Putting Ridesharing to the Test: Efficient and Scalable Solutions and the Power of Dynamic Vehicle Relocation

PANAYIOTIS DANASSIS, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

MARIJA SAKOTA, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

ARIS FILOS-RATSIKAS, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland and University of Liverpool, United Kingdom

BOI FALTINGS, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

We perform a *systematic* evaluation of a diverse set of algorithms for the ridesharing problem which is, to the best of our knowledge, one of the *largest* and most *comprehensive* to date. In particular, we evaluate 12 different algorithms over 12 metrics related to *global efficiency*, *complexity*, *passenger*, *driver*, and *platform* incentives. Our evaluation setting is specifically designed to resemble reality as closely as possible. We achieve this by (a) using actual data from the NYC's yellow taxi trip records, both for modeling customer requests, and taxis (b) following closely the pricing model employed by ridesharing platforms and (c) running our simulations to the scale of the actual problem faced by the ridesharing platforms.

Our results provide a clear-cut recommendation to ridesharing platforms on which solutions can be employed in practice, and demonstrate the large potential for efficiency gains. Moreover, we show that simple, lightweight *relocation schemes* – which can be used as independent components to any ridesharing algorithm – can significantly improve Quality of Service metrics by up to 50%. As a highlight of our findings, we identify a *scalable*, *on-device* heuristic that offers an efficient, end-to-end solution for the Dynamic Ridesharing and Fleet Relocation problem.

1 INTRODUCTION

The emergence and widespread use of *ridesharing* in recent years has had a profound impact on urban transportation in a variety of ways. Amongst others, it has mitigated congestion costs (such as commute times, fuel usage, accident propensity, etc.), it has enabled marketplace optimization for both passengers and drivers, and it has provided great environmental benefits. Ridesharing however results in some passenger disruption as well, due to compromise in flexibility, increased travel time, and loss of privacy and convenience. Thus, in the core of any ridesharing platform lies the need for an efficient balance between the incentives of the passengers, the drivers, and those of the platform.

Optimizing the usage of transportation resources is not an easy task, especially for cities like New York, with more than 13000 taxis and 270 ride requests per minute. For example, [20] estimates that 45000 customer requests remain unmet each day in New York, despite the fact that approximately 5000 taxis are vacant at any time. In fact, on aggregate, drivers spend about 47% of their time not serving any passengers. Moreover, up to 80% of the taxi rides in Manhattan could be shared by two riders, with only a few minutes increase in travel time [3]. A *more sophisticated matching policy* could mitigate these costs by better allocating available supply to demand. As a second example, *coordinated vehicle relocation* could also be employed to bridge the gap on the spatial supply/demand imbalance and improve passenger satisfaction and Quality of Service (QoS) metrics. Drivers often relocate to find passengers: 61.3% of trips begin in a different neighborhood than the drop-off location of the last passenger [20], yet currently drivers move without any coordinated search behavior, resulting in spatial search frictions.

Given the importance of the ridesharing problem for transportation and the economy, it is not surprising that the related literature is populated with a plethora of papers, proposing different solutions along different axes, such as efficiency [2–4, 15, 30, 44, 60], platform revenue [6, 21], driver incentives [51, 71] or fairness [49, 66]. Most of the related work can be broadly categorized as either

(a) *empirical papers*, that propose heuristics tailored for the ridesharing problem and evaluate their performance on experimental scenarios or (b) *theoretical papers*, that design algorithms for more abstract versions of the problem (e.g., matching under uncertainty, in the presence of deadlines, etc.) and provide worst-case guarantees for their performance.

Despite the seeming heterogeneity of the solutions that have been proposed in this vast literature, the algorithmic primitives of the ridesharing problem can be pinpointed to the following two. First, we need to use efficient algorithms to *match* passengers with other passengers, or passengers with vehicles, and secondly, we need to *account for the future* when choosing which requests to service, in order to achieve a better placement of vehicles for potential future requests. The former challenge is more broadly studied by the extensive literature of *matching* algorithms, whereas the latter is analogous to the central objective of the classical k-*taxi* or k-*server problem*, for which several solutions have been proposed. Additionally, if one has access to some distributional knowledge of the domain, 'accounting for the future' could also be alternatively interpreted as a matching setting, where idle vehicles are 'provisionally matched' with requests that are *expected to appear* in the near future.

In this context, we are interested in the following fundamental questions:

> *How can we adapt and combine the classical algorithms for these fundamental settings, to apply them to the ridesharing problem? Which of these algorithms work well in practice, in a realistic ridesharing scenario, for a host of different objectives? Can we leverage historical data for dynamic vehicle relocation to close the gap on the spatial supply/demand imbalance?*

In order to properly answer these questions, we strongly believe that there is a need for a thorough empirical evaluation of different approaches in a quite realistic setting, and this is what we do in the present paper.

1.1 Our Contributions

(1) We perform a *comprehensive,* and systematic evaluation of a diverse set of algorithms for the ridesharing problem. We have evaluated **12** different algorithms over **12** metrics. We put extra emphasis on designing an evaluation setting which resembles reality as closely as possible, in every aspect of the problem. Our list of metrics includes amongst others the total distance saved as a result of ridesharing, the pick-up times and delay incurred by the passengers, the profit and search frictions of drivers, and the platform revenue. To the best of our knowledge, this is the first end-to-end experimental evaluation of this magnitude.

(2) We examine the extent to which *relocation* of idle taxis can improve QoS objectives, by closing the gap on spatial supply/demand imbalance. We propose relocation schemes which are based on matching algorithms and make use of the historical data to predict future requests. Our results here are irrefutable: Relocation schemes based on lightweight matching algorithms improve several QoS metrics radically (exceeding 50% in certain cases), suggesting that relocation should be a vital part of any efficient ridesharing algorithm.

(3) As a highlight of our results, we identify a *scalable, on-device* heuristic (the ALMA algorithm of [28]) that offers an efficient, end-to-end solution for the Dynamic Ridesharing and Fleet Relocation problem.

We firmly believe that our findings provide a clear-cut recommendation to ridesharing platforms on which solutions they should employ in practice.

1.2 Discussion and Related Work

The dynamic ridesharing – and the closely related *dynamic dial-a-ride* (see [1]) – problem has drawn the attention of diverse disciplines over the past few years, from operations research to transportation engineering, and computer science. Solution approaches include constrained optimization [2, 3, 56, 64], weighted matching [4, 11, 28, 30, 72], other heuristics [10, 50, 56, 61, 62], reinforcement learning [40], or model predictive control [22], among others. We refer the interested reader to the following surveys [1, 27, 38, 42, 55, 63] for a review on the optimization challenges, various algorithmic designs adopted over the years, a classification of existing ridesharing systems, models and algorithms for shared mobility, and finally models and solution methodologies for the dial-a-ride problem, respectively.

As we mentioned in the introduction, the key algorithmic components of ridesharing are the following. First, it is an *online* problem, as the decisions made at some point in time clearly affect the possible decisions in the future, and therefore the general approach of the field of online algorithms and competitive analysis is applicable [17, 52]. Secondly, it is clearly a *matching* setting, both for bipartite graphs (for matching passengers with taxis) and for general graphs (for matching passengers to shared rides). In fact, several of the algorithms that have been proposed in the literature for the problem are for different variants of online matching. Finally, ridesharing displays an inherent connection to the *k-taxi problem* [26, 37, 45], which, in turn, is a generalization of the well-known *k-server problem* [46, 47][1]. In the k-taxi problem, once a request appears (with a source and a destination), one of the k taxis at the platform's disposal must serve the request. Viewing shared rides (multiple passengers that have already been matched in a previous step) as requests, one can clearly apply the k-taxi (and k-server algorithms) to the ridesharing setting. Granted, the k-server algorithms have been designed to operate in a more challenging setting in which (a) the requests have to be served *immediately*, whereas normally there is some leeway in that regard, often at the expense of customer satisfaction, and (b) the positions of requests are typically *adversarially* chosen, rather than following some distribution, as is the case in reality. Despite those facts, the fundamental idea behind these algorithms is a pivotal part of ridesharing, as it aims to *serve existing requests efficiently, but at the same time place the vehicles as well as possible to serve future requests*. This is also the main principle of the relocation strategies for idle taxis.

The algorithms that we consider in this paper are appropriate modifications of the most significant ones that have been proposed for the aforementioned key algorithmic primitives of the ridesharing problem, as well as heuristic approaches which are based on the same principles, but were specifically designed with the ridesharing application in mind. We emphasize that such modifications are needed, primarily because many of these algorithms were tailored for sub-problems of the ridesharing setting, and end-to-end solutions in the literature are rather scarce.

Much of the related work in the literature focuses on approaches that are inherently centralized and require knowledge of the full ridesharing network, which makes them rather computationally intensive. As an additional goal of our investigation, we would like to identify solutions that are lightweight, decentralized, and which ideally run *on-device*. Of course there have been proposed some hybrid, and decentralized approaches for the ridesharing problem (e.g., [40, 64]), and several of the algorithms that we include in our experimental evaluation can be implemented in a decentralized manner. As it turns out though, the ALMA algorithm of [28], which has been designed with precisely these objectives in mind (low computational complexity, scalability, and low communication cost), performs very well across the board with respect to our objectives.

---

[1]In fact the latter two problems are quite closely connected, and algorithms for the k-server problem can be used to solve the k-taxi problem. See [26] for more details.

An important component of a successful ridesharing application is relocation. Many studies in shared mobility systems have shown that the adoption of a relocation strategy can help improve the system performance for their specific context [3, 12, 20, 40, 54, 58, 68]. Strategies include using a short window of known active requests [3], historical demand [40], or prediction techniques to predict future demand [65]. Yet, relocation by nature increases vehicle travel distance, leading to undesirable consequences (economical, environmental, maintenance, management of human resources, etc.), thus a balance needs to be struck. Most of the employed relocation approaches are course-grained; the network is generally divided into several zones, blocks, etc. [40, 54, 68] and the entities (e.g., the vehicles) move between the zones. However, compared to other shared mobility systems, dynamic ridesharing posses unique challenges, meaning that such coarse-grained approaches are not appropriate: most of them are centralized – thus computationally intensive and not scalable –, they might not take into account the behavior of other drivers, potentially leading to over-saturation of high demand areas, and, most importantly, they are *slow to adapt* to the highly dynamic nature of the problem (e.g., responding to high demand generated by a concert, or the fact that vehicles remain free for only a few minutes at a time). The problem clearly calls for fine-grained solutions, yet such approaches in the literature are still rather scarce. In this paper, we employ such a fine-grained relocation scheme (similarly to [3]), based on matching between the idle taxis and the *potential* requests, which is better suited for the problem at hand.

Finally, we emphasize that, while several papers in the literature provide detailed evaluations on realistic datasets, (e.g., see [2, 3, 28, 60, 61]), they either (a) only consider parts of the ridesharing problem and therefore do not propose end-to-end solutions, (b) only evaluate a few newly-proposed algorithms against some basic baselines, (c) only consider a limited number of performance metrics, predominantly with regard to the overall efficiency and often without regard to QoS metrics or (d) perform evaluations on a much smaller scale, thus not capturing the real-life complexity of the problem. On the contrary, our work provides a comprehensive evaluation of a large number of proposed algorithms, over multiple different metrics, and for real-world scale, end-to-end problems.

2 PROBLEM STATEMENT & MODELING

In this section we formally present the Dynamic Ridesharing, and Fleet Relocation (DRSFR) problem. To avoid introducing unnecessary notation, we only present the description of the model here; precise notation and details are provided in the respective sections where they are used.

In the DRSFR problem there is a (potentially infinite) metric space $\mathcal{X}$ representing the topology of the environment, equipped with a distance function $\delta : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$. Both are known in advance. At any moment, there is a (dynamic) set of available taxi vehicles $\mathcal{V}_t$, ready to service customer requests (i.e., drive to the pick-up, and subsequently to the destination location). Between servicing requests, vehicles can relocate to locations of potentially higher demand, to mitigate spatial search frictions between drivers. Customer requests appear in an online manner at their respective pick-up locations, wait to potentially be matched to a shared ride, and finally are serviced by a taxi to their respective destination. In order for two requests to be able to share a ride, they must satisfy *spatial*, and *temporal* constraints. The former dictates that requests should be matched only if there is good spatial overlap among their routes. Yet, due to the latter constraint, requests cannot be matched even if they have perfect spatial overlap, if they are not both 'active' at the same time. Finally, the DRSFR is an inherently *online* problem, as we are unaware of the requests that will appear in the future, and need to make decisions before the requests expire, while taking into account the dynamics of the fleet of taxis. The *goal* is to *minimize the cumulative distance driven* by the fleet of taxis, while maintaining *high QoS*, given that we *serve all requests*. Serving all requests improves passenger satisfaction, and, most importantly, allows us to ground our evaluation to a common scenario, ensuring a fair comparison.

2.1 Performance Metrics

2.1.1 *Global Metrics*.

Distance Driven: Minimize the cumulative distance driven by all vehicles for serving all the requests. We chose this objective as it directly correlates to passenger, driver, company, and environmental objectives (minimize cost, delay, CO_2 emissions, maximize the number of shared rides, improve QoS, etc.). All of the evaluated algorithms have to *serve all the requests*, either as shared, or single rides.

Complexity: Real-world time constraints dictate that the employed solution produces results in a reasonable time-frame[2].

2.1.2 *Passenger Specific Metrics – Quality of Service (QoS)*.

Time to Pair: Expected time to be paired in a shared ride, i.e., $\mathbb{E}[t_{paired} - t_{open}]$, where t_{open}, t_{paired} denote the time the request appeared, and was paired as a shared ride respectively. If the request is served as a single ride, then t_{paired} refers to the time the algorithm chose to serve it as such.

Time to Pair with Taxi: Expected time to be paired with a taxi, i.e., $\mathbb{E}[t_{taxi} - t_{paired}]$, where t_{taxi} denotes the time the (shared) ride was paired with a taxi.

Time to Pick-up: Expected time to passenger pickup, i.e., $\mathbb{E}[t_{pickup} - t_{taxi}]$, where t_{pickup} denotes the time the request was picked-up.

Delay: Additional travel time over the expected direct travel time (when served as a single ride, instead of a shared ride), i.e., $\mathbb{E}[(t_{dest} - t_{pickup}) - (t'_{dest} - t_{pickup})]$. t_{dest}, and t'_{dest} denote the time the request reaches, and would have reached as a single ride, its destination.

Research conducted by ridesharing companies shows that passengers' satisfaction level remains sufficiently high as long as the pick-up time is less than a certain threshold. The latter is corroborated by data on booking cancellation rate against pick-up time [67]. In other words, passengers would rather have a short pick-up time and long detour, than vice-versa [19]. This also suggests that an effective relocation scheme can considerably improve passenger satisfaction by reducing the average pick-up time (see Section 5.1).

2.1.3 *Driver Specific Metrics*.

Driver Profit: Total revenue earned minus total travel costs.

Number of Shared Rides: Directly related to the profit. By carrying more than one passenger at a time, drivers can serve more requests in a day, which consequently, increases their income [69].

Frictions: Waiting time experienced by drivers between serving requests (i.e., time between dropping-off a ride, and getting matched with another). Search frictions occur when drivers are unable to locate rides due to spatial supply and demand imbalance. Even though in our scenario matchings are performed automatically, without any searching involved by the drivers, lower frictions indicate lower regret by the drivers, thus lower temptation to potentially switch to an alternative ridesharing platform.

2.1.4 *Platform Specific Metrics*.

Platform Profit: Usually a commission on the driver's fee[3], and passenger fees (which, given that we serve all the requests, the latter would be constant across all the employed algorithms).

---

[2]For example UberPool has a waiting period of at most 2 minutes until you get a match (https://www.uber.com/au/en/ride/uberpool/), thus any algorithm has to run in under that time to be applicable in real life.

[3]E.g., Uber charges partners 25% fee on all fares (https://www.uber.com/en-GH/drive/resources/payments/).
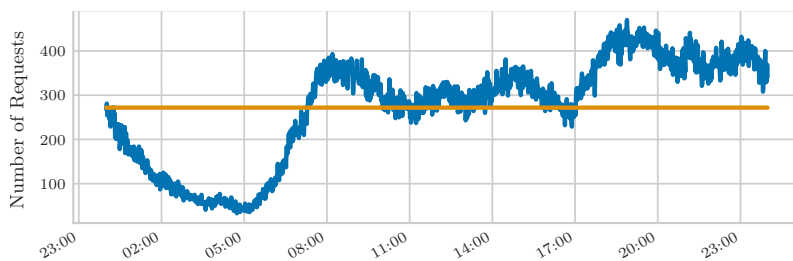
Fig. 1. Request per minute on January 15, 2016 (blue line). Mean value = 272 requests (yellow line).

Quality of Service (QoS): Refer to the aforementioned, passenger specific metrics. Improving the QoS to their costumers correlates to the growth of the company.

Number of Shared Rides: The matching rate is important especially in the nascent stage of a ridesharing platform [31].

We do not report separate values on the aforementioned metrics, as they directly correlate to their respective passenger, and driver specific ones.

2.2 Modeling

2.2.1 **Dataset**. We have used the yellow taxi trip records of 2016, provided by the NYC Taxi and Limousine Commission[4]. For every request, the dataset provides amongst others the pick-up and drop-off times, and geo-location coordinates. Time is discrete, with granularity of 1 minute (same as the dataset). On average, there are 272 new requests per minute, totaling to 391479 requests on the broader NYC area (352455 in Manhattan) on the evaluated day (Jan, 15). Figure 1 depicts the request arrival per minute on the aforementioned day.

2.2.2 **Taxi Vehicles**. A unique feature of the NYC Yellow taxis is that they may only be hailed from the street and are not authorized to conduct pre-arranged pick-ups. This provides an ideal setting for a counter-factual analysis since (1) we can assume a realistic position of each taxi at the beginning of the simulation (last drop-off location), and (2) all observed rides are obtained through search, thus – assuming reasonable prices, and delays – customers do not have nor are willing to take an alternative means of transportation. Thus, validating our choice that all of the algorithms considered will have to eventually serve all the requests. By law, there are 13, 587 taxis in NYC[5]. The majority of the results presented in this paper use a much lower number of vehicles (what we call *base number*) for three reasons: (1) to reduce the complexity of the problem, given that most of the employed algorithms can not handle such a large number of vehicles, (2) to evaluate under resource scarcity – making the problem harder – to better differentiate between the results, and (3) to investigate the possibility of a more efficient utilization of resources, with minimal cost to the consumers. However, we still present simulations for a wide range of vehicles, up to close to the total number. The number, initial location, and speed of the taxi vehicles were calculated as follows:

- We calculated the *base number* of taxis, as the minimum number of taxis required to serve all requests as single rides (no ridesharing). If a request appears, and all taxis are occupied serving other requests, we increase the required number of taxis by one. This resulted to around 4000 − 5000 vehicles (depending on the size of the simulation, see Section 5). Simulations were conducted for {×0.5, ×0.75, ×1.0, ×2.0, ×3.0} the base number.

---

[4]https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page
[5]https://www1.nyc.gov/site/tlc/businesses/yellow-cab.page

- Given a number of taxis, V, the initial position of each taxi is the drop-off location of the last V requests, prior to the starting time of the simulation. To avoid cold start, we compute the drop-off time of each request, and assume the vehicle occupied until then.
- The vehicles' average speed is estimated to 6.2 m/s (22.3 km/h), based on the trip distance and time per trip as reported in the dataset, and corroborated by the related literature (in [60] the authors estimated the speed to be between $5.5 - 8.5$ m/s depending on the time of day).

2.2.3 ***Customer Requests****.* A request, r, is a tuple $\langle t_r, s_r, d_r, k_r \rangle$. Request r appears (becomes *open*) at its respective pick-up time (t_r), and geo-location (s_r). Let d_r denote the destination. Each request admits a willingness to wait (k_r) to find a match (rideshare), i.e., we assume *dynamic* waiting periods per request. The rationale behind k_r is that requests with longer trips are more willing to wait to find a match than requests with destinations near-by. After k_r time-steps we call request r, *critical*. If a critical request is not matched, it has to be served as a single ride. Recall that in our setting *all* of the requests must be served. Let $\mathcal{R}_t^{\text{open}}, \mathcal{R}_t^{\text{critical}}$ denote the sets of open, and critical requests respectively, and let $\mathcal{R}_t = \mathcal{R}_t^{\text{open}} \cup \mathcal{R}_t^{\text{critical}}$.

We calculate k_r as in related literature [28]. Let $w_{\min}$, and $w_{\max}$ be the minimum and maximum possible waiting time, i.e., $w_{\min} \leq k_r \leq w_{\max}, \forall r$. Knowing s_r, d_r, we can compute the expected trip time ($\mathbb{E}[t_{\text{trip}}]$). Assuming people are willing to wait proportional to their trip time, let $k_r = q \times \mathbb{E}[t_{\text{trip}}]$, where $q \in [0, 1]$. $w_{\min}, w_{\max}$, and q can be set by the ridesharing company, based on customer satisfaction (following [28], let $w_{\min} = 1$, $w_{\max} = 3$, and $q = 0.1$).

2.2.4 ***Rides****.* A (shared)ride, ρ, is a pair $\langle r_1, r_2 \rangle$, composed of two requests. If a request r is served as a single ride, then $r_1 = r_2 = r$. Let $\mathcal{P}_t$ denote the set of rides waiting to be matched to a taxi at time t. Contrary to some recent literature on high capacity ridesharing (e.g., [3, 50]), we purposefully restricted ourselves to rides of at most two requests for two reasons: complexity, and passenger satisfaction. The complexity of the problem grows rapidly as the number of potential matches increases, while most of the proposed/evaluated approaches already struggle to tackle matchings of size two on the scale of a real-world application. Moreover, even though a fully utilized vehicle would ultimately be a more efficient use of resources, it diminishes passenger satisfaction (a frequent worry being that the ride will become interminable, according to internal research by ridesharing companies) [18, 69]. Given that a hard constraint is the servicing of all requests, we do not assume a time limit on matching rides with taxis; instead we treat it as a QoS metric.

2.2.5 ***Distance Function****.* The optimal choice for a distance function would be the actual driving distance. Yet, our simulations require trillions of distance calculations, which is not attainable. Given that the locations are given in latitude and longitude coordinates, it is tempting to use the Haversine formula[6] to estimate the Euclidean distance, as in related literature [18, 61]. We have opted to use the Manhattan distance, given that the simulation takes place mostly in Manhattan. To evaluate our choice, we collected more than 12 million actual driving distances using the Open Source Routing Machine (project-osrm.org), which computes the shortest path in road networks. Manhattan distance's error, compared to the actual driving distance, was -0.5 ± 2.9 km, while Euclidean distance's was -3.2 ± 3.8 km.

2.2.6 ***Pricing****.* A combination of an one-time flag drop fee ($\beta = 2.2$ \$[7]), distance fare ($\pi_I = 0.994$ \$/km for a single ride, $\pi_{II} = 0.8$ \$/km shared[7]), fuel price (3.2 \$/gal[8]), and vehicle mileage (46.671

(a) Request – Request Matching   (b) (Shared) Ride – Taxi Matching   (c) Idle Taxi Relocation
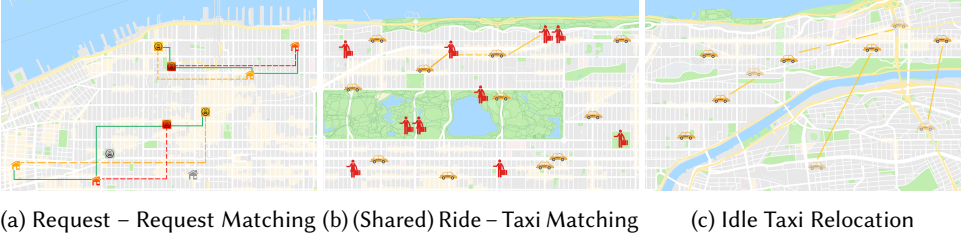
Fig. 2. The three separate components of the DRSFR problem.

km/gal [20]). The aforementioned fuel price and mileage result in a cost per km $c = 0.0686$ \$/km. The revenue $M(\rho)$ of a taxi driver from serving ride ρ is given by the following equation [20]:

$$M(\rho) = \begin{cases} \beta + \pi_I \delta(s_r, d_r) - c\delta(s_v, s_r, d_r) & \text{, if } \rho \text{ single} \\ 2\beta + \pi_{II} \delta(s_{r_1}, d_{r_1} | r_2) + \pi_{II} \delta(s_{r_2}, d_{r_2} | r_1) - c\delta(s_v, s_{r_1}, s_{r_2}, d_{r_1}, d_{r_2}) & \text{, if } \rho \text{ shared} \end{cases} \quad (1)$$

where, with some slight abuse of notation, $\delta(s_v, s_r, d_r)$ denotes the distance from the current location of the taxi s_v, to the pick-up and subsequently drop-off location of the ride, $\delta(s_{r_1}, d_{r_1} | r_2)$ denotes the distance driven from the pick-up to the destination of r_1, given that r_1 will share the ride with r_2 (similarly $\delta(s_{r_2}, d_{r_2} | r_1)$ for r_2), and finally, $\delta(s_v, s_{r_1}, s_{r_2}, d_{r_1}, d_{r_2})$ denotes the total driving distance of the taxi for serving the two requests starting from s_v.

2.2.7 **Embedding into HSTs**. A starting point of many of the employed k-server algorithms is embedding the input metric space $\mathcal{X}$ into a distribution μ over σ-hierarchically well-separated trees (HSTs), with separation $\sigma = \Theta(\log |\mathcal{X}| \log(k \log |\mathcal{X}|))$, where $|\mathcal{X}|$ denotes the number of points. It has been shown that solving the problem on HSTs suffices, as any finite metric space can be embedded into a probability distribution over HSTs with low distortion [34]. The distortion is of order $O(\sigma \log_\sigma |\mathcal{X}|)$, and the resulting HSTs have depth $O(\log_\sigma \Delta)$, where Δ is the diameter of $\mathcal{X}$ [7].

Given the popularity of the aforementioned method, it is worth examining the size of the resulting trees. Given that the geo-coordinate system is a discrete metric space, we could directly embed it into HSTs. Yet, the size of the space is huge, thus for better discretization we have opted to generate the graph of the street network of NYC. To do so, we used data from openstreetmap.org. Similarly to [60], we filtered the streets selecting only primary, secondary, tertiary, residential, unclassified, road, and living street classes, using those as undirected edges and street intersections as nodes. The resulting graph for NYC contains 66543 nodes, and 95675 edges (5018, and 8086 for Manhattan). Given that graph, we generate the HSTs [60].

3   EMPLOYED ALGORITHMS & CHALLENGES

3.1   Problem Decomposition

The DRSFR problem can be decomposed into the following three parts (Figure 2):

   (a) Request – request matching to create a (shared) ride,
   (b) Ride to taxi matching,
   (c) Relocation of the idle fleet.

Complexity issues make the simultaneous consideration of all three problems when taking a decision impractical. Instead, a more realistic approach is to tackle each component individually, under minimum consideration of the remaining two. Each of the aforementioned components is

a significant problem in its own right. Step (a) refers to the problem of *Online Maximum Weight Matching with Delays* – given a non-bipartite graph, nodes appear in an online manner and leave after some time-steps. Nodes can be matched only while being present, and the goal is to maximize the cumulative utility over a finite time horizon [5, 33]. Step (b) can be viewed either as an *Online Maximum Weight Bipartite Matching with Delays* [4] – the difference here being that the graph is bipartite, with rides on one side, and taxis on the other – or as a *k-Taxi Problem* [26] (and by extension as a *k-Server Problem* [46, 47]). In the latter formulation, k taxis (servers) have to move to serve all the requests while minimizing the total distance traveled, with the caveat being that each will end up at the destination of the served request. Finally, step (c) can be either viewed as the *k-center* problem or the more general *k-Facility Location Problem* [41], concerned with the optimal placement of facilities (taxis) to minimize transportation costs, or as an *Online Maximum Weight Matching* problem, performed on the *history* of requests. Given the high complexity of the former problems (they are both NP-hard, in fact, APX-hard [36, 43]), we have opted for the latter interpretation.

3.2 Algorithms

We have evaluated a variety of approaches ranging from offline maximum weight matching (MWM), and greedy solutions, to online MWM, k-Taxi/Server algorithms, and linear programming. Offline algorithms (e.g., MWM, ALMA, Greedy) can be run either in a just-in-time (JiT) manner – i.e., when a request becomes critical – or in batches, i.e., every x minutes (given that our dataset has granularity of 1 minute, we run in batches of 1, and 2 minutes). We solve each of the three steps of the DRSFR problem (Figure 2) individually. When possible, we use the same algorithm for both steps (a) and (b). k-Taxi/Server algorithms, though, can not solve step (a), thus we opted to use the best performing algorithm for step (a) (namely the offline MWM run in batches). Step (c) was treated separately; due to the computational complexity of most of the evaluated approaches, we opted to evaluate only the most promising solutions. In what follows, we will start by describing the employed approaches for the ridesharing part, i.e., steps (a) and (b). In the Section 4 we describe the employed techniques for dynamic relocation.

Matching Graphs: At time t, let $\mathcal{G}_a = (\mathcal{R}_t, \mathcal{E}_t^a)$, where $\mathcal{E}_t^a$ denotes the weighted edges between requests. With a slight abuse of notation, let $\delta(s_{r_1}, s_{r_2}, d_{r_1}, d_{r_2})$ denote the minimum distance required to serve both r_1, and r_2 (as a shared ride, i.e., excluding the case of first serving one of them and then the other) with a single taxi located either in s_1, or s_2. The weight w_{r_1, r_2} of an edge $(r_1, r_2) \in \mathcal{E}_t^a$ is defined as $w_{r_1, r_2} = \delta(s_1, d_1) + \delta(s_2, d_2) - \delta(s_{r_1}, s_{r_2}, d_{r_1}, d_{r_2})$ (similarly to [3, 28]). If $r_1 = r_2$, let $w_{r_1, r_2} = 0$ (single passenger ride). Intuitively, this number represents an approximation (given that it is impossible to know in advance the location of the taxi that will serve the ride) on the travel distance saved by matching requests r_1, and r_2[9].

Similarly, at time t, let $\mathcal{G}_b = (\mathcal{P}_t \cup \mathcal{V}_t, \mathcal{E}_t^b)$, where $\mathcal{E}_t^b$ denotes the weighted edges between rides and taxis. With a slight abuse of notation, let $\delta(s_v, s_{r_1}, s_{r_2}, d_{r_1}, d_{r_2})$ denote the minimum distance required (out of all the possible pick-up and drop-off combinations) to serve both r_1, and r_2 with a single taxi located at s_v. The weight w_{r_1, r_2} of an edge $(r_1, r_2) \in \mathcal{E}_t^b$ is defined as $w_{r_1, r_2} = 1/\delta(s_v, s_{r_1}, s_{r_2}, d_{r_1}, d_{r_2})$. If $r_1 = r_2$ (single passenger ride), let $\delta(s_v, s_{r_1}, s_{r_2}, d_{r_1}, d_{r_2}) = \delta(s_v, s_{r_1}, d_{r_1})$. For the step (b) of the DRSFR problem, we run the offline algorithms every time the set of rides ($\mathcal{P}_t$) is not empty.

3.2.1 *Maximum Weight Matching (MWM).* To match requests into shared rides (step (a) of the DRSFR problem), we find the maximum weight matching on $\mathcal{G}_a$. To match rides with taxis (step

---

[9]It also ensures that the shared ride will cost less than the single ride option.

(b)), we find the maximum weight matching on $\mathcal{G}_b$. On both cases we use the *blossom algorithm* [32]. Not surprisingly, MWM results in high quality allocations, but that comes with an overhead in running time, compared to simpler, 'local' solutions (see Section 5). This is because blossom's running time – on a graph (V, E) – is $O(|E||V|^2)$, and we have to run it three times, one for each step of the DRSFR problem. Additionally, the MWM algorithm inherently requires a global view of the whole request set in a time window, and is therefore not a good candidate for the fast, decentralized solutions that are more appealing for real-life applications.

3.2.2 **ALtruistic MAtching Heuristic (ALMA)** *[28].* ALMA is a recently proposed lightweight heuristic for weighted matching. A distinctive characteristic of ALMA is that agents (in our context: requests / rides) make decisions locally, based solely on their own utilities. While contesting for a resource (in our context: request / taxi), each agent will back-off with probability that depends on their own utility loss of switching to their respective remaining resources. Agents that do not have good alternatives will be less likely to back-off and vice versa. It is inherently decentralized, requires only a 1-bit partial feedback, and has constant in the total problem size running time, under reasonable assumptions on the preference domain of the agents. Thus, it is an ideal candidate for an *on-device* solution. Moreover, in [28] it was shown to achieve high quality results on a simpler version of step (a) of the DRSFR problem.

3.2.3 **Greedy***.* Greedy is a very simple algorithm, which selects a node i randomly, and matches it with $j = \arg\max(w_{i,j})$. Greedy approaches are appealing[10], not only due to their low complexity, but also because real-time constraints dictate a short planning windows which diminish the benefit of batch optimization solutions compared to myopic approaches [69].

3.2.4 **Approximation (Appr)** *[11].* Similar to MWM, this is a recently-proposed offline algorithm for solving steps (a), and (b) of the DRSFR problem. It takes a two-phase approach: first, it matches requests to shared rides using minimum weight matching based on the shortest distance to serve any request pair but on the worst pickup choice. Then it matches rides to taxis using again minimum weight matching, and assuming the weight to be the distance of the closest pick-up location of the two. The authors of [11] prove a worst-case approximation guarantee of 2.5 for the algorithm.

3.2.5 **Postponed Greedy (PG)** *[5].* This is another very recently proposed, 1/4-competitive algorithm for maximum weight online matching with deadlines (step (a) of the DRSFR problem). Contrary to our setting, it was designed for fixed deadlines, i.e., $k_r = c, \forall r \in \mathcal{R}$. When a request r appears, the algorithm creates a virtual seller and a virtual buyer for that request. The buyer matches greedily with the best seller so far, and the choice remains fixed. When r becomes critical, it's role will be randomly finalized either as a seller, or buyer. If r is a seller, and a subsequent buyer was matched with r, the match is finalized. The major difference is that in our setting requests become critical out-of-order, and a critical request can not be matched later. Thus, when a request becomes critical, if determined to be a seller, the match is finalized (if one has been found), otherwise the request is treated as a single ride.

3.2.6 **Greedy Dual (GD)** *[15].* An online algorithm for solving the Min-cost (Bipartite) Perfect Matching with Delays, i.e., both steps (a), and (b) of the DRSFR problem, based on the popular Primal-Dual technique [39]. The weight (cost) of an edge in this setting includes arrival times as well, specifically $w_{r_1, r_2} = (\delta(s_1, s_2) + \delta(d_1, d_2))/u_{\text{average}} + |t_1 - t_2|$, where u_{average} is the average speed (see Section 2.2.2). The algorithm partitions all the requests into active sets, starting with the singleton $\{r\}$ for a newly arrived request r. Every time-step these actives sets grow, until the

---

[10][69] reports that GrabShare's scheduling component has used an entirely greedy approach to allocate bookings to drivers. Lyft also started with a greedy system [18].

weight of edges of different active sets make the dual constraints of the problem tight. Then the active sets merge, and the algorithm matches as many pairs of free requests in these sets as possible. It's a $O(|\mathcal{R}|)$-competitive algorithm, that works with infinite metric spaces, potentially making the algorithm better suited for applications like the DRSFR problem. Yet, it does not take into account the willingness to wait (k_r), missing matches of requests that became critical. Despite being designed for bipartite matchings as well, we opted out from using it for step (b) since it would require to create a new node every time a taxi vehicle drops-off a ride and becomes available.

3.2.7 **Balance (Bal)** *[53].* Balance is simple and classical k-server algorithm from the literature of competitive analysis. A ride is served by the taxi that has the minimum sum of the distance traveled so far plus its distance to the source of the ride (chosen randomly between the sources of the two requests composing the ride). It is min-max fair, i.e., it greedily minimizes the maximum accumulated distance among the taxis. The competitive ratio of the algorithm is $|\mathcal{X}| - 1$ in arbitrary metric spaces with $|\mathcal{X}|$ points [53].

3.2.8 **Harmonic (Har)** *[57].* Harmonic is another classical randomized algorithm from the k-server literature, which is simple and memoryless. It matches a taxi with a ride with probability inversely proportional to the distance from its source (chosen randomly between the sources of the two requests composing the ride). The trade-off for the simplicity is the high competitive ratio, which is $O(2^{|\mathcal{V}|} \log |\mathcal{V}|)$ [9].

3.2.9 **Double Coverage (DC)** *[23].* Double-coverage is one of the two most famous k-server algorithms in the literature. The algorithm is designed for HSTs and extends to general finite metric spaces $\mathcal{X}$ via HST embeddings. First we perform the embedding [8, 35] and then, to determine which taxi will serve a ride, all unobstructed taxis move towards its source (chosen randomly between the sources of the two requests composing the ride) with equal speed. When during this process, a taxi becomes obstructed (i.e., its path is blocked by another taxi), it stops, while the rest keep moving. When a taxi reaches the leaf with the ride, the process stops, and each taxi maintains it's position on the HST. Given that only leafs correspond to locations on $\mathcal{X}$, we chose to implement the *lazy* version of the algorithm (which is equivalent to the original definition e.g., see [46]), i.e. only the taxi serving the request will move on $\mathcal{X}$. This is also on par with the main goal of minimizing the distance driven. The algorithm is k-competitive on all tree metrics [24].

3.2.10 **Work Function (WFA)** *[25, 47].* WFA is perhaps the most important k-server algorithm, as it provides the best competitive ratio to date, due to the celebrated result of [47]. It is a dynamic programming approach which, intuitively, computes the optimal solution until time $t - 1$, plus a greedy cost for switching taxi locations. An obvious obstacle that makes the algorithm intractable in practice is that the complexity rises from step to step, resulting in computation and/or memory issues. We implemented an efficient implementation using network flows, as described in [59]. Yet, as the authors of [59] state as well, the only practical way of using the WFA is switching to its window version w-WFA, where we only optimize for the last w rides. Even though the complexity of w-WFA does not change between time-steps, it does change with the number of taxis. The resulting network has $2|\mathcal{P}| + 2|\mathcal{V}| + 2$ nodes, and we have to run the Bellman–Ford algorithm [13] at least once to compute the potential of nodes and make the costs positive (Bellman–Ford runs in $O(|\mathcal{P}||\mathcal{V}|)$). We refer the reader to [14]) for more details on network optimization. As before, the source of the ride is chosen randomly between the sources of the two requests composing the ride.

3.2.11 k-**Taxi** *[26].* This is a very recent algorithm for the k-taxi problem, which provides the best possible competitive ratio for the problem. The algorithm operates on HSTs, where the rides and taxis at any time are placed at its leaves. First, it generates a Steiner tree that spans the leaves that

have taxis or rides, and then uses this tree to schedule rides, by simulating an electrical circuit. In particular, whenever a ride appears at a leaf, the algorithm interprets the edges of the tree with length R as resistors with resistance R, which determine the fraction of the current flow that will be routed from the node corresponding to the taxi towards the ride. These fractions are then interpreted as probabilities which determine which taxi will be chosen to pick up the ride.

3.2.12 **High Capacity (HC)** *[3].* Highly cited paper, and the only one in our evaluated approaches that addresses vehicle relocation (step (c)). Contrary to our approach, they tackle steps (a), and (b) simultaneously, leaving step (c) as a separate sub-problem. Their method consists of five steps: (i) computing a pairwise request-vehicle shareability graph (RV-graph) [60], (ii) computing a graph of feasible trips and the vehicles that can serve them (RTV-graph), (iii) computing a greedy solution for the RTV-graph, (iv) solving an ILP to compute the best assignment of vehicles to trips, using the previously computed greedy solution as an initial solution, and finally (v - optional) if there remain any unassigned requests, solving an ILP to optimally assign them to idle vehicles based on travel times. Given the ILP formulation, this is the most promising approach in terms of solution quality, but it is not scalable, and effectively impractical to apply in the real world. Worse case, the number of variables in the ILP is $O(|\mathcal{V}||\mathcal{R}|^2)$ – which results in 27 - 216 million variables, given that every time-step we have approximately 300 - 600 requests, and as many taxis – and the number of constraints is $|\mathcal{V}| + |\mathcal{R}|$. The latter make hard to even compute the initial greedy solution in real-time. The authors of [3] circumvent this problem by enforcing delay constraints, but in our modeling every algorithm has to serve all requests, resulting in a prohibited large ILP. We use IBM-CPLEX [16] to solve the resulting ILPs.

3.2.13 **Baseline: Single Ride**. Uses MWM to schedule the serving of single rides to taxis (there is no ridesharing, i.e., we omit step (a) of the DRSFR problem).

3.2.14 **Baseline: Random**. Makes random matches, provided that the edge weight is non-negative.

While our evaluation contains many recently proposed algorithms for matching, the observant reader might notice that, with the exception of k-taxi, our k-server algorithms are from the classical literature. We did consider more recent k-server algorithms (e.g., [7, 29, 48]), but their complexity turns out to be prohibitive. This is mainly because they proceed via an 'online rounding' of an LP-relaxation of the problem, which maintains a variable for every (time-step, point in the metric space) pair. Even for one hour (3600 time-steps) and only for our discretization of Manhattan (5018 nodes), we need more than 18 million variables (230 million for NYC).

4 DYNAMIC VEHICLE RELOCATION

The aim of any relocation strategy is to improve the spatial allocation of supply. Serving requests redistributes the taxis, resulting in an inefficient allocation. One can assume a 'lazy' approach, relocating vehicles only to serve requests. While this minimizes the cost of serving a request (e.g., distance driven, fuel, etc.), it results in sub-optimal QoS. Improving the QoS (especially the time to pick-up, since it highly correlates to passenger satisfaction, see Section 2.1.2) plays a crucial role in the growth of a company. The *goal* then is to:

Improve the QoS metrics, while minimizing the excess distance driven.

There are two ways to enforce relocation: *passive*, and *active*. Ridesharing platforms, like Uber and Lyft, have implemented market-driven pricing as a passive form of relocation. Counterfactual analysis performed in [20] shows that implementing pricing rules can result in daily net surplus gains of up to 232000 and 93000 additional daily taxi-passenger matches. While the gains are

Fig. 3. Percentage of similar trips per hour in Manhattan, January 15, 2016 (blue line). Mean value = 13.3% (yellow line).

substantial, the market might be slow to adapt, and drivers and passengers do not always follow equilibrium policies. Contrary to that, our approach is *active*, in the sense that we directly enforce relocation. Moreover, we adopt a more *anthropocentric* approach: in our setting, the demand is fixed, thus the goal is not to increase revenue as a result of serving more rides, but rather to improve the QoS[11].

There are many ways to approach dynamic relocation (part (c) of the DRSFR problem). High Capacity [3] solves an ILP, which could reach high quality results, but it is not scalable nor practical. Ideally, we would like a solution that can run *on-device*. The k-server algorithms perform an implicit relocation, yet they are primarily developed for adversarial scenarios, and do not utilize the plethora of historic data[12]. In reality, requests follow patterns that emerge due to human habituality (e.g., during the first half of the day in Manhattan, there are many more drop-offs in Midtown compared to pickups [20]). Density based clustering [70] is a natural approach, yet, due to the vast number of requests, the only discernible clusters were of large regions (Manhattan, Bronx, Staten Island, Brooklyn, or Queens), which does not allow for fine-grained relocation. Given the high density of the requests, and the low frictions of the taxis (i.e., taxis remain free for relocation only for a short time window), we opted for a simple, fine-grained, matching approach. As a matter of fact, we tried zone based relocation (generating zones based on historical data using the OPTICS clustering algorithm [70], or using pre-defined clusters based on population density according to the NYC census data[13]), but achieved significantly inferior results.

4.1 Patterns in Customer Requests

To confirm the existence of transportation patterns, we performed the following analysis: For each request r on January 15[14], we searched the past three days for requests r' such that $|t_r - t_{r'}| \leq 10$, $\delta(s_r, s_{r'}) \leq 250$, and $\delta(d_r, d_{r'}) \leq 250$. The results are depicted in Figure 3. On average, 13.3% of the trips are repeated across all three previous days, peaking at 43.7% on rush hours (e.g., 6-8 in the morning).

4.2 Proposed Approach

We propose a fine-grained, weighted matching relocation, which is *plug-and-play* and can be combined with *any* algorithm for steps (a) and (b). Given the existence of transportation patterns, we use the history to predict a set of *expected future requests*. Specifically, let D, and T be the

---

[11]Decreased delays can also in turn improve revenue by serving more requests in a fixed time window.

[12]NYC TLC has been proving data on yellow taxi trips since 2009.

[13]https://guides.newman.baruch.cuny.edu/nyc_data/nbhoods

[14]January 15, 2016 was selected as a representative date for our simulations since it is not a holiday, and it is a Friday thus sampling for past requests results in a representative pattern (contrary to sampling on a weekend for example).

Table 1. Employed algorithms for each step of the DRSFR problem.

	Step (a)	Step (b)	Step (c)
Maximum Weight Matching (MWM)	MWM	MWM	MWM/ALMA/Greedy
ALtruistic MAtching Heuristic (ALMA) [28]	ALMA	ALMA	ALMA
Greedy	Greedy	Greedy	Greedy
Approximation (Appr) [11]	Appr	Appr	-
Postponed Greedy (PG) [5]	PG	MWM	-
Greedy Dual (GD) [15]	GD	MWM	-
Balance (Bal) [53]	MWM	Bal	-
Harmonic (Har) [57]	MWM	Har	-
Double Coverage (DC) [23]	MWM	DC	-
Work Function (WFA) [25, 47]	MWM	WFA	-
k-Taxi [26]	MWM	k-Taxi	-
High Capacity (HC) [3]	HC	HC	(HC)

sampling windows, in days and minutes respectively (we used $D = 3$, and $T = 2$). Let t denote the current time-step. The set of past requests on our sampling window is $\mathcal{R}_{\text{past}} = \{r : t_r - t \leq T\}$, as long as r appeared at most D number of days prior to t. The set of expected future requests $\mathcal{R}_{\text{future}}$ is generated by sampling from $\mathcal{R}_{\text{past}}$. Relocation is performed in a just-in-time manner, every time the set of idle vehicles is not empty. We generate similar matching graphs as in Section 3.2, and then we proceed to match requests to shared rides, and rides to idle taxis. The difference being that now the set of nodes of $\mathcal{G}_a$ is $\mathcal{R}_{\text{future}} \cup \mathcal{R}_t$. Finally, each idle taxi starts moving towards the source of its match (given that these are expected rides, the source is picked at random between the sources of the two requests composing the ride).

We use the **MWM**, **ALMA**, and **Greedy** algorithms for the weighted matching. It is worth noting that we evaluated different approaches for the edge weights of $\mathcal{G}_b$ (ride – taxi matching). The best performing one in our scenario was the inverse of the distance, which makes sense given that we want to improve QoS, while *minimizing the extra distance driven*. Yet, depending on objectives, one might use different weights (e.g., the expected profit).

Table 1 presents all of the evaluated algorithms, subdivided into the three parts of the DRSFR problem. For example, k-Server algorithms can not solve step (a), thus we use the best performing algorithm for step (a), i.e., MWM. Similarly for PG, and GD for step (b).

5 SIMULATION RESULTS

In this section we present the results of our evaluation. For every metric we report the average value out of 8 runs. In this section we shortly detail only the most relevant results. Please refer to Section A.1 of the appendix for the complete results including larger test-cases on the broader NYC area and omitted metrics, standard deviation values, algorithms (e.g., WFA, and HC had to be evaluated in smaller test-cases), etc.

We first present our results on one hour, and base number (see Section 2.2.2) of taxis (Figure 4). Then we show that the results are robust at a larger time-scale (one day, Figure 5), and varying number of vehicles (2138 - 12828, Figure 6). Finally we present results on the step (c) of the DRSFR problem: dynamic relocation (Table 2, Figure 7).

Distance Driven: In the small test-case (Figure 4a) MWM performs the best, followed by Bal (+7%). ALMA comes second (+19%), and then Greedy (+21%). The high performance of Bal in this metric is because it uses MWM for step (a), which has a more significant impact on the distance driven.

(a) Distance Driven (m)   (b) Elapsed Time (ns)   (c) Time to Pick-up (s)   (d) Delay (s)

Fig. 4. January 15, 2016 – 08:00 - 09:00 – Manhattan – #Taxis = 4276 (base number).

(a) Distance Driven (m)   (b) Driver Profit ($)   (c) Frictions (s)   (d) Delay (s)

Fig. 5. January 15, 2016 – 00:00 - 23:59 (full day) – Manhattan – #Taxis = 5081 (base number).

(a) Distance Driven (m)   (b) Time to Pick-up (s)   (c) Delay (s)   (d) Cumulative Delay (s)

Fig. 6. January 15, 2016 – 08:00 - 09:00 – Manhattan – #Taxis = {2138, 3207, 4276, 8552, 12828}.

Similar results are observed for the whole day (Figure 5a), with Bal, ALMA, and Greedy achieving +4%, +18%, and +22% compared to MWM, respectively. Figure 6a shows that as we decrease the number of taxis, Bal loses its advantage, Greedy is pulling away from ALMA (9% worse than ALMA), while ALMA closes the gap to MWM (+17%).

Complexity: To estimate the complexity, we measured the elapsed time of each algorithm. Greedy is the fastest one (Figure 4b), closely followed by Har, Bal, and ALMA. ALMA is inherently decentralized. The red overlay denotes the parallel time for ALMA, which is 2.5 orders of magnitude faster than Greedy.

Time to Pick-up: MWM exhibits exceptionally low time to pick-up (Figure 4c), lower than the single ride baseline. ALMA, Greedy, and Bal have +69%, +76%, and +33% compared to MWM, respectively. As before, Figure 6b shows that as we decrease the number of taxis, Bal loses its advantage, and Greedy is pulling further away from ALMA. Note that to improve visualization, we removed DC's pick-up time as it was one order of magnitude larger than Appr.

Delay: PG exhibits the lowest delay (Figure 4d), but this is because it makes 26% fewer shared rides than the rest of the high performing algorithms. ALMA has the smallest delay (−13% compared

to MWM), with Greedy following at −1%, while Bal has +63% (both compared to MWM). As the number of taxis decrease (Figure 6c), ALMA's gains increase further (−22% compared to MWM).

Figure 6d depicts the cumulative delay, which is the sum of all delays described in Section 2.1.2, namely the time to pair, time to pair with taxi, time to pick-up, and delay. An interesting observation is that reducing the fleet size from 12828 (×3.0 of the base number) to just 3207 (×0.75 of the base number) vehicles (75% reduction) results in only approximately 2 minutes of additional delay. This goes to show the great potential for efficiency gains such technologies have to offer.

Profit & Frictions: Contrary to their performance in QoS metrics, GD, and Appr achieve the highest driver profit, 12% and 8% higher than MWM, respectively (although the low QoS and increased distance driven suggest low quality matchings, which can explain the higher revenue, yet deems them undesirable). Bal, and Har follow with +2 − 3%. ALMA and Greedy achieve the similar profit to MWM. PG exhibits significantly worse results (−13%), due to the lower number of shared rides it matches.

Small differences in driver profit can have a significant impact on the platform's profit. There are 13587 taxis in NYC[5], 67 − 85% of which are on the road at one time (i.e., 9103 - 11549 taxis). The additional 2% profit of Bal translates to $32.3 additional revenue in a day. Multiplied by the total number of taxis, and assuming that the platform keeps 25% as commission[3], this results in $73506 - $93258 additional revenue per day for the platform.

Figure 5b also depicts the maximum (red dot), and minimum (green dot) value of a driver's profit. Closer to the mean maximum value suggests a fairer algorithm for the drivers. Moreover, it is worth noting that the minimum value for all the algorithms is zero, meaning that there are taxis which remain unutilized (in spite of the fact that the number of taxis − in this scenario 5081 − is considerably lower than the current fleet size of yellow taxis).

Finally, Figure 5c shows the driver frictions. Just like with the profit, k-server algorithms seem to outperform matching algorithms by far. Compared to MWM, Bal and Har achieve a 97% decrease, while ALMA and Greedy achieve a 31%, and 23% decrease respectively. Given that we have a fixed supply, lower frictions indicate a more even distribution of rides amongst taxis.

Time to Pair with Taxi & Number of Shared Rides: Excluding the test-case with the smallest taxi fleet (×0.5 the base number), the time to pair with taxi was zero, or close to zero, for all the evaluated algorithms. The latter comes to show the potential for efficiency gains and better utilization of resources using smart technologies. The number of shared rides is approximately the same for all the employed algorithms, with notable exception the PG which makes 26% fewer shared rides.

*5.0.1 **MWM vs. Greedy Approaches:*** MWM seems to perform the best in the total distance driven, and the QoS metrics, which is reasonable since it makes optimal matches amongst passengers. Yet, MWM is hard to scale and requires a centralized solution. Greedy approaches are appealing[15], not only due to their low complexity, but also because real-time constraints dictate short planning windows which would potentially diminish the benefit of batch optimization solutions compared to myopic approaches [69].

*5.0.2 **ALMA vs. Greedy Approaches:*** ALMA was inherently developed for multi-agent applications. Agents make decisions locally, using completely uncoupled learning rules, and require only a 1-bit partial feedback [28], making it an ideal candidate for an *on-device* implementation. This is fundamentally different than a decentralized implementation of the Greedy algorithm for example. Even in decentralized algorithms, the number of communication rounds required grows

---

[15][69] reports that GrabShare's scheduling component has used an entirely greedy approach to allocate bookings to drivers. Lyft also started with a greedy system [18].

	MWM	ALMA	Greedy
Time to Pick-up	-48.95%	-55.18%	-55.03%
Time to Pick-up SD	-52.97%	-58.22%	-58.21%
Delay	-15.95%	-17.79%	-17.73%
Delay SD	-19.25%	-20.96%	-20.98%
Cumulative Delay	-38.37%	-43.23%	-43.11%
Total Distance	5.48%	6.25%	6.24%

Table 2. Relocation Gains.

Fig. 7. Time to Pick-up (s) – End-To-End Solution. +R in the x-axis labels indicates the use of relocation. January 15, 2016 – 00:00 - 23:59 – Manhattan – #Taxis = 5081.

with the size of the problem. However, in practice the real-time constraints impose a limit on the number of rounds, and thus on the size of the problem that can be solved within them. In fact, ALMA is of a greedy nature as well, albeit it utilizes a more intelligent backing-off scheme, thus there are scenarios where ALMA significantly outperforms the greedy, as proven by the simulation results. For example, in more challenging scenarios (smaller taxi fleet, or potentially different types of taxis) the smarter back off mechanism results in a more profound difference.

5.1 Relocation

A crucial trade-off of any relocation scheme is improving the QoS metrics, while minimizing the excess distance driven. Table 2 shows that our proposed scheme successfully balances this trade-off. In particular, ALMA – the best performing overall – radically improves the QoS metrics by more than 50% (e.g., ALMA decreases the pick-up time by 55%, and its standard deviation (SD) by 58%), while increasing the driving distance by only 6%. The cumulative delay is decreased by 43%. Recall that the proposed approach is *plug-and-play* and can be combined with *any algorithm* for steps (a) and (b) of the DRSFR problem. In this case, Table 2 uses MWM for steps (a) - (b). This provides the evaluated algorithms with a common ground, and allows for fair comparison focused only on the relocation part.

As a final step, we evaluate end-to-end solutions, using MWM, ALMA, and Greedy to solve all three of the steps of the DRSFR problem. Figure 7 depicts the time to pick-up (error bars denote one SD of uncertainty), a metric highly correlated to passenger satisfaction level [19, 67]. We compare against the single ride baseline (see Section 3.2.13). Once more, the proposed relocation scheme results in radical improvements, as the time to pick-up drops (compared to the single ride) from +14.09% to −41.76% for MWM, from +74.14% to −9.33% for ALMA, and finally, from +86.10% to −7.97% for Greedy. The latter comes to show that utilizing *a simple relocation scheme can eliminate the negative effects of ridesharing on the QoS metrics.*

6 CONCLUSION

The next technological revolution will be interwoven to the proliferation of intelligent systems. As we bridge the gap between physical and cyber worlds, we will give rise to decentralized, multi-agent based technologies, ideally run *on-device*. To gain insight into the problem, it is highly important to evaluate a diverse set of candidate solutions in settings designed to closely resemble reality. In this work, we focused on the key algorithmic components of the Dynamic Ridesharing and Fleet Relocation problem. To the best of our knowledge, our evaluation setting is one of the *largest* and most *comprehensive* to date. Our findings provide a clear-cut recommendation to ridesharing platforms, and validates the capacity for deployment of our proposed approach. As a highlight of our paper, we show that a recently proposed heuristic (ALMA), which exhibits the aforementioned

desired properties, offers an efficient, end-to-end solution for the Dynamic Ridesharing and Fleet Relocation problem.

REFERENCES

[1] Niels Agatz, Alan Erera, Martin Savelsbergh, and Xing Wang. 2012. Optimization for dynamic ride-sharing: A review. *European Journal of Operational Research* 223, 2 (2012), 295–303.

[2] Niels Agatz, Alan L Erera, Martin WP Savelsbergh, and Xing Wang. 2011. Dynamic ride-sharing: A simulation study in metro Atlanta. *Procedia-Social and Behavioral Sciences* 17 (2011), 532–550.

[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* (2017).

[4] Itai Ashlagi, Yossi Azar, Moses Charikar, Ashish Chiplunkar, Ofir Geri, Haim Kaplan, Rahul Makhijani, Yuyi Wang, and Roger Wattenhofer. 2017. Min-cost bipartite perfect matching with delays. (2017).

[5] Itai Ashlagi, Maximilien Burq, Chinmoy Dutta, Patrick Jaillet, Amin Saberi, and Chris Sholley. 2019. Edge Weighted Online Windowed Matching. In *Proceedings of the 2019 ACM Conference on Economics and Computation (EC '19)*. ACM.

[6] Siddhartha Banerjee, Daniel Freund, and Thodoris Lykouris. 2017. Pricing and Optimization in Shared Vehicle Systems: An Approximation Framework. In *Proceedings of the 2017 ACM Conference on Economics and Computation*. ACM.

[7] Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. 2015. A polylogarithmic-competitive algorithm for the k-server problem. *J. ACM* 62, 5 (2015), 1–49.

[8] Yair Bartal. 1996. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proceedings of 37th Conference on Foundations of Computer Science*. IEEE, 184–193.

[9] Yair Bartal and Eddie Grove. 2000. The harmonic k-server algorithm is competitive. *Journal of the ACM (JACM)* (2000).

[10] Kanika Bathla, Vaskar Raychoudhury, Divya Saxena, and Ajay D Kshemkalyani. 2018. Real-time distributed taxi ride sharing. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2044–2051.

[11] Xiaohui Bei and Shengyu Zhang. 2018. Algorithms for trip-vehicle assignment in ride-sharing. In *Thirty-Second AAAI*.

[12] Valérie Bélanger, Yannick Kergosien, Angel Ruiz, and Patrick Soriano. 2016. An empirical comparison of relocation strategies in real-time ambulance fleet management. *Computers & Industrial Engineering* 94 (2016), 216–229.

[13] Richard Bellman. 1958. On a routing problem. *Quarterly of applied mathematics* 16, 1 (1958), 87–90.

[14] Dimitri P Bertsekas. 1998. *Network optimization continuous and discrete models*. Athena Scientific Belmont.

[15] Marcin Bienkowski, Artur Kraska, Hsiang-Hsuan Liu, and Paweł Schmidt. 2018. A primal-dual online deterministic algorithm for matching with delays. (2018), 51–68.

[16] Christian Bliek1ú, Pierre Bonami, and Andrea Lodi. 2014. Solving mixed-integer quadratic programming problems with IBM-CPLEX: a progress report. In *Proceedings of the twenty-sixth RAMP symposium*. 16–17.

[17] Allan Borodin and Ran El-Yaniv. 2005. *Online computation and competitive analysis*. cambridge university press.

[18] Timothy Brown. [n.d.]. Matchmaking in Lyft Line — Part 1. eng.lyft.com/matchmaking-in-lyft-line-9c2635fe62c4.

[19] Timothy Brown. [n.d.]. Matchmaking in Lyft Line — Part 2. eng.lyft.com/matchmaking-in-lyft-line-691a1a32a008.

[20] Nicholas Buchholz. 2018. *Spatial equilibrium, search frictions and dynamic efficiency in the taxi industry*. Technical Report. mimeo, Princeton University.

[21] Mengjing Chen, Weiran Shen, Pingzhong Tang, and Song Zuo. 2019. Dispatching through pricing: modeling ride-sharing and designing dynamic prices. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*.

[22] Rui Chen and Christos G Cassandras. 2019. Optimization of ride sharing systems using event-driven receding horizon control. *arXiv preprint arXiv:1901.01919* (2019).

[23] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. 1990. New Results on Server Problems. (1990), 291–300.

[24] Marek Chrobak and Lawrence L Larmore. 1991. An optimal on-line algorithm for k servers on trees. *SIAM J. Comput.*

[25] Marek Chrobak and Lawrence L Larmore. 1991. The Server Problem and On-Line Games. *On-line algorithms* (1991).

[26] Christian Coester and Elias Koutsoupias. 2018. The Online k-Taxi Problem. *arXiv preprint arXiv:1807.06645* (2018).

[27] Jean-François Cordeau and Gilbert Laporte. 2007. The dial-a-ride problem: models and algorithms. *Ann. of op. research*.

[28] Panayiotis Danassis, Aris Filos-Ratsikas, and Boi Faltings. 2019. Anytime Heuristic for Weighted Matching Through Altruism-Inspired Behavior. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*.

[29] Sina Dehghani, Soheil Ehsani, MohammadTaghi Hajiaghayi, Vahid Liaghat, and Saeed Seddighin. 2017. Stochastic k-Server: How Should Uber Work? *arXiv preprint arXiv:1705.05755* (2017).

[30] John P Dickerson, Karthik A Sankararaman, Aravind Srinivasan, and Pan Xu. 2018. Allocation problems in ride-sharing platforms: Online matching with offline reusable resources. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[31] Chinmoy Dutta and Chris Sholley. 2018. Online matching in a ride-sharing platform. *arXiv preprint arXiv:1806.10327*.

[32] Jack Edmonds. 1965. Maximum matching and a polyhedron with 0 1-vertices. *Journal of research of the National Bureau of Standards B* (1965).

[33] Yuval Emek, Shay Kutten, and Roger Wattenhofer. 2016. Online matching: haste makes waste! (2016), 333–344.

[34] Jittat Fakcharoenphol, Satish Rao, Satish Rao, and Kunal Talwar. 2003. A Tight Bound on Approximating Arbitrary Metrics by Tree Metrics. In *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing (STOC '03).*

[35] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. 2004. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. System Sci.* 69, 3 (2004), 485–497.

[36] Tomás Feder and Daniel Greene. 1988. Optimal algorithms for approximate clustering. In *Proceedings of the twentieth annual ACM symposium on Theory of computing.* ACM, 434–444.

[37] Amos Fiat, Yuval Rabani, and Yiftach Ravid. 1994. Competitive k-server algorithms. *J. Comput. System Sci.* (1994).

[38] Masabumi Furuhata, Maged Dessouky, Fernando Ordóñez, Marc-Etienne Brunet, Xiaoqing Wang, and Sven Koenig. 2013. Ridesharing: The state-of-the-art and future directions. *Transportation Research Part B: Methodological* 57 (2013).

[39] Michel X Goemans and David P Williamson. 1997. The primal-dual method for approximation algorithms and its application to network design problems. *Approximation algorithms for NP-hard problems* (1997), 144–191.

[40] Maxime Guériau and Ivana Dusparic. 2018. SAMoD: Shared autonomous mobility-on-demand using decentralized reinforcement learning. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC).* IEEE.

[41] Sudipto Guha and Samir Khuller. 1999. Greedy strikes back Improved facility location algorithms. *Journal of algorithms.*

[42] Sin C Ho, WY Szeto, Yong-Hong Kuo, Janny MY Leung, Matthew Petering, and Terence WH Tou. 2018. A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research Part B: Methodological.*

[43] Wen-Lian Hsu and George L Nemhauser. 1979. Easy and hard bottleneck location problems. *Discrete Applied Mathematics* 1, 3 (1979), 209–215.

[44] Taoan Huang, Bohui Fang, Xiaohui Bei, and Fei Fang. 2019. Dynamic Trip-Vehicle Dispatch with Scheduled and On-Demand Requests. In *The Conference on Uncertainty in Artificial Intelligence (UAI.*

[45] Andrew P Kosoresow. 1997. Design and analysis of online algorithms for mobile server applications. (1997).

[46] Elias Koutsoupias. 2009. The k-server problem. *Computer Science Review* 3, 2 (2009), 105 – 118.

[47] Elias Koutsoupias and Christos H Papadimitriou. 1995. On the k-server conjecture. *Journal of the ACM (JACM)* (1995).

[48] James R Lee. 2018. Fusible HSTs and the randomized k-server conjecture. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS).* IEEE, 438–449.

[49] Nixie Lesmana, Xuan Zhang, and Xiaohui Bei. 2019. Balancing Efficiency and Fairness in On-Demand Ridesourcing. In *Proceedings of the 33rd Conference on Neural Information Processing Systems (NEURIPS).* to appear.

[50] Meghna Lowalekar, Pradeep Varakantham, and Patrick Jaillet. 2019. ZAC: A Zone pAth Construction Approach for Effective Real-Time Ridesharing. (2019).

[51] Hongyao Ma, Fei Fang, and David C Parkes. 2019. Spatio-Temporal Pricing for Ridesharing Platforms. In *Proceedings of the 2019 ACM Conference on Economics and Computation.* ACM, 583–583.

[52] Mark Manasse, Lyle McGeoch, and Daniel Sleator. 1988. Competitive algorithms for on-line problems. In *Proceedings of the twentieth annual ACM symposium on Theory of computing.* ACM, 322–333.

[53] Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. 1990. Competitive algorithms for server problems. *J. Algorithms* 11, 2 (1990), 208–230.

[54] L Miguel Martínez, Gonçalo Homem de Almeida Correia, Filipe Moura, and Mafalda Mendes Lopes. 2017. Insights into carsharing demand dynamics: Outputs of an agent-based model application to Lisbon, Portugal. *International Journal of Sustainable Transportation* 11, 2 (2017), 148–159.

[55] Abood Mourad, Jakob Puchinger, and Chengbin Chu. 2019. A survey of models and algorithms for optimizing shared mobility. *Transportation Research Part B: Methodological* (2019).

[56] 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 (2017), 208–226.

[57] Prabhakar Raghavan and Marc Snir. 1989. Memory versus randomization in on-line algorithms. (1989), 687–703.

[58] Claudio Ruch, Sebastian Hörl, and Emilio Frazzoli. 2018. Amodeus, a simulation-based testbed for autonomous mobility-on-demand systems. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC).* IEEE.

[59] Tomislav Rudec, Alfonzo Baumgartner, and Robert Manger. 2013. A fast work function algorithm for solving the k-server problem. *Central European Journal of Operations Research* 21, 1 (01 Jan 2013), 187–205.

[60] 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* (2014).

[61] Douglas Oliveira Santos and Eduardo Candido Xavier. 2013. Dynamic taxi and ridesharing: A framework and heuristics for the optimization problem. In *Twenty-Third International Joint Conference on Artificial Intelligence.*

[62] Douglas O Santos and Eduardo C Xavier. 2015. Taxi and ride sharing: A dynamic dial-a-ride problem with money as an incentive. *Expert Systems with Applications* 42, 19 (2015), 6728–6737.

[63] Shrawani Silwal, Md Osman Gani, and Vaskar Raychoudhury. 2019. A Survey of Taxi Ride Sharing System Architectures. In *2019 IEEE International Conference on Smart Computing (SMARTCOMP).* IEEE, 144–149.

[64] Andrea Simonetto, Julien Monteil, and Claudio Gambella. 2019. Real-time city-scale ridesharing via linear assignment problems. *Transportation Research Part C: Emerging Technologies* 101 (2019), 208–232.

[65] Kevin Spieser, Samitha Samaranayake, Wolfgang Gruel, and Emilio Frazzoli. 2016. Shared-vehicle mobility-on-demand systems: a fleet operator's guide to rebalancing empty vehicles. In *Transportation Research Board 95th Annual Meeting*.
[66] Tom Sühr, Asia J Biega, Meike Zehlike, Krishna P Gummadi, and Abhijnan Chakraborty. 2019. Two-Sided Fairness for Repeated Matchings in Two-Sided Markets: A Case Study of a Ride-Hailing Platform. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 3082–3092.
[67] M. Tang, S. Ow, W. Chen, Y. Cao, K. Lye, and Y. Pan. 2017. The Data and Science behind GrabShare Carpooling. In *2017 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*.
[68] Reza Vosooghi, Jakob Puchinger, Marija Jankovic, and Anthony Vouillon. 2019. Shared Autonomous Vehicle Simulation and Service Design. *Transportation Research Part C: Emerging Technologies* 107 (2019), 15–33.
[69] Dominic Widdows, Jacob Lucas, Muchen Tang, and Weilun Wu. 2017. GrabShare: The construction of a realtime ridesharing service. In *2017 2nd IEEE International Conference on Intelligent Transportation Engineering (ICITE)*.
[70] Dongkuan Xu and Yingjie Tian. 2015. A Comprehensive Survey of Clustering Algorithms. *Annals of Data Science*.
[71] Chak Fai Yuen, Abhishek Pratap Singh, Sagar Goyal, Sayan Ranu, and Amitabha Bagchi. 2019. Beyond Shortest Paths: Route Recommendations for Ride-sharing. In *The World Wide Web Conference*. ACM, 2258–2269.
[72] Boming Zhao, Pan Xu, Yexuan Shi, Yongxin Tong, Zimu Zhou, and Yuxiang Zeng. 2019. Preference-aware task assignment in on-demand taxi dispatching: An online stable matching approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 2245–2252.

A APPENDIX

A.1 Simulation Results in Detail

In this section we present in detail the results of our evaluation of Section 5 including, but not limited to, larger test-cases (broader NYC area), and the omitted algorithms, graphs, and tables. For every metric we report the average value out of 8 runs. The dataset was cleaned to remove requests with travel time shorter than one minute, or invalid geo-locations (e.g., outside Manhattan, Bronx, Staten Island, Brooklyn, or Queens).

Section A.2 08:00 - 09:00 – Manhattan: We begin with our small test-case: one hour (08:00 - 09:00), base number of taxis (i.e., 4276, see Section 2.2.2), limited to Manhattan. Figure 8, and Table 2 depict all the evaluated metrics, while the latter also includes the standard deviation of each value. Finally, Table 3 presents the relative difference (percentage of gain or loss) compared to MWM (first line of the table). In what follows, we will adhere to the same pattern, i.e., presenting two tables for the same evaluation, one containing the absolute values, and one presenting the relative difference compared to the algorithm in the first line of the table. We were able to run most of the algorithms in this test-case, except for WFA which we run only for {×0.5, ×0.75} the base number of taxis, and HC which is so computationally heavy, that we had to run a separate test-case of only 10 minutes (see Section A.6).

Offline algorithms (e.g., MWM, ALMA, Greedy) can be run either in a just-in-time (JiT) manner – i.e., when a request becomes critical – or in batches. The following two tables (Tables 4, and 5) evaluate the performance of each algorithm for each option. Given that our dataset has granularity of one minute, we run in batches of one, and two minutes. Moreover, due to the large number of requests, at least one request turns critical in every time-step. Thus, JiT and in batches of one minute produced the exact same results. To allow for the evaluation of every algorithm (except HC), we run the evaluation in a smaller scale, i.e., 2138 taxis ({×0.5} the base number of taxis). These tables also include the results for the WFA algorithm. Every other result presented in this paper assumes the best performing option for each of the algorithms (usually batch size of two minutes).

Finally, Figure 9 shows that our results are robust to a varying number of vehicles (2138 - 12828).

Section A.3 00:00 - 23:59 (full day) – Manhattan: We continue to show that the results are robust to a larger time-scale. As before, Figure 10, and Tables 6, and 7 depict all the evaluated metrics.

Sections A.4 08:00 - 09:00, and A.5 00:00 - 23:59 (full day) – Broader NYC Area: In the following two sections, we show that our results are robust to larger geographic areas, specifically in the broader NYC Area, including Manhattan, Bronx, Staten Island, Brooklyn, and Queens. Figure 11, and Tables 8, and 9, and Figure 12, and Tables 10, and 11 depict all the evaluated metrics, for one hour, and one day respectively.

Section A.6 08:00 - 08:10 – Manhattan: This is a limited test-case aimed to evaluate the HC algorithm, due to its high computational complexity. Figure 13, and Tables 12, and 13 depict all the evaluated metrics.

Section A.7 Dynamic Vehicle Relocation – 00:00 - 23:59 (full day) – Manhattan: In this section, we present results on the step (c) of the DRSFR problem: dynamic relocation. We fix an algorithm for steps (a), and (b) – specifically MWM – to allow for a common ground and a fair comparison, focused only on the relocation part. Figure 14, and Tables 14, and 15 depict all the evaluated metrics.

Section A.8 End-To-End Solution – 00:00 - 23:59 (full day) – Manhattan: As a final step, we evaluate end-to-end solutions, using MWM, ALMA, and Greedy to solve all three of the steps of the DRSFR problem. Figure 15, and Tables 16, and 17 present all the evaluated metrics.

A.2 08:00 - 09:00 – Manhattan

(a) Total Distance Driven (m)

(b) Elapsed Time (ns) [LOG]

(c) Time to Pair (s)

(d) Time to Pair with Taxi (s)

(e) Time to Pick-up (s)

(f) Delay (s)

(g) Cumulative Delay (s)

(h) Driver Profit ($)

(i) Number of Shared Rides

(j) Frictions (s)

Fig. 8. January 15, 2016 – 08:00 - 09:00 – Manhattan – #Taxis = 4276 (base number).

Table 2. January 15, 2016 – 08:00 - 09:00 – Manhattan – #Taxis = 4276 (base number).

	Distance Driven (m)	SD	Elapsed Time (ns)	SD	Time to Pair (s)	SD	Time to Pair with Taxi (s)	SD	Time to Pick-up (s)	SD	Delay (s)	SD	Cumulative Delay (s)	Driver Profit ($)	SD	Number of Shared Rides	SD	Frictions (s)	SD
MWM	4.76E+07	0.00E+00	1.54E+12	0.00E+00	32.05	30.90	0.00	0.00	122.78	146.36	28.56	76.94	183.39	92.42	59.75	9.54E+03	0.00	232.30	420.13
ALMA	5.67E+07	1.14E+05	8.09E+10	4.36E+09	31.95	30.99	0.00	0.00	206.93	246.41	24.79	77.61	263.67	91.65	53.22	9.61E+03	10.46	206.59	387.78
Greedy	5.77E+07	8.97E+04	4.37E+10	2.73E+09	32.16	31.02	0.00	0.00	215.82	249.18	28.19	79.89	276.17	92.05	52.55	9.55E+03	18.52	200.17	381.71
Appr	7.90E+07	0.00E+00	1.06E+12	0.00E+00	30.29	30.19	0.00	0.00	580.48	427.45	71.13	133.34	681.90	98.89	40.52	1.00E+04	0.00	203.84	315.85
PG	6.27E+07	1.05E+05	8.55E+11	2.18E+10	59.69	43.21	0.00	0.00	219.16	282.64	13.77	61.12	292.62	80.08	45.69	7.11E+03	28.92	190.24	384.91
GD	6.62E+07	0.00E+00	7.54E+12	9.01E+10	52.91	32.09	14.67	18.27	225.96	267.47	143.82	313.95	437.36	103.65	56.84	9.01E+03	0.00	166.78	348.97
Bal	5.11E+07	5.16E+04	4.60E+10	2.09E+09	32.05	30.89	0.00	0.00	163.20	156.45	46.67	120.62	241.91	94.98	30.95	9.54E+03	0.00	621.14	490.55
Har	7.61E+07	2.47E+05	4.28E+10	2.41E+09	32.05	30.89	0.00	0.00	540.38	479.35	50.84	129.03	623.27	95.18	41.59	9.54E+03	0.00	279.55	282.88
DC	1.23E+08	5.41E+06	1.79E+13	1.05E+12	32.05	30.89	0.00	0.00	0.00	10458.42	52.29	125.51	6051.60	96.94	196.20	9.54E+03	0.00	133.54	349.94
k-Taxi	5.97E+07	2.31E+05	5.23E+13	3.00E+12	32.05	30.89	0.00	0.00	288.89	372.65	47.09	120.88	368.02	94.90	62.00	9.54E+03	0.00	188.68	343.92
Single	8.51E+07	0.00E+00	8.12E+11	0.00E+00	0.00	0.00	0.02	1.04	133.36	201.19	0.00	0.00	133.38	24.88	7.30	0.00E+00	0.00	119.80	291.01

Table 3. January 15, 2016 – 08:00 - 09:00 – Manhattan – #Taxis = 4276 (base number). Each column presents the relative difference compared to the first line, i.e., the MWM (algorithm - MWM) / MWM, for each metric.

	Distance Driven (m)	SD	Elapsed Time (ns)	SD	Time to Pair (s)	SD	Time to Pair with Taxi (s)	SD	Time to Pick-up (s)	SD	Delay (s)	SD	Cumulative Delay (s)	Driver Profit ($)	SD	Number of Shared Rides	SD	Frictions (s)	SD
MWM	0.00%	–	0.00%	–	0.00%	0.00%	–	–	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	–	0.00%	0.00%
ALMA	19.15%	–	-94.75%	–	-0.29%	0.30%	–	–	68.54%	68.36%	-13.19%	0.88%	43.78%	-0.84%	-10.93%	0.72%	–	-11.07%	-7.70%
Greedy	21.24%	–	-97.16%	–	0.35%	0.40%	–	–	75.78%	70.25%	-1.30%	3.84%	50.59%	-0.41%	-12.06%	0.08%	–	-13.83%	-9.14%
Appr	65.95%	–	-30.85%	–	-5.47%	-2.28%	–	–	372.79%	192.06%	149.01%	73.30%	271.83%	6.99%	-32.18%	4.81%	–	-12.25%	-24.82%
PG	31.69%	–	-44.50%	–	86.27%	39.86%	–	–	78.51%	93.12%	-51.81%	-20.56%	59.57%	-13.35%	-23.54%	-25.48%	–	-18.10%	-8.38%
GD	39.03%	–	389.79%	–	65.11%	3.87%	–	–	84.04%	82.75%	403.50%	308.04%	138.49%	12.15%	-4.87%	-5.63%	–	-28.21%	-16.94%
Bal	7.41%	–	-97.01%	–	0.00%	0.00%	–	–	32.92%	6.90%	63.39%	56.77%	31.91%	2.76%	-48.19%	0.00%	–	167.39%	16.76%
Har	59.98%	–	-97.22%	–	0.00%	0.00%	–	–	340.13%	227.52%	78.00%	67.70%	239.87%	2.98%	-30.39%	0.00%	–	20.34%	-32.67%
DC	158.13%	–	1061.10%	–	0.00%	0.00%	–	–	-100.00%	7045.82%	83.09%	63.13%	3199.91%	4.88%	228.37%	0.00%	–	-42.51%	-16.71%
k-Taxi	25.48%	–	3293.85%	–	0.00%	0.00%	–	–	135.29%	154.62%	64.85%	57.11%	100.68%	2.68%	3.76%	0.00%	–	-18.78%	-18.14%
Single	78.81%	–	-47.28%	–	-100.00%	-100.00%	–	–	8.62%	37.46%	-100.00%	-100.00%	-27.27%	-73.08%	-87.78%	-100.00%	–	-48.43%	-30.73%

Table 4. January 15, 2016 – 08:00 - 09:00 – Manhattan – #Taxis = 2138.

Offline algorithms are run either in Just-in-Time (JiT) manner, or in batches (with batch size 1, or 2 min). Because of the density of the dataset, requests become critical every time-step, thus JiT is the same as in batches with batch size 1.

	Distance Driven (m)	SD	Elapsed Time (ms)	SD	Time to Pair (s)	SD	Time to Pair with Taxi (s)	SD	Time to Pick-up (s)	SD	Delay (s)	SD	Cumulative Delay (s)	SD	Driver Profit ($)	SD	Number of Shared Rides	SD	Frictions (s)	SD
MWM (1)	5.46E+07	0.00E+00	9.91E-10	1.45E-11	5.17	18.08	102.26	407.94	318.23	542.59	45.92	123.45	471.57	52.46	190.44	0.00	9.72E+03	0.00	29.68	12.38
MWM (2)	5.27E+07	0.00E+00	1.45E-11	5.96E-10	32.05	30.90	88.64	308.08	288.94	441.85	38.34	112.20	447.97	49.91	187.50	0.00	9.54E+03	0.00	34.32	16.06
ALMA (1)	6.30E+07	1.38E-05	4.12E+09	4.12E+09	3.97	15.71	356.07	614.16	654.41	806.25	39.48	119.40	1053.93	36.01	188.79	8.36	9.83E+03	8.36	29.43	9.91
ALMA (2)	6.18E+07	2.28E-05	6.02E-10	1.57E+09	31.91	32.05	323.09	566.04	611.59	738.26	29.84	102.38	996.43	35.59	184.46	6.88	9.62E+03	6.13	30.00	10.28
Greedy (1)	6.76E+07	2.28E-05	6.78E+09	1.68E+09	4.41	16.52	577.85	706.95	932.92	813.98	44.64	102.38	1559.82	12.06	189.90	6.13	9.82E+03	6.13	29.54	9.76
Greedy (2)	6.66E+07	1.01E-05	1.31E-10	3.75E+09	32.14	31.04	536.36	668.99	881.67	783.43	34.77	106.58	1484.94	36.99	185.72	6.13	9.55E+03	17.72	29.97	10.00
Appr (1)	8.04E+07	0.00E+00	5.41E-11	0.00E+00	27.94	30.07	852.44	1048.80	1454.91	1185.15	71.59	137.37	2406.88	36.05	198.00	0.00	1.00E+04	0.00	45.28	18.08
Appr (2)	8.05E+07	0.00E+00	5.42E-11	0.00E+00	30.29	30.19	894.67	995.86	1410.40	1145.77	69.81	128.89	2315.17	35.61	197.35	0.00	1.00E+04	0.00	29.69	10.13
PG	6.74E+07	1.17E-05	5.20E-11	1.81E+10	59.53	43.27	297.99	764.61	664.30	1053.38	19.90	333.21	1041.73	39.97	161.90	31.39	7.12E+03	0.00	29.56	8.29
GD	6.92E+07	0.00E+00	7.11E-12	3.28E+11	52.91	32.09	358.38	801.79	626.96	945.13	153.49	120.74	1191.73	54.47	210.00	0.00	9.01E+03	0.00	30.21	9.70
Bal (1)	6.22E+07	9.71E-04	1.35E+10	4.56E+09	5.17	18.08	403.38	535.51	725.18	637.49	55.53	134.36	1189.26	39.97	192.96	0.00	9.72E+03	0.00	30.23	11.05
Bal (2)	5.99E+07	1.38E-05	3.42E-10	7.09E+09	32.05	30.89	336.45	466.17	635.90	569.04	46.20	120.74	1050.59	41.81	189.53	0.00	9.54E+03	0.00	31.76	12.20
Har (1)	8.10E+07	2.36E-05	1.37E-10	4.70E+09	5.17	18.08	946.56	1058.46	1555.42	1190.40	61.98	146.65	2569.12	49.97	194.22	0.00	9.72E+03	0.00	29.45	9.67
Har (2)	7.96E+07	2.87E-05	3.36E-10	6.90E+09	32.05	30.89	906.49	1024.29	1501.74	1153.03	51.28	130.04	2491.55	51.20	190.37	0.00	9.54E+03	0.00	29.65	9.65
DC (1)	1.41E+08	1.90E-06	1.16E-13	5.48E-11	5.17	30.89	0.00	0.00	7660.58	10890.62	62.70	14.271	7728.45	280.85	192.45	0.00	9.72E+03	0.00	85.76	269.59
DC (2)	1.38E+08	1.71E-06	8.99E-12	3.74E-11	5.17	30.89	0.00	0.00	7290.07	10196.66	51.77	124.76	7373.89	272.89	188.37	0.00	9.54E+03	0.00	93.37	287.47
k-Taxi (1)	7.10E+07	3.00E-05	3.41E-12	2.12E-11	5.17	18.08	646.48	528.54	1099.19	702.35	58.32	141.50	1809.15	46.97	193.48	0.00	9.72E+03	0.00	29.33	9.74
k-Taxi (2)	6.92E+07	2.04E-05	9.24E-12	4.05E-11	32.05	30.89	586.22	491.05	1020.87	670.76	48.51	126.93	1687.65	48.16	189.90	0.00	9.54E+03	0.00	30.14	10.10
WFA (1)	8.48E+07	3.25E-05	1.28E-14	6.77E-12	5.17	18.08	0.00	0.00	39066.93	48847.98	64.04	145.43	31035.75	597.69	194.69	0.00	9.72E+03	0.00	63.79	254.48
WFA (2)	8.09E+07	0.00E+00	9.69E-13	0.00E+00	32.05	30.90	0.00	0.00	28964.94	39066.93	51.85	127.05	29048.84	593.79	190.49	0.00	9.54E+03	0.00	69.68	264.10
Single	8.63E+07	0.00E+00	2.02E-12	0.00E+00	0.00	0.00	0.00	0.00	843.80	1444.94	0.00	0.00	1544.30	6.90	49.72	0.00	0.00E+00	0.00	29.41	6.11
Random	1.14E+08	2.56E-05	6.26E+09	1.42E+09	3.61	15.05	1963.08	1457.54	2908.86	1580.37	161.07	328.92	5031.62	47.27	222.47	6.54	9.88E+03	9.37	29.56	9.37

Table 5. January 15, 2016 – 08:00 - 09:00 – Manhattan – #Taxis = 2138.

Offline algorithms are run either in Just-in-Time (JiT) manner, or in batches (with batch size 1, or 2 min). Because of the density of the dataset, requests become critical every time-step, thus JiT is the same as in batches with batch size 1.

Each column presents the relative difference compared to the first line, i.e., MWM of batch size one (algorithm - MWM(1)) / MWM(1), for each metric.

	Distance Driven (m)	SD	Elapsed Time (ms)	SD	Time to Pair (s)	SD	Time to Pair with Taxi (s)	SD	Time to Pick-up (s)	SD	Delay (s)	SD	Cumulative Delay (s)	SD	Driver Profit ($)	SD	Number of Shared Rides	SD	Frictions (s)	SD
MWM (1)	0.00%	–	0.00%	–	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
MWM (2)	-3.44%	–	46.61%	–	520.26%	70.89%	-13.32%	-24.48%	-9.20%	-18.57%	-16.49%	-9.12%	-5.00%	-4.86%	-1.54%	-4.86%	-1.82%	15.65%	29.68%	29.68%
ALMA (1)	15.59%	–	-60.03%	–	-23.20%	-13.08%	248.20%	50.55%	105.64%	48.59%	-14.01%	-3.29%	123.49%	-31.36%	-0.87%	-31.36%	1.36%	-0.84%	29.43%	-20.00%
ALMA (2)	13.27%	–	-39.29%	–	517.65%	71.03%	215.95%	38.76%	92.19%	39.75%	-35.02%	-17.07%	111.30%	-32.16%	-3.04%	-32.16%	-1.03%	1.08%	30.00%	-16.96%
Greedy (1)	23.91%	–	-93.16%	–	-14.58%	-8.61%	465.07%	73.30%	193.16%	50.02%	-2.78%	-1.94%	230.77%	-30.70%	-0.24%	-30.70%	1.01%	-0.47%	29.54%	-21.15%
Greedy (2)	21.98%	–	-86.81%	–	522.14%	71.70%	424.50%	63.99%	177.06%	44.39%	-24.29%	-13.67%	214.89%	-29.48%	-2.48%	-29.48%	-1.72%	0.99%	29.97%	-19.22%
Appr (1)	47.32%	–	445.49%	–	440.80%	66.34%	733.59%	157.10%	357.19%	118.42%	55.92%	11.28%	410.39%	-31.28%	3.97%	-31.28%	2.92%	52.57%	45.28%	46.03%
Appr (2)	47.52%	–	447.16%	–	486.39%	66.99%	686.88%	144.12%	343.20%	111.17%	52.04%	4.40%	390.95%	-32.13%	3.63%	-32.13%	2.90%	0.02%	29.69%	-18.16%
PG	23.56%	–	7071.20%	–	1052.23%	139.31%	191.40%	87.43%	108.75%	94.14%	-56.65%	-12.87%	120.90%	-23.81%	-14.99%	-23.81%	-26.70%	-0.41%	29.56%	-33.08%
GD	26.84%	–	424.36%	–	924.12%	77.50%	250.45%	96.55%	97.01%	74.19%	234.27%	169.91%	152.19%	3.83%	1.33%	3.83%	-7.35%	1.79%	30.21%	-21.63%
Bal (1)	14.02%	–	-46.42%	–	0.00%	0.00%	294.47%	31.27%	127.88%	17.49%	20.94%	8.83%	152.19%	-21.13%	1.98%	-21.13%	0.00%	1.87%	30.23%	-10.75%
Bal (2)	9.88%	–	-65.49%	–	520.26%	70.88%	229.01%	14.28%	99.83%	4.87%	0.61%	-2.20%	122.78%	-20.31%	-0.48%	-20.31%	-1.82%	7.00%	31.76%	-1.50%
Har (1)	48.42%	–	-86.16%	–	0.00%	0.00%	825.64%	159.47%	388.77%	119.39%	34.98%	18.79%	444.80%	-4.75%	1.05%	-4.75%	0.00%	-0.76%	29.45%	-21.90%
Har (2)	45.93%	–	-66.14%	–	520.26%	70.88%	786.45%	151.09%	371.90%	112.50%	11.68%	5.34%	428.35%	-2.41%	-0.04%	-2.41%	-1.82%	-0.08%	29.65%	-22.10%
DC (1)	159.34%	–	11636.41%	–	-100.00%	-100.00%	-100.00%	-100.00%	2307.25%	1907.15%	36.55%	15.59%	1538.87%	435.35%	1.65%	435.35%	0.00%	188.95%	85.76	2076.91%
DC (2)	152.49%	–	8969.76%	–	520.26%	70.88%	-100.00%	-100.00%	2190.83%	1779.25%	1.06%	12.75%	1463.68%	210.17%	-1.09%	210.17%	-1.82%	214.60%	93.37	2221.34%
k-Taxi (1)	30.13%	–	3339.41%	–	0.00%	0.00%	532.19%	29.56%	245.41%	29.44%	27.00%	14.61%	283.64%	-10.47%	1.60%	-10.47%	0.00%	-1.18%	29.33%	-21.38%
k-Taxi (2)	26.82%	–	4258.51%	–	520.26%	70.88%	473.26%	20.37%	220.80%	23.62%	5.65%	2.81%	257.88%	-8.20%	-0.28%	-8.20%	-1.82%	1.54%	30.14%	-18.41%
WFA (1)	55.42%	–	129083.20%	–	1052.23%	-100.00%	-100.00%	-100.00%	9630.90%	7428.32%	39.48%	17.80%	6481.32%	1039.29%	2.23%	1039.29%	0.00%	114.92%	63.79	1954.90%
WFA (2)	48.26%	–	97654.76%	–	520.26%	70.89%	-100.00%	-100.00%	9001.91%	7100.07%	12.93%	2.91%	6059.99%	1031.86%	0.03%	1031.86%	-1.82%	134.77%	69.68	2032.61%
Single	58.20%	–	1935.09%	–	-100.00%	-100.00%	585.01%	222.97%	166.30%	166.30%	-100.00%	-100.00%	227.48%	-86.85%	-73.89%	-86.85%	-100.00%	-0.91%	29.41	-50.64%
Random	108.44%	–	-93.69%	–	-30.10%	-16.76%	1819.69%	257.30%	812.51%	191.26%	250.78%	166.43%	966.99%	-9.89%	16.82%	-9.89%	1.60%	-0.40%	29.56	-24.38%

(a) Total Distance Driven (m)

(b) Elapsed Time (ns) [LOG]

(c) Time to Pair (s)

(d) Time to Pair with Taxi (s)

(e) Time to Pick-up (s)

(f) Delay (s)

(g) Cumulative Delay (s)

(h) Driver Profit ($)

(i) Number of Shared Rides

(j) Frictions (s)

Fig. 9. January 15, 2016 – 08:00 - 09:00 – Manhattan – Varying #Taxis = {2138, 3207, 4276, 8552, 12828}.

A.3 00:00 - 23:59 (full day) – Manhattan

(a) Total Distance Driven (m) (b) Elapsed Time (ns) [LOG] (c) Time to Pair (s) (d) Time to Pair with Taxi (s)

(e) Time to Pick-up (s) (f) Delay (s) (g) Cumulative Delay (s) (h) Driver Profit ($)

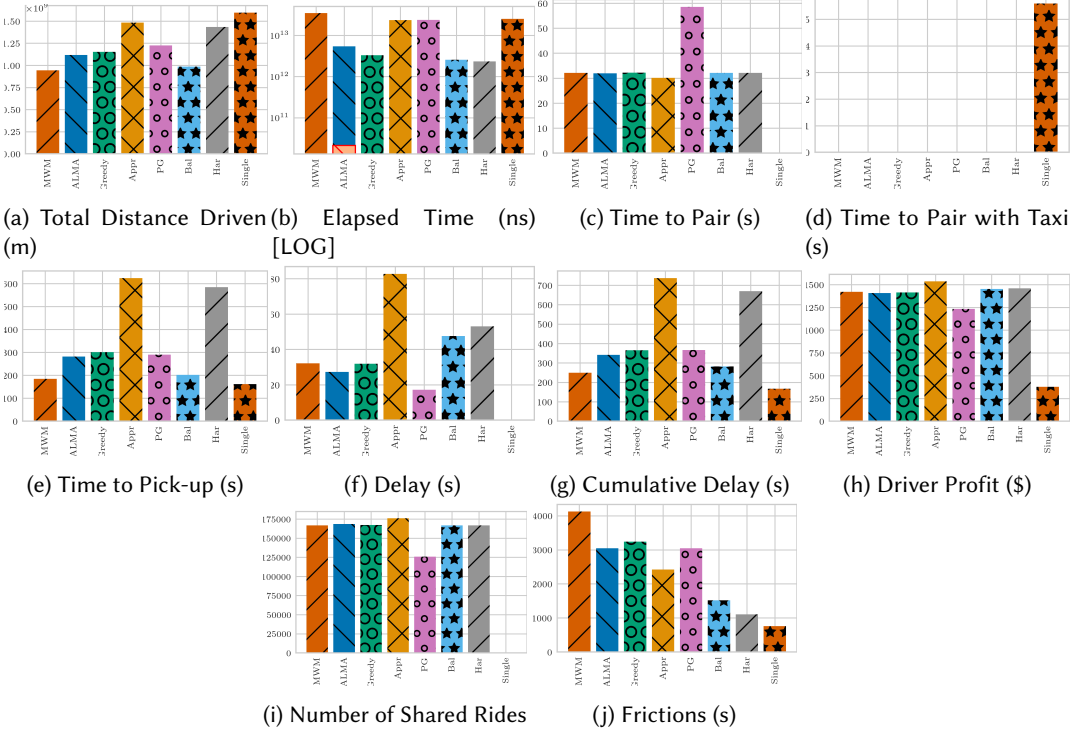(i) Number of Shared Rides (j) Frictions (s)

Fig. 10. January 15, 2016 – 00:00 - 23:59 (full day) – Manhattan – #Taxis = 5081 (base number).

Table 6. January 15, 2016 – 00:00 - 23:59 (full day) – Manhattan – #Taxis = 5081 (base number).

	Distance Driven (m)	SD	Elapsed Time (ns)	SD	Time to Pair (s)	SD	Time to Pair with Taxi (s)	SD	Time to Pick-up (s)	SD	Delay (s)	SD	Cumulative Delay (s)	Driver Profit ($)	SD	Number of Shared Rides	SD	Frictions (s)	SD
MWM	9.45E+08	0.00E+00	3.48E+13	0.00E+00	32.10	30.84	0.00	0.00	184.55	274.34	32.14	87.69	248.79	1420.37	895.19	1.67E+05	0.00	4127.47	10597.84
ALMA	1.12E+09	2.73E+05	5.42E+12	3.31E+11	31.98	31.01	0.00	0.00	281.70	405.36	27.32	86.78	341.00	1406.70	736.46	1.68E+05	29.39	3047.45	7264.56
Greedy	1.15E+09	5.93E+05	3.30E+12	3.86E+11	32.18	31.05	0.00	0.00	301.04	407.70	31.93	90.56	365.15	1414.66	719.44	1.67E+05	26.29	3242.70	8167.00
Appr	1.48E+09	0.00E+00	2.37E+13	0.00E+00	30.14	30.18	0.00	0.00	624.13	473.31	82.75	156.13	737.02	1536.97	478.35	1.76E+05	0.00	2421.75	4505.68
PG	1.22E+09	4.64E+05	2.39E+13	1.50E+12	58.58	42.99	0.00	0.00	290.18	431.80	17.16	84.29	365.91	1234.17	603.66	1.26E+05	95.74	3044.98	8561.98
Bal	9.85E+08	2.42E+05	2.56E+12	1.39E+11	32.10	30.84	0.00	0.00	201.85	225.85	47.51	126.16	281.47	1452.66	107.31	1.67E+05	0.00	1516.33	221.62
Har	1.43E+09	1.46E+06	2.34E+12	3.85E+11	32.10	30.84	0.00	0.00	584.05	533.66	53.08	132.72	669.23	1458.51	187.05	1.67E+05	0.00	1106.16	246.76
Single	1.60E+09	0.00E+00	2.54E+13	0.00E+00	0.00	0.00	5.59	99.05	161.77	355.37	0.00	0.00	167.36	376.90	116.72	0.00E+00	0.00	756.43	1216.88

Table 7. January 15, 2016 – 00:00 - 23:59 (full day) – Manhattan – #Taxis = 5081 (base number). Each column presents the relative difference compared to the first line, i.e., the MWM (algorithm - MWM) / MWM, for each metric.

	Distance Driven (ns)	SD	Elapsed Time (ns)	SD	Time to Pair (s)	SD	Time to Pair with Taxi (s)	SD	Time to Pick-up (s)	SD	Delay (s)	SD	Cumulative Delay (s)	Driver Profit ($)	SD	Number of Shared Rides	SD	Frictions (s)	SD
MWM	0.00%	–	0.00%	–	0.00%	0.00%	–	–	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	–	0.00%	0.00%
ALMA	18.29%	–	-84.43%	–	-0.39%	0.56%	–	–	52.64%	47.76%	-15.00%	-1.04%	37.06%	-0.96%	-17.73%	1.08%	–	-26.17%	-31.45%
Greedy	21.92%	–	-90.52%	–	0.23%	0.68%	–	–	63.12%	48.61%	-0.65%	3.27%	46.77%	-0.40%	-19.63%	0.41%	–	-21.44%	-22.94%
Appr	57.08%	–	-32.07%	–	-6.12%	-2.16%	–	–	238.19%	72.53%	157.50%	78.05%	196.24%	8.21%	-46.56%	5.69%	–	-41.33%	-57.48%
PG	29.57%	–	-31.48%	–	82.46%	39.38%	–	–	57.23%	57.40%	-46.61%	-3.87%	47.08%	-13.11%	-32.57%	-24.49%	–	-26.23%	-19.21%
Bal	4.24%	–	-92.64%	–	0.00%	0.00%	–	–	9.37%	-17.67%	47.85%	43.87%	13.13%	2.27%	-88.01%	0.00%	–	-63.26%	-97.91%
Har	51.67%	–	-93.28%	–	0.00%	0.00%	–	–	216.47%	94.53%	65.19%	51.36%	168.99%	2.68%	-79.10%	0.00%	–	-73.20%	-97.67%
Single	69.00%	–	-27.23%	–	-100.00%	-100.00%	–	–	-12.35%	29.54%	-100.00%	-100.00%	-32.73%	-73.46%	-86.96%	-100.00%	–	-81.67%	-88.52%

A.4   08:00 - 09:00 – Broader NYC Area (Manhattan, Bronx, Staten Island, Brooklyn, Queens)

(a) Total Distance Driven (m)

(b) Elapsed Time (ns) [LOG]

(c) Time to Pair (s)

(d) Time to Pair with Taxi (s)

(e) Time to Pick-up (s)

(f) Delay (s)

(g) Cumulative Delay (s)

(h) Driver Profit ($)

(i) Number of Shared Rides
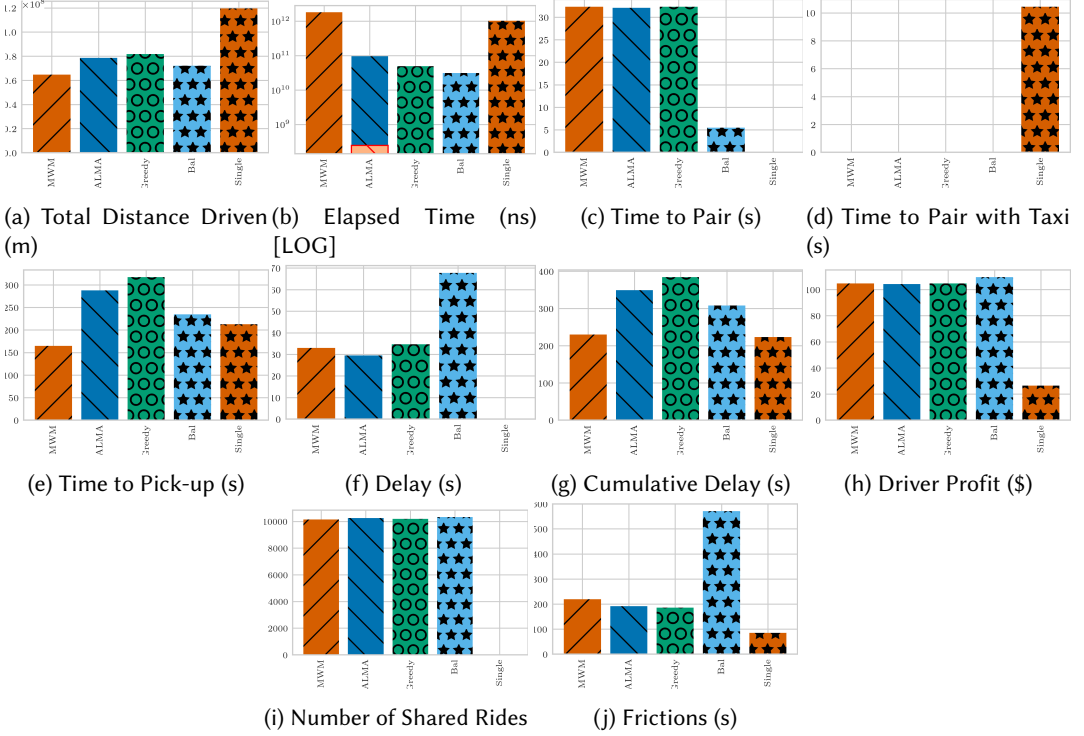
(j) Frictions (s)

Fig. 11. January 15, 2016 – 08:00 - 09:00 – Broader NYC Area – #Taxis = 4972 (base number).

Table 8. January 15, 2016 – 08:00 - 09:00 – Broader NYC Area – #Taxis = 4972 (base number).

	Distance Driven (m)	SD	Elapsed Time (ns)	SD	Time to Pair (s)	SD	Time to Pair with Taxi (s)	SD	Time to Pick-up (s)	SD	Delay (s)	SD	Cumulative Delay (s)	Driver Profit ($)	SD	Number of Shared Rides	SD	Frictions (s)	SD
MWM	6.48E+07	0.00E+00	1.81E+12	0.00E+00	32.34	31.34	0.00	0.00	164.59	401.29	33.01	104.44	229.94	104.67	81.54	1.02E+04	0.00	219.19	415.07
ALMA	7.86E+07	2.48E+05	9.61E+10	1.02E+10	32.09	31.32	0.00	0.00	287.93	646.99	29.55	167.88	349.58	104.13	72.74	1.03E+04	5.88	191.41	379.85
Greedy	8.18E+07	2.96E+05	4.88E+10	7.00E+09	32.32	31.40	0.00	0.00	317.48	720.24	34.73	170.53	384.53	104.68	69.88	1.02E+04	12.28	185.35	374.04
Bal	7.22E+07	1.15E+05	3.09E+10	6.01E+09	5.49	18.97	0.00	0.00	234.85	428.34	67.79	219.01	308.14	109.57	65.24	1.03E+04	0.00	571.11	516.22
Single	1.20E+08	0.00E+00	1.03E+12	0.00E+00	0.00	0.00	10.44	83.19	212.61	577.74	0.00	0.00	223.06	26.37	9.21	0.00E+00	0.00	85.07	211.52

Table 9. January 15, 2016 – 08:00 - 09:00 – Broader NYC Area – #Taxis = 4972 (base number). Each column presents the relative difference compared to the first line, i.e., the MWM (algorithm - MWM) / MWM, for each metric.

	Distance Driven (m)	SD	Elapsed Time (ns)	SD	Time to Pair (s)	SD	Time to Pair with Taxi (s)	SD	Time to Pick-up (s)	SD	Delay (s)	SD	Cumulative Delay (s)	Driver Profit ($)	SD	Number of Shared Rides	SD	Frictions (s)	SD
MWM	0.00%	–	0.00%	–	0.00%	0.00%	–	–	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	–	0.00%	0.00%
ALMA	21.37%	–	-94.68%	–	-0.75%	-0.08%	–	–	74.94%	61.23%	-10.48%	60.74%	52.03%	-0.52%	-10.79%	1.02%	–	-12.67%	-8.49%
Greedy	26.34%	–	-97.30%	–	-0.04%	0.20%	–	–	92.90%	79.48%	5.20%	63.27%	67.24%	0.01%	-14.31%	0.42%	–	-15.44%	-9.88%
Bal	11.44%	–	-98.29%	–	-83.01%	-39.48%	–	–	42.69%	6.74%	105.35%	109.70%	34.01%	4.68%	-20.00%	1.77%	–	160.56%	24.37%
Single	84.67%	–	-42.94%	–	-100.00%	-100.00%	–	–	29.18%	43.97%	-100.00%	-100.00%	-2.99%	-74.80%	-88.71%	-100.00%	–	-61.19%	-49.04%

A.5 00:00 - 23:59 (full day) – Broader NYC Area (Manhattan, Bronx, Staten Island, Brooklyn, Queens)

(a) Total Distance Driven (m)

(b) Elapsed Time (ns) [LOG]

(c) Time to Pair (s)

(d) Time to Pair with Taxi (s)

(e) Time to Pick-up (s)

(f) Delay (s)

(g) Cumulative Delay (s)

(h) Driver Profit ($)

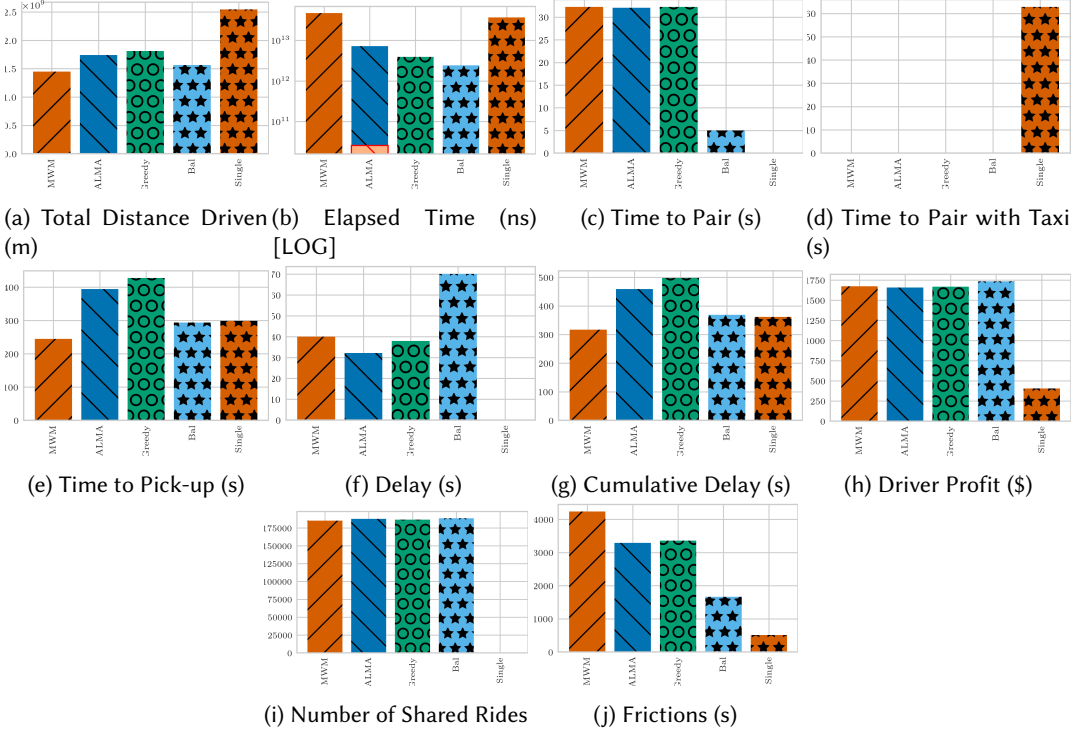(i) Number of Shared Rides

(j) Frictions (s)

Fig. 12. January 15, 2016 – 00:00 - 23:59 (full day) – Broader NYC Area – #Taxis = 6533 (base number).

Table 10. January 15, 2016 – 00:00 - 23:59 (full day) – Broader NYC Area – #Taxis = 6533 (base number).

	Distance Driven (m)	SD	Elapsed Time (ns)	SD	Time to Pair (s)	SD	Time to Pair with Taxi (s)	SD	Time to Pick-up (s)	SD	Delay (s)	SD	Cumulative Delay (s)	Driver Profit ($)	SD	Number of Shared Rides	SD	Frictions (s)	SD
MWM	1.45E+09	0.00E+00	4.71E+13	0.00E+00	32.26	31.24	0.00	0.00	244.90	433.03	40.01	131.68	317.17	1675.21	949.14	1.85E+05	0.00	4238.58	10671.94
ALMA	1.74E+09	5.28E+05	7.20E+12	1.37E+11	32.09	31.44	0.00	0.00	394.30	701.06	32.17	133.83	458.57	1659.01	741.69	1.88E+05	22.08	3286.78	8808.76
Greedy	1.81E+09	2.36E+06	3.92E+12	3.31E+11	32.28	31.49	0.00	0.00	427.74	750.89	37.92	137.07	497.93	1667.74	672.62	1.87E+05	16.76	3357.00	9464.19
Bal	1.57E+09	2.60E+05	2.42E+12	1.60E+11	4.97	18.17	0.00	0.00	293.91	457.48	70.17	216.02	369.04	1736.02	153.93	1.89E+05	0.00	1666.22	570.15
Single	2.55E+09	0.00E+00	3.70E+13	0.00E+00	0.00	0.00	62.85	753.27	298.52	1147.60	0.00	0.00	361.37	405.82	80.73	0.00E+00	0.00	512.34	437.98

Table 11. January 15, 2016 – 00:00 - 23:59 (full day) – Broader NYC Area – #Taxis = 6533 (base number). Each column presents the relative difference compared to the first line, i.e., the MWM (algorithm - MWM) / MWM, for each metric.

	Distance Driven (m)	SD	Elapsed Time (ns)	SD	Time to Pair (s)	SD	Time to Pair with Taxi (s)	SD	Time to Pick-up (s)	SD	Delay (s)	SD	Cumulative Delay (s)	Driver Profit ($)	SD	Number of Shared Rides	SD	Frictions (s)	SD
MWM	0.00%	–	0.00%	–	0.00%	0.00%	–	–	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	–	0.00%	0.00%
ALMA	20.13%	–	-84.69%	–	-0.53%	0.64%	–	–	61.01%	61.90%	-19.59%	1.63%	44.58%	-0.97%	-21.86%	1.34%	–	-22.46%	-17.46%
Greedy	25.04%	–	-91.66%	–	0.05%	0.80%	–	–	74.66%	73.41%	-5.22%	4.09%	56.99%	-0.45%	-29.13%	0.68%	–	-20.80%	-11.32%
Bal	8.15%	–	-94.85%	–	-84.61%	-41.85%	–	–	20.01%	5.65%	75.38%	64.05%	16.35%	3.63%	-83.78%	1.83%	–	-60.69%	-94.66%
Single	75.86%	–	-21.44%	–	-100.00%	-100.00%	–	–	21.90%	165.02%	-100.00%	-100.00%	13.94%	-75.78%	-91.49%	-100.00%	–	-87.91%	-95.90%

A.6   08:00 - 08:10 – Manhattan

(a) Total Distance Driven (m)

(b) Elapsed Time (ns) [LOG]

(c) Time to Pair (s)

(d) Time to Pair with Taxi (s)

(e) Time to Pick-up (s)

(f) Delay (s)

(g) Cumulative Delay (s)

(h) Driver Profit ($)

(i) Number of Shared Rides
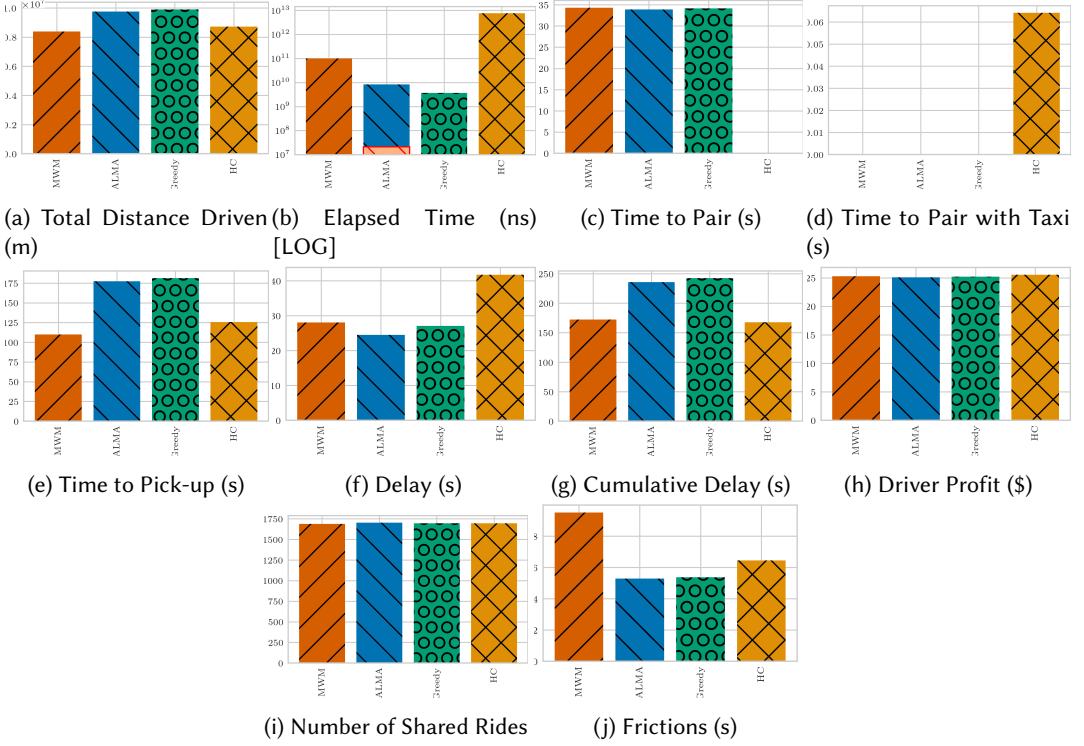
(j) Frictions (s)

Fig. 13. January 15, 2016 – 08:00 - 08:10 – Manhattan – #Taxis = 2779 (base number).

Table 12. January 15, 2016 – 08:00 - 09:00 – Manhattan – #Taxis = 2779 (base number).

	Distance Driven (m)	SD	Elapsed Time (ns)	SD	Time to Pair (s)	SD	Time to Pair with Taxi (s)	SD	Time to Pick-up (s)	SD	Delay (s)	SD	Cumulative Delay (s)	Driver Profit ($)	SD	Number of Shared Rides	SD	Frictions (s)	SD
MWM	8.38E+06	0.00E+00	9.92E+10	0.00E+00	34.32	30.93	0.00	0.00	109.83	125.56	28.08	80.29	172.23	25.30	29.59	1.69E+03	0.00	9.51	31.75
ALMA	9.76E+06	6.10E+04	8.28E+09	1.93E+09	33.88	30.71	0.00	0.00	177.21	216.99	24.49	75.40	235.58	25.09	26.38	1.70E+03	9.76	5.29	22.26
Greedy	9.91E+06	1.06E+04	3.66E+09	7.61E+08	34.16	30.88	0.00	0.00	181.43	216.19	27.05	74.92	242.64	25.19	26.14	1.70E+03	7.55	5.37	21.50
HC	8.72E+06	0.00E+00	7.52E+12	0.00E+00	0.13	4.02	0.06	2.78	125.69	155.09	41.77	106.45	167.65	25.56	29.61	1.70E+03	0.00	6.45	27.19

Table 13. January 15, 2016 – 08:00 - 08:10 – Manhattan – #Taxis = 2779 (base number). Each column presents the relative difference compared to the first line, i.e., the MWM (algorithm - MWM) / MWM, for each metric.

	Distance Driven (m)	SD	Elapsed Time (ns)	SD	Time to Pair (s)	SD	Time to Pair with Taxi (s)	SD	Time to Pick-up (s)	SD	Delay (s)	SD	Cumulative Delay (s)	Driver Profit ($)	SD	Number of Shared Rides	SD	Frictions (s)	SD
MWM	0.00%	–	0.00%	–	0.00%	0.00%	–	–	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	–	0.00%	0.00%
ALMA	16.40%	–	-91.65%	–	-1.28%	-0.70%	–	–	61.34%	72.81%	-12.80%	-6.09%	36.78%	-0.83%	-10.87%	0.95%	–	-44.37%	-29.90%
Greedy	18.17%	–	-96.31%	–	-0.45%	-0.15%	–	–	65.18%	72.18%	-3.67%	-6.68%	40.88%	-0.43%	-11.65%	0.44%	–	-43.57%	-32.29%
HC	3.98%	–	7474.61%	–	-99.61%	-87.01%	–	–	14.44%	23.52%	48.74%	32.59%	-2.66%	1.02%	0.05%	0.47%	–	-32.16%	-14.37%

A.7 Dynamic Vehicle Relocation – 00:00 - 23:59 (full day) – Manhattan

(a) Total Distance Driven (m)

(b) Time to Pair (s)

(c) Time to Pick-up (s)

(d) Delay (s)

(e) Cumulative Delay (s)

(f) Driver Profit ($)

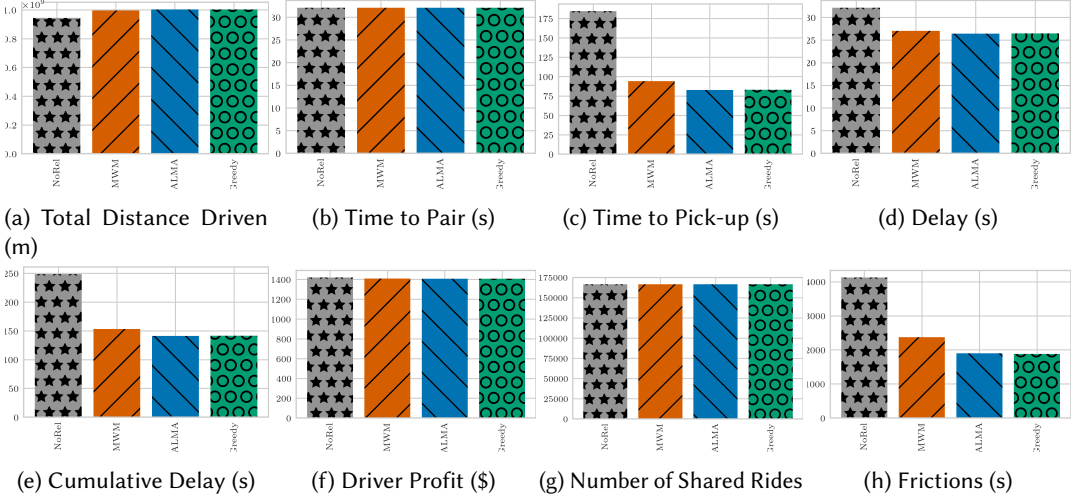(g) Number of Shared Rides

(h) Frictions (s)

Fig. 14. Dynamic Vehicle Relocation – January 15, 2016 – 00:00 - 23:59 (full day) – Manhattan – #Taxis = 5081 (base number).

Table 14. Dynamic Vehicle Relocation – January 15, 2016 – 00:00 - 23:59 (full day) – Manhattan – #Taxis = 5081 (base number).

	Distance Driven (m)	SD	Elapsed Time (ns)	SD	Time to Pair (s)	SD	Time to Pair with Taxi (s)	SD	Time to Pick-up (s)	SD	Delay (s)	SD	Cumulative Delay (s)	Driver Profit ($)	SD	Number of Shared Rides	SD	Frictions (s)	SD
NoRel	9.45E+08	–	–	–	32.10	30.84	0.00	0.00	184.55	274.34	32.14	87.69	248.79	1420.37	895.19	1.67E+05	0.00	4127.47	10597.84
MWM	9.97E+08	–	–	–	32.10	30.84	0.00	0.00	94.21	129.01	27.01	70.81	153.33	1408.73	674.38	1.67E+05	0.00	2372.57	5366.92
ALMA	1.00E+09	–	–	–	32.10	30.84	0.00	0.00	82.72	114.61	26.42	69.31	141.24	1407.37	447.49	1.67E+05	0.00	1898.47	2739.70
Greedy	1.00E+09	–	–	–	32.10	30.84	0.00	0.00	82.99	114.65	26.44	69.29	141.53	1407.41	440.30	1.67E+05	0.00	1880.72	2962.94

Table 15. Dynamic Vehicle Relocation – January 15, 2016 – 00:00 - 23:59 (full day) – Manhattan – #Taxis = 5081 (base number). Each column presents the relative difference compared to not using relocation (algorithm - NoRel) / NoRel, for each metric.

	Distance Driven (m)	SD	Elapsed Time (ns)	SD	Time to Pair (s)	SD	Time to Pair with Taxi (s)	SD	Time to Pick-up (s)	SD	Delay (s)	SD	Cumulative Delay (s)	Driver Profit ($)	SD	Number of Shared Rides	SD	Frictions (s)	SD
NoRel	0.00%	–	–	–	0.00%	0.00%	–	–	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	–	0.00%	0.00%
MWM	5.48%	–	–	–	0.00%	0.00%	–	–	-48.95%	-52.97%	-15.95%	-19.25%	-38.37%	-0.82%	-24.67%	0.00%	–	-42.52%	-49.36%
ALMA	6.25%	–	–	–	0.00%	0.00%	–	–	-55.18%	-58.22%	-17.79%	-20.96%	-43.23%	-0.92%	-50.01%	0.00%	–	-54.00%	-74.15%
Greedy	6.24%	–	–	–	0.00%	0.00%	–	–	-55.03%	-58.21%	-17.73%	-20.98%	-43.11%	-0.91%	-50.81%	0.00%	–	-54.43%	-72.04%

A.8 End-To-End Solution – 00:00 - 23:59 (full day) – Manhattan

(a) Total Distance Driven (m)

(b) Time to Pair (s)

(c) Time to Pick-up (s)

(d) Delay (s)

(e) Cumulative Delay (s)

(f) Driver Profit ($)
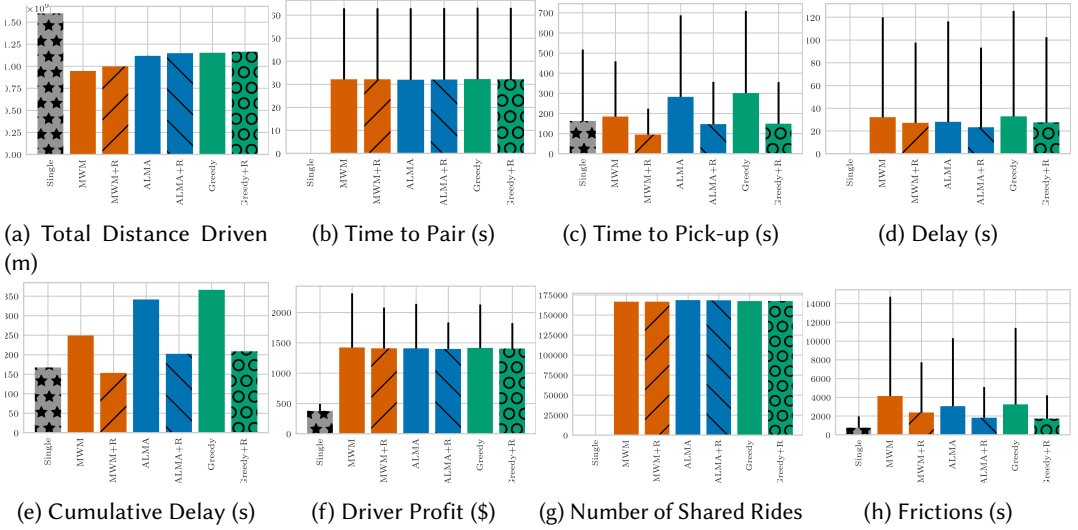
(g) Number of Shared Rides

(h) Frictions (s)

Fig. 15. End-To-End Solution – January 15, 2016 – 00:00 - 23:59 (full day) – Manhattan – #Taxis = 5081 (base number)

Table 16. End-To-End Solution – January 15, 2016 – 00:00 - 23:59 (full day) – Manhattan – #Taxis = 5081 (base number).

	Distance Driven (m)	SD	Elapsed Time (ns)	SD	Time to Pair (s)	SD	Time to Pair with Taxi (s)	SD	Time to Pick-up (s)	SD	Delay (s)	SD	Cumulative Delay (s)	Driver Profit ($)	SD	Number of Shared Rides	SD	Frictions (s)	SD
Single	1.60E+09	–	–	–	0.00	0.00	5.59	99.05	161.77	355.37	0.00	0.00	167.36	376.90	116.72	0.00E+00	–	756.43	1216.88
MWM	9.45E+08	–	–	–	32.10	30.84	0.00	0.00	184.55	274.34	32.14	87.69	248.79	1420.37	895.19	1.67E+05	–	4127.47	10597.84
MWM+R	9.97E+08	–	–	–	32.10	30.84	0.00	0.00	94.21	129.01	27.01	70.81	153.33	1408.73	674.38	1.67E+05	–	2372.57	5366.92
ALMA	1.12E+09	–	–	–	31.98	31.01	0.00	0.00	281.70	405.36	28.02	88.30	341.70	1406.70	736.46	1.68E+05	–	3047.45	7264.56
ALMA+R	1.15E+09	–	–	–	32.02	31.04	0.00	0.00	146.68	210.60	23.22	70.06	201.92	1397.14	440.45	1.68E+05	–	1815.54	3295.10
Greedy	1.15E+09	–	–	–	32.18	31.05	0.00	0.00	301.04	407.70	32.85	92.71	366.07	1414.66	719.44	1.67E+05	–	3242.70	8167.00
Greedy+R	1.16E+09	–	–	–	32.17	31.02	0.00	0.00	148.88	207.00	27.46	75.18	208.52	1404.95	422.37	1.67E+05	–	1726.67	2487.20

Table 17. End-To-End Solution – January 15, 2016 – 00:00 - 23:59 (full day) – Manhattan – #Taxis = 5081 (base number). Each column presents the relative difference compared to the Singe Ride baseline (algorithm - Singe) / Singe, for each metric.

	Distance Driven (m)	SD	Elapsed Time (ns)	SD	Time to Pair (s)	SD	Time to Pair with Taxi (s)	SD	Time to Pick-up (s)	SD	Delay (s)	SD	Cumulative Delay (s)	Driver Profit ($)	SD	Number of Shared Rides	SD	Frictions (s)	SD
Single	0.00%	–	–	–	–	–	0.00%	0.00%	0.00%	0.00%	–	–	0.00%	0.00%	0.00%	–	–	0.00%	0.00%
MWM	-40.83%	–	–	–	–	–	-100.00%	-100.00%	14.09%	-22.80%	–	–	48.66%	276.85%	666.95%	–	–	445.65%	770.90%
MWM+R	-37.59%	–	–	–	–	–	-100.00%	-100.00%	-41.76%	-63.70%	–	–	-8.39%	273.76%	477.77%	–	–	213.66%	341.04%
ALMA	-30.01%	–	–	–	–	–	-100.00%	-100.00%	74.14%	14.07%	–	–	104.17%	273.23%	530.96%	–	–	302.88%	496.98%
ALMA+R	-28.17%	–	–	–	–	–	-100.00%	-100.00%	-9.33%	-40.74%	–	–	20.65%	270.69%	277.36%	–	–	140.02%	170.78%
Greedy	-27.86%	–	–	–	–	–	-100.00%	-100.00%	86.10%	14.73%	–	–	118.73%	275.34%	516.38%	–	–	328.69%	571.14%
Greedy+R	-27.17%	–	–	–	–	–	-100.00%	-100.00%	-7.97%	-41.75%	–	–	24.59%	272.76%	261.87%	–	–	128.27%	104.39%