Heuristics for Improving Trip-Vehicle Fitness in On-demand Ride-Sharing Systems

Sevket Gökay[1,2], Andreas Heuvels[1,2] and Karl-Heinz Krempels[1,2]

[1]*Informatik 5 (Information Systems), RWTH Aachen University, Aachen, Germany*

[2]*CSCW Mobility, Fraunhofer FIT, Aachen, Germany*

Keywords: Demand-Responsive Transport, Dial-a-Ride Problem with Time Windows, Ride-Sharing.

Abstract: On-demand ride-sharing services are emerging alternatives to classical transport modes. Combined with self-driving vehicles, this movement has potential to shape the future of our mobility. To make full use of the potential, such services need to be scalable with growing demand. Assigning real-time trip requests to vehicles such that the driving costs are minimized is computationally expensive, but has to be done fast. This work proposes an approach to reduce the processing time it takes to assign a trip request to a vehicle. The solution is a trip-vehicle fitness estimation framework that is flexible enough to utilize any fitness measure and is self-adjusting through feedback loops. We analyze the placement of a trip request within a vehicle schedule, present and implement three fitness measures. The resulting system is evaluated based on performance, customer satisfaction and vehicle costs criteria by running simulations. The evaluation results indicate significant performance improvement and noticeable improvements in terms of customer satisfaction and vehicle costs.

1 INTRODUCTION

Personal transportation, both private and public, is essential for the well-being of a society. Private transportation is flexible (w. r. t. time and location) and convenient since it imposes no vehicle changes as in its public counterpart. But it comes with traffic congestion, parking place problems, increased gas emissions and additional costs attached to owning a vehicle. Public transportation is usually cheaper and eliminates the parking place problem, but is not as flexible and convenient. Moreover, the quality of the service can be poor in rural areas compared to urban areas. In this Internet-driven information age, classical systems are being challenged by new ideas that take advantage of information technology (IT) and the collective vision hints at a paradigm shift towards Mobility-as-a-Service (MaaS) where two trends meet: On-demand shared mobility and self-driving vehicles (Greenblatt and Shaheen, 2015; IFT, 2015).

On-demand transportation services pick up the customers at their desired time and bring them from any location to any location, and therefore offer an emerging middle ground between private and public transportation. Uber[1] and Lyft[2] are IT-embracing, rising alternatives to the classical example for on-demand transportation, that is the taxi service. These are functionality-wise similar to a taxi service: Each vehicle services one trip request after another, sequentially. Furthermore, they also offer *ride-sharing* (UberPOOL and Lyft Line) where multiple trip requests are combined into one vehicle ride by picking up or dropping off another customer while servicing a trip request. This turns the classical taxi service into a more efficient offering (better use of resources with increasing demand) and has potential for a larger scale deployment. There are also companies that concentrate on providing a ride-sharing service such as Via[3], allygator shuttle[4] and CleverShuttle[5]. Since the customers can benefit from, e. g., reduced prices, but also should be able to tolerate some newly-introduced downsides (i. e. potentially increased waiting and ride times), a delicate trade-off has to be made between customer satisfaction and service/vehicle costs, which are inherently in opposition to each other. Hereafter, customer satisfaction refers to the number of satisfied trip requests, waiting and ride times. Likewise, service/vehicle costs refer to the number of vehicles in service, distances driven by vehicles and vehicle ca-

[1]https://www.uber.com
[2]https://www.lyft.com
[3]https://ridewithvia.com
[4]https://www.allygatorshuttle.com
[5]http://clevershuttle.org

Gökay, S., Heuvels, A. and Krempels, K.
Heuristics for Improving Trip-Vehicle Fitness in On-demand Ride-Sharing Systems.
DOI: 10.5220/0006690203230334
In *Proceedings of the 4th International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS 2018)*, pages 323-334
ISBN: 978-989-758-293-6

pacity utilization.

There are two variants of on-demand transportation services with respect to how trip requests become apparent to the system. While in *offline* solutions all requests are known before service and vehicle schedules do not change after calculation (later during service), *online* solutions process requests as they appear in real-time without knowledge about the future and constantly update the route of vehicles during service.

The performance of an online on-demand ride-sharing system (i. e. how fast it processes the request and finds a fitting vehicle or not) is particularly important, since the customers are requesting rides in real-time and waiting for responses. While most of the theoretical related work concentrates on the offline variant, the online variant offerings of aforementioned private companies are niche such that the performance and scalability are not big concerns. This work identifies shortcomings of an existing system (Gökay et al., 2017) and improves it in this regard. While Section 2 provides an overview of the research field, Section 3 motivates this work. It describes in detail how the problem has been diagnosed by performing a simulation using a real data set, collecting and analyzing statistics. Section 4 explains the taken optimization approach on an abstract level. The proposed solution is a flexible framework that combines multiple measures which can influence trip-vehicle assignments. We identify three measures and integrate them into the framework. Section 5 presents the details of the technical realization, as well as the implementation improvements that have been carried out to take the optimization efforts further. Section 6 illustrates the evaluation methodology and results. We let the previous and the current, improved implementation perform a simulation using identical configuration and data sets. We compare and discuss the outcome of both versions. Finally, Section 7 concludes the paper.

2 RELATED WORK

On-demand ride-sharing addresses the dial-a-ride problem (DARP) (Cordeau and Laporte, 2007) which is a generalization of a number of problems such as Pickup and Delivery Problem (PDP) (Savelsbergh and Sol, 1995), Vehicle Routing Problem (VRP) (Laporte, 1992b) and Traveling Salesman Problem (TSP) (Laporte, 1992a). While TSP considers only one server (i. e. vehicle), VRP works with a fleet of vehicles. VRP only considers deliveries to locations, whereas in PDP, vehicles transport goods from pickup to delivery locations. All three problems have variants that consider time constraints by employing *time windows*, i. e. location visits have to occur within given time windows, (Savelsbergh, 1985; Cordeau et al., 2001a; Parragh et al., 2008). Most DARP models already impose time windows, so we consider it as part of the problem definition. The main difference of DARP from TSP, VRP and PDP is the human perspective: These problems aim to minimize vehicle route costs and maximize the number of satisfied requests. DARP additionally aims to minimize customer waiting and ride time in order to maximize customer satisfaction.

DARP has two configurations: Online or offline, single- or multi-vehicle. In (Cordeau and Laporte, 2007), the authors show that earliest works concentrate on offline single-vehicle DARP, most of the work addresses offline multi-vehicle DARP and only very few of the algorithms solve online multi-vehicle DARP. We believe that online multi-vehicle variant is the most promising when considering real-time, real-world practical usage.

Since DARP generalizes TSP, determining an optimal solution is NP-hard (Coja-Oghlan et al., 2005; Gørtz, 2006). Even though there are exact algorithms that can find optimal solutions, these either solve simplified DARPs by leaving some constraints out or work on small to medium problem instances (Psaraftis, 1983; Desrosiers et al., 1986; Ropke et al., 2007). Many exact algorithms use a branch-and-cut technique (Lysgaard et al., 2004; Cordeau, 2006). However, due to the need in practical applications, most related work focuses on development of heuristics and metaheuristics. There are some techniques that are widely used such as neighborhood search (Pisinger and Ropke, 2007), tabu search (Cordeau et al., 2001b; Attanasio et al., 2004), simulated annealing (Baugh et al., 1998), insertion heuristics (Jaw et al., 1986; Madsen et al., 1995; Tsubouchi et al., 2010), and genetic algorithms (Jorgensen et al., 2007).

Solving DARP with big problem instances in a performant and scalable way only recently started to get attention. The work in (Alonso-Mora et al., 2017) presents a solution approach to online multi-vehicle DARP that starts with a greedy solution and improves it incrementally. The authors evaluate the algorithm by using 1000, 2000, 3000 vehicles and $\sim$3 million trip requests ($\sim$500,000 per day). The requests are collected for a time period (i. e. 10–50 seconds) and then assigned to vehicles in batches, which detracts from its real-time applicability. Another algorithm, presented in (Ota et al., 2017), is evaluated by processing $\sim$260 million trip requests using between $\sim$500 and $\sim$6500 vehicles. It provides good performance, but requires extraordinary resources (i. e. a 1200 CPU core cluster). Both works evaluate their approaches with trips extracted from New York City taxi trip data (2010–2013)[6].

3 MOTIVATION

Having developed the initial version of an online multi-vehicle DARP solution approach in (Gökay et al., 2017) but tested it with only small problem instances (10-70 vehicles and up to 5000 trip requests), we wanted to challenge the system and run a simulation test with the same data set (one day with 524,845 trip requests) and similar configuration (3000 vehicles each with a capacity of 10 and a time window of 5 min for pick-up and drop-off events) as presented in (Alonso-Mora et al., 2017). The simulation was not finished even after two and half days (we stopped it after processing 292,200 requests).

The fact, that a request data set of one day could not be processed at the very worst within a day, implies that the system cannot scale and is not applicable for real-time, practical usage. This led us to investigate the possible sources of the problem.

Problem Description

Our approach employs insertion heuristics: A request consists of a pick-up and drop-off event. Each event comprises of a location, a time window and an actual time within the time window, that represents the point of time when the location will be visited by the vehicle[7]. The processing pipeline of a request contains a first stage, that sorts the vehicles according to vehicle-request fitness, and a second stage that iterates the sorted list of vehicles, inserts the request into the schedule of the considered vehicle and tries to adjust the schedule in a way that the detour caused by this new request does not violate the time constraints of existing requests. After satisfying time constraints, it is checked whether this new schedule violates the capacity constraints of the vehicle. The request is assigned to the first vehicle that has no violations. A detailed description of the working principle, and the scheduling algorithm, can be found in (Gökay et al., 2017), which bases on (Tsubouchi et al., 2010).

During the adjustment of the schedule, the order of events is changed several times until finding a feasible order without time constraint violations. Moreover, changing the order of events (and thus locations) causes many routing calculations to determine the ride times between newly ordered events so that the algorithm can check whether time constrains are still held. This many live routing calculations caused significant runtime overhead, since we do not precompute and store the routes between all locations in a lookup table for the following reasons:

1. The boundaries of the service area are defined by the imported map during application startup, which can be as big as the service operator desires. Within the service area the trip requests can be made from *any* location to *any* location. A predefined set of allowed pick-up and drop-off locations would make pre-computation possible, but also force customers to walk which is inconvenient.

2. Since the system is an *online* on-demand service, there is no knowledge about the future requests. Pre-computing routes from each location to each location and storing them is only manageable (w. r. t. computation space and time) for *offline* variants with small problem instances. At best, the online system can cache computed routes to prevent costly re-calculation of the same route. Our implementation already utilizes route caches.

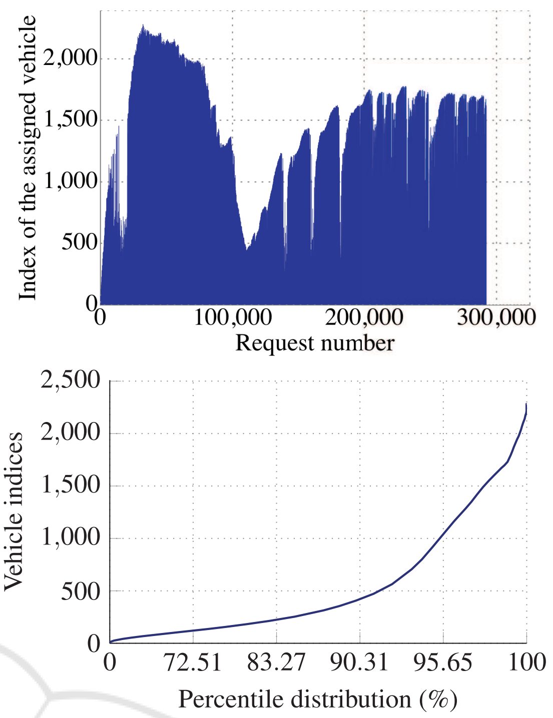

Figure 1: Vehicle index statistics of the initial implementation for 292,200 requests with 3000 vehicles, vehicle capacity of 10 and time window of 5 min (mean≈164, stdev≈335, max=2281). The vehicle sorter is inefficient about finding the fitting bus to which a request can be assigned.

[6]https://databank.illinois.edu/datasets/IDB-9610843

[7]Maximum ride time constraints are implied, since both pick-up and drop-off events have time windows.

Interestingly enough, most existing DARP solutions neglect the route calculation cost aspect of the problem, since they consider the routes to be *given*, and focus on the multi-criteria combinatorial optimization. The aforementioned recent works, (Alonso-Mora et al., 2017; Ota et al., 2017), utilize the gridlike street geography of Manhattan to pre-compute routes, which is not the case for most service areas.

Having established that, we investigated whether we can reduce the number of routing calculations. The investigation led to the discovery that the first stage in the pipeline, the vehicle sorter, was inefficient (Figure 1): The indices of the vehicles, to which the requests were assigned, were so high within the sorted vehicle list, that the second stage considered many, apparently unfitting, vehicles and thus has done many unnecessary routing calculations.

4 APPROACH

Our optimization approach starts with the analysis *where* pick-up and drop-off events are inserted into a schedule. Figure 2 illustrates one section of a vehicle schedule, in which $e_{k-1}, e_k, e_{k+1}, e_{k+2}$ are consecutive events and the actual times are ordered:

$$t_{first} < \ldots < t_{k-1} < t_k < t_{k+1} < t_{k+2} < \ldots < t_{last}$$

When a new event e_{new} has to be inserted into a schedule, it should fall between two events, based on the actual times of events: $t_k < t_{new} < t_{k+1}$. We call the position (i. e. the pair (e_k, e_{k+1})), where the new event e_{new} falls, a *pocket*. While the schedule contained the path $e_k \to e_{k+1}$ before the insertion, afterwards it will be changed to $e_k \to e_{new} \to e_{k+1}$ due to the detour.

The schedule of a vehicle contains the future requests that are assigned to it. So, e_{first} is the first event that the vehicle has to service. Likewise, e_{last} is the last event in the schedule. An event cannot fall before the first event e_{first}, because in this case it implies that the event is in the past. In our implementation, a vehicle is waiting idle at e_{last} until a new request is assigned to it (in the related problem space of VRP the variant, where vehicles are not required to return to a depot after the last delivery, is called *open* VRP (Li et al., 2007)).

There are three possibilities where the pick-up and drop-off events of a request might fall:

(P1) Both within the schedule (Figure 3a):

$$t_{first} < t_{new_{pick-up}} < t_{new_{drop-off}} < t_{last}$$

(P2) Pick-up in the schedule, drop-off after the schedule (Figure 3b):

$$t_{first} < t_{new_{pick-up}} < t_{last} < t_{new_{drop-off}}$$

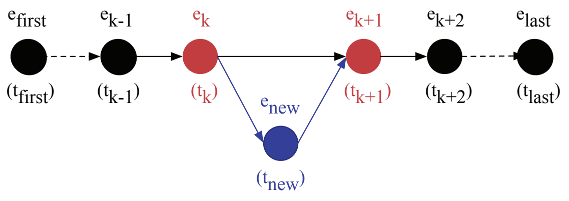

Figure 2: Illustration of a pocket. The event pair (e_k, e_{k+1}) is the pocket for e_{new}.

Since the drop-off event does not fall into an actual pocket, we treat $(e_{last}, e_{new_{drop-off}})$ as such.

(P3) Both after the schedule (Figure 3c):

$$t_{last} < t_{new_{pick-up}} < t_{new_{drop-off}}$$

Since the pick-up event does not fall into an actual pocket, we treat $(e_{last}, e_{new_{pick-up}})$ as such.

To reduce the number of false positives of the vehicle sorting stage and therefore to have a more precise strategy, we incorporate the information where the events fall into our approach. Based on this observation, we define three measures for sorting vehicles with respect to a request for each event:

Detour Distance: This measure ensures that we favor vehicles with minimal additional route costs imposed by detours. It is calculated as the difference between new and old path distance:

$$dist_{e_k \to e_{new}} + dist_{e_{new} \to e_{k+1}} - dist_{e_k \to e_{k+1}}$$

However, the cases **(P2)** and **(P3)** contain situations in which the old path does not exist. We address this issue in Section 4.1. Values returned by this measure are treated as *the lower the better*. Since we want the vehicle sorting stage to be fast, we calculate all distances between two geolocations based on the haversine formula.

Largest Pocket: This measure ensures that we disfavor vehicles with potential event constraint violations. Event constraint violations occur when the actual times of two events are too close to each other, while the locations of the events are too far from each other. In such cases, our routing engine calculates the shortest path duration from the first to the subsequent location and our scheduling algorithm decides that the vehicle cannot make it to the subsequent event *in time*. Preventing these cases reduces the number of routing calculations. Largest pocket is calculated as the division of time available in pocket by the new path distance:

$$\frac{t_{k+1} - t_k}{dist_{e_k \to e_{new}} + dist_{e_{new} \to e_{k+1}}}$$

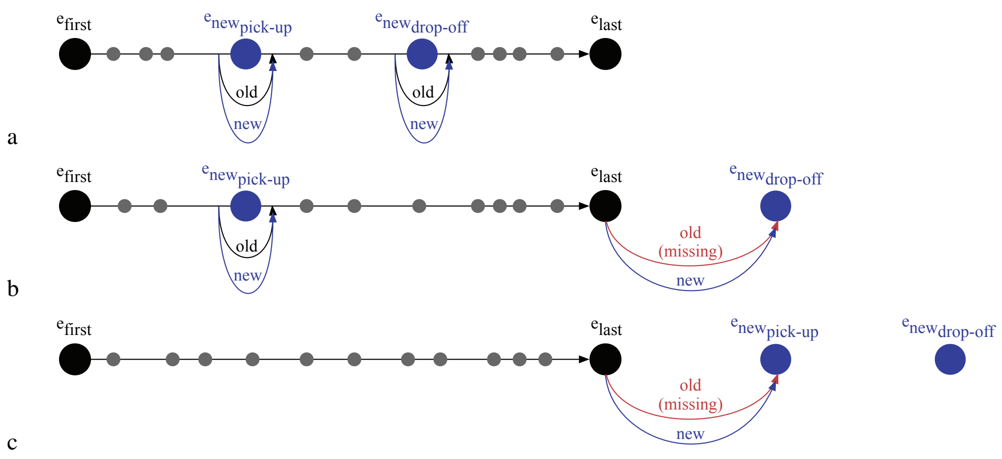

Figure 3: Possible event pockets: (a) Pick-up and drop-off events within the schedule (b) Pick-up in schedule, drop-off after the schedule (c) Both events after the schedule

Values returned by this measure are treated as *the higher the better.*

Reward: Our vehicle sorting stage is intended to be fast and therefore its calculations are approximate. In the second stage, shortest path durations are calculated and actual times are adjusted. This measure ensures that we disfavor vehicles, where potentially more event adjustments are needed, by rewarding cases with less events within the schedule. It assigns a constant value to each case following the preference **(P3)** $>$ **(P2)** $>$ **(P1)** and values returned by this measure are treated as *the higher the better.*

4.1 Old Path Distance Estimation

We define a mechanism to estimate old path distances in cases where we have only a new path distance (drop-off in **(P2)** and pick-up in **(P3)**). For this, we make use of the situations, i. e. both events in **(P1)** and pick-up event in **(P2)**, where both the old and new distances are present. The idea is to store and always update the average of old and new distance ratios. Then, when it comes to estimate the old path distance, we multiply this ratio with the new path distance to approximate what the old path distance would have been.

The distance values can be as small as tens of meters and can grow up to tens of kilometers. Therefore, having only one ratio for the whole value spectrum would not have been precise. In order to handle the ratios better, we divide the spectrum into multiple buckets that are 200m wide.

4.2 Combining Measures

After establishing the basic building blocks of our approach, this section presents how we put everything together. The optimized vehicle sorter follows these steps for each trip request and for each vehicle:

1. Determine the pockets for pick-up and drop-off events, i. e. find out the case (**(P1)**, **(P2)** or **(P3)**) for this request-schedule mapping.
2. Calculate the three measures for this request-schedule mapping.
3. Construct the weighted linear combination of the three measures to derive the final fitness score.
4. Sort vehicles by their fitness score.

In the field of forecasting, the weights for combining the features are commonly determined by incorporating domain knowledge of experts and analyzing historical data (Adya et al., 2001). One problem with weighting is that past data may not reflect the current characteristics and therefore using static weights can result in poor accuracy (Miller et al., 1992). We get around this problem by employing a feedback mechanism into our processing pipeline in order to inform the vehicle sorter after processing each request how good or bad the sorting was based on the index of the assigned vehicle (Figure 4). This approach can be defined as *dynamic linear combination* (Lobrano et al., 2010). In addition to sorting vehicles after the linear combination step, we also sort vehicles by each measure, separately. By this means, we can compare the global index a vehicle receives with the index each measure gave the same vehicle. If the measure gave the vehicle a smaller (or higher) index than the final index, the measure increases (or decreases) its weight. Moreover, adjusting the weights based on only the

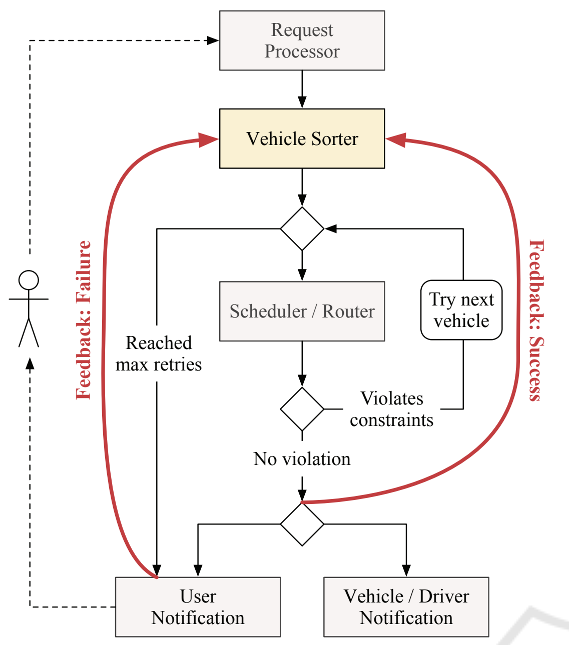

Figure 4: Overview of the general framework how the trip requests are processed with the added feedback mechanism.

last request or all past requests would be misleading, since we rather want to recognize a *trend* and adjust accordingly. In order to realize this, we use a *moving average* of last 1000 indices for each measure.

4.3 Request Rejection Costs

Another important aspect is the processing costs of a request rejection. A rejection means iterating all the vehicles in the fleet, where no vehicle is a valid candidate for accepting the request into schedule. When the number of vehicles is huge, it takes an excessive amount of time to reject a single request. With the optimization techniques illustrated thus far, the improved vehicle sorting stage produces less deviation between the indices of assigned vehicles. We use this outcome to our advantage and employ a filter to reduce the number vehicles that are to be considered. We keep track of vehicle indices for accepted requests and update the *moving average* for subsequent requests. The average index is used as a cut-off point when the second stage iterates the sorted list of vehicles. The average index is relaxed with each rejected request, since it could mean that the cut-off point was set too greedily.

5 IMPLEMENTATION

The system is developed as a standalone Java 8 application. It uses GraphHopper[8] as the routing engine, since it can be embedded into a Java application. GraphHopper imports OpenStreetMap (OSM)[9] maps and builds the underlying graph to be used for routing. Moreover, GraphHopper utilizes *contraction hierarchies* to speed up routing and we specify the transportation mode as *car*. The computed routes are easily cacheable, because the graph is not time-dependent (as it would be in the case of public transportation modes such as bus or train).

We make use of parallelization at all steps where it is possible. One step worth mentioning is the second stage, the scheduler in Figure 4. The initial implementation iterated the list of vehicles sequentially. This takes too long, if the viable vehicle has a high index in the list. Parallel processing with parallelization factor set to the number of vehicles to be considered can also be costly due to context switching, if this number is huge. We partition the sorted list of vehicles into batches with size set to the number of CPU cores of the system. The scheduler runs for each vehicle within a batch in parallel. If no viable vehicle is found in this batch, we continue with the next batch. If there are results, we calculate the total time cost for each result and pick the vehicle with the lowest cost, which is determined by the sum of several criteria:

- Increased costs of the vehicle
  $$\Delta_v = travelTime(S_{new}) - travelTime(S_{prev}),$$
  where $travelTime(S)$ is the sum of all trips in schedule S.
- Increased costs for each customer due to the schedule adjustment
  $$\Delta_S = \sum_{c \in S_{new}} costs_{S_{new}}(c) - \sum_{c \in S_{prev}} costs_{S_{prev}}(c),$$
  where $costs_S(C)$ denotes the cost for a customer C in schedule S and is calculated as the sum of waiting and ride time.

6 EVALUATION

We perform two simulations, one with the old and one with the new vehicle sorting strategy implementation, using the same simulation configuration and data sets, and compare their results. The old vehicle sorting strategy is the one presented in (Tsubouchi

[8]https://www.graphhopper.com/
[9]http://www.openstreetmap.org/

et al., 2010), which prefers vehicles with the closest direction to the new trip request. Both simulations benefit from the improvements outlined in Section 5. The simulations are run inside a virtual machine configured with 8 vCPUs (Intel Xeon E5-2650 clocked at 2.20 GHz) and 8 GB system memory. Used Java Virtual Machine parameters are `-Xms2048m -Xmx4096m -XX:+UseG1GC`. As of writing, this system configuration represents an entry level off-the-shelf server.

The OSM data used for New York City contains all information and changes up to 2017-04-09T15:01:34Z. We extract trips from New York City taxi trip data (2010–2013) for the day Saturday May 11, 2013 (524,845 trips after clean-up). From each data point we read the pick-up date time (we treat this as the beginning of the pick-up time window) and the geolocations of pick-up and drop-off to construct the trip requests. Number of passengers is set to 1. Since the previous vehicle sorting strategy could not cope with processing all trips of a day in reasonable time, we prepare three data sets:

1. Reduced number of trips by selecting *every 10th* (from 00:00 to 23:59 totaling to 52,552 trips). The aim is to keep the distribution as close as possible to Figure 5 with less data points. Hereby, we assume not to introduce a selection bias.
2. Picking a time interval with *high density* of requests to represent peak demand (with pick-up times from 19:00 to 20:00 totaling to 30,884 trips)
3. Picking a time interval with *low density* of requests to represent off-peak demand (with pick-up times from 05:00 to 07:00 totaling to 10,388 trips).

The configuration consists of number of vehicles (depending on the data set this parameter varies between 100 and 3000), vehicle capacity (3 or 10) and time window (5 or 10 min). For all data sets, all vehicles are initialized at the same location and at the start of the data set's time range (e. g. for low density data set at 05:00). We evaluate our results in three categories: Performance improvement (Figure 6), customer satisfaction (Figure 7) and service/vehicle costs (Figure 8). We also discuss how the changes aiming at performance improvements affect the customer satisfaction and service costs. The detailed results are presented in Tables 1 and 2.

Figure 6 shows that the improved implementation is always faster in terms of request-vehicle assignment calculation duration. The average calculation duration is reduced from 2707 ms to 320 ms (worst case). This is due to the fact that it considers less false positive vehicles until finding a viable vehicle for a request (i. e. reduced average vehicle index). This improves the total run time substantially (Table 1). As

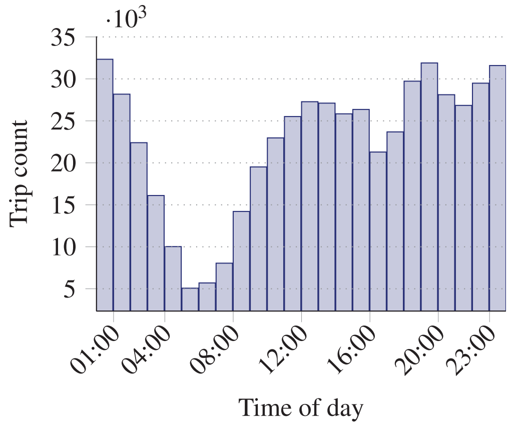

Figure 5: Hourly trip distribution throughout the day on Saturday May 11, 2013.

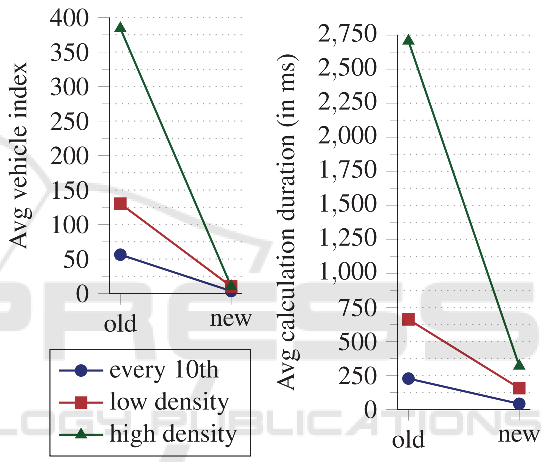

Figure 6: Performance comparison of the old and new implementation. Average vehicle index is reduced, since the new implementation considers less false positives. This significantly reduces the calculation duration of request-vehicle assignments.

can be seen in Figure 7, the customer satisfaction aspect benefits from improved implementation. We can recognize that the new version almost always reduces the waiting time and ride delay of customers while producing similar or better success rate[10]. Ride delay improvements are not that prominent or slightly worse with the low density data set, because the new implementation consistently has higher success rate (see Table 2).

These improvements can come at slightly increased vehicle costs. Vehicle statistics in Table 2 show that the new implementation has somewhat higher vehicle costs when the configuration is too generous (e. g. more vehicles in service than actually needed), because it distributes requests more evenly across all

[10]The number of accepted requests divided by the number of all requests

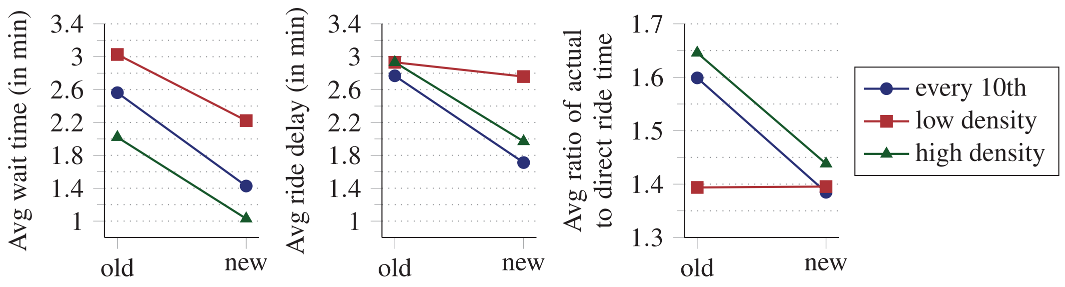

Figure 7: Customer satisfaction comparison of the old and new implementation. Both wait time and ride delay are lower with the new implementation. Average ratio of actual to direct ride time expresses the relative ride delay.

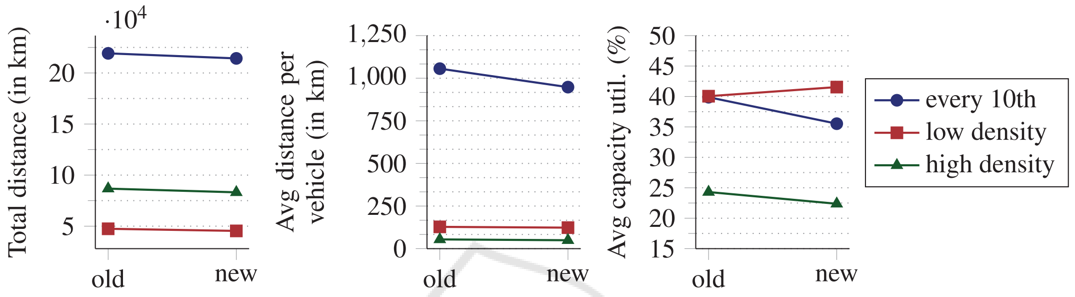

Figure 8: Service/vehicle costs comparison of the old and new implementation. Average capacity utilization decreases with the data sets *every 10th* and *high density* because with some simulation configurations the new implementation is at a disadvantage (see Table 2).

vehicles (as highlighted by the column *# of used vehicles*). Even though the driven distance per vehicle is lower, there are more vehicles that are used. This situation results in a slightly higher sum of driven distance of all vehicles. Moreover, we capture capacity utilization statistics for each customer-vehicle assignment by counting total customers in a vehicle after picking up the new customer. Figure 8 illustrates that average capacity utilization decreases with the data sets *every 10th* and *high density*. This is due to the huge differences between the new and old implementation with generous configurations. Since the events in vehicle schedules are not that *dense* with the new implementation (a consequence of even distribution of requests to all vehicles), capacity utilization is lower. Even though the new implementation yields better capacity utilization results with diminishing configurations, they are not dominant enough to counterbalance the overall average. With diminishing configurations the new implementation starts to produce better results in all aspects. If the number of vehicles is set to a value such that both implementations have to make use of all the vehicles, then in most cases the new implementation satisfies more customers (i. e. achieves higher success rate) with less vehicle costs. As a result, the total distance driven by all vehicles or the average distance driven per vehicle stays more or less the same (Figure 8).

7 CONCLUSION

On-demand ride-sharing systems are becoming increasingly popular. This entails a potential scalibility problem which only recently started to get attention. In this work, we presented an optimization approach for an online on-demand ride-sharing system that was intended to be a runtime improvement without sacrificing quality of the results, but the evaluation showed that the new approach can yield better results as well. The system's working principle consists of a trip request processing pipeline with two stages. The first stage, the vehicle sorter, sorts the vehicles according to their request-vehicle assignment fitness. The second stage, the scheduler, iterates the list of sorted vehicles and adjusts the schedule of each vehicle (after inserting the new request into the schedule) such that the detours imposed by the inclusion of the new request do not break the time constraints of other requests. Schedule adjustments include a large number of routing calculations which are costly. The scheduler iterates the list until finding one viable vehicle that can tolerate the inclusion of the new request and to which the request is assigned. The need for a runtime improvement is based on the evaluation with huge data sets (∼500,000 trip requests) and thousands of vehicles. The evaluation took such a long time to process the requests, that the solution was not applicable for real-time usage. We identified that the cause of this problem was the former vehicle sorting stage, which inefficiently considered false positive vehicles as fitting and therefore the second stage, the scheduler, had to do many schedule adjustments of unfitting vehicles.

The proposed optimization replaces the vehicle sorting stage and tries to avoid time constraint violations. At first, we determine the *pockets* (i. e. potential placements) of the pick-up and drop-off events of a new request within a schedule and use the information gathered from neighboring events for a more precise judgment. Based on this, we calculate three measures (detour distance, largest pocket and reward) and derive the final fitness score by the weighted linear combination of these three measures. Moreover, we introduce a feedback loop into the processing pipeline for the system to learn from its successes and failures in order to regulate the measure weights accordingly. For the evaluation, we extract trip requests from New York City taxi trip data and run two simulations, i. e., we let the old and new implementations with otherwise identical configurations process the same requests. The evaluation results are categorized in three groups: Performance improvement, customer satisfaction and service costs. As the main goal of this work, the new implementation outperforms the old one in all simulations. Moreover, the new implementation provides better customer satisfaction in almost all simulations. Since the new implementation aims to minimize the possibility of time constraint violations, it distributes requests more evenly to all vehicles. With generous configurations (e. g. there are more vehicles in service than actually needed), it has slightly higher vehicle costs. When reducing the number of vehicles, the new implementation starts to yield better results in all aspects. We observe that a time window of 10 min and 2000 vehicles each with a capacity of 10 can satisfy 98% of the demand at peak periods. At off-peak time periods, the number of vehicles can be reduced to 500 to accomplish the same.

This work serves as a basis for further experimentation and improvements. Having a fast evaluation cycle enables extensive testing with big problem instances to better assess the real-world applicability. Event though the performance gain is particularly evident with the data set *high density*, the test runtime is still suboptimal with some configurations. Furthermore, the disadvantageous consequences of generous configurations (making use of *all* vehicles, decreased shared rides and capacity utilization) should be avoided. Addressing these issues is part of future work.

ACKNOWLEDGMENTS

This work was partially funded by the German Federal Ministry of Transport and Digital Infrastructure (BMVI) for the project “Digitalisierte Mobilität – die Offene Mobilitätsplattform” (19E16007B).

REFERENCES

Adya, M., Collopy, F., Armstrong, J., and Kennedy, M. (2001). Automatic identification of time series features for rule-based forecasting. *International Journal of Forecasting*, 17(2):143–157.

Alonso-Mora, J., Samaranayake, S., Wallar, A., Frazzoli, E., and Rus, D. (2017). On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 114(3):462–467.

Attanasio, A., Cordeau, J.-F., Ghiani, G., and Laporte, G. (2004). Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. *Parallel Computing*, 30(3):377 – 387.

Baugh, J. W., Kakivaya, G. K. R., and Stone, J. R. (1998). Intractability of the dial-a-ride problem and a multiobjective solution using simulated annealing. *Engineering Optimization*, 30(2):91–123.

Coja-Oghlan, A., Krumke, S. O., and Nierhoff, T. (2005). A Hard Dial-a-Ride Problem that is Easy on Average. *Journal of Scheduling*, 8(3):197–210.

Cordeau, J.-F. (2006). A branch-and-cut algorithm for the dial-a-ride problem. *Oper. Res.*, 54(3):573–586.

Cordeau, J.-F., Desaulniers, G., Desrosiers, J., Solomon, M. M., and Soumis, F. (2001a). VRP with Time Windows. In Toth, P. and Vigo, D., editors, *The Vehicle Routing Problem*, pages 157–193. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

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

Cordeau, J.-F., Laporte, G., and Mercier, A. (2001b). A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society*, 52(8):928–936.

Desrosiers, J., Dumas, Y., and Soumis, F. (1986). A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows. *American Journal of Mathematical and Management Sciences*, 6(3-4):301–325.

Gökay, S., Heuvels, A., Rogner, R., and Krempels, K.-H. (2017). Implementation and Evaluation of an On-Demand Bus System. In *Proceedings of the 3rd International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS 2017)*, Porto, Portugal.

Gørtz, I. L. (2006). Hardness of preemptive finite capacity dial-a-ride. In in Computer Science, L. N., editor, *Proceedings of 9th International Workshop on Ap-*

proximation Algorithms for Combinatorial Optimization Problems (APPROX 2006).

Greenblatt, J. B. and Shaheen, S. (2015). Automated vehicles, on-demand mobility, and environmental impacts. *Current Sustainable/Renewable Energy Reports*, 2(3):74–81.

IFT (2015). Urban Mobility System Upgrade – How shared self-driving cars could change city traffic. Research report, International Transport Forum at the Organisation for Economic Co-operation and Development (OECD). Available at https://www.itf-oecd.org/sites/default/files/docs/15cpb_self-drivingcars.pdf.

Jaw, J.-J., Odoni, A. R., Psaraftis, H. N., and Wilson, N. H. (1986). A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research Part B: Methodological*, 20(3):243 – 257.

Jorgensen, R. M., Larsen, J., and Bergvinsdottir, K. B. (2007). Solving the dial-a-ride problem using genetic algorithms. *Journal of the Operational Research Society*, 58(10):1321–1331.

Laporte, G. (1992a). The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231 – 247.

Laporte, G. (1992b). The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(3):345 – 358.

Li, F., Golden, B., and Wasil, E. (2007). The open vehicle routing problem: Algorithms, large-scale test problems, and computational results. *Comput. Oper. Res.*, 34(10):2918–2930.

Lobrano, C., Tronci, R., Giacinto, G., and Roli, F. (2010). *Dynamic Linear Combination of Two-Class Classifiers*, pages 473–482. Springer Berlin Heidelberg, Berlin, Heidelberg.

Lysgaard, J., Letchford, A. N., and Eglese, R. W. (2004). A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100(2):423–445.

Madsen, O. B. G., Ravn, H. F., and Rygaard, J. M. (1995). A heuristic algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. *Annals of Operations Research*, 60(1):193–208.

Miller, C. M., Clemen, R. T., and Winkler, R. L. (1992). The effect of nonstationarity on combined forecasts. *International Journal of Forecasting*, 7(4):515 – 529.

Ota, M., Vo, H., Silva, C., and Freire, J. (2017). STaRS: Simulating Taxi Ride Sharing at Scale. *IEEE Transactions on Big Data*, 3(3):349–361.

Parragh, S. N., Doerner, K. F., and Hartl, R. F. (2008). A survey on pickup and delivery problems - Part I: Transportation between customers and depot. *Journal für Betriebswirtschaft*.

Pisinger, D. and Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403 – 2435.

Psaraftis, H. (1983). An exact algorithm for the single vehicle many-to-many dial-a-ride problem with time windows. *Transportation Science*, 17(3):351–357.

Ropke, S., Cordeau, J.-F., and Laporte, G. (2007). Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks*, 49(4):258–272.

Savelsbergh, M. and Sol, M. (1995). The general pickup and delivery problem. *Transportation Science*, 29(1):17–29.

Savelsbergh, M. W. P. (1985). Local search in routing problems with time windows. *Annals of Operations Research*, 4(1):285–305.

Tsubouchi, K., Yamato, H., and Hiekata, K. (2010). Innovative on-demand bus system in Japan. *IET Intelligent Transport Systems*, 4(4):270–279.

Table 1: Performance overview of test cases. Each cell contains a value that represents the result of the simulation with the new implementation and another value within parentheses that expresses the difference of the this value from the result of the simulation with the old implementation. The green (or red) cells indicate that the new implementation produced a better (or worse) result.

Test case setup				Indices of assigned vehicles			Calculation durations for each request (in ms)			
Data set	# of vehicles	Vehicle capacity	Time window (in min)	Mean	Stdev	Max	Mean	Stdev	Max	Test runtime (in h)
every 10th	500	10	10	1.22 (−55.94)	0.81 (−46.46)	8 (−262)	4.86 (−191.60)	2.24 (−167.09)	91 (−2718)	0.07 (−2.80)
every 10th	500	10	5	1.27 (−58.03)	0.86 (−47.46)	8 (−280)	4.96 (−189.41)	2.41 (−162.50)	78 (−2559)	0.07 (−2.76)
every 10th	500	3	10	1.21 (−57.51)	0.80 (−47.46)	8 (−267)	4.82 (−191.67)	2.11 (−166.13)	73 (−2730)	0.07 (−2.80)
every 10th	500	3	5	1.30 (−58.72)	0.89 (−47.66)	8 (−267)	5.02 (−191.44)	2.46 (−163.75)	98 (−2481)	0.07 (−2.79)
every 10th	250	10	10	2.84 (−58.82)	2.61 (−47.77)	32 (−218)	17.44 (−226.61)	16.54 (−220.10)	245 (−3084)	0.25 (−3.31)
every 10th	250	10	5	4.17 (−55.10)	4.04 (−43.97)	50 (−200)	23.54 (−194.77)	21.91 (−187.08)	326 (−2351)	0.34 (−2.84)
every 10th	250	3	10	3.59 (−60.97)	3.44 (−48.15)	42 (−208)	19.70 (−233.80)	18.70 (−222.30)	349 (−2626)	0.29 (−3.41)
every 10th	250	3	5	4.06 (−56.42)	4.13 (−44.35)	54 (−196)	22.57 (−201.22)	22.69 (−190.94)	318 (−2841)	0.33 (−2.94)
every 10th	175	10	5	5.73 (−50.03)	6.05 (−35.75)	79 (−96)	52.94 (−215.54)	66.30 (−138.40)	664 (−1857)	0.77 (−3.15)
every 10th	175	3	5	5.97 (−50.86)	6.41 (−35.72)	95 (−80)	55.93 (−214.78)	68.63 (−133.00)	704 (−1705)	0.82 (−3.14)
every 10th	100	10	10	7.97 (−32.31)	8.66 (−17.71)	59 (−41)	94.95 (−140.11)	66.29 (−49.36)	825 (−527)	1.39 (−2.05)
every 10th	100	10	5	5.36 (−32.17)	5.36 (−20.45)	56 (−44)	90.50 (−131.09)	66.45 (−47.32)	739 (−676)	1.32 (−1.91)
every 10th	100	3	10	8.15 (−32.55)	8.81 (−17.05)	63 (−37)	94.11 (−139.91)	64.53 (−48.09)	620 (−711)	1.37 (−2.04)
every 10th	100	3	5	5.39 (−32.81)	5.38 (−20.55)	60 (−40)	89.56 (−132.29)	65.67 (−47.10)	726 (−791)	1.31 (−1.93)
high density	3000	10	10	4.31 (−393.26)	3.05 (−382.55)	78 (−2385)	9.53 (−1850.68)	8.08 (−2048.44)	235 (−28 836)	0.08 (−15.88)
high density	3000	10	5	4.17 (−353.80)	3.16 (−352.93)	107 (−2078)	10.84 (−1639.17)	11.40 (−1898.11)	322 (−24 989)	0.09 (−14.06)
high density	2000	10	10	5.01 (−457.82)	2.76 (−434.98)	60 (−1940)	12.66 (−2977.16)	12.95 (−3485.42)	414 (−25 105)	0.11 (−25.54)
high density	2000	10	5	4.84 (−375.56)	3.41 (−379.04)	99 (−1901)	17.80 (−2049.96)	19.73 (−2692.67)	271 (−24 320)	0.15 (−17.58)
high density	1500	10	10	4.50 (−432.18)	2.71 (−381.56)	45 (−1455)	20.03 (−3474.81)	17.66 (−3327.30)	267 (−18 772)	0.17 (−29.81)
high density	1500	10	5	11.44 (−371.43)	22.10 (−338.67)	301 (−1199)	106.96 (−2819.85)	256.50 (−2788.13)	2775 (−16 664)	0.92 (−24.19)
high density	1250	10	10	18.62 (−372.60)	41.91 (−290.92)	477 (−773)	546.16 (−2874.99)	1071.62 (−1833.93)	6587 (−9732)	4.68 (−24.66)
high density	1250	10	5	19.12 (−330.01)	31.86 (−286.81)	465 (−785)	516.76 (−2464.63)	853.35 (−1821.24)	5299 (−10 876)	4.43 (−21.14)
high density	1000	10	10	26.24 (−302.14)	49.13 (−223.95)	515 (−485)	1064.31 (−1909.83)	1314.02 (−897.47)	6655 (−6080)	9.13 (−16.38)
high density	1000	10	5	18.86 (−279.96)	28.61 (−233.94)	376 (−624)	888.25 (−1817.94)	1068.72 (−1017.07)	5043 (−8092)	7.62 (−15.59)
low density	500	10	10	4.42 (−145.12)	2.93 (−121.60)	22 (−478)	22.45 (−747.65)	18.64 (−596.09)	437 (−3588)	0.06 (−2.16)
low density	500	10	5	12.39 (−136.96)	17.88 (−105.74)	199 (−301)	110.63 (−618.04)	169.06 (−380.36)	1481 (−2342)	0.32 (−1.78)
low density	500	3	10	8.88 (−148.04)	11.32 (−112.75)	155 (−345)	50.27 (−745.42)	84.76 (−515.78)	856 (−2825)	0.15 (−2.15)
low density	500	3	5	13.57 (−139.01)	19.29 (−106.86)	214 (−286)	122.25 (−608.67)	176.22 (−372.48)	1426 (−2149)	0.35 (−1.76)
low density	250	10	10	12.48 (−74.18)	17.59 (−49.66)	133 (−117)	239.66 (−346.40)	181.12 (−103.61)	943 (−757)	0.69 (−1.00)
low density	250	10	5	11.23 (−72.56)	14.97 (−49.80)	149 (−101)	230.23 (−315.45)	165.81 (−104.03)	748 (−970)	0.66 (−0.91)
low density	250	3	10	13.80 (−75.61)	18.24 (−48.01)	141 (−109)	242.78 (−343.95)	173.74 (−103.24)	938 (−724)	0.70 (−0.99)
low density	250	3	5	10.68 (−75.48)	14.24 (−51.25)	141 (−109)	232.70 (−315.65)	169.40 (−91.04)	959 (−990)	0.67 (−0.91)

Table 2: Statistics w. r. t. vehicle/service costs and customer satisfaction. Each cell contains a value that represents the result of the simulation with the new implementation and another value within parentheses that expresses the difference of the this value from the result of the simulation with the old implementation. The green (or red) cells indicate that the new implementation produced a better (or worse) result.

Test case setup				Vehicle/service costs					Customer satisfaction			
Data set	# of vehicles	Vehicle capacity	Time window (in min)	# of used vehicles	Mean driven distance per vehicle (in km)	Stdev driven distance per vehicle (in km)	Shared rides (%)	Avg capacity util (%)	Avg waiting time (in min)	Avg ride delay (in min)	Avg ratio of actual to direct ride times	Success rate (%)
every 10th	500	10	10	500 (+155)	607.88 (−211.80)	43.58 (−355.50)	2.57 (−59.73)	10.27 (−9.54)	0.05 (−3.04)	0.12 (−3.72)	1.11 (−0.76)	99.85 (−0.14)
every 10th	500	10	5	500 (+166)	607.87 (−200.56)	44.09 (−366.71)	2.49 (−54.98)	10.26 (−8.02)	0.04 (−1.86)	0.09 (−2.09)	1.08 (−0.45)	99.76 (−0.16)
every 10th	500	3	10	500 (+140)	608.51 (−196.46)	44.91 (−360.39)	2.46 (−60.74)	34.22 (−29.00)	0.05 (−3.04)	0.11 (−3.71)	1.11 (−0.76)	99.86 (−0.13)
every 10th	500	3	5	500 (+159)	607.93 (−189.41)	43.49 (−373.54)	2.56 (−54.88)	34.23 (−25.16)	0.04 (−1.86)	0.09 (−2.06)	1.08 (−0.44)	99.75 (−0.17)
every 10th	250	10	10	250 (0)	839.13 (−212.19)	168.96 (−0.99)	68.92 (+6.25)	20.94 (+1.35)	1.67 (−1.55)	2.80 (−0.89)	1.64 (−0.17)	97.63 (+3.50)
every 10th	250	10	5	250 (0)	913.86 (−113.43)	138.01 (−68.26)	61.12 (+3.38)	18.16 (+0.17)	1.43 (−0.52)	1.79 (−0.35)	1.43 (−0.09)	94.62 (−0.91)
every 10th	250	3	10	250 (0)	864.12 (−195.25)	167.90 (+3.74)	70.48 (+7.72)	65.88 (+3.65)	1.79 (−1.47)	2.73 (−0.92)	1.62 (−0.18)	96.92 (+3.85)
every 10th	250	3	5	250 (0)	925.91 (−104.31)	134.76 (−60.93)	58.59 (+1.03)	57.84 (−0.30)	1.30 (−0.67)	1.69 (−0.43)	1.40 (−0.11)	94.87 (−0.38)
every 10th	175	10	5	175 (0)	1122.03 (−35.36)	58.93 (−46.49)	65.36 (+5.14)	18.90 (+0.69)	1.60 (−0.47)	1.96 (−0.14)	1.44 (−0.02)	79.01 (+4.69)
every 10th	175	3	5	175 (0)	1126.32 (−32.44)	57.42 (−47.45)	65.29 (+4.59)	61.22 (+1.95)	1.67 (−0.42)	1.93 (−0.15)	1.44 (−0.02)	77.49 (+4.34)
every 10th	100	10	10	100 (0)	1257.59 (−13.62)	43.64 (−12.48)	69.82 (+3.61)	20.75 (+0.69)	3.09 (−0.38)	3.38 (−0.06)	1.62 (+0.01)	44.18 (+1.85)
every 10th	100	10	5	100 (0)	1259.97 (−0.90)	41.71 (−11.23)	65.33 (+2.17)	18.88 (+0.30)	2.01 (−0.13)	2.03 (−0.05)	1.41 (−0.01)	44.02 (+1.01)
every 10th	100	3	10	100 (0)	1265.45 (−11.20)	41.36 (−4.53)	69.31 (+2.65)	64.90 (+1.29)	3.20 (−0.34)	3.24 (−0.14)	1.59 (−0.01)	43.13 (+1.69)
every 10th	100	3	5	100 (0)	1261.03 (−0.42)	43.36 (−6.86)	64.87 (+2.28)	60.98 (+1.03)	2.02 (−0.16)	1.99 (−0.06)	1.40 (0.00)	43.95 (+1.16)
high density	3000	10	10	3000 (+528)	42.70 (−9.20)	8.74 (−3.26)	14.39 (−62.94)	11.61 (−11.07)	0.46 (−2.10)	0.50 (−3.16)	1.09 (−0.78)	99.19 (−0.50)
high density	3000	10	5	3000 (+801)	42.60 (−7.84)	8.84 (−1.75)	13.07 (−59.77)	11.39 (−10.00)	0.35 (−1.24)	0.38 (−1.86)	1.08 (−0.49)	98.26 (−0.53)
high density	2000	10	10	2000 (0)	44.84 (−9.71)	8.86 (−0.66)	61.89 (−18.93)	18.55 (−5.62)	0.71 (−1.80)	1.45 (−2.16)	1.32 (−0.50)	98.76 (+6.54)
high density	2000	10	5	2000 (0)	45.19 (−6.33)	8.90 (−0.20)	59.91 (−14.41)	17.97 (−3.84)	0.58 (−1.03)	1.24 (−0.99)	1.29 (−0.27)	97.59 (+1.52)
high density	1500	10	10	1500 (0)	48.57 (−7.45)	8.93 (−0.24)	81.60 (−3.47)	25.62 (−0.20)	1.01 (−1.37)	2.23 (−1.39)	1.52 (−0.24)	98.34 (+23.06)
high density	1500	10	5	1500 (0)	50.97 (−2.07)	9.43 (+0.70)	81.35 (+0.64)	24.22 (+0.51)	1.03 (−0.54)	2.03 (−0.21)	1.49 (−0.01)	91.84 (+13.29)
high density	1250	10	10	1250 (0)	54.79 (−2.29)	8.81 (−0.38)	88.85 (+2.56)	30.39 (+3.74)	1.59 (−0.78)	3.52 (−0.12)	1.78 (+0.05)	82.10 (+17.54)
high density	1250	10	5	1250 (0)	53.07 (−1.43)	9.06 (−0.03)	87.10 (+3.57)	25.84 (+1.34)	1.30 (−0.30)	2.22 (−0.02)	1.50 (+0.02)	74.98 (+8.12)
high density	1000	10	10	1000 (0)	56.61 (−1.19)	9.08 (−0.44)	90.96 (+3.76)	31.62 (+4.39)	1.84 (−0.53)	3.86 (+0.23)	1.82 (+0.12)	62.25 (+9.88)
high density	1000	10	5	1000 (0)	54.69 (−0.66)	9.39 (+0.46)	88.60 (+3.53)	26.54 (+1.48)	1.42 (−0.21)	2.27 (+0.05)	1.48 (+0.03)	57.90 (+3.83)
low density	500	10	10	500 (0)	106.38 (−18.18)	13.78 (+0.17)	69.76 (+5.00)	21.68 (+1.95)	1.29 (−2.25)	3.19 (−0.57)	1.54 (0.00)	98.07 (+23.42)
low density	500	10	5	500 (0)	121.12 (−2.13)	14.35 (+0.58)	60.33 (+0.93)	18.05 (+0.05)	1.73 (−0.55)	1.95 (−0.14)	1.31 (0.00)	78.48 (+8.25)
low density	500	3	10	500 (0)	115.93 (−8.52)	14.77 (+1.51)	67.23 (+2.49)	64.39 (+1.97)	1.94 (−1.65)	3.16 (−0.48)	1.49 (−0.01)	91.77 (+19.78)
low density	500	3	5	500 (0)	121.77 (−1.12)	14.99 (+0.22)	58.31 (−0.92)	57.45 (−0.48)	1.85 (−0.46)	1.87 (−0.20)	1.30 (−0.01)	75.90 (+5.42)
low density	250	10	10	250 (0)	130.47 (−3.79)	12.95 (−0.30)	72.40 (+4.86)	22.79 (+2.34)	3.17 (−0.56)	3.93 (+0.03)	1.49 (+0.01)	40.14 (+3.88)
low density	250	10	5	250 (0)	129.99 (−0.40)	13.33 (−0.11)	64.67 (+3.10)	19.20 (+0.58)	2.23 (−0.15)	2.13 (−0.04)	1.29 (+0.01)	37.03 (+0.61)
low density	250	3	10	250 (0)	131.27 (−1.41)	12.49 (−1.43)	70.65 (+5.07)	67.22 (+3.82)	3.34 (−0.56)	3.73 (−0.02)	1.46 (0.00)	37.62 (+2.10)
low density	250	3	5	250 (0)	129.46 (−0.91)	14.11 (+0.40)	64.38 (+2.51)	61.55 (+1.75)	2.24 (−0.26)	2.12 (+0.02)	1.28 (+0.01)	37.24 (+2.63)