevant state variables. In *Proceedings of Nineteenth International Joint Conference on Artificial Intelligence*, pages 752–757, 2005.

M. J. Kearns, Y. Mansour, and A. Y. Ng. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine Learning*, 49(2-3):193–208, 2002.

P. W. Keller, S. Mannor, and D. Precup. Automatic basis function construction for approximate dynamic programming and reinforcement learning. In *Proceedings of the Twenty-Third International Conference on Machine Learning*, pages 449–456, 2006.

F. H. Liu and S. Y. Shen. The fleet size and mix vehicle routing problem with time windows. *Journal of the Operational Research Society*, 50:721–732, 1999.

M. S. Maxwell, M. Restrepo, S. G. Henderson, and H. Topaloglu. Approximate dynamic programming for ambulance redeployment. *INFORMS Journal on Computing*, 22(2):266–281, 2010.

A. S. Minkoff. A Markov decision model and decomposition heuristic for dynamic vehicle dispatching. *Operations Research*, 41(1):77–90, 1993.

C. Novoa and R. Storer. An approximate dynamic programming approach for the vehicle routing problem with stochastic demands. *European Journal of Operational Research*, 196(2):509–515, 2009.

V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1–11, 2013.

A. P. Rivera and M. Mes. Dynamic multi-period freight consolidation. In F. Corman, S. Voß, and R. R. Negenborn, editors, *Computational Logistics*, Lecture Notes in Computer Science, pages 370–385. Springer International Publishing, 2015.

S. Salhi and G. Nagy. A cluster insertion heuristic for single and multiple depot vehicle routing problems with backhauling. *Journal of the Operational Research Society*, 50:1034–1042, 1999.

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

H. P. Simão, J. Day, A. P. George, T. Gifford, J. Nienow, and W. B. Powell. An approximate dynamic programming algorithm for large-scale fleet management: a case application. *Transportation Science*, 43(2):178–197, 2009.

S. Singh and D. P. Bertsekas. Reinforcement learning for dynamic channel allocation in cellular telephone systems. In *Proceedings of the Conference in Advances in Neural Information Processing Systems*, pages 974–980, 1997.

M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35:254–265, 1987.

H. Topaloglu and W. B. Powell. Dynamic-programming approximations for stochastic time-staged integer multicommodity-flow problems. *INFORMS Journal on Computing*, 18(1):31–42, 2006.

M. W. Ulmer, D. C. Mattfeld, M. Henning, and J. C. Goodson. A rollout algorithm for vehicle routing with stochastic customer requests. In D. Mattfeld, T. Spengler, J. Brinkmann, and M. Grunewald, editors, *Logistics Management*, pages 217–227. Springer International Publishing, 2015.

M. W. Ulmer, D. C. Mattfeld, and F. Köster. Budgeting time for dynamic vehicle routing with stochastic customer requests. *Transportation Science*, 52(1):20–37, 2018.

W. van Heeswijk, M. Mes, and M. Schutten. An approximate dynamic programming approach to urban freight distribution with batch arrivals. In F. Corman, S. Voß, and R. R. Negenborn, editors, *Computational Logistics*, Lecture Notes in Computer Science, pages 61–75. Springer International Publishing, 2015.

W. van Heeswijk, M. Mes, and M. Schutten. The delivery dispatching problem with time windows for urban consolidation centers. *Transportation Science*, to appear, 2017.

D. Vigo. A heuristic algorithm for the asymmetric capacitated vehicle routing problem. *European Journal of Operational Research*, 89(1):108–126, 1996.

Y. Virin, G. Shani, S. E. Shimony, and R. Brafman. Scaling up: solving POMDPs through value based clustering. In *Proceedings of AAAI Twenty-Second Conference on Artificial Intelligence*, pages 1290–1295, 2007.

H. Wang and A. Odoni. Approximating the performance of a "last mile" transportation system. *Transportation Science*, 50(2):659–675, 2016.

Appendix: MIQCP Formulation for the Deterministic Last-Mile Ride-Sharing Planning Problem

The deterministic last-mile ride-sharing planning problem is a variant of the DARP, where the inputs are the set of demands to be served and available vehicles. Compared to the classical DARP, there are two new features in our formulation: 1) every request starts from the same location (as illustrated in Figure 1), and 2) vehicle availability can be controlled using user-supplied parameters. In this section, we give a mixed integer quadratically constrained programming formulation for the problem.

The deterministic problem is defined on a directed graph $G = (V, E)$, where $V = \{0\} \cup a \cup a' \cup \{n+1\}$ is the vertex set such that 0 and $n+1$ denote the hub and a and a' both denote the n requests, where for every $i \in a$, there is a corresponding $i' \in a'$; we denote by H the set $a \cup a'$. With each vertex $i \in V$ is associated a load q_i where $q_0 = q_{n+1} = 0$, $q_i = 1$ for $i \in a$, and $q_i = -1$ for $i \in a'$. The edge set E is defined as $\{(0,j)|j \in a\} \cup \{(i,j)|i,j \in H, i \neq j\} \cup \{(i, n+1)|i \in a'\}$. The travel time on edge $(i,j) \in E$ is denoted by t_{ij}, where t_{ij} is 0 for the following edges: $\{(0,j)|j \in a\} \cup \{(i,n+1)|i \in a'\} \cup \{(i,j)|i,j \in a, i \neq j\}$. The travel time between two vertices in a is zero because all requests start from the same hub. For $\{(i,j)|i \in a, j \in a'\}$, t_{ij} is the travel time from the hub to passenger j's destination. And for $\{(i,j)|i \in a', j \in a\}$, t_{ij} is the travel time from passenger i's destination to the hub. For each request $i \in a$, p_i denotes its arrival period, and for each vehicle $k \in K$, v_k denotes the time when it will be available. Each vehicle has a fixed capacity, which is denoted by Q.

We now define the decision variables. Let x_{ij}^k equals 1 if edge $(i,j) \in E$ is traversed by vehicle $k \in K$, and 0 otherwise; w_i^k the load of vehicle k upon leaving vertex $i \in V$; d_i^k the time at which vehicle k leaves vertex $i \in V$; and r_i^k the travel time (waiting + riding time) of passenger $i \in a$ on vehicle k. Here, the waiting time refers to the time from arriving at the hub to being picked up. When a vehicle passes through vertex $i \in a$, it means that the vehicle is assigned to serve request i, and the time it leaves vertex i is the time it picks up

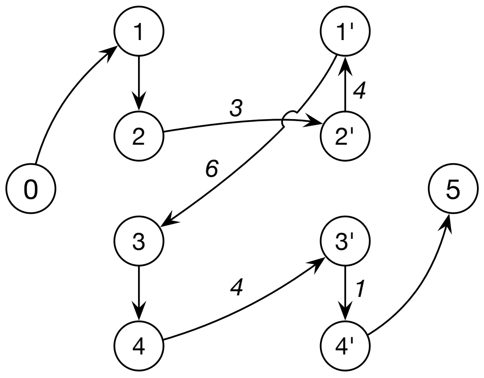


Figure 11: A simple graph with four demands and a service vehicle with a capacity of 2. Two trips are made, serving demands 2 and 1 in the first trip and demands 3 and 4 in the second trip. The numbers above edges are the travel costs. Unlabeled edges have zero cost.

the passenger. Thus, if a vehicle visits two vertices $i \in a$ and $j \in a$ consecutively, it means that the vehicle picks up i and j at the same time. Each vehicle is allowed to make multiple trips. Visiting vertex $i' \in a'$ means dropping passenger i at its destination, and the time it leaves vertex i' is the drop-off time. A simple illustrative example with four demands and a single service vehicle with capacity of 2 can be found in Figure 11. In this example, the vehicle first picks up demands 1 and 2, dropping off 2 followed by 1. After returning to the hub, the vehicle then picks up demands 3 and 4, dropping off 3 followed by 4. According to the labeled edge costs, travel times for all demands should be 7, 3, 17, and 18. The total cost is thus 45.

The MIQCP formulation is given as follows:

$$\min \sum_{i \in a} \sum_{k \in K} r_i^k \tag{4}$$

$$\text{Subject to} \tag{5}$$

$$\sum_{j \in H} \sum_{k \in K} x_{ij}^k = 1 \qquad i \in a \tag{6}$$

$$\sum_{j \in H} x_{ij}^k = \sum_{j \in V} x_{i'j}^k \qquad i \in a, k \in K \tag{7}$$

$$\sum_{j \in a} x_{0j}^k \leq 1 \qquad k \in K \tag{8}$$

$$\sum_{j \in V} x_{ji}^k = \sum_{j \in V} x_{ij}^k \qquad i \in H, k \in K \tag{9}$$

$$d_j^k \geq (d_i^k + t_{ij}) x_{ij}^k \qquad (i,j) \in E, k \in K \tag{10}$$

$$d_0^k \geq v_k \qquad k \in K \tag{11}$$

$$d_{i'}^k \geq d_i^k \qquad i \in a, k \in K \tag{12}$$

$$d_i^k \geq (p_i - 1) T x_{0i}^k \qquad i \in a, k \in K \tag{13}$$

$$r_i^k \geq d_{i'}^k \sum_{j \in H} x_{ij}^k - (p_i - 1) T \qquad i \in a, k \in K \tag{14}$$

$$w_j^k \geq (w_i^k + q_j) x_{ij}^k \qquad (i,j) \in E, k \in K \tag{15}$$

$$w_i^k \leq Q \qquad i \in H, k \in K \tag{16}$$

$$x_{ij}^k \in \{0,1\}, w_i^k \in \mathbb{Z}_{\geq 0}, d_i^k \in \mathbb{R}_{\geq 0}, r_i^k \in \mathbb{R}_{\geq 0}$$

Constraint (6) ensures that each request is served by exactly one vehicle. Constraint (7) ensures that if a passenger is picked up by a vehicle, then the same vehicle must deliver that passenger to its destination. Constraints (8)–(9) ensure that the flow of vehicles through the graph is consistent. Constraints (10)–(12) are concerned with the time when vehicles leave vertices: (10) ensures that they are consistent, (11) ensures that vehicles are available after they return to the hub (based on given vehicle states), and (12) ensures that a passenger is delivered only after it has been picked up. Constraint (13) ensures that a request can be picked up only after it has arrived at the hub. Constraint (14) defines passenger travel time, while constraints (15)–(16) ensure that the number of passengers in a vehicle is not greater than its capacity.