DemandProp: a scalable algorithm for real-time predictive positioning of fleets in dynamic ridesharing systems

Jesper Provoost
provoost@kth.se
University of Twente
Enschede, The Netherlands

Győző Gidófalvi
gyozo.gidofalvi@abe.kth.se
Royal Institute of Technology (KTH)
Stockholm, Sweden

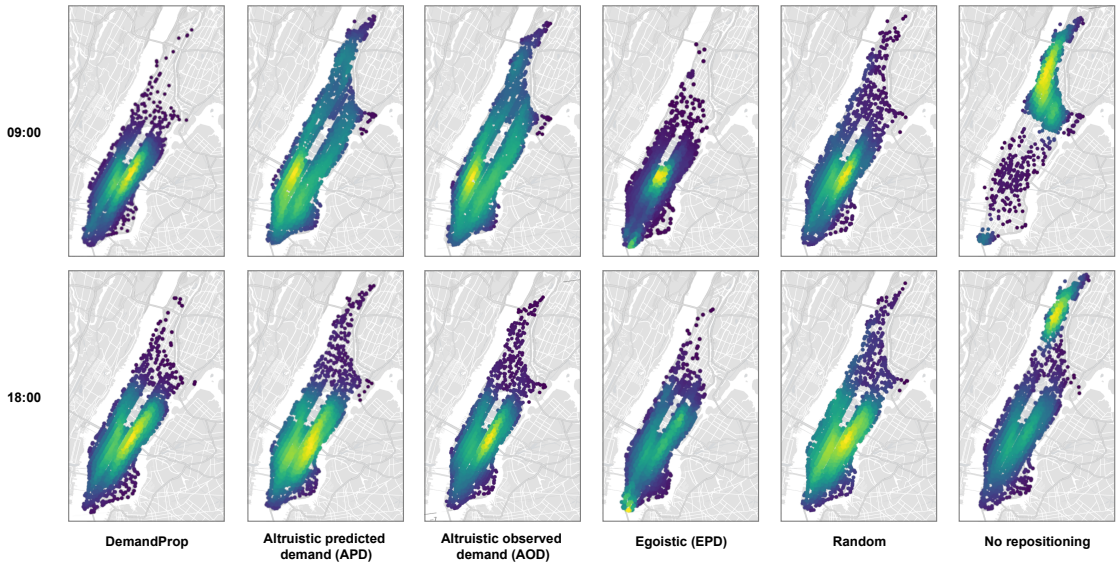


Figure 1: Positions of fleet subject to DemandProp and baseline repositioning strategies

ABSTRACT

In dynamic ridesharing systems, repositioning of idle vehicles is an essential operational task to maximize resource efficiency and improve customer satisfaction. In most practical taxi services, drivers position themselves to maximize their individual benefits, leading to decentralized decision-making. Moreover, knowledge about current and past demands is already outdated by the time that drivers arrive at their target position. As a consequence, the system-wide performance is likely to be suboptimal. Most existing approaches to intelligent fleet repositioning only make decisions based on past observations, do not consider shareability of trips or are unsuitable for real-time operation in networks with thousands of road segments. We therefore present a real-time predictive fleet positioning technique that relies on deep neural networks (DNNs) and a stochastic model to optimally (re)position idle vehicles in a ridesharing system. Our approach, called DemandProp, works on large-scale graph representations of the road network and enables fine-grained and precise decision-making. We evaluated our approach through extensive simulation studies on real-world datasets from New York City, USA. The results demonstrate that our approach outperforms all baseline methods with regard to resource efficiency and transport efficiency. Compared to an egoistic strategy, which is most prevalent in traditional taxi systems, delays can be reduced by 69% and on-time performance can be increased by more than 15%. Also, the daily idle time for vehicles can be reduced by 60 minutes. DemandProp enables operators to reduce the number of vehicles in the fleet by more than 25% while ensuring satisfactory on-time performance and reject rates. Our results also show that DemandProp excels especially in situations where most vehicles are already occupied, demonstrating its ability of performing fine-grained and proactive optimization also during peak hours.

1 INTRODUCTION

Shared transportation services allow multiple persons or goods to complete their trip inside the same vehicle. It is regarded as an essential component in sustainable urban transportation, as it increases vehicle utilization while alleviating the burden on resources and infrastructure. Implementing shared transportation systems can therefore provide substantial benefits with regard to operational costs, traffic congestion and consequently pollution. [2].

Ridesharing is a clear example of shared transportation which concerns movement of people. Traditional ridesharing approaches, where drivers offer a seat in their vehicles to passengers who are travelling in the same direction, are suitable for large-distance travel but cannot provide enough flexibility to be implemented for short, high-frequency trip requests. Dynamic ridesharing, which has been facilitated by the worldwide growth of smartphone usage

and social networking, provides more flexibility by fulfilling on-demand trip requests in real-time. Customers of such a service can request a trip through their smartphone (by providing a real-time GPS location), after which the trip requests are assigned to a vehicle [5]. As of today, most work in this area has focused on optimizing the allocation of vehicles to trip requests, i.e. solving the request-trip-vehicle assignment problem [4, 7, 18, 21].

Another substantial influence on the transport and resource efficiency of shared transportation networks is the positioning of idle vehicles inside the network [16]. This will be the focus of this paper. In traditional taxi systems, drivers are individual agents who decide where they position themselves, primarily with the objective of maximizing their individual likelihood of finding a customer. Their decision is often based on observations of the past or knowledge about the current situation. However, with this *egoistic* and *reactive* strategy, drivers cannot properly harmonize their decisions and the knowledge about demand will already be outdated by the time that drivers arrive at their target position. Hence, the system-wide performance is likely to be suboptimal. To illustrate, when drivers are informed that the current demand at a train station is high due to a large number of incoming trains, many might reposition themselves to this area, leading to an oversupply at the station and a shortage of vehicles in areas with smaller but scattered demands. As a consequence, system-wide customer delays will increase and more vehicles will remain underutilized.

In shared transportation systems, optimizing repositioning strategies is even more essential to ensure that requests which are shareable can be served by as little vehicles as possible, thus making optimal use of the resources that are available throughout the network. Currently, repositioning in ridesharing systems is often done by raising prices in areas with higher demand, and therefore providing an incentive for drivers to position themselves in those areas. Examples of this are the "dynamic pricing" strategies used by companies like Lyft and Uber [6]. However, even though they can be effective to balance the supply and demand in a decentralized system, these approaches can still be regarded as egoistic and lack the ability to perform system-wide optimization at fine granularity. The characteristics of dynamic shared transportation, i.e. real-time availability of knowledge about trip requests and vehicle dispatch decisions, make it feasible to move from an egoistic and reactive repositioning strategy to a more altruistic and proactive one, in order to obtain a better system-wide performance in terms of resource efficiency and QoS [5].

This paper introduces *DemandProp*; a real-time fleet positioning technique that relies on deep neural networks (DNNs) and a stochastic model to optimally (re)position vehicles inside a ridesharing system. DemandProp is able to predict demands for all edges in the network with high accuracy, and applies algorithms to propagate the expected demand values towards the idle vehicles in the network, with the aim of maximizing cumulative expected demand served by all vehicles while minimizing the distance travelled to serve those demands. DemandProp is scalable and resilient and is able to optimize in real-time (i.e. at minute interval) for systems with more than 5,000 vehicles and 9,000 edges in the road network. DemandProp is also ready for processing streamed input data regarding demand, as well as computation offloading to vehicles.

The contributions of this paper can be summarized as follows:

- We propose a scalable real-time ridesharing repositioning algorithm with the aim of minimizing customer delays and maximizing resource utilization. To the best of our knowledge, this is the first work which is able to deliver fine-grained and precise decision-making at real-world scale level of a ridesharing service, by incorporating the complete graph structure of the road network into the prediction and repositioning process.
- We propose algorithms that can match the trip requests from real-world datasets to vertices in the graph. By means of shortest path routing, the estimated flows on the edges can be computed for every timestep. Subsequently, aggregation is performed which yields an estimation of the flow on every edge at any given time.
- We propose two deep neural networks configurations which, based on real-world trip datasets, are able to predict for any timestamp: 1) the aggregated flow for every edge, and 2) the demand for every vertex in the network. The models can predict using temporal inputs and a lookback window of demand in the past hour, hence supporting real-time streaming input data for improved accuracy. Both DNNs deliver predictions for the entire graph (i.e. > 3500 vertices and > 8500 edges) with high precision ($R^2 > 0.9$) while maintaining low prediction times (< 0.1 seconds).
- We experimentally evaluate the DemandProp algorithm and demonstrate its effectiveness and efficiency by means of simulations of a large-scale real-world ridesharing system. The simulation is powered by a full-scale graph representation of the road network, an extensive trip dataset, as well as a travel time prediction model.

2 RELATED WORK

In recent years, several techniques for on-demand vehicle dispatching and repositioning have been proposed in literature. However, most of these techniques target traditional taxi systems and do not consider the concept of shareability which is essential in making shared transportation services effective [14, 15, 17]. Moreover, many of the works are aimed at the objectives of the individual drivers instead of the system-wide performance [12]. Others consider only fixed locations as possible start and end positions of trips (e.g. bikesharing systems with permanent stations) [11], which differs in complexity from the problem under study where new customers can be picked up at any position inside the network. There are other works available which consider shareability inside ridesharing networks. For instance, Santi et al. [19] propose shareability networks as a method to quantify the benefits of ridesharing. Vazifeh et al. [23] build further upon this work, using classic graph-theoretic methods to compute the potential reduction of fleet size. Even though the authors argue that their methods can be used for real-time vehicle dispatching, the repositioning of idle vehicles is not considered.

Some work has been done regarding idle repositioning in ridesharing systems, for example Braverman et al. [9] who integrate a closed queueing network model to route empty vehicles. Furthermore, [20] and [24] also propose a queueing theory approach to idle vehicle repositioning. However, even though the authors demonstrate that

their respective techniques outperform others, the experiments were performed on small and simplistic models and convergence occurs on the order of hours [9]. Therefore, such approaches are inadequate for real-time optimization of repositioning strategies with the scale level and granularity that are required by full-scale ridesharing services.

One solution that is proposed in literature is to route idle vehicles towards areas where trip requests have been previously rejected while minimizing the repositioning distance [3]. Even though this method makes sure that the demand-supply mismatch will be balanced eventually, the downside is that the decision-making is highly reactive and does not take into account the future evolution of the demand and supply. In a follow-up paper [4], the authors present a more sophisticated approach which takes future demands into account. They sample the expected demand from a probability distribution and compute dispatching routes such that potential demand is maximized along the route. This essentially integrates the repositioning process with the vehicle routing. Although a decrease in waiting no significant increase of service rate was observed as compared to a reactive strategy. In addition, with a fleet size of 3000 vehicles the computation time increases to more than 60 seconds, which impacts the scalability of the proposed solution. The authors suggest "improving the rebalancing of vehicles" (i.e. as a separate process, not integrated with vehicle routing) in future work. In another paper, Jung and Chow [13] compare three different repositioning strategies: driving towards areas with high pickup probabilities, driving to depots and remaining at the last dropoff location (i.e. no repositioning). They simulate the different strategies using real-world taxi trip data from New York City. Both the first and second strategies improve the request acceptance rate in comparison to the scenario where no repositioning takes place. However, their approach does not consider the current vehicle state, nor the current or future demand on the road network. As a consequence, it lacks the ability to optimize repositioning routes in a proactive and vehicle-specific manner.

In [16], Pouls, Meyer and Ahuja propose an intelligent repositioning algorithm for dynamic ridesharing which they simulate on large real-world data. The algorithm is forecast-driven and is therefore able to make proactive decisions. The authors demonstrate that their approach is suitable for real-time usage even in large-scale scenarios. However, the authors use grid-based partitioning (i.e. of 1 km^2) as the environment for simulating vehicle movements as well as repositioning decisions. This method neglects a large amount of information inside the graph structure of the road network, and therefore does not yield fine-grained repositioning policies, nor accurate simulation of vehicle positions and routing. Furthermore, the algorithm needs several parameters which need to be estimated empirically, such as a discount for the demand coverage if a vehicle is already partially occupied. This is a potentially vulnerable aspect of the proposed methodology, since wrongly chosen parameters can yield inefficient repositioning and unrealistic simulation results. It suggests a gap for repositioning algorithms which can directly generate optimized policies based on predicted demand data and probabilistic models for the spatiotemporal evolution of demand.

3 METHODOLOGY

3.1 Overview

The methodology can be divided in five different components. First, we preprocess and clean the real-world trip datasets and map the trips to the vertices in the network under study. Subsequently, we algorithmically compute the flow (i.e. number of passengers) that travels on every edge for every time period. We build and train two deep learning models to predict the aforementioned edge flows as well as the demands on every vertex. Finally, the proposed repositioning algorithm DemandProp is described.

3.2 Matching Trips to Vertices

We start by defining the set R of trip requests. An individual trip request $r \in R$ contains multiple attributes: a pickup timestamp, pickup coordinates and dropoff coordinates. We also define the road network as a directed graph G, with a set V of vertices and a set E of edges. We simply match the trip's pickup coordinates with the nearest vertex based on the haversine distance $d(P_1, P_2)$, and do the same for the dropoff coordinates. Upon implementation, the performance of this algorithm is enhanced using vectorization. The proposed approach is shown in Appendix A.
The algorithm yields an origin-destination mapping (v_o, v_d) for all $r \in R$, with v_o being the pickup node and v_d being the dropoff node. With the resulting origin-destination mapping, we then perform the following tasks:

- We determine the travel times (i.e. the time difference in seconds between pickup and dropoff time) for all origin-destination pairs of vertices in the dataset. Accordingly, we train an XGBoost model which can predict the travel times between an arbitrary pair of vertices in G.
- We aggregate the demands for every vertex in intervals Δ_t of 15 minutes. This results in a *vertex demand* dataset which can be used to train the corresponding neural network.
- According to the methodology proposed in Section 3.3, we aggregate the flows on every edge in intervals Δ_t of 15 minutes. This results in a *edge flow* dataset which can be used to train the corresponding neural network. The model configuration and training processes are further described in Section 3.4.

3.3 Computing Aggregated Edge Flows

In this paper, an aggregation interval $\Delta_t = 15$ minutes is selected. We aim to compute the flows which traverse every edge $e \in E$ for each aggregation interval in the dataset. Based on the timestamps of the first and last request in the dataset, we know the number of intervals K. Since the dataset only contains the origin-destination mapping and not the intermediate vertices or edges that the vehicles traversed along, we need to estimate the path of the vehicles between the pickup and dropoff point. We do this by applying Dijkstra's shortest path algorithm, where the weights w_e for $e \in E$ are defined as the predicted travel times between the start and end vertices v_0 and v_1 of edge e. Then, for every request $r \in R$, we compute the path as a sequence of edges $(e_0, e_1, ..., e_n)$. This path can subsequently be traversed, keeping track of the travel time from

the initial time at which the request was made. Based on this intermediate travel time, the number of passengers for request r can be added to the flow matrix at that timestep and for the edge(s) which it passes along at that particular time. When the algorithm is finished, the flow matrix L will therefore contain, for every 15-minute timestep, the total number of passengers that traverses along all edges in the network.

3.4 Predicting Vertex Demand and Edge Flow

In order to enable proactive repositioning with DemandProp, the expected demands on the vertices and flows on the edges must be computed at every timestep. Therefore, two separate deep neural network configurations are proposed which have the ability to predict with a horizon of maximally one hour ahead.

Both the vertex demand model and the edge flow model accept the same inputs: the cyclic time variables (hour, minute, weekday, month, day) and a lookback window. The lookback window is defined as the mean observed demand at every vertex in the network during the past 60 minutes. The decision was made to use the same input space for both models, since both models are expected to benefit from the observations of demand and since it is computationally easy to keep track of the incoming trip requests. Also, with this technique input arrays have to be computed only once per iteration of the simulator, which facilitates faster overall prediction (and thus repositioning) times. Hence, the number of inputs to both models is $5 + |V|$. The vertex demand model predicts the demand at a 15-minute aggregation interval, and therefore generates $|V|$ outputs in total. The edge flow model predicts a value for every edge, and therefore has an output layer of size $|E|$. Both models predict the aggregated value (either demand or flow) for the 15 minutes immediately after the timestamp that was provided as input to the model. In order to find a balance between model complexity and the potential threat of underfitting, the vertex demand model was chosen to have two hidden layers of 3,600 neurons each, approximating the number of vertices of the road network under study. For the same reason, the edge flow model is equipped with two hidden layers of 3,600 and 6,000 neurons, respectively.

3.5 Repositioning of fleet

The vertex demand model and the edge flow model provide the fundaments of the DemandProp repositioning algorithm. To make predictions, the current time as well as the lookback window of observed demands serve as input variables. The initial prediction step yields two matrices containing vertex demands and edge flows for all prediction horizons. The first step is to convert the edge flows into a transition probability matrix P_{path}. Since we know the passenger flow for every edge in E, we can determine the probabilities that an edge is traversed from its source node. Hence, for every vertex the probabilities of its outgoing edges sum up to one. Subsequently, multiplying these probabilities with the predicted vertex demands yields a $|H|$x$|E|$ matrix of estimated edge demands, which we denote as D. The matrices P_{path} and D are further used during the repositioning process.

Instead of optimizing from the perspective of the vehicles, our proposed approach is edge-centric: the algorithm aims to satisfy the predicted demands at the edges in decreasing order. DemandProp essentially performs a small-scale simulation within discretized time intervals (i.e. horizons H with an interval of $\Delta_t = 15$ minutes), serving and propagating demands while keeping track of intermediate vehicle locations and building the repositioning paths along the way. At every iteration, it performs a backwards traversal from that edge through the graph to find the optimal vehicle(s) to serve the demand on the edge. To determine how much of the demand on the edge can be served by a vehicle, we first compute the expected load (i.e. number of passengers inside the vehicle) $\mathbb{E}[L]$ based on its current load, the transition probabilities P and predicted travel time. If we let X be a random variable denoting the travel time in minutes, then based on the TLC dataset we use a lognormal distribution to model the probability that a ride ends after a given time.

After updating the vehicle's capacity and position, and serving (a part of) the demand, the path from the vehicle's location to the edge is added to the planned repositioning route of that vehicle. The demand $\bar{d}$ that is served on an intermediate edge can be computed by keeping track of the backwards (i.e. from the source of the edge to the vehicle) transition probabilities P_{path} along the shortest path and the necessary capacity C of the vehicle at the end of the path.

The basic functionality of the algorithm is illustrated in Figure 2. Suppose that we have a small sample graph of seven vertices A-G. A fleet of four vehicles (each with a capacity of five seats) is operating in the environment. Two vehicles are initially located at vertex D (with 4 and 5 free seats, respectively), one at B (with 2 seats remaining) and another at E (with all 5 seats remaining). The edge demands (contained in D) are displayed as **boldfaced** for every edge in the graph. For the sake of simplicity, we let the travel time for every edge be 2 minutes. The algorithm will first sort the edges by their remaining predicted demand for the coming 15 minutes. In the first iteration of the algorithm, the edge from D to F (which we will call e_{ref}) has the highest remaining demand of 12. The algorithm first tries to satisfy this demand using the vehicles that are currently located at the source of the edge (i.e. vertex D). The red and green vehicle are located at D and have a remaining capacity of 4 and 5, respectively. Therefore, the remaining demand on e_{ref} can be computed as $12 - 4 - 5 = 3$ and the sequence from D to F is added to the repositioning paths of both vehicles. Both vehicles are fully utilized to serve the demand, so their remaining capacity will be 0. However, there is still an unserved demand of 3 remaining on e_{ref}. The DemandProp algorithm will traverse the graph using breadth-first search (BFS) until all demand is satisfied or until the accumulated travel time in all branches of the search has exceeded the 15-minute horizon. From e_{ref}, it searches the preceding edges (i.e. incoming edges at vertex D), which means that we first examine the edge from B to D. The blue vehicle, with currently 2 remaining seats, is located at the source of this edge. Hence, in step 2, another part of the demand at e_{ref} can be served. The probability that the current load can be retained at vertex D can be computed as $p_{load} \approx 0.99088$. The expected load at vertex D is consequently computed as $\mathbb{E}[L] = 2 \cdot 0.99088 = 2.97264$, and the expected number of free seats is $5 - 2.97264 = 2.02736$. We can therefore satisfy another part of the demand on e_{ref}, with $3 - 2.02736 = 0.97264$ remaining. Since all of the seats in the blue vehicle are now either reserved or used, no intermediate demands

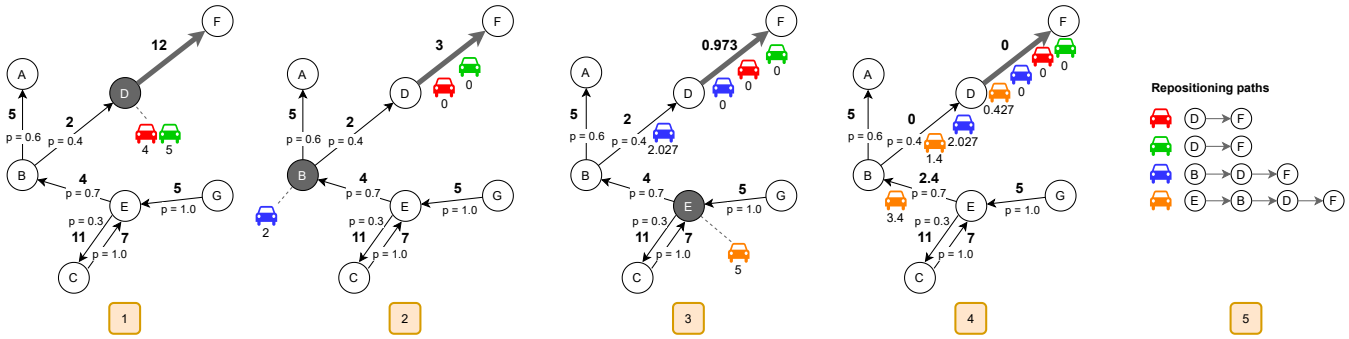


Figure 2: Illustrative example of generating repositioning paths using DemandProp

can be served. The sequence B-D-F is added to the repositioning path of the vehicle.

In step 3, we aim to satisfy the remaining demand of 0.973 at e_{ref}. The algorithm traverses further through the graph and finds the orange vehicle with 5 seats available. By repositioning this vehicle, all remaining demand on e_{ref} can be served while the vehicle still has $5 - 0.983 = 4.027$ remaining seats. Therefore the algorithm can use this vehicle to serve intermediate demands, based on the probability that these demands can be shared with the main demand from D to F. The intermediate demands are computed in backwards order, from D to E. Since there is a 1.0 probability p_{path} that the demand from B to D ends up on e_{ref}, the demand which can be served from B to D can be computed as $min(2 \cdot 1.0, 4.027) = 2.0$. Consequently, the demand which can be served from E to B can be computed as $min(4 \cdot 0.4, 2.027) = 1.6$. Overall, the remaining demand on the edge from E to B is therefore $4 - 1.6 = 2.4$. For all four vehicles, this yields the repositioning paths shown in 5.

4 EXPERIMENTAL SETUP

For the experiments, we determine New York City, USA to be the city under study to simulate the movements of vehicles in a ridesharing service. Since it is impossible to capture the movement of every customer on a microscopic scale, we combine knowledge from real-world trip requests, geographical data and travel time prediction to model the demand and determine the realistic paths that vehicles take inside the road network. Trip requests from the raw dataset are 'played back' in the simulator, and simulated vehicles can serve these trips based on their current state and planned path. Inside the simulation environment, multiple relevant metrics concerning resource and transport efficiency (such as vehicle occupancy, acceptance rate and customer delays) are measured at every timestep. These metrics can be evaluated and compared for different repositioning strategies in several scenarios. The baseline algorithms, experimental scenarios and performance metrics are introduced and explained in the next subsection.

4.1 Datasets and pre-processing

Our experiments are based on a real-world taxi trip request dataset which contains more than 100 million taxi trip requests from the city of New York City, USA. The data is released by the New York City Taxi and Limousine Commission (NYC TLC) on a monthly basis [22]. We decide to use 'traditional' taxi data instead of specific data about ridesharing for various reasons. First of all, the volume of data is much larger for regular taxi systems, which benefits the accuracy of our simulations and also facilitates assessment of the proposed approach and its scalability in a real-world scenario. Moreover, the high usage of TLC taxi data in related work means that results can be verified and compared with other approaches. Since our approach and simulation environment integrate shareability at runtime, we argue that data of regular taxi trips can be used to represent realistic demands. This also provides advantages for the majority of potential real-world implementations where trip data about ridesharing is not (yet) available. Since mid-2016, TLC uses area codes instead of coordinates to describe the origin and destination of trips. This makes it harder to accurately match trips to vertices in the graph. Hence, we use one year of trip data from July 2015 until July 2016, where the coordinates are still available. The relevant attributes that are available for every trip are: *pickup date/time*, *pickup coordinates*, *dropoff coordinates* and *number of passengers*.

The graph representing the road network was retrieved from OpenStreetMap using the OSMnx Python library [8]. Only the Manhattan area of New York City was selected in order to have a clear and well-defined study-area. Subsequently, irregularities and redundancies were treated: duplicate vertices and edges were consolidated and small isolated sections were removed. Dead-ends were removed from the graph in order to prevent vehicles in the simulator environment from getting stuck. The resulting directed graph contains 3,555 vertices and 8,535 edges.

Based on the road network that is represented by the graph, only the trips within Manhattan (i.e. both the pickup and dropoff coordinates located in Manhattan) were queried from the trip request dataset. Also, invalid trips are filtered from The trips are matched to the vertices of the graph in accordance with the methodology proposed in Section 3.2. From the resulting dataset, the edge flow model and vertex demand model were trained according to Sections 3.3 and 3.4.

4.2 Simulator

We have developed a simulator which can simulate passenger demand and the trajectories of a ridesharing fleet at intervals of 1 minute. All trip requests in the simulation are exactly reproduced according to the trip request dataset as described in Section 4.1.

Based on the planned paths (i.e. either when serving requests or repositioning), vehicles move from vertex to vertex inside the simulated road network. The path of a vehicle is defined as a queue containing a sequence of vertices, where the first vertex in the queue is removed once that vertex has been reached. Their speed of movement along the edges is determined by the predicted travel times, which are recomputed at an hourly interval. Vehicles are objects which individually keep track of the current vertex and the time that they have spent at that vertex - if the vehicle has been there longer than the travel time towards the next vertex in the queue, the vehicle will move to the next location and the first element will be removed from the queue. This process of moving vehicles through the network while taking expected travel times into account forms the fundamental core of the simulator. During a single iteration in the simulator, several actions are performed:

(1) **Request retrieval** Retrieve the requests that occur at the current timestep from the dataset, merge with requests that could not be served during the last five minutes.
(2) **Request handling** Handle the outstanding requests and compute the optimal vehicles to serve them, based on the number of vacant seats, distance from the new customer and the expected delay for existing customers. If there are no available vehicles within 2 km from the pickup location and the request has been pending for more than 5 minutes, it is rejected.
(3) **Vehicle dispatching** Dispatch the vehicles which have accepted a trip request in Step 2. Compute their updated path, number of remaining vacant seats and estimated travel time.
(4) **Vehicle repositioning** This is the primary action under study in this paper. Idle vehicles (i.e. with no occupied seats and no planned route) are determined and repositioned along a route which is computed by the repositioning algorithm (either DemandProp or one of the baseline algorithms listed in 4.3).
(5) **Moving vehicles** Based on the sum of time spent at the current vertex and the time advancement in the simulator, it is determined which vehicles should move to their next vertex. If a vehicle moves to a new vertex, we check if the vehicle has reached one of the following:
   (a) The *pickup vertex* of its customer, in which case the customer is picked up and the number of occupied seats is increased.
   (b) The *dropoff vertex* of its customer, in which case the customer is dropped off and the number of occupied seats is decreased. If the vehicle is now empty, it is marked as *idle*, and will be repositioned in the next iteration (unless it can serve a new request).
(6) **Saving metrics** The relevant metrics are stored in memory. All measurements are saved to disk periodically at an interval of 60 minutes (simulator time).

4.3 Baseline methods

The DemandProp algorithm proposed in this paper was compared to five other repositioning strategies. These algorithms are outlined in decreasing order of complexity:

- The **altruistic predicted demand (APD)** algorithm performs the repositioning in a centralized way by solving an assignment problem using a heuristic method. Using the vertex demand model, the demands for the upcoming 30 minutes are predicted by which the vertices are sorted in decreasing order. The predicted demands as well as the supply (i.e. number of seats available in idle vehicles) are then normalized such that demand can be fully covered by supply, regardless of the current state of the system. Moving down the list of sorted vertices, the algorithm attempts to satisfy the demands by allocating idle vehicles to that vertex until the demand has been satisfied, based on the greedy approximation algorithm proposed by Dantzig [10] for solving the unbounded knapsack problem. To determine which vehicle(s) should serve the demand at a particular vertex, a heuristic is used. With C_n being the number of remaining seats of vehicle n, X_n, Y_n being its current position, $d(P_1, P_2)$ being the haversine distance in meters between two coordinate pairs and v being the vertex considered, the heuristic is defined as:

$$h_v(n) = \frac{C_n}{d((X_n, Y_n), v) + 1}$$

  The algorithm keeps track of the residue, such that no demand is left unsatisfied. Finally, for every repositioned vehicle the path towards the vertex is computed using Dijkstra's algorithm (the predicted travel times being the weights).
- The **altruistic observed demand (AOD)** algorithm is nearly identical to the APD algorithm, except that a lookback window is used instead of predicted demands. A buffer is maintained which contains the number of passengers that were made in the last hour. As a consequence, the algorithm is reactive instead of proactive.
- The **egoistic predicted demand (EPD)** algorithm bears the closest resemblance to the traditional repositioning strategies in regular taxi systems, where drivers move to areas where high demands are expected in the near future. Therefore, unlike the APD and AOD algorithms, this is a decentralized algorithm where decisions are not harmonized between vehicles. A heuristic is used to reposition a vehicle to the most attractive. With D_v being the demand at vertex v, X_n, Y_n being the current position of vehicle n, $d(P_1, P_2)$ being the haversine distance in meters between two coordinate pairs and v being the vertex considered, the heuristic represents a tradeoff between the demand and the distance to the vertex:

$$h_n(v) = \frac{D_v}{d((X_n, Y_n), v) + 1}$$

  Hence, unlike the APD and AOD algorithms, we look from the 'egoistic' perspective of the drivers instead of aiming to fill the demands from the network's perspective.
- The **random repositioning** strategy routes idle vehicles to randomly selected vertices. Therefore, this repositioning algorithm contains no intelligence and takes neither the objectives of individual drivers nor system-wide benefits into consideration.

- In the **no repositioning** strategy, no action is taken when a vehicle becomes idle. Vehicles will therefore remain curbside at the same location until a new request can be served.

4.4 Experimental design

Using simulator runs, we aim to evaluate the effectiveness of the DemandProp algorithm in comparison to other baseline methods. For our experiments, we selected two periods of one week: Monday November 2 until Sunday November 8, 2015 as well as April 11 until April 17, 2016. These periods were selected based on the criteria that no public holidays, events or special weather conditions occurred. Therefore, the simulations are performed based on normal taxi movements within the city of New York. The vehicles inside the ridesharing services are initialized with a fixed seating capacity. This capacity is sampled from a probability distribution which approximates the real-world taxi seating capacity in the New York City taxi system [1]: with a 0.9 probability the taxi has a capacity of 4, and with 0.1 probability the taxi has a capacity of 5. We want to evaluate the performance of DemandProp in two different aspects of efficiency. First of all, we consider *transport efficiency*, which concerns the delays and comfort experienced by customers of the ridesharing service. Furthermore, we assess *resource efficiency*, which involves the cost implications for the operator as well as the environmental impact on society. Particularly, we want to discover to what extent DemandProp can reduce fleet size while maintaining transport efficiency at satisfactory levels.

4.4.1 Transport efficiency. At each iteration of the simulation (i.e. at a minute resolution), the following metrics which concern transport efficiency are recorded:

- The average **waiting time** experienced by customers. This is the time between the moment that the request was made until the customer is picked up.
- The average **detour time** experienced by customers. This is the delay that occurs during the trip (i.e. between pickup and dropoff) caused by serving other customers along the way.
- The **on-time performance**, which is defined as the percentage of requests which is served with less than 5 minutes of delay. The delay is composed of the *extra waiting time* (i.e. when the customer is picked up later than initially planned due to the pickup of a new customer) and the *detour time*.
- The average **number of customers per vehicle**, which can be regarded as an indicator of the comfort level experienced by customers. It is preferred when customers spread out more efficiently across the fleet.

4.4.2 Resource efficiency. The following metrics regarding resource efficiency are recorded at each iteration of the simulation:

- The **reject rate** of requests, which is defined as the percentage of trip requests that must be rejected by a vehicle. It therefore provides a measure for the availability of vehicles and the demand-supply mismatch.
- The **empty vehicle rate**, which is defined as the percentage of vehicles that is currently carrying no passengers. It provides a measure of how efficiently the space in the vehicles is utilized.
- The average daily **idle time** of vehicles. This is the time per day that a vehicle spends while carrying no passenger. From the operator's perspective, this is the time in which no revenue is being generated and therefore should be minimized.
- The average daily **distance covered** of vehicles, i.e. the number of kilometers that a vehicle drives on an average day.

4.4.3 Experiments. In the principal experiment regarding transport and resource efficiency, we use a fixed number of vehicles $N = 5000$. For every repositioning strategy (i.e. DemandProp and the baseline methods), we then simulate the previously mentioned two weeks of taxi trip requests and store the metrics after every simulated minute. The results are subsequently aggregated, both for an entire week as well as every weekday, hence facilitating the comparison of repositioning strategies during different parts of the week.

To assess the possible fleet size reduction that can be achieved by implementing DemandProp, we run the previous experiments again, but for different values of the fleet size N. Because of time and resource constraints, we choose N to range from 4,000 to 6,000 with intervals of 500. We determine a minimum on-time performance of 80% and a maximum reject rate of 1.0% as reference values. We then determine to what extent the fleet size can be reduced while still adhering to these reference values.

5 RESULTS AND DISCUSSION

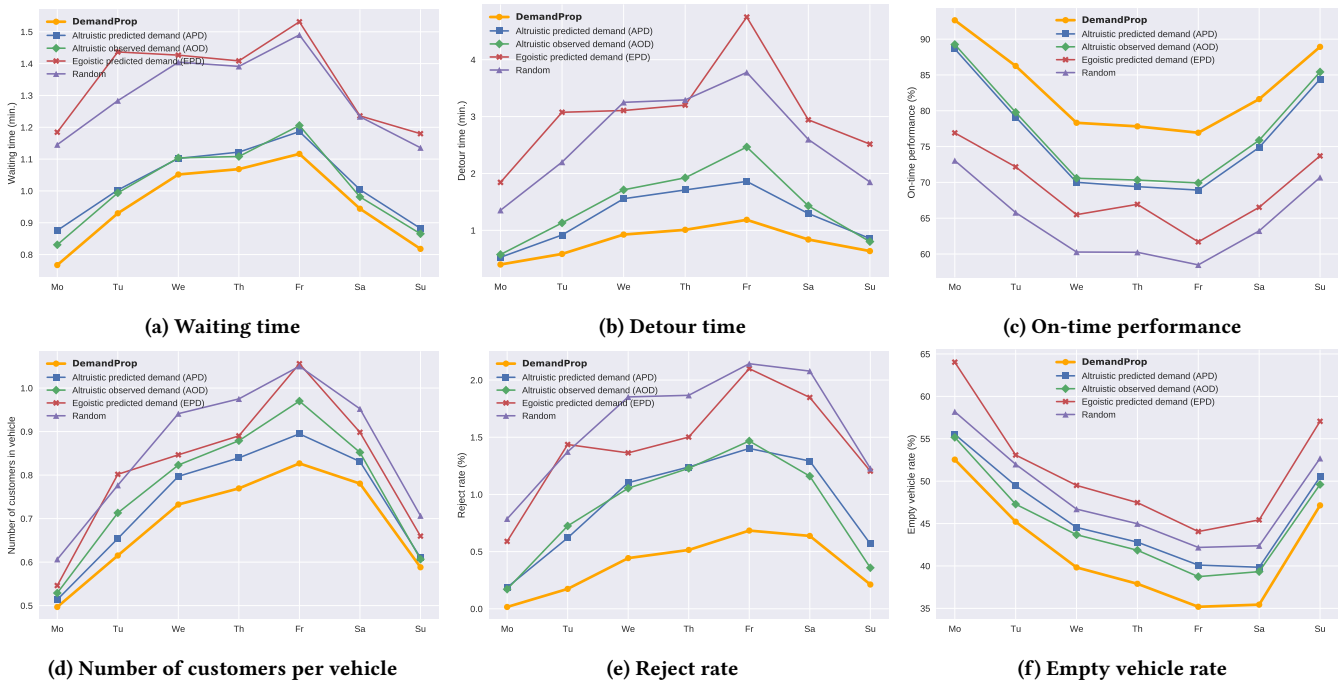
5.1 Model training and validation

We split both the vertex demand and edge flow datasets into 80% training set and a 20% test set. 10% of the training set is used as a validation set during the training process. Both the vertex demand model and the edge flow model are trained for 2,000 epochs using the Adam optimizer with a learning rate $\alpha = 5 \cdot 10^{-6}$ and using the mean squared error (MSE) as the loss function. On the test dataset, both models perform well, with the vertex demand model achieving an R^2 score of 0.80 and the edge flow model achieving an R^2 score of 0.91. The training results are shown in Table 4.

5.2 Simulation runs

First, we look at the results of the two simulator runs of the weeks November 2-8 and April 11-17, respectively. These simulator runs were executed with fleet size $N = 5000$. Figure 3 shows the aggregated metrics per weekday, while Table 1 provides the conclusive metrics for the complete simulator runs. The table also contains the increase and decrease percentages, which indicate the gains or losses that DemandProp yields in comparison to the baseline algorithms proposed in Section 4.3.

From the results in Figure 3 and 1, it emerges that the DemandProp algorithm outperforms the baseline methods concerning transport efficiency for every day of the week. With regard to the total delay, DemandProp yields a 45.21% advantage over the most intelligent baseline method (i.e. APD) and a 1501% advantage against the strategy where vehicles are not repositioned at all. The egoistic predicted demand (EPD) repositioning strategy, which is most prevalent in traditional taxi systems, produces 224.9% higher delays than DemandProp. Most of the difference in delay can be explained by the increase in detour time, which is 47.79 seconds on average for DemandProp and 74.73 seconds for the APD repositioning strategy



(a) Waiting time **(b) Detour time** **(c) On-time performance**

(d) Number of customers per vehicle **(e) Reject rate** **(f) Empty vehicle rate**

Figure 3: Performance of the repositioning strategies for N = 5000, aggregated by weekday

Repositioning method	Delay (min.)	On-time performance (%)	Customers per vehicle	Idle time	Distance covered (km)
DemandProp	**1.077**	**83.22**	**0.687**	**12 h. 34 min.**	**351.5**
(B) Altruistic predicted demand	1.564 (+45.21%)	76.49 (-6.730%)	0.734 (+6.841%)	13 h. 19 min. (+45 min.)	315.9 (-10.12%)
(B) Altruistic observed demand	1.755 (+62.95%)	77.82 (-5.400%)	0.768 (+11.79%)	13 h. 48 min. (+74 min.)	320.3 (-8.876%)
(B) Egoistic predicted demand	3.499 (+224.9%)	68.14 (-15.08%)	0.814 (+18.49%)	13 h. 34 min. (+60 min.)	192.9 (-45.12%)
(B) Random	3.033 (+181.6%)	64.54 (-18.68%)	0.858 (+24.89%)	14 h. 5 min. (+91 min.)	387.2 (+10.16%)
(B) No repositioning	17.25 (+1501%)	50.60 (-32.62%)	1.343 (+95.49%)	13 h. 49 min. (+75 min.)	169.1 (-51.89%)

Table 1: Performance of the repositioning strategies for N = 5000

	Vertex demand model	Edge load model
R^2	0.8023	0.9111
MSE	0.1884	0.0639
MAE	0.2993	0.1663

Figure 4: Performance of prediction models on their respective test sets

- a 56.37% increase. This is also shown in the on-time performance: with DemandProp, 83.22% of all simulated requests is served with less than 5 minutes of delay, which is an increase of more than 5% in comparison to the altruistic repositioning strategies and more than 15% for the other baseline methods. Based on the number of customers per vehicle, the DemandProp repositioning strategy also yields a better spread of customers across the fleet and therefore ensures a better comfort level for customers. When DemandProp is used, 0.687 customers are on average carried by a single vehicle, compared to the APD and AOD strategies where 6.841% and 11.79% more customers are carried, respectively. When no repositioning is performed, the number of carried customers is almost doubled. This can be explained by a demand-supply mismatch (caused by the fact that vehicles are not repositioned to high-demand areas) which forces the vehicles in high-demand areas to pick up more passengers to satisfy all remaining demand. Another explanation could be that system-wide delays become higher (as visible in 1), causing vehicles to carry more passengers simultaneously. Altogether, our simulations suggest that the DemandProp algorithm outperforms the baseline methods by a considerable margin with regard to transport efficiency (in terms of delays and comfort experienced by customers of the ridesharing service).

When we consider resource efficiency, DemandProp outperforms the baseline methods for three of the four metrics that were evaluated. The reject rate (i.e. the percentage of requests that could not be served by any vehicle) across the complete two-week simulation period was just 0.38% for the DemandProp algorithm, while the

baseline methods result in at least a doubling of the number of request rejections: more specifically, the best-performing baseline method (APD) yields a reject rate of 0.92% while no repositioning results in the worst reject rate of 5.13%. Similarly, DemandProp demonstrates a lower empty vehicle rate than any of the baseline methods throughout the simulator runs. When DemandProp is used, on average 41.89% of the vehicles is carrying no passengers. This percentage increases by more than 10% for the egoistic repositioning method (EPD), which yields an empty vehicle rate of 52.80%. A similar phenomenon can be observed with regard to the idle time. If DemandProp repositioning is used, a vehicle spends on average 12 hours and 34 minutes per day without serving customers and therefore not generating revenue. The baseline methods produce at least 45 minutes and at most 91 minutes more idle time per day. Ideally, we would like to decrease the idle time further, in pursuance of higher resource-efficiency. However, idle times are largely determined by the trade-off between fleet size and the magnitude of demand throughout the network. Possible fleet size reductions will be evaluated in more detail later in this section. In a real-world situation the fleet size will always be dynamic, and less vehicles will be operated at night which automatically leads to less idle time for the entire fleet.

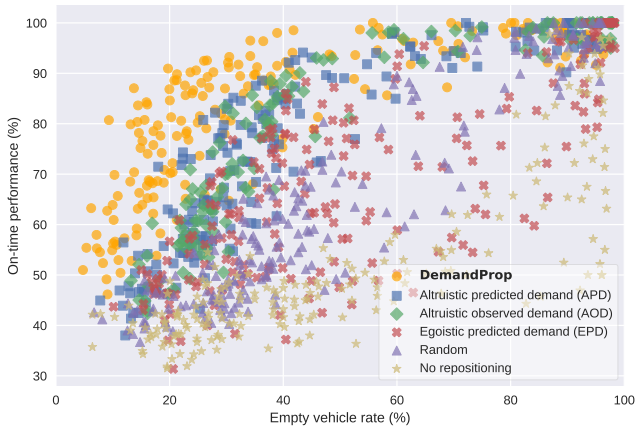
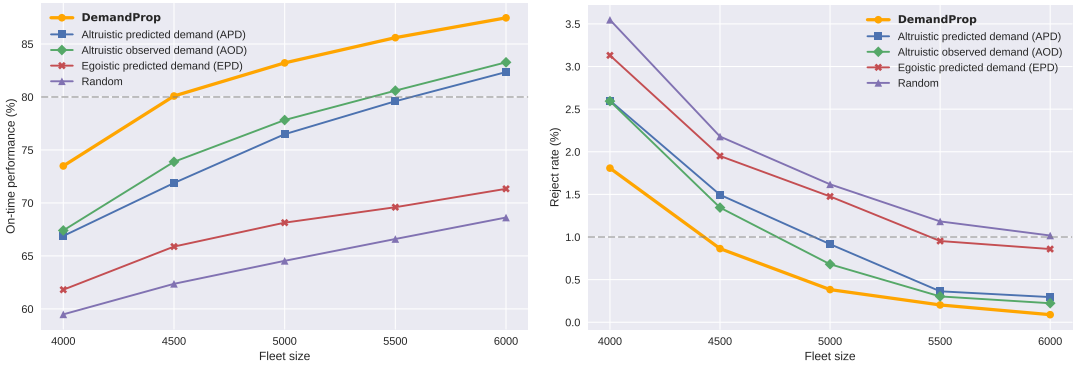


Figure 5: On-time performance of repositioning algorithms for increasing levels of vehicle utilization

In Figure 5, we can observe the performance and resilience of the repositioning strategies in situations where high demand levels occur. Hence, we plot the empty vehicle rate against the on-time performance. We evaluate the ability of the repositioning strategy to maintain high on-time performance even when most vehicles are currently busy serving passengers. We argue that this approach can demonstrate whether the repositioning algorithm has the capability to proactively and altruistically route the vehicles along the paths where many shareable trips are expected, with minimal overhead as a consequence. From the graph it is evident that DemandProp excels in situations where the empty vehicle rate is low (i.e. between 0% and 40%). As a matter of fact, the DemandProp repositioning is the only evaluated method which yields an on-time performance over 80% when only 10% of vehicles is empty (i.e. when 90% of the vehicles is serving passengers). By comparison to the baseline methods, the simulated results therefore suggest that using DemandProp to reposition the fleet allows the vehicles to serve demand with a relatively small delay, even when there is little capacity remaining in the fleet.

The only metric on which the DemandProp performs worse than the baseline algorithms is the average daily distance covered. When DemandProp is used to reposition the vehicles, a vehicle travels a daily average of 351.5 kilometers. With the exception of the random repositioning method, this is significantly more than the driving distance when the baseline methods are applied. For instance, under the egoistic repositioning method (EPD) the vehicles traverse a daily average of 192.9 kilometers, which is a decrease of 45.12% in comparison to DemandProp. For the more advanced baseline methods APD and AOD, the differences are smaller (10.12% and 8.876%, respectively) but still substantial. A likely explanation for these differences is that the DemandProp is able to evaluate and act upon demand in a more fine-grained manner than the baseline methods, and is therefore generating more advanced repositioning paths for the vehicles. By proposing paths with a higher distance, more (expected) demand can be covered by the available fleet. This also explains why egoistic repositioning algorithm (EPD) results in shorter repositioning paths: since the algorithm does not altruistically coordinate the demand-supply mismatch between different vehicles, vehicles will often be repositioned towards the same location. This makes the repositioning parts simple and relatively short, but also results in higher residual demand throughout the network. The extra driving distance causes higher overall fuel consumption, and we could therefore argue that DemandProp is less resource-efficient in this aspect than most of the baseline methods. However, the results regarding reject rate, idle time and vehicle utilization suggest that the benefits outweigh the potential overhead.

In another experiment, we assess the performance from the perspective of reducing the fleet size. By running the simulation with different fleet sizes N and recording the on-time performance and reject rate, insights can be gained into the reductions that can be made with DemandProp while still being able satisfy operational criteria. In Figure 6, the results of this experiment can be observed. Again, the *no repositioning* baseline is omitted from the graphs to ensure interpretability and emphasize on the other baseline methods. From the graphs it becomes clear that there exists a non-linear relationship between fleet size and the metrics under study: for both on-time performance and reject rate, the performance improves at a smaller rate as the fleet size grows. From the operator's perspective, a trade-off should therefore be sought between the cost of resources and the transport efficiency. The results show that DemandProp outperforms the baseline methods for every fleet size, both in on-time performance as well as reject rate. When DemandProp repositioning is used, the previously mentioned criterium of 80% on-time performance is reached for a fleet-size of 4,500. For the best-performing baseline methods APD and AOD, this level is reached at a fleet size of approximately 5,500. Hence, when compared to the most intelligent alternatives, DemandProp allows for a fleet reduction of nearly 1,000 vehicles while maintaining the same on-time performance. Looking at the reject rate, DemandProp reaches the criterium of 1.0% at a fleet size of 4,500, while the APD and AOD baselines reach that at a fleet size of 5,000. Therefore, in



(a) On-time performance **(b) Reject rate**

Figure 6: Performance under fleet size reduction (criteria marked using gray dashed line)

comparison to these baselines, DemandProp allows for a fleet reduction of nearly 500 vehicles while maintaining the same reject rate. With some reservations, it is also possible to compare DemandProp to other approaches proposed in related work. For instance, Vazifeh et al. [23] demonstrate that a fleet size of 6,000 yields an on-time performance of 85%, even though the authors define the on-time performance using a maximum delay of six minutes instead of five. Therefore, according to our definition, the on-time performance of their proposed method would almost certainly be lower than 85%. Hence, we could argue that DemandProp performs better, serving 87.46% of the requests with a delay shorter than five minutes. However, due to possible differences in the simulator environment, vehicle dispatching and the parameters used, it is impracticable to justify this claim with confidence.

6 CONCLUSIONS AND FUTURE WORK

In this work, we have presented a predictive repositioning algorithm for dynamic ridesharing services. This algorithm, which we call DemandProp, relies on deep neural networks (DNNs) and a stochastic model to optimally (re)position the fleet, with the aim of maximizing cumulative expected demand served by all vehicles while minimizing the distance travelled to serve those demands. Our results show that our approach is scalable and is capable of running in real-time on large-scale networks of more than 5,000 vehicles and 9,000 roads, representing the complete road network of Manhattan, New York City. Our results on a real-world trip request dataset demonstrate that DemandProp is able to deliver fine-grained and precise decision-making at real-world scale level of a ridesharing service. DemandProp outperforms all baseline methods by a significant margin regarding transport and resource efficiency. Compared to an egoistic repositioning strategy, which is prevalent in traditional taxi systems, the on-time performance can be increased by 15% and total customer delays can be reduced by 69%. The idle time of vehicles is reduced by at least 45 minutes per day in comparison to baseline methods. Our results also demonstrate that DemandProp is able to deliver high on-time performance (>80%) even when there is little capacity remaining in the fleet (i.e. less than 20% vehicles without passengers), in contrast to other baseline methods. Furthermore, a performance evaluation under different fleet sizes suggests that DemandProp enables operators to reduce the fleet by more than 1,000 vehicles in order to reach the same on-time performance as the most intelligent baseline method (i.e. the altruistic and predictive one). Compared to the egoistic repositioning strategy, the results even suggest that the fleet can be reduced by more than 2,000 vehicles to maintain the same level of on-time performance. When looking at the reject rate, the possible fleet size reductions are approximately 500 vehicles in order to maintain the performance of the altruistic predictive baseline strategy. It is therefore evident that DemandProp can provide considerable benefits to the performance of a ridesharing service, both from the operator's and customer's perspective.

In the future, we aim to study how the DemandProp algorithm reacts to days with special demand patterns, such as holidays or big events. Since the algorithm is driven by neural networks that predict demand, it is likely that demand predictions will become less accurate on such occasions. Hence, we aim to assess whether DemandProp is able to handle such demand patterns better than baseline models. Additionally, we aim to study if DemandProp is able to handle a higher diversity of fleet size and seating capacities. We are especially interested in the implications of large differences in the number of seats (i.e. emphasizing on mixed-size fleets), and whether DemandProp is able to structurally outperform the baseline models in such scenarios. Lastly, we are interested in incorporating other constraints and objectives into our models, such as EV charging and integration of depots and time windows, such that DemandProp is able to optimize under a wide range of different scenarios of which the importance is determined by the stakeholder.

REFERENCES

[1] About.com. 2007. New York Taxis – Getting Around New York City in a Taxi. https://www.tripsavvy.com/new-york-city-taxis-4026457
[2] Niels A. H. Agatz, A. Erera, M. Savelsbergh, and Xing Wang. 2010. Sustainable Passenger Transportation: Dynamic Ride-Sharing. *Urban Economics & Regional Studies eJournal* (2010).
[3] Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. 2017. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences* 114, 3 (2017), 462–467. https://doi.org/10.1073/pnas.1611675114 arXiv:https://www.pnas.org/content/114/3/462.full.pdf
[4] J. Alonso-Mora, A. Wallar, and D. Rus. 2017. Predictive routing for autonomous mobility-on-demand systems with ride-sharing. In *2017 IEEE/RSJ International*

Conference on Intelligent Robots and Systems (IROS). 3583–3590. https://doi.org/10.1109/IROS.2017.8206203

[5] Andrew Amey, John Attanucci, and Rabi Mishalani. 2011. Real-Time Ridesharing: Opportunities and Challenges in Using Mobile Phone Technology to Improve Rideshare Services. *Transportation Research Record* 2217, 1 (2011), 103–110. https://doi.org/10.3141/2217-13 arXiv:https://doi.org/10.3141/2217-13

[6] Siddhartha Banerjee, Ramesh Johari, and Carlos Riquelme. 2016. Dynamic Pricing in Ridesharing Platforms. *SIGecom Exch.* 15, 1 (Sept. 2016), 65–70. https://doi.org/10.1145/2994501.2994505

[7] Xiaohui Bei and Shengyu Zhang. 2018. Algorithms for Trip-Vehicle Assignment in Ride-Sharing. In *AAAI*.

[8] Geoff Boeing. 2017. OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems* 65 (2017), 126 – 139. https://doi.org/10.1016/j.compenvurbsys.2017.05.004

[9] Anton Braverman, J. G. Dai, Xin Liu, and Lei Ying. 2019. Empty-Car Routing in Ridesharing Systems. *Oper. Res.* 67, 5 (Sept. 2019), 1437–1452. https://doi.org/10.1287/opre.2018.1822

[10] George B. Dantzig. 1957. Discrete-Variable Extremum Problems. *Operations Research* 5, 2 (1957), 266–288. https://doi.org/10.1287/opre.5.2.266 arXiv:https://doi.org/10.1287/opre.5.2.266

[11] Mauro Dell'Amico, Eleni Hadjicostantinou, Manuel Iori, and Stefano Novellani. 2014. The bike sharing rebalancing problem: Mathematical formulations and benchmark instances. *Omega* 45 (2014), 7 – 19. https://doi.org/10.1016/j.omega.2013.12.001

[12] Nandani Garg and Sayan Ranu. 2018. Route Recommendations for Idle Taxi Drivers: Find Me the Shortest Route to a Customer!. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '18)*. Association for Computing Machinery, New York, NY, USA, 1425–1434. https://doi.org/10.1145/3219819.3220055

[13] Jae Young Jung and Joseph Chow. 2019. Large-Scale Simulation-Based Evaluation of Fleet Repositioning Strategies for Dynamic Rideshare in New York City. In *WCX SAE World Congress Experience*. SAE International. https://doi.org/10.4271/2019-01-0924

[14] B. Li, D. Zhang, L. Sun, C. Chen, S. Li, G. Qi, and Q. Yang. 2011. Hunting or waiting? Discovering passenger-finding strategies from a large-scale real-world taxi dataset. In *2011 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*. 63–68. https://doi.org/10.1109/PERCOMW.2011.5766967

[15] F. Miao, S. Han, S. Lin, J. A. Stankovic, D. Zhang, S. Munir, H. Huang, T. He, and G. J. Pappas. 2016. Taxi Dispatch With Real-Time Sensing Data in Metropolitan Areas: A Receding Horizon Control Approach. *IEEE Transactions on Automation Science and Engineering* 13, 2 (2016), 463–478. https://doi.org/10.1109/TASE.2016.2529580

[16] Martin Pouls, Anne Meyer, and Nitin Ahuja. 2020. Idle Vehicle Repositioning for Dynamic Ride-Sharing. In *ICCL*.

[17] Jason W. Powell, Yan Huang, Favyen Bastani, and Minhe Ji. 2011. Towards Reducing Taxicab Cruising Time Using Spatio-Temporal Profitability Maps. In *Advances in Spatial and Temporal Databases*, Dieter Pfoser, Yufei Tao, Kyriakos Mouratidis, Mario A. Nascimento, Mohamed Mokbel, Shashi Shekhar, and Yan Huang (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 242–260.

[18] Xinwu Qian, Wenbo Zhang, Satish V. Ukkusuri, and Chao Yang. 2017. Optimal assignment and incentive design in the taxi group ride problem. *Transportation Research Part B: Methodological* 103, C (2017), 208–226. https://doi.org/10.1016/j.trb.2017.03.001

[19] Paolo Santi, Giovanni Resta, Michael Szell, Stanislav Sobolevsky, Steven H. Strogatz, and Carlo Ratti. 2014. Quantifying the benefits of vehicle pooling with shareability networks. *Proceedings of the National Academy of Sciences* 111, 37 (2014), 13290–13294. https://doi.org/10.1073/pnas.1403657111 arXiv:https://www.pnas.org/content/111/37/13290.full.pdf

[20] Hamid R. Sayarshad and Joseph Y.J. Chow. 2017. Non-myopic relocation of idle mobility-on-demand vehicles as a dynamic location-allocation-queueing problem. *Transportation Research, Part E: Logistics and Transportation Review* 106 (Oct. 2017), 60–77. https://doi.org/10.1016/j.tre.2017.08.003

[21] Andrea Simonetto, Julien Monteil, and Claudio Gambella. 2019. Real-time city-scale ridesharing via linear assignment problems. *Transportation Research Part C Emerging Technologies* 101 (02 2019), 208–232. https://doi.org/10.1016/j.trc.2019.01.019

[22] New York City Taxi and Limousine Commission. 2016. Trip Record Data. https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

[23] M. Vazifeh, P. Santi, G. Resta, S. Strogatz, and C. Ratti. 2018. Addressing the minimum fleet problem in on-demand urban mobility. *Nature* 557 (05 2018). https://doi.org/10.1038/s41586-018-0095-1

[24] Tai yu Ma, Joseph Y.J. Chow, and Saeid Rasulkhani. 2018. An integrated dynamic ridesharing dispatch and idle vehicle repositioning strategy on a bimodal transport network. In *Proceedings of 7th Transport Research Arena TRA 2018, April 16-19, 2018, Vienna, Austria*. https://doi.org/10.5281/zenodo.2155709

A TRIP MATCHING ALGORITHM

Algorithm 1: Trip matching

Input: Trip request dataset R, set of vertices V from graph G
Output: Origin-destination mapping (v_o, v_d) for all $r \in R$
Initialize trip requests R from dataset;
for $r \in R$ **do**
  Get pickup coordinates (x_o, y_o);
  Get dropoff coordinates (x_d, y_d);
  Set $v_o = \text{argmin}_{v \in V}\, d((x_o, y_o), v)$;
  Set $v_d = \text{argmin}_{v \in V}\, d((x_d, y_d), v)$;
  Store origin-destination mapping (v_o, v_d);
end