Optimizing Vehicle Distributions and Fleet Sizes for Mobility-on-Demand

Alex Wallar*, Javier Alonso-Mora†, and Daniela Rus*

***Abstract*— Mobility-on-demand (MoD) systems are revolutionizing urban transit with the introduction of ride-sharing. Such systems have the potential to reduce vehicle congestion and improve accessibility of a city's transportation infrastructure. Recently developed algorithms can compute routes for vehicles in real-time for a city-scale volume of requests with ride-sharing. However, these algorithms focus on optimizing the performance for a given fleet of vehicles and do not tell us how many vehicles we need to service all the requests. In this paper, we present a method to optimize the vehicle distributions and fleet sizes for MoD systems that allows requests to share vehicles. We present an algorithm to determine how many vehicles are needed, where they should be initialized, and how they should be routed to service all the travel demand for a given period of time. Evaluation using 23,529,740 historical taxi requests from Manhattan shows that on average 2864 four passenger vehicles are needed to service all of the taxi demand in a day with an average waiting time of 41.9 secs and added travel delay of 2.8 mins.**

I. Introduction

Autonomous vehicles and transportation network companies are revolutionizing personal mobility by making transportation available anywhere at anytime. Mobility-on-demand (MoD) has the potential to provide faster and more efficient transportation using fleets of coordinated autonomous vehicles. State of the art algorithms are able to efficiently manage fleets of vehicles to service large volumes of requests as is needed in dense urban areas.

Ride-sharing, where more than one passenger is able to use the same vehicle at the same time, is improving the efficiency of these algorithms by allowing less vehicles to service more requests. Services such as UberPool and Lyft Line have demonstrated how effective ride-sharing can be at extending our means of transportation within a city. Global vehicle dispatching algorithms have been developed that make use of ride-sharing by considering batches of requests, and optimizing routes for a set of vehicles to service these requests. These algorithms can ensure that passengers do not need to wait too long to be picked up and that by sharing a ride, the passenger will not go too far out of their way. However, these approaches cannot guarantee that all requests are serviced and do not tell us how many vehicles we need.

* The authors are at the Computer Science and Artificial Intelligence Laboratory of the Massachusetts Institute of Technology, 32 Vassar St, 02139 Cambridge MA, USA {wallar,rus}@csail.mit.edu

† The author is at the Department of Cognitive Robotics, Delft University of Technology, Mekelweg 2, 2628 CD Delft, Netherlands J.AlonsoMora@tudelft.nl

*This work was supported in part by the Netherlands Organisation for Scientific Research NWO Veni 15961, the Amsterdam Institute for Advanced Metropolitan Solutions, and the MIT-Singapore Alliance on Research and Technology under the Future of Urban Mobility.

Fig. 1: A cropped view of the initial vehicle distributed needed to service all of the taxi demand in Manhattan for a particular day

In what follows, we address the problem of determining how many vehicles are needed and where they should be for a MoD fleet to service *all* the taxi demand at city-scale with a maximum waiting time and maximum incurred delay while allowing multiple requests to be serviced by the same vehicle. We show that optimizing the number of vehicles and their distribution, we can significantly improve the efficiency of a MoD fleet.

A. Related Work

Early work for MoD systems focused on developing algorithmic techniques for fleets with single occupancy vehicles [1]–[6]. However, with the explosion of the sharing economy [7], [8], there is great potential for ride-sharing to make our fleets more affordable and efficient.

Recently, algorithms for large-scale ride-sharing have been developed. The work in [9] showed that 80% of rides in Manhattan could be pairwise shared while increasing the travel time by only a couple minutes. They later extended the analysis to multiple cities [10]. Alonso-Mora et al. developed a scalable approach to allow more than two passengers to share a vehicle by finding an optimal assignment of requests to a given fleet of vehicles [11]. The efficiency of the fleet improved in [12] by proactively routing the vehicles through

areas of high demand. However, these approaches did not determine how many vehicles would be needed to service all the travel demand.

There has been some research focussing on fleet sizing but either analyzes how different fleet sizes will perform to satisfy travel demand [13] or use a fixed start and end point model [14]. A recent breakthrough in [15] provides a method to optimally solve the minimum fleet problem, but assumes a vehicle can only transport one passenger at a time and cannot be easily extended to ride-sharing. Čáp and Alonso-Mora [16] used multi-objective analysis to determine the required fleet size for a set of requests while allowing multiple requests to share a ride, but the method was not scalable and they only presented experiments using 1 minute of request data.

B. Method Overview

The method for optimizing the fleet size and vehicle distribution is split into multiple steps. First we select a set of starting locations we call *vehicle deposits* such that every node in the road network is reachable from at least one deposit within given amount of time. These deposits are computed once offline and remain constant. We then iterate over the set of requests in batches backwards in request time. The batch size is fixed for all iterations and we only iterate over the batches once. We can think of the set of requests as a stack remaining requests to satisfy and at each iteration we are popping off the latest batch.

For each batch, we compute a set of travel schedules to pick up and drop off requests in that batch using the method from [11]. We assume vehicles are located at the vehicle deposits when computing these schedules.

While iterating over batches of requests, we maintain a set of initial travel schedules. These are the first schedules vehicles would execute after initializing. For the first iteration, this set is empty. For each iteration, we determine how vehicles can transition between schedules computed for the current batch of requests to the set of initial travel schedules. We then select a set of schedules and transitions from those computed that minimizes a cost function, satisfies certain constraints, and services all travel requests from the batch. Any initial travel schedule that receives an incoming transition is then removed from the set and all selected schedules are added to the set.

After iterating over all requests, we compute a maximum bipartite matching between travel schedules that have no outgoing transitions and our set of initial travel schedules using more lenient transition constraints. Schedules in the initial travel schedule set that receive incoming transitions from this matching are removed. Afterwards, this set of initial travel schedules will tell us how many vehicles we need and where they should be to service all the travel demand.

II. Preliminaries

A. Definitions

We assume that the vehicles travel along a road-network, $G = (N, E)$, represented as a directed graph, and that we have a function $\tau(n_i, n_j)$ for $n_i, n_j \in N$ that gives the shortest travel times between nodes in the graph.

We consider a set of travel requests, $\mathcal{R}$, and a set of vehicle deposits, $\mathcal{D}$. A travel request is a tuple $r = (p_r, d_r, t_r^r)$, where $p_r \in N$ is the pickup location, $d_r \in N$ is the dropoff location, and t_r^r is the time the request was made. $\mathcal{R}$ is sorted in ascending order by request time. $\mathcal{R}$ is iterated over in batches of n requests backwards in time. That means the first batch will consist of requests from $r_{|\mathcal{R}|-n}$ to $r_{|\mathcal{R}|}$.

A vehicle deposit is a location on the graph, $\mathcal{D} \subseteq N$, where vehicles are initialized. Vehicle deposits can be thought of as starting locations for vehicles before they have been assigned any requests.

For the formulation we consider a travel request to be satified if 1) the waiting time, ω_r, given by the difference between the pickup time, t_r^p, and the request time, t_r^r is less than a specified maximum waiting time, T_{wait} and 2) the total travel delay for the request given by $\delta_r = t_r^d - t_r^*$ is less than a specified maximum travel delay, T_{delay}, where t_r^d is the time when the request is dropped off and $t_r^* = t_r^r + \tau(p_r, d_r)$ is the earliest possible time the destination could be reached.

For the approach, we will compute a set of *travel schedules* and *schedule transitions* that vehicles will follow to satisfy the travel requests in $\mathcal{R}$. A travel schedule, $S = \{(l, t), ...\}$, for a given vehicle, v, is a sequence of pick up and drop off locations along with the time the pick up or drop off will occur and $R(S) \subseteq \mathcal{R}$ is the set of requests for a given schedule S. For convenience, let's call $\ell_S(k)$ and $t_S(k)$ the location and time of the kth event in the schedule respectively. We assume that the vehicle takes the shortest path between those locations. For a schedule to be valid, all the travel requests in the schedule must be satisfied given the maximum waiting time and maximum travel delay constraints and the number of passengers in the vehicle must not surpass the vehicle's capacity at any given time. We define the cost of each schedule as the sum of the travel delays for the requests associated with the schedule $\delta(S) = \sum_{r \in R(S)} \delta_r$.

A schedule transition is a pair of schedules, (S_i, S_j), such that a vehicle is able to satisify S_i then S_j without idling for longer than a given maximum vehicle idle time, T_{idle}. Let's define the transition time and idle time between schedules as $\xi(S_i, S_j) = \tau(\ell_{S_i}(|S_i|), \ell_{S_j}(1))$ and $\theta(S_i, S_j) = t_{S_j}(1) - t_{S_i}(|S_i|) - \xi(S_i, S_j)$ respectively.

B. Selecting Vehicle Deposits

Due to the maximum waiting time and maximum delay constraints for travel requests, it is not possible to guarantee a prescribed service rate for an arbitrary set of vehicle deposits. For instance, if the travel time from the closest vehicle deposit to a request's pick up location is larger than the maximum waiting time, that request may be impossible to service. Therefore, in order to provide a guaranteed service rate, we must intelligently select the locations for the vehicle deposits.

Using the road-network, $G = (N, A)$, we can select a set $D \subseteq N$ as vehicle deposit locations such that $\forall n \in N$,

$\exists d \in D$ with $\tau(d, n) \leq T_{\text{depos}}$ where $0 \leq T_{\text{depos}} \leq T_{\text{wait}}$. To reduce the computational overhead for computing schedules, we select the minimum number of vehicle deposits needed. We can do this by solving an integer linear program. First let's define a reachability matrix as follows,

$$H_{ij} = \begin{cases} 1, & \text{if } \tau(n_i, n_j) \leq T_{\text{depos}} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

This matrix describes which nodes are reachable from a given node within a specified amount a time. Let's also define a set of binary variables x where $x_i = 1$ if n_i is used as a deposit location and 0 otherwise. We can now solve an ILP to determine the minimum number of nodes to use as vehicle deposits such that the reachability constraint is satisfied:

$$\min_{x} \quad \sum_{i=1}^{|N|} x_i \tag{2}$$

$$\text{s.t.} \quad \sum_{i=1}^{|N|} x_i \cdot H_{ij} \geq 1 \quad \forall j \in [1, |N|] \tag{3}$$

Eq. (3) guarantees that every node in N is reachable within T_{depos} travel time from at least one vehicle deposit. Now we can define the set of vehicle deposits as $\mathcal{D} = \{n_i : 1 \leq i \leq |N| \wedge x_i = 1\}$

III. Fleet Optimization

In this section we describe how schedules and schedule transitions are selected for each batch using an integer linear program (ILP) and how are set of initial travel schedules is updated after each iteration.

A. Schedule Chaining Overview

We compute the schedule chains using an iterative batch process working backwards in time. At each iteration we are selecting a set of schedules and schedule transitions that minimizes the cost function and satisfies the constraint set. After each iteration, the selected schedules and schedule transitions are fixed and must remain selected. Any schedule that is selected that does not have an incoming transition is called a *start* as it requires a new vehicle. At each iteration we will maintain a set of starts that could be used for future transitions. This reduces the computational overhead of the approach.

At the kth iteration, we are given a set of requests, $\mathcal{R}_k \subseteq \mathcal{R}$, ordered by request time. We can consider $\mathcal{R}_k$ the remaining requests left to service after $k-1$ batches have been processed. For the first iteration, $k = 1$ and $\mathcal{R}_1 = \mathcal{R}$. The batch of requests for the kth iteration, $\mathcal{R}_k^{\text{latest}} \subseteq \mathcal{R}_k$, are the n latest requests from $\mathcal{R}_k$, that is the n requests in $\mathcal{R}_k$ with the latest request times. We then compute valid schedules using the requests in $\mathcal{R}_k$ that service requests in $\mathcal{R}_k^{\text{latest}}$ using the method from [11]. We limit the number of schedules by only considering requests in $\mathcal{R}_k$ that occurred within some time, T_{reqs} since the requests in $\mathcal{R}_k^{\text{latest}}$ were made. Note that these schedules will contain all the requests in $\mathcal{R}_k^{\text{latest}}$ but may also contain other requests from $\mathcal{R}_k$. We can think of this set of schedules as the candidate schedules to be selected that service batch k. We will call this set of candidate schedules, $\mathcal{S}_k$ where $\mathcal{S}_{k,i}$ is the ith schedule in that set. We will define the set of associated requests for the set of schedules, $\mathcal{S}_k$ as $\mathcal{R}_k^{\text{assoc}} = \bigcup_{i=1}^{|\mathcal{S}_k|} R(\mathcal{S}_{k,i})$.

We want to select a set of schedules, $\mathcal{S}_k^*$, at each iteration that would require new vehicles (i.e. schedules that have no incoming transitions). We select these schedules from the computed schedules for the current iteration, $\mathcal{S}_k$, and schedules selected from the previous iteration, $\mathcal{S}_{k-1}^*$. At the first iteration, $\mathcal{S}_{k-1}^* = \emptyset$. In conjunction with selecting initial schedules, we select a set of schedule transitions out of candidate transitions from $\mathcal{S}_k$ to $\mathcal{S}_{k-1}^*$. The transitions and schedules, $\mathcal{S}_k^*$ are solved for at the same time. Note that not all schedules in $\mathcal{S}_{k-1}^*$ need to be in $\mathcal{S}_k^*$. If a transition is computed between some $u_k \in \mathcal{S}_k^*$ and $v_{k-1} \in \mathcal{S}_{k-1}^*$, then v_{k-1} will not be in $\mathcal{S}_k^*$ since it would not require a new vehicle. Since we are iterating through the request set backwards in time, at the final iteration, we will have a set of schedules where vehicles will begin their routes. This will give us an initial distribution of vehicles and by iterating through the transitions, will provide the temporal distribution.

B. Integer Linear Program Formulation

We formulate this schedule selection problem as an ILP. Let's first define a set of binary variables $S = \{s_i : \forall i \in [1, |\mathcal{S}_k|]\}$ where $s_i = 1$ if schedule i is selected from $\mathcal{S}_k$ and zero otherwise. Let's also define a set of binary transition variables, $E = \{\epsilon_{ij} : \forall i \in [1, |\mathcal{S}_k|], \forall j \in [1, |\mathcal{S}_{k-1}^*|]\}$, where $\epsilon_{ij} = 1$ if schedule i in $\mathcal{S}_k$ should transition to schedule j in $\mathcal{S}_{k-1}^*$ and zero otherwise. For convenience, we also define two more sets of binary variables, $X = \{\chi_i : \forall i \in [1, |\mathcal{R}_k^{\text{latest}}|]\}$ and $H = \{\eta_i : \forall i \in [1, |\mathcal{S}_{k-1}^*|]\}$ where $\chi_i = 1$ if the ith request in $\mathcal{R}_k^{\text{latest}}$ was not used and $\eta_i = 1$ if the ith schedule in $\mathcal{S}_{k-1}^*$ was not used. With these variables, $\mathcal{Y} = (S, E, H, X)$, we can define a cost function for ILP to minimize as:

$$\begin{aligned} \mathcal{C}(\mathcal{Y}) = \sum_{i=1}^{|\mathcal{S}_k|} \Big[\delta(\mathcal{S}_{k,i}) \cdot s_i + \sum_{j=1}^{|\mathcal{S}_{k-1}^*|} \xi(\mathcal{S}_{k,i}, \mathcal{S}_{k-1,j}^*) \cdot \epsilon_{ij} \Big] \\ + K_r \cdot \sum_{i=1}^{|\mathcal{R}_k^{\text{assoc}}|} \chi_i + K_s \cdot \sum_{i=1}^{|\mathcal{S}_{k-1}^*|} \eta_i \end{aligned} \tag{4}$$

This cost combines the sum of the delays for the selected schedules from $\mathcal{S}_k$, the transition times for schedule transitions selected from $\mathcal{S}_k$ to $\mathcal{S}_{k-1}^*$, and additive costs for ignoring requests in $\mathcal{R}_k^{\text{assoc}}$ and schedules in $\mathcal{S}_{k-1}^*$. In, Eq. (4), K_r and K_s are constant cost values for ignoring requests and schedules respectively.

1) Requests Are Serviced At Most Once: For a schedule selection to be valid, we must ensure that each request in $\mathcal{R}_k^{\text{assoc}}$ is serviced at most once. This is described in the constraint:

$$\chi_i + \sum_{j \in \mathcal{I}_{r=i}^{\mathcal{S}_k}} s_j = 1 \quad \forall i \in [1, |\mathcal{R}_k^{\text{assoc}}|] \tag{5}$$

In Eq. (5), the set $\mathcal{I}_{r=i}^{\mathcal{S}_k}$ corresponds to the indices of schedules in $\mathcal{S}_k$ that service the ith request in $\mathcal{R}_k^{\text{assoc}}$.

2) Minimal Acceptable Service Rate: We also want to select schedules that will guarantee at least a minimum acceptable service rate is achieved. We do this by ensuring at least $\beta \cdot n$ are serviced out of the n latest requests, $\mathcal{R}_k^{\text{latest}}$. This constraint is formulated as:

$$\sum_{i=1}^{|\mathcal{R}_k^{\text{latest}}|} \chi_i \leq \beta \cdot n \tag{6}$$

3) Maximum Idle Time: To ensure that the vehicles do not idle for more than a specified amount of time, T_{idle}, between transitions, we must constrain transition variables. Though we already take into account the transition times into the cost function, direcly constraining the amount of vehicle idling allows us to provide guarantees for the maximum idle times for vehicles. This constraint is formulated as:

$$\begin{aligned} \epsilon_{ij} \cdot \theta(\mathcal{S}_{k,i}, \mathcal{S}^*_{k-1,j}) \leq T_{\text{idle}} \\ \forall i \in [1, |\mathcal{S}_k|] \text{ and } \forall j \in [1, |\mathcal{S}^*_k|] \end{aligned} \tag{7}$$

4) Schedules Start On Time: To gaurantee that requests in schedules selected in previous iterations are still satisfied, we must only select transitions such that the previously selected schedules, $\mathcal{S}^*_{k-1}$, start on time. This constraint over the transition variables is formulated as:

$$\begin{aligned} \epsilon_{ij} \cdot \left(t_{\mathcal{S}_{k,i}}(|\mathcal{S}_{k,i}|) + \xi(\mathcal{S}_{k,i}, \mathcal{S}^*_{k-1,j})\right) \leq t_{\mathcal{S}^*_{k-1,j}}(1) \\ \forall i \in [1, |\mathcal{S}_k|] \text{ and } \forall j \in [1, |\mathcal{S}^*_k|] \end{aligned} \tag{8}$$

This constraint ensures that the following schedule will be reached in time such that the waiting time and delay constraints are satisfied. This constraint also guarantees that all other schedules in the chain are also satisfied.

5) Schedule Flow Constraints: Also, to enable valid transitions, we must apply flow constraints to the transition variables. There can only be an outgoing transition from schedule i if the ith schedule is selected. This is described as:

$$\sum_{j=1}^{|\mathcal{S}^*_{k-1}|} \epsilon_{ij} \leq s_i \quad \forall i \in [1, |\mathcal{S}_k|] \tag{9}$$

Likewise, we can only have at most one incoming transititon to a any given schedule in $\mathcal{S}^*_{k-1}$ from $\mathcal{S}_k$. This is described by the constraint:

$$\eta_j + \sum_{i=1}^{|\mathcal{S}_k|} \epsilon_{ij} = 1 \quad \forall j \in [1, |\mathcal{S}^*_{k-1}|] \tag{10}$$

Note that we have introduced the variable η_j here to indicate if the jth schedule in $\mathcal{S}^*_{k-1}$ is not selected. This is used in the cost function.

6) Complete ILP Description: Combining the cost function from Eq. (4) with the constraints described in this section, we can formulate an ILP that will select a set of schedules and schedule transitions that will minimize the cost function while satisfying service rate, delay, waiting time, and transition constraints for the set of given requests. The full ILP is then formulated as:

$$\begin{aligned} \min_{\mathcal{Y}} \quad & \mathcal{C}(\mathcal{Y}) \\ \text{s.t.} \quad & \text{constraints (5)} - \text{(10)} \end{aligned} \tag{11}$$

C. Preparing For Next Iteration

After solving this ILP, we need to gather the selected schedules in preparation for the next iteration. To determine $\mathcal{S}^*_k$, we first add all schedules from $\mathcal{S}_k$ for which the corresponding variable $s_i = 1$ We also add all schedules from $\mathcal{S}^*_{k-1}$ that do not have any incoming transitions. These are all the schedules which would require a new vehicle to service since they are not going to be transitioned to by another vehicle after finishing its schedule. This set is constructed as:

$$\begin{aligned} \mathcal{S}^*_k = \{\mathcal{S}_{k,i} : s_i = 1, \forall i \in [1, |\mathcal{S}_{k,i}|]\} \\ \cup \{\mathcal{S}^*_{k-1,i} : \eta_i = 1, \forall i \in [1, \mathcal{S}^*_{k-1,j}]\} \end{aligned} \tag{12}$$

For the next iteration we also need to remove all the requests from the set of requests that have already been serviced by schedules selected in the current iteration:

$$\mathcal{R}_{k+1} = \mathcal{R}_k \setminus \bigcup_{i=1}^{|\mathcal{S}^*_k|} R(\mathcal{S}^*_{k,i}) \tag{13}$$

Finally, to keep track of all the schedules and associated transitions, we maintain a set of all schedules selected, $\mathcal{S}$, and a function $\mathcal{N} : \mathcal{S} \to \mathcal{S}$, that maps all schedules to their transitions. In the case that a given schedule $s \in \mathcal{S}$ has no outgoing transitions, $\mathcal{N}(s) = \emptyset$.

Now that the remaining requests have been gathered and schedules have been selected, we can move on to the next iteration. An overview of the schedule chaining algorithm is shown in Algo. 1. In the algorithm we use the short hand functions LatestRequests($\mathcal{R}_k$) to represent the n latest requests from $\mathcal{R}^k$, ComputeSchedules($\mathcal{R}_k^{\text{latest}}$) to represent all the schedules computed from $\mathcal{R}_k^{\text{latest}}$, and SolveScheduleSelectionILP($\mathcal{S}_k, \mathcal{S}^*_{k-1}, \mathcal{R}_k^{\text{latest}}, \mathcal{R}_k^{\text{assoc}}$) to represent solving Eq. (11) and extracting the schedules selected from the optimization variables.

IV. Long Term Rebalancing

After computing the starting schedules using the schedule chaining algorithm, there may be more starting schedules selected than necessary due to the constraint on the maximum idling time for a vehicle. We perform what we call *long term rebalancing*, which relaxes the constraint on the maximum idling time after the last iteration of the schedule chaining algorithm to reduce the total number of vehicles needed in the fleet. Long term rebalancing computes a matching from schedules which have no outgoing transitions to schedules that have no incoming transitions using a larger maximum idling time, denoted as T_{reb}, while ensuring the delay and waiting time constraints for the future schedules remain satisfied.

Let us define the two sets for the matching problem, the set of schedules with no outgoing transitions, $\mathcal{A} = \{s :$

Algorithm 1 Overview of the Schedule Chaining Algorithm

$\mathcal{S} \leftarrow \{\}$
$\mathcal{S}_0 \leftarrow \{\}$
$\mathcal{R}_1 \leftarrow$ AllRequests()
$k \leftarrow 1$
while $|\mathcal{R}_k| > 0$ **do**
  $\mathcal{R}_k^{\text{latest}} \leftarrow$ LatestRequests($\mathcal{R}_k$)
  $\mathcal{S}_k \leftarrow$ ComputeSchedules($\mathcal{R}_k^{\text{latest}}$)
  $\mathcal{R}_k^{\text{assoc}} = \bigcup_{i=1}^{|\mathcal{S}_k|} R(\mathcal{S}_{k,i})$
  $\mathcal{S}_k^* \leftarrow$ ScheduleSelectionILP($\mathcal{S}_k, \mathcal{S}_{k-1}^*, \mathcal{R}_k^{\text{latest}}, \mathcal{R}_k^{\text{assoc}}$)
  $\mathcal{R}_{k+1} \leftarrow \mathcal{R}_k \setminus \bigcup_{i=1}^{|\mathcal{S}_k^*|} R(\mathcal{S}_{k,i}^*)$
  $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{S}_k^*$
  for $i = 1$ **to** $|\mathcal{S}_k|$ **do**
    for $j = 1$ **to** $|\mathcal{S}_{k-1}^*|$ **do**
      if $s_i = 1 \wedge \epsilon_{ij} = 1$ **then**
        $\mathcal{N}(\mathcal{S}_{k,i}) \leftarrow \mathcal{S}_{k-1,j}^*$
      end if
    end for
  end for
  $k \leftarrow k + 1$
end while

$\mathcal{N}(s) = \emptyset, \forall s \in \mathcal{S}\}$, and the set of schedules without any incoming transitions, $\mathcal{B} = \{s : \nexists u \mathcal{N}(u) = s, \forall s \in \mathcal{S}\}$. To minimize the total number of vehicles needed, we will perform a maximum cardinality bipartite matching between sets $\mathcal{A}$ and $\mathcal{B}$. We will do so using an ILP.

To formulate this ILP, let's define a set of binary variables, $E = \{\epsilon_{ij} : \forall i \in |\mathcal{A}|, \forall j \in |\mathcal{B}|\}$ which represent if schedule $\mathcal{A}_i$ should transition to schedule $\mathcal{B}_j$. With these variables, we can maximize the number of transitions to minimize the total number of vehicles needed in the fleet. The utility function is given as:

$$\mathcal{J}(E) = \sum_{i=1}^{|\mathcal{A}|} \sum_{j=1}^{|\mathcal{B}|} \epsilon_{ij} \tag{14}$$

We also need to constrain the matching so that the idling and waiting times for the schedules are respected. These are the same constraints as in Eq. (7) and (8), however a larger vehicle idling time, T_{reb}, is used:

$$\epsilon_{ij} \cdot \theta(\mathcal{A}_i, \mathcal{B}_j) \leq T_{\text{reb}} \tag{15}$$

$$\epsilon_{ij} \cdot \left(t_{\mathcal{A}_i}(|\mathcal{A}_i|) + \xi(\mathcal{A}_i, \mathcal{B}_j)\right) \leq t_{\mathcal{B}_j}(1) \tag{16}$$
$$\forall i \in [1, |\mathcal{A}|] \text{ and } \forall j \in [1, |\mathcal{B}|]$$

Since we are computing a matching, we need to apply flow constraints to the transition variables to guarantee that there will be at most one outgoing transition for any schedule in $\mathcal{A}$ and at most one incoming transition to any schedule in $\mathcal{B}$:

$$\sum_{i=1}^{|\mathcal{A}|} \epsilon_{ij} \leq 1 \quad \forall j \in [1, |\mathcal{B}|] \tag{17}$$

$$\sum_{j=1}^{|\mathcal{B}|} \epsilon_{ij} \leq 1 \quad \forall i \in [1, |\mathcal{A}|] \tag{18}$$

Now we can combine these constraints along with the cost function in Eq. (14) to formulate the ILP for long term rebalancing:

$$\max_{E} \quad \mathcal{J}(E) \tag{19}$$
$$\text{s.t.} \quad \text{constraints (15)} - \text{(18)}$$

V. Evaluation

We evaluate the proposed algorithm using historical taxi request data from Manhattan [17] and compare the performance to the actual efficiency of the taxi fleet from the data.

A. Experimental Setup

For the experiments, we used one month of historical taxi data from 00:00 on May 1st to 23:59 on May 31st, 2013. The data contains the origin, destination, pick up time and drop off time for all taxi requests in Manhattan. From this raw data, we use the reported pick up time as the request time since request time was not provided. The road network we use is extracted from OpenStreetMap [18] and the travel times were queried from Google Maps. The shortest paths and travel times between every pair of nodes in the road network were computed offline. We ran the algorithm independently for each day of the month. A computer with a 3.0 GHz, 36 core (72 thread) processor and 144GB of memory was used to run the experiments and we ran four instances of the algorithm at the same time (18 threads per instance).

We assess the performance of the algorithm using vehicles of capacity two and four. We use a fixed maximum waiting time of $T_{\text{wait}} = 3$ minutes and a maximum delay of $T_{\text{delay}} = 6$ minutes. The vehicle deposits were computed using a maximum travel time of $T_{\text{depos}} = 1$ minute and we use a batch size of $n = 10$ requests. The minimum acceptable service rate is $\beta = 1$ (i.e. we want to service all requests). The maximum idle time for schedule chaining is $T_{\text{idle}} = 30$ seconds and for rebalancing $T_{\text{reb}} = 24$ hours.

We compare the fleet sizes produced by the proposed algorithm with the actual fleet sizes used to service the requests.

B. Results

We collect several metrics to access the performance of the proposed algorithm including the travel delay, waiting time, vehicle idle time, passenger load, computational time, and fleet size. The travel delay and waiting time are defined in Sec. II-A. The vehicle idle time is the amount of time a vehicle would have to idle between schedules and is also precisely defined in Sec. II-A. The passenger load is the maximum number of passengers in a vehicle at any time for a given schedule. The computation time includes the time required to compute schedules and solve the schedule chaining ILP for all batches. The fleet size is the total number of vehicles needed to service all of the requests. The averages of these metrics for the whole month are shown in Table I. In the table we show the actual metrics from the raw data for comparison. We also plotted the fleet size and computational time as a function of the number of requests in Fig. 2.

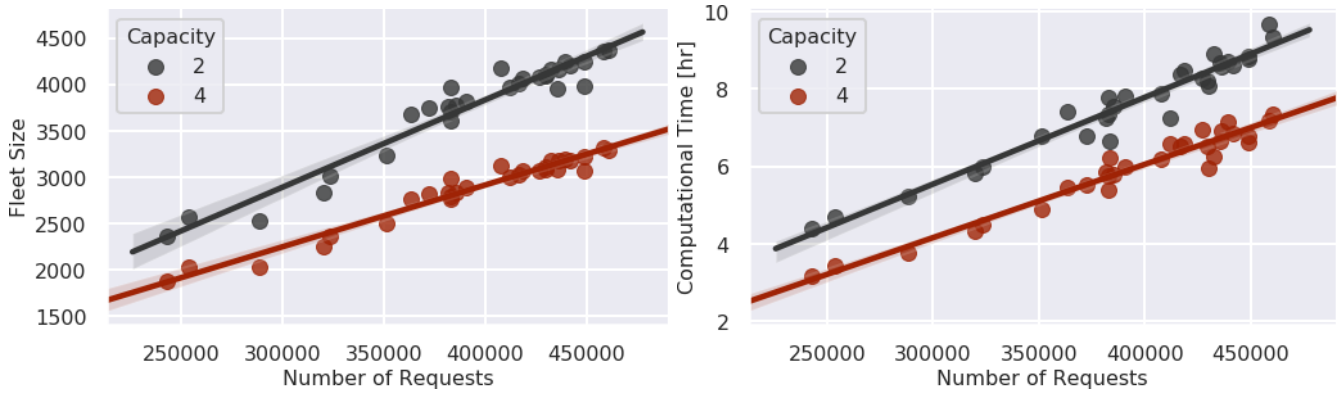

Fig. 2: A comparison of the fleet size and computational time needed for each day of the month for vehicle capacites of two and four passengers as a function of the number of requests in the day

	Avg. Delay [m]	Avg. Veh. Idle Time [m]	Avg. Load	Avg. Wait Time [s]	Avg. Comp. Time [hr]	Avg. Fleet Size
Capacity=2	1.7	4.8	1.9	35.4	7.6	3761.5
Capacity=4	2.8	4.7	3.1	41.9	5.9	2864.0
Actual	0.0	22.3	1.0	0.0	-	12237.4

TABLE I: Performance metrics and fleet sizes for vehicle capacities of two and four passengers compared to the performance of the current fleet of NYC taxis

We observe a large reduction in the required fleet size to service all the requests when using the proposed algorithm against the baseline. For a fleet with a two passenger vehicle capacity, on average we need 3761 vehicles and for a four passenger vehicle capacity fleet, we need 2864 vehicles. This is a 69% and 77% reduction in fleet size respectively compared to the baseline. The proposed algorithm also reduces the average vehicle idle time significantly from 22.3 mins to 4.7 mins for capacity two and 4.8 mins for capacity four. This is due to the fleet operating more efficiently.

The reduction in fleet size comes at the cost of added waiting time and associated travel delay, however our data shows that these costs are very low. Passengers on average would expect to experience an average travel delay 1.7 mins for a capacity two fleet and 2.8 mins for a capacity four fleet. The average waiting times are both less than a minute at 35.4 secs and 41.9 secs respectively.

We can see that the proposed algorithm is efficiently utilizing the vehicles by inspecting the average passenger load. For capacity two fleets, the average load is 1.9 passengers per schedule (95% seat utilization) and for capacity four fleets, it is 3.1 passengers per schedule (76% seat utilization).

The time it took to for the algorithm to compute and select the schedules for a given day took on average 7.6 hours and 5.9 hours for fleets of capacity two and capacity four vehicles respectively. The time is higher for capacity two fleets because it takes longer to iterate through the set of requests since more schedules are needed. This algorithm is not inteded for online use, but to generate statistics on historical demand to make decisions for fleet sizing. For this case, we think the computational times are sufficiently low to run the algorithm.

We plot the fleet size and computational time against the number of requests for a given day Fig. 2. In our experiments we observe that fleets of capacity two always have higher fleet sizes and take longer to compute than fleets of capacity four for a given number of requests. For our sample, we observe a linear trend in the fleet size against the number of requests for both vehicle capacities. We see that fleet size grows more rapidly for capcity two than for capacity four. For all days the fleet size remains under 4500 vehicles for capacity two and 3500 for capacity four.

VI. Conclusion

In this paper, we presented a method to optimize the vehicle distributions for a mobility-on-demand fleet. We presented an algorithm to determine how many vehicles are needed, where they should be initialized, and how they should move to service all the travel demand for a given period of time while allowing multiple passengers to be serviced by the same vehicle. The algorithm can be used to inform a ride-sharing fleet operator of how vehicles should be distributed to handle the demand for an area.

We demonstrated a significant improvement in the vehicle efficiency of the taxi fleet when using the proposed algorithm as compared to how taxis are currently utilized in Manhattan. On average, we can reduce the fleet size by 69% by allowing up to two passengers per vehicle and by 77% for up to four passengers per vehicle while guaranteeing all travel requests are serviced compared to the baseline. These benefits come at only a small cost to the user with an added travel delay of 1.7 mins and 2.8 mins and waiting times of 35.4 secs and 41.9 secs for vehicle capacities of two and four respectively.

Future work will focus on determining a lower bound on the fleet size so we have a range to inform fleet operators. We also plan to investigate how this method can be extended for online use for real-time fleet management.

REFERENCES

[1] M. Pavone, S. L. Smith, E. Frazzoli, and D. Rus, "Robotic load balancing for mobility-on-demand systems," *The International Journal of Robotics Research*, vol. 31, no. 7, pp. 839–854, 2012.

[2] K. Spieser, K. Treleaven, R. Zhang, E. Frazzoli, D. Morton, and M. Pavone, "Toward a systematic approach to the design and evaluation of automated mobility-on-demand systems: A case study in singapore," in *Road vehicle automation*, pp. 229–245, Springer, 2014.

[3] K. Treleaven, M. Pavone, and E. Frazzoli, "Asymptotically optimal algorithms for one-to-one pickup and delivery problems with applications to transportation systems," *IEEE Transactions on Automatic Control*, vol. 58, no. 9, pp. 2261–2276, 2013.

[4] A. Prorok and V. Kumar, "Privacy-preserving vehicle assignment for mobility-on-demand systems," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pp. 1869–1876, IEEE, 2017.

[5] G. Clare and A. G. Richards, "Optimization of taxiway routing and runway scheduling," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 4, pp. 1000–1013, 2011.

[6] R. Zhang and M. Pavone, "Control of robotic mobility-on-demand systems: a queueing-theoretical perspective," *Proceedings of Robotics: Science and Systems Conference*, July 2014.

[7] T. Rosenberg, "Its not just nice to share, its the future," 2013.

[8] A. Sundararajan, "From zipcar to the sharing economy," *Harvard Business Review*, vol. 1, 2013.

[9] P. Santi, G. Resta, M. Szell, S. Sobolevsky, S. H. Strogatz, and C. Ratti, "Quantifying the benefits of vehicle pooling with shareability networks.," *Proceedings of the National Academy of Sciences*, vol. 111, no. 37, pp. 13290–4, 2014.

[10] R. Tachet, O. Sagarra, P. Santi, G. Resta, M. Szell, S. Strogatz, and C. Ratti, "Scaling law of urban ride sharing," *Scientific reports*, vol. 7, p. 42868, 2017.

[11] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus, "On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment," *Proceedings of the National Academy of Sciences*, vol. 114, no. 3, pp. 462–467, 2017.

[12] A. Wallar, M. van der Zee, J. Alonso-Mora, and D. Rus, "Vehicle rebalancing for mobility-on-demand systems with ride-sharing," 2018.

[13] P. M. Boesch, F. Ciari, and K. W. Axhausen, "Autonomous vehicle fleet sizes required to serve different levels of demand," *Transportation Research Record: Journal of the Transportation Research Board*, no. 2542, pp. 111–119, 2016.

[14] K. Winter, O. Cats, G. H. d. A. Correia, and B. Van Arem, "Designing an automated demand-responsive transport system: Fleet size and performance analysis for a campus–train station service," *Transportation Research Record: Journal of the Transportation Research Board*, no. 2542, pp. 75–83, 2016.

[15] M. M. Vazifeh, P. Santi, G. Resta, S. H. Strogatz, and C. Ratti, "Addressing the minimum fleet problem in on-demand urban mobility," *Nature*, vol. 557, no. 7706, pp. 534–538, 2018.

[16] M. Čáp and J. Alonso-Mora, "Multi-objective analysis of ridesharing in automated mobility-on-demand," *Proceedings of Robotics: Science and Systems Conference*, 2018.

[17] B. Donovan and D. B. Work, "New York City Taxi Trip Data (2010-2013)." http://dx.doi.org/10.13012/J8PN93H8, 2014.

[18] OpenStreetMap contributors, "Planet dump retrieved from https://planet.osm.org ." https://www.openstreetmap.org , 2017.

[19] O. Tange, "Gnu parallel - the command-line power tool," *;login: The USENIX Magazine*, vol. 36, pp. 42–47, Feb 2011.