Proactive rebalancing and speed-up techniques for on-demand high capacity vehicle pooling

Yang Liu, Samitha Samaranayake

Abstract—By proposing speed-up techniques and a proactive rebalance algorithm, we improve the Mobility-on-Demand fleet management approach in [1] on both computational performance and system performance. The speed-up techniques comprise search space pruning and parallelization Input/Output reduction, which reduce the computation time by up to 97.67% in experiments on taxi trips in Manhattan, New York City. The proactive rebalancing algorithm guides idle vehicles to future demand based on a probability distribution estimated using historical data, which increases the service rate by 4.8% on average, and decreases the waiting time and total delay by 5.0% and 10.7% on average respectively.

Index Terms—Ridesharing, Mobility-on-Demand system, Vehicle routing, Proactive rebalancing

I. INTRODUCTION

The rapid development of on-demand ridepolling services, such as UberPool and Lyft Line, is disrupting urban mobility. These services allow multiple passengers traveling on similar routes to share a ride. They not only provide passengers with a reliable and low-cost travel mode but also help reduce the congestion and pollution associated with transportation. High capacity ridesharing services such as Via also offer affordable alternatives to public transit services.

These services are currently operated by vehicles with human drivers. However, the fleet will probably to comprise autonomous vehicles since millions of dollars are invested in the autonomous vehicle industry these years. Google planned to build a fleet of autonomous vehicles, and in Europe, low-speed (25 miles per hour) twelve-seat automated vehicles deployments were also underway through the CityMobil2 project [2]. With such advancements in automation technologies, demand-responsive autonomous mobility services are also gaining research interest [2]-[5].

To provide high-quality on-demand ridesharing service, efficient fleet management methods to optimize the assignment from the vehicles to travel requests are required. Most of the literature on Mobility-on-Demand (MoD) systems have focused on the case without pooling travel requests until recently [6], [7]. A recent study showed that 80% taxi trips in Manhattan, New York City (NYC) could be shared by two riders with only a few minutes travel time increase [8], which quantified the benefits of vehicle pooling on a real-size network. However, the method only provides the optimal solution when a ride can be shared by at most 2 riders, and is intractable for high capacity vehicle pooling. To overcome this difficulty, a computationally efficient anytime optimal fleet management algorithm was proposed to address both the problem of assigning vehicles to travel requests and the problem of rebalancing the idling vehicles for a high capacity fleet [1]. Traditional methods for fleet management problem usually formulate the problem as an Integer Linear Program (ILP), which is not tractable for large-scale instances due to the large solution space. The method in [1] decoupled the problem by first employing the pairwise shareability graph [8] to compute feasible trips and then using a compacted ILP to find the optimal assignment from the vehicles to feasible trips. Extensive experiments using taxi trips in Manhattan, NYC showed that the algorithm could provide a satisfying solution in real time for most of the cases. However, the computation time could still be high when the fleet size was large. For example, considering a fleet of 3000 four-seat vehicles, the simulation for a day’s travel demand could be as high as about 41 hours using a 24-core 2.5GHz computer [9]. In addition, MoD system performance can benefit from a more efficient vehicle rebalancing method since the method in [1] was simply matching idle vehicles to the locations of the unserved requests assuming that these requests may request again.

In this work, we present a series of techniques to improve this state-of-the-art framework in [1], which includes: i) search space pruning techniques when computing feasible trips; ii) a parallelization Input/Output (I/O) reduction technique; iii) improved vehicle assignment and vehicle rebalancing formulation; iv) a proactive rebalancing method. We demonstrate the performance of the techniques using taxi trips in Manhattan, NYC in the numerical experiments. These techniques would not only help researchers conduct simulation-based studies more quickly but also provide insights for the implementation of the method in the industry.

The rest of this article is structured as follows. In Section II, we briefly review the fleet management method in [1]. In Section III, we introduce speed-up techniques to improve computation performance. Section IV proposes a proactive rebalancing method to improve MoD system performance. Section V consists of numerical experiments on the Manhattan network to show the efficiency of the techniques. Finally, Section VI provides closing remarks and discusses possible directions for future research.

II. PRELIMINARIES

In this section, we introduce the state-of-the-art high capacity MoD service simulation framework in [1].
A. Problem definition

We consider a fleet of vehicles with a given capacity to satisfy the demand for MoD services. Two main tasks of the MoD fleet management are: i) to assign the vehicles to travel requests; ii) to rebalance_idle vehicles to areas where would likely have travel demand.

The set of vehicles is denoted by $\mathcal{V} = \{v_1,\ldots,v_n\}$. Each vehicle $v_i$ has a corresponding capacity $c_i$. The set of travel requests is denoted by $\mathcal{R} = \{r_1,\ldots,r_n\}$. Multiple travelers may share one ride. A passenger is defined as a past request that has been picked up by a vehicle and is on its path to the destination. For each request $r$, the waiting time denoted by $w_r$ is defined as the difference between the pickup time $t^p_r$ and the request time $t^r_r$. For each request $r$ that has been picked up (i.e. passenger), the total delay is defined as $\delta_r = t^d_r - t^r_r$, where $t^d_r$ is the dropoff time and $t^r_r$ is the earliest possible time at which the destination could be reached. We assume $t^r_r$ to be the travel time between the origin $o_r$ and the destination $d_r$ by following the shortest path. The goal is to find the optimal assignment that minimizes a cost function $C$ under the following constraints:

- For each request $r$, the waiting time $w_r$ must be below the maximum waiting time $\Omega_r$.
- For each request $r$ that has been picked up, the total delay $\delta_r$ must not exceed the maximum delay $\Delta_r$.
- For each vehicle $v$, the number of passengers in the vehicle has to be smaller than or equal to the capacity $c_v$.

If a request is not served under the above constraints, it is discarded in the simulation, and a large penalty $c_ko$ will occur in the cost function. The cost function $C$ is set to the sum of total delay and the penalty for unassigned requests.

The total delay includes both the waiting time for all assigned requests and the in-vehicle delay caused by sharing with other passengers.

$$C = \sum_{v \in \mathcal{V}} \sum_{r \in \mathcal{P}_v} \delta_r + \sum_{r \in \mathcal{R}_{ok}} \delta_r + \sum_{r \in \mathcal{R}_{ko}} c_{ko}$$

where $\mathcal{P}_v$ is the set of passengers in vehicle $v$, $\mathcal{R}_{ok}$ is the set of requests that have been assigned vehicles, and $\mathcal{R}_{ko}$ is the set of requests that are not assigned any vehicles.

The MoD fleet assignment problem is similar to the dynamic pickup and delivery problem with time windows, which has been shown difficult to solve for large-scale travel demand.

B. Method overview

The anytime optimal algorithm in [1] decouples the problem by first computing feasible trips based on a pairwise shareability graph and then finding the optimal trip-vehicle assignment by solving an ILP of reduced dimensionality. The assignment of the travel requests is processed every 30 seconds. After an assignment round, requests that have not been picked up remain in the request pool until the waiting time constraint is violated. Specifically, each round of the assignment simulation includes the following steps.

1) Construct a pair-wise request-vehicle graph (RV-graph). RV-graph describes possible pairwise matchings between vehicles and requests. In the graph, two requests $r_i$ and $r_j$ are connected if a virtual vehicle at the origin of either request can serve both under the constraints discussed in Section II-A. Similarly, a request $r$ is connected to a vehicle $v$ if $v$ can serve $r$ without violating the constraints.

2) Compute the Request-Trip-Vehicle graph (RTV-graph) using the cliques of the RV-graph. A feasible trip is defined as a set of requests that can be served by one vehicle without violating the constraints. RTV-graph consists of all the feasible trips and the vehicles that can serve them. A request $r$ is connected to a trip $T$ if $T$ contains $r$. A trip $T$ is connected to a vehicle $v$ if $v$ can serve $T$ under the constraints. For each vehicle, the feasible trips are computed incrementally in trip length, which reduces the search space significantly.

3) Solve an ILP to find the optimal assignment from the vehicles to feasible trips. The ILP in [1] is as follows:

$$\min \sum_{i,j \in TV} c_{ij} \cdot x_{ij} + \sum_{k \in \{1,...,n\}} c_{ko} \cdot \chi_k$$

subject to

$$\sum_{i \in TV_{r_j}} x_{i,j} \leq 1, \quad \forall v_j \in \mathcal{V} \quad (1)$$

$$\sum_{j \in TV_{r_k}} \sum_{i \in \mathcal{V}} x_{i,j} + \chi_k = 1, \quad \forall r_k \in \mathcal{R}$$

$$x_{i,j} \in \{0,1\}, \quad \forall i,j \in TV$$

where $TV_M$ is the set of all feasible assignments between trips and vehicles, $c_{ij}$ is the cost of vehicle $j$ serving trip $i$. The decision variables are $x_{i,j}$ and $\chi_k$, where $x_{i,j} = 1$ if vehicle $v_j$ is assigned to trip $i$, and $\chi_k = 1$ if the request $r_k$ is unassigned in this round of assignment simulation. In the constraints, there are three sets: the set of trips that can be served by vehicle $j$ is $TV_{r_j}$; the set of trips that contains request $r_k$ is $TV_{r_k}$; the set of vehicles that can serve trip $i$ is $TV_{i}$.

The constraints ensure that: i) each vehicle will serve at most one trip; ii) each request is either served by one vehicle or ignored ($\chi_k = 1$) in the assignment.

4) Rebalance idle vehicles. After the assignment, there may be idle vehicles (no trips assigned to the vehicle) and unsatisfied requests (no vehicles assigned). Assuming that more requests will be likely to appear in the neighborhood of the unsatisfied requests, a linear program (LP) is solved to match the idle vehicles to the locations of unsatisfied requests while minimizing the rebalancing cost.

$$\min \sum_{v \in \mathcal{V}_{idle}} \sum_{r \in \mathcal{R}_{ko}} \tau_{v,r} y_{v,r}$$

subject to

$$\sum_{v \in \mathcal{V}_{idle}} \sum_{r \in \mathcal{R}_{ko}} y_{v,r} = \min(|\mathcal{V}_{idle}|, |\mathcal{R}_{ko}|) \quad (2)$$

$$0 \leq y_{v,r} \leq 1, \quad \forall v_{r}, r \in \mathcal{V}$$

where $\mathcal{V}_{idle}$ is the set of idle vehicles, $\tau_{v,r}$ is the travel time between the vehicle $v$’s location and the origin of the unassigned request $r$, $y_{v,r}$ is the decision variable where $y_{v,r} = 1$ if the vehicle $v$ is assigned the task to travel to
the origin of \( r \) and 0 otherwise, and \( \mathcal{Y} \) is the set of decision variables.

III. SPEED-UP TECHNIQUES TO IMPROVE ON-DEMAND HIGH CAPACITY VEHICLE POOLING

In this section, we propose speed-up techniques to improve the on-demand high capacity vehicle pooling framework. The techniques in this section do not impair the anytime optimality of the framework. Readers can also refer to [13] for approximation methods.

A. Search space pruning

In [1], an exhaustive search is conducted to check if a vehicle can serve a trip. It is time-consuming for a trip with a high number of requests. In addition, the number of trips increases exponentially with the number of requests. Therefore, we propose techniques to: i) reduce the number of times required to conduct an exhaustive search; ii) decrease the computation time of each exhaustive search.

Claim 1. Given a network with static travel time, if no vehicle can serve trip \( T \) under the constraints in the current round of assignment simulation, there will not be any vehicles that can serve trip \( T \) in the future.

As the travel time is not changing dynamically, the above claim is intuitively true. In [1], when a trip is not feasible for vehicle \( v \) in the current round of assignment simulation, it is still considered for \( v \) in the following rounds. To reduce such duplicate computation, we propose the following technique:

Claim 2. For any trip \( i \), an array \( A_i \) can be used to record the set of feasible vehicles. In the following rounds of assignment simulation, only the vehicles in set \( A_i \) are considered for trip \( i \). We keep updating \( A_i \) and discard trip \( i \) whenever \( |A_i| = 0 \).

As the number of trips increases exponentially with the number of requests, it can be memory-consuming if we consider all trips. In addition, the set of feasible vehicles is small for a trip of multiple requests. Therefore, the computation time saved by the technique is diminishing as the length of the trip increases. In our work, we use the technique only for the trips with one request. Given a network with dynamic travel time, the technique can be used as an efficient approximation since the travel time will not change significantly during a trip.

The exhaustive search in [1] checks the feasibility of every possible route (order of pickups and dropoffs for the requests in the trip) for a trip. For each route, the framework lets the vehicle virtually follow it and check the constraints during the process. In [1], the waiting time constraint is checked when traversing a request’s origin, and the delay constraint is checked when traversing a request’s destination. The following technique can potentially decrease the computation time of the exhaustive search process for each route.

Claim 3. During checking a trip’s feasibility, whenever the vehicle traverses an origin or a destination, the constraints are checked for every node in the route that has not been traversed.

If any constraint is violated, the route is not feasible.

The intuition is that when the vehicle arrives at some new location, some requests may already be infeasible due to the vehicle’s detour on the prior route section, and continuing to follow the route causes redundant computation.

B. Parallelization I/O reduction

In [1], when constructing the RV-graph, each request is connected to all the vehicles that can serve it under the constraints. Specifically, for each request-vehicle pair, the framework checks if exists a feasible route to serve the request and all the passengers in the vehicle. This is conducted via the exhaustive search and is parallelized among the requests. Each request is assigned to a worker in the pool of worker processes, and the set of possible vehicles is also input to it. Here possible vehicles represents that if the vehicle travels directly to the origin of the request, it can pick up the request within the waiting time constraint. As the set of possible vehicles for different requests may overlap with each other, each worker can reduce I/O communication by processing multiple requests in a batch. To minimize the I/O overhead, we propose the problem of optimizing the request-worker assignment:

\[
\text{minimize } \sum_{i \in \mathcal{N}} \bigcup_{r \in \mathcal{R}, z_{i,r} = 1} \mathcal{V}_r \\
\text{subject to } \sum_{i \in \mathcal{N}} z_{i,r} = 1 \quad \forall r \in \mathcal{R} \\
z_{i,r} \in \{0, 1\} \quad \forall i \in \mathcal{N}, r \in \mathcal{R}
\]

where \( \mathcal{N} \) is the set of workers, \( \mathcal{V}_r \) is the set of possible vehicles for request \( r \), \( z_{i,r} \) is the decision variable, where \( z_{i,r} = 1 \) represents that request \( r \) is assigned to worker \( i \), and 0 otherwise. The aim is to minimize the number of vehicles input to the workers in total. The problem is similar to the weighted set partitioning problem but more difficult since the weight for each set is not known before \( z_{i,r} \) is set.

The set of possible vehicles should be similar for two requests that are spatially close to each other. Therefore, instead of solving the above problem directly, we use the K-means clustering algorithm [14] to cluster the requests based on their coordinates, and assign each cluster of requests to different workers. The number of clusters \( k \) is chosen according to the number of workers in the computer.

C. Improved ride-vehicle assignment formulation

In this work, we use the following formulation for trip-vehicle assignment which is mentioned in [15]. The formulation reduces the number of variables by \( |\mathcal{R}| \) compared to the formulation discussed in Section II-B.

2Enumerate every possible order of pickups and dropoffs for the requests to find the optimal route and check if it violates any constraints.
tasks in consecutive rounds, and circle around inefficiently in the algorithm in [1] is conducted in each round of assignment. Therefore, we first add a constraint to ensure that the spatial coverage of the fleet. We add one constraint to avoid vehicles rebalancing to one neighborhood and decrease the number of variables if multiple vehicles can be assigned to one unsatisfied request if the procedure in the current round of simulation.

In the original formulation discussed in Section II-B, multiple vehicles can be assigned to one unsatisfied request if we do not need $\xi_k$ for each request in the formulation, but the solution is equivalent.

IV. PROACTIVE REBALANCING FOR RIDE SHARING FLEETS

In this section, we propose a proactive rebalancing vehicle rebalancing framework, which can potentially improve MoD system performance. The framework comprises the following changes: i) we use a new rebalancing formulation, which guarantees that the solution is a matching between the idle vehicles and unsatisfied requests; ii) we propose a way to incorporate proactive rebalancing in the rebalancing formulation.

1) New vehicle rebalancing formulation: The rebalancing algorithm in [11] is conducted in each round of assignment simulation. A vehicle may be assigned different rebalancing tasks in consecutive rounds, and circle around inefficiently in the network. Therefore, we first add a constraint to ensure that if a vehicle is assigned a rebalancing task in previous rounds of simulation, it will not be considered in the rebalancing procedure in the current round of simulation.

In the original formulation discussed in Section II-B multiple vehicles can be assigned to one unsatisfied request if the request is close to these vehicles. This can make the vehicles rebalance to one neighborhood and decrease the spatial coverage of the fleet. We add one constraint to avoid this disadvantage.

Formulating the problem for the solver will be time-consuming when $|R_{ko}|$ is large. In our experiments, we set an upper bound to both the number of vehicles and requests in the formulation, which are denoted by $V_{\text{max}}$ and $R_{\text{max}}$ respectively. Specifically, this is done via the following steps:

1) New vehicle rebalancing formulation: The rebalancing algorithm in [11] assumes new requests will appear in the same areas where the unsatisfied requests are. However, the assumption may not be true if the demand does not show short-term recurring patterns. Therefore, we propose a way to conduct proactive rebalancing based on the probability distribution estimated using historical data.

A recent study proposes a predictive routing method for MoD system by adding virtual request samples. A potential reason is that after sampling, the probability for virtual requests to appear in reality is not considered. Some virtual requests may be rare historically, which harm the system performance; ii) the computation time increases significantly when adding several hundred virtual requests in each round of assignment simulation. Out proactive rebalancing method will also help guide vehicles to the locations of potential future requests, but it does not have the above two disadvantages.

Since the demand is sparsely distributed on thousands of network nodes in a real-size network, it is hard to predict the future demand at the node level. As we only need to guide the vehicles to areas where requests are likely to appear, we use the K-means clustering algorithm to spatially cluster the nodes according to their geo-coordinates. We assume that each passenger has a walking range of $\alpha$ miles. Then, the number of clusters $k$ is determined by satisfying $\text{Total area} \approx 2\pi\alpha^2$. Based on historical data, we build the probability distribution $P(n | o, \xi)$, which is the probability of $n$ requests appearing in cluster $o$ given a time interval $\xi$. A time interval $\xi$ is defined by a tuple $(t_s, t_e)$ where $t_s$ and $t_e$ are the start time and the end time of the time interval respectively. Let $n^5$ be the maximum number of requests appearing in cluster $o$ at time interval $\xi$ according to historical data. We can generate $\sum_{o \in O} n^5$ virtual

\begin{align}
\text{minimize} & \sum_{i,j \in X_{TV}} (c_{i,j} - c_{ko} \cdot l_i \cdot x_{i,j}) \\
\text{subject to} & \sum_{i \in X_{V,i}} x_{i,j} \leq 1, \quad \forall v_j \in V \\
& \sum_{i \in X_{V,i}} \sum_{j \in X_{V,j}} x_{i,j} \leq 1 \quad \forall v_k \in V \\
& x_{i,j} \in \{0, 1\}, \quad \forall i, j \in X_{TV}
\end{align}

where $l_i$ is the number of requests in trip $i$. We track $l_i$ for each trip $i$. We consider all the requests unassigned before the assignment, and thus a total penalty $c_{ko} \cdot |R|$ is imposed to the objective function. Whenever we assign a vehicle $j$ to a trip $i$, the system gets a reward $c_{ko} \cdot l_i$ for picking up $l_i$ requests in the trip. The number of variables is reduced by $|R|$ because we do not need $\xi_k$ for each request in the formulation.

Note that in this section we are still rebalancing idle vehicles to the locations of unsatisfied requests as in [11]. We will replace $R_{ko}$ by potential future requests in Section IV-2. The new formulation ensures that the solution is a valid matching between the vehicles and the requests. However, it increases the number of constraints by $|R_{ko}|$, and for each of these constraints, we have to calculate a summation of $|V_{\text{idle}}|$ terms.

3This term is ignored in the objective function since adding constants does not affect the optimal solution.
requests for time interval $\xi$, where $O$ is the set of clusters. For each virtual request $r$, we set the probability of it appearing at time interval $\xi$, which is denoted by $p_r$, as follows.

Assume that we are considering cluster $o$ and time interval $\xi$. The virtual requests are denoted by $r_1, r_2, \ldots, r_n$. As the definition reveals, $P(n \mid o, \xi)$ is the joint probability of exactly $n$ requests occurring among all $n^o$ requests. What we want to use is the marginal probability $p_r$, for $1 \leq i \leq n^o$. We cannot solve all these marginal probabilities because the correlations between the requests are unknown. To help get a solution, we assume that $r_i$ can only appear when all requests in the set \{r$_j$ | 1 $\leq$ j $<$ i\} appear. This assumption implies the order on these requests, i.e., when there are $i$ requests appearing, they would be $r_1, r_2, \ldots, r_i$. Under this assumption, we derive that $p_{r_i} = \sum_{n=0}^{n^o} P(n \mid o, \xi)$.

We can use formulation in Section IV.C considering the set of virtual requests with a probability higher than $p_{\text{min}}$.

The number of virtual requests can be large when the demand is high. To reduce the scale of the optimization problem, we ignore the virtual requests if there exists an idle vehicle within a predefined travel time range. In summary, this method has the following advantages: i) Compared to the predictive method in [16], our proactive rebalancing method requires predictions of only the origins instead of origin-destination (O-D) pairs. The probabilistic prediction is under less noise. Since we only consider the requests with a high probability, we are more confident that the vehicles are guided to areas with potential future requests; ii) The method does not increase the scale of the trip-vehicle simulation; iii) Since we proactively send idle vehicles to areas where future requests appear, the waiting time and delay for those requests can be potentially decreased.

V. NUMERICAL EXPERIMENTS

A. Experimental setup

We use taxi trip data in Manhattan, NYC from 6 am to 12 am on an arbitrary day (Monday, May 6th, 2013) as the travel demand. The network we use is the entire road network of Manhattan (4092 nodes and 9453 edges) [1], [8]. The link travel time is the daily mean travel time, which is computed using the method in [8]. The shortest paths between every two nodes in the network are precomputed and stored in a look-up table. We evaluate the performance of the techniques by comparing the metrics of interest in varying cases (different fleet sizes, capacities, maximum waiting time, maximum delay) between three algorithms: i) the framework in [1], denoted by original; ii) our framework with the speed-up techniques, denoted by speed-up; iii) our framework with the speed-up techniques and the proactive rebalancing algorithm, denoted by speed-up+proactive. The system is implemented using Python 3.5, and all experiments are conducted on a 4 core 3.4GHz computer. The maximum waiting time and the maximum total delay are assumed to be the same for all requests, which are denoted by $\Omega$ and $\Delta$ respectively.

B. Experimental results

We conduct two sets of experiments with a fleet of capacity 4 vehicles and a fleet of capacity 10 vehicles respectively. During the experiments, we collect the following metrics: the mean computation time (average computation time for a round of assignment simulation), service rate, mean waiting time and mean total delay. These metrics are shown in Table I.

We first analyze the performance of the speed-up framework. The speed-up techniques decrease the computation time significantly in all cases with no optimality loss. Specifically, the average computation time reduction is as high as 87.46%, while the mean waiting time and mean in-vehicle delay remain at the same level. The maximum computation time reduction is 97.67% when operating a fleet of 1000 capacity 10 vehicles with $\Omega = 300$ and $\Delta = 600$. When $\Omega$ and $\Delta$ increases, the computation time of the original framework increases significantly, but the computation time of our speed-up framework remains at the same magnitude. In addition, the service rate is increased by 3.5% on average. For our experiment time span, this represents that 4068 more passengers are served in 6 hours. Two potential reasons for this to happen: i) The search space pruning technique discussed in Claim[3] allows us to use less time to conduct the exhaustive search to check a trip’s feasibility. Therefore, we can explore more feasible solutions in the RTV-graph given a computation time limitation; ii) the rebalancing formulation is more efficient due to the newly added constraints. The maximum gap (11.9%) happens when operating a fleet of 3000 capacity 4 vehicles with $\Omega = 120$ and $\Delta = 240$.

The speed-up+proactive framework offers a 81.44% computation reduction on average compared to the original framework. The average computation reduction is slightly lower than the speed-up framework because the number of virtual requests can be higher than the number of unassigned requests. However, the service rate is increased by 4.8% on average, which is higher than the speed-up framework. In addition, the waiting time and total delay are decreased by 5.0% and 10.7% respectively. The largest gap happens when operating a fleet of 3000 capacity 4 vehicles when $\Omega = 120$ and $\Delta = 300$, where the speed-up+proactive framework reduces the waiting time and the total delay on average by 24.28 seconds and 68.51 seconds respectively.

VI. CONCLUSION

We presented a series of techniques to improve ride-vehicle assignment and fleet rebalancing, which can be seamlessly integrated with the state-of-the-art method in [1]. We showed experimentally that these techniques reduce the computation time by up to 97.67%. We also propose a proactive rebalancing algorithm, which increases the service rate by 4.8%, and
decreases the waiting time and total delay on average by 5% and 10.7% on average. We believe the techniques will not only help researchers to conduct MoD system simulation more quickly but also provide insights for implementing the framework in the industry. Future work will be but not limited to: i) designing speed-up techniques that can be employed in networks with dynamic travel time; ii) analyzing the sensitivity of the hyperparameters in the rebalancing algorithms.

REFERENCES

[1] J. Alonso-Mora, S. Samarayake, 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.
[2] D. J. Fagnant and K. M. Kockelman, “Dynamic ride-sharing and fleet sizing for a system of shared autonomous vehicles in austin, texas,” Transportation, vol. 45, no. 1, pp. 143–158, 2018.
[3] M. W. Levin and S. D. Boyles, “A multiclass cell transmission model for shared human and autonomous vehicle roads,” Transportation Research Part C: Emerging Technologies, vol. 62, pp. 103–116, 2016.
[4] K. Spieser, S. Samarayake, W. Gruel, and E. Frazzoli, “Shared-vehicle mobility-on-demand systems: a fleet operator’s guide to rebalancing empty vehicles,” in Transportation Research Board 95th Annual Meeting, no. 16-5987, 2016.
[5] H. Dia and F. Javanmohr, “Autonomous shared mobility-on-demand: Melbourne pilot simulation study,” Transportation Research Procedia, vol. 22, pp. 285–296, 2017.
[6] 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. Springer, 2014, pp. 229–245.
[7] R. Zhang and M. Pavone, “Control of robotic mobility-on-demand systems: a queueing-theoretical perspective,” The International Journal of Robotics Research, vol. 35, no. 1-3, pp. 186–203, 2016.
[8] P. Santi, G. Resta, M. Szel, 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. 13 290–13 294, 2014.
[9] J. Alonso-Mora, S. Samarayake, A. Wallar, E. Frazzoli, and D. Rus, “On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment–supplemental material–,” 2017.
[10] M. W. Savelsbergh and M. Sol, “The general pickup and delivery problem,” Transportation science, vol. 29, no. 1, pp. 17–29, 1995.
[11] W. P. Nancy and J. W. Barnes, “Solving the pickup and delivery problem with time windows using reactive tabu search,” Transportation Research Part B: Methodological, vol. 34, no. 2, pp. 107–121, 2000.
[12] S. Ropke and D. Pisinger, “An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows,” Transportation science, vol. 40, no. 4, pp. 455–472, 2006.
[13] C. Dutta, “When hashing met matching: Efficient search for potential matches in ride sharing,” arXiv preprint arXiv:1809.02680, 2018.
[14] J. MacQueen et al., “Some methods for classification and analysis of multivariate observations,” in Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, vol. 1, no. 14. Oakland, CA, USA, 1967, pp. 281–297.
[15] Y. Liu, P. Bansal, R. Daziano, and S. Samarayake, “A framework to integrate mode choice in the design of mobility-on-demand systems,” https://doi.org/10.1016/j.trc.2018.09.022, 2018.
[16] J. Alonso-Mora, A. Wallar, and D. Rus, “Predictive routing for autonomous mobility-on-demand systems with ride-sharing,” in Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on, IEEE, 2017, pp. 3583–3590.
[17] B. Donovan and D. Work, “New york city taxi trip data (2010–2013),” 2014.

Table I: Algorithm performance comparison.

| Number of vehicles | Capacity | Original | Speed-up | Speed-up+Proactive | Algorithm | Computation time (s) | Service rate | Waiting time (s) | Total delay (s) |
|--------------------|---------|----------|----------|-------------------|----------|---------------------|-------------|----------------|----------------|
| 1000               | 120     | 2,40     | 1,37     | 1,02              | Speed-up | 1.46                | 10.67      | 73.58          | 132.43         |
| 2000               | 4       | 300      | 2,94     | 2,41              | Speed-up | 0,97               | 0,97       | 0,97           | 0,97           |
| 3000               | 10      | 300      | 600      | 5,04              | Speed-up | 3,95               | 3,95       | 3,95           | 3,95           |

**Revised Table I: Algorithm performance comparison.**

| Number of vehicles | Capacity | Original | Speed-up | Speed-up+Proactive | Algorithm | Computation time (s) | Service rate | Waiting time (s) | Total delay (s) |
|--------------------|---------|----------|----------|-------------------|----------|---------------------|-------------|----------------|----------------|
| 1000               | 120     | 2,40     | 1,37     | 1,02              | Speed-up | 1.46                | 10.67      | 73.58          | 132.43         |
| 2000               | 4       | 300      | 2,94     | 2,41              | Speed-up | 0,97               | 0,97       | 0,97           | 0,97           |
| 3000               | 10      | 300      | 600      | 5,04              | Speed-up | 3,95               | 3,95       | 3,95           | 3,95           |

**Table II: Speed-up and Proactive results.**