On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment

Javier Alonso-Mora,1,2 Samitha Samaranayake,3 Alex Waller,4 Emilio Frazzoli,5 and Daniela Rus6

1Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139; 2School of Civil and Environmental Engineering, Cornell University, Ithaca, NY 14853; and 3Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139

Edited by Michael F. Goodchild, University of California, Santa Barbara, CA, and approved November 22, 2016 (received for review July 20, 2016)

Ride-sharing services are transforming urban mobility by providing timely and convenient transportation to anybody, anywhere, and anytime. These services present enormous potential for positive societal impacts with respect to pollution, energy consumption, congestion, etc. Current mathematical models, however, do not fully address the potential of ride-sharing. Recently, a large-scale study highlighted some of the benefits of car pooling but was limited to static routes with two riders per vehicle (optimally) or three (with heuristics). We present a more general mathematical model for real-time high-capacity ride-sharing that (i) scales to large numbers of passengers and trips and (ii) dynamically generates optimal routes with respect to online demand and vehicle locations. The algorithm starts from a greedy assignment and improves it through a constrained optimization, quickly returning solutions of good quality and converging to the optimal assignment over time. We quantify experimentally the trade-off between fleet size, capacity, waiting time, travel delay, and operational costs for low- to medium-capacity vehicles, such as taxis and van shuttles. The algorithm is validated with ~3 million rides extracted from the New York City taxi cab public dataset. Our experimental study considers ride-sharing with rider capacity of up to 10 simultaneous passengers per vehicle. The algorithm applies to fleets of autonomous vehicles and also incorporates rebalancing of idling vehicles to areas of high demand. This framework is general and can be used for many real-time multivehicle, multitask assignment problems.

Significance

Ride-sharing services can provide not only a very personalized mobility experience but also ensure efficiency and sustainability via large-scale ride pooling. Large-scale ride-sharing requires mathematical models and algorithms that can match large groups of riders to a fleet of shared vehicles in real time, a task not fully addressed by current solutions. We present a highly scalable anytime optimal algorithm and experimentally validate its performance using New York City taxi data and a shared vehicle fleet with passenger capacities of up to ten. Our results show that 2,000 vehicles (15% of the taxi fleet) of capacity 10 or 3,000 of capacity 4 can serve 98% of the demand within a mean waiting time of 2.8 min and mean trip delay of 3.5 min.


The authors declare no conflict of interest.

This article is a PNAS Direct Submission.

Freely available online through the PNAS open access option.

1Present address: Delft Center for Systems and Control, Delft Technical University, 2628 CD, Delft, Netherlands.

2To whom correspondence should be addressed. Email: J.AlonsoMora@tudelft.nl.

This article contains supporting information online at www.pnas.org/lookup/suppl/doi:10.1073/pnas.1611675114/-/DCSupplemental.
Here, we consider the problem of using a fleet of vehicles with varying passenger capacities, and, in contrast to ref. 5, we address both the problems of assigning vehicles to matched passengers and rebalancing—or repositioning—the fleet to service demand. We show how the unified problem of passenger and vehicle assignment can be solved in a computationally efficient manner at a large scale, thereby demonstrating the capability to operate a real-time MoD system with multiple service tiers (shared-taxi, shared-vans, and shared-buses) of varying capacity.

Whereas previous approaches to this problem have focused on heuristic-based solutions (16–18), we present a reactive anytime optimal algorithm. That is, an algorithm that efficiently returns a valid assignment of travel requests to vehicles and then refines it over time, converging to an optimal solution. If enough computational resources are available, the optimal assignment for the current requests and time would be found; otherwise, the best solution found so far is returned.

Traditional approaches that rely on an integer linear program (ILP) formulation, such as ref. 19, also provide anytime guarantees for the multivehicle-routing problem. However, in contrast to our approach, their applicability is limited to small problem instances, which in ref. 19 was 32 requests and 4 vehicles, with a computation cost of several minutes. We also rely on an ILP formulation, but because we do not explicitly model the edges of the road network in the ILP, our approach scales to much larger problem instances. We observe that instances such as New York City, with thousands of vehicles, requests, and road segments, can be solved in real time.

Our approach decouples the problem by first computing feasible trips from a pairwise shareability graph (5) and then assigning trips to vehicles. We show that this assignment can be posed as an ILP of reduced dimensionality. The framework allows for flexibility in terms of prescribing constraints such as (but not limited to) maximum user waiting times and maximum additional delays due to sharing a ride. We also extend the method to proactively rebalance the vehicle fleet by moving idle vehicles to areas of high demand. In summary, we present a framework for solving the real-time ride-pooling problem with (i) arbitrary numbers of passengers and trips, (ii) anytime optimal rider allocation and routing dependent on the fleet location, and (iii) online rerouting and assignment of riders to existing trips.

We quantify experimentally the performance tradeoffs between fleet size, capacity, waiting time, travel delay, and operational costs for low- and medium-capacity vehicles (such as taxis, vans, or minibuses) in a large urban setting. Detailed experimental results are presented for a subset of ~3 million rides extracted from the New York City taxicab public dataset. We show that 3,000 vehicles with a capacity of 2 and 4 could serve 94 and 98% of the demand with a mean waiting time of 3.2 and 2.7 min, and a mean delay of 1.5 and 2.3 min, respectively. To achieve 98% service rate, with comparable waiting time (2.8 min) and delay (3.5 min), a fleet of just 2,000 vehicles with a capacity of 10 was required. This fleet size is 15% of the active taxis in New York City (Movie S1). We also show that our approach is robust with respect to the density of requests and could therefore be applied to other cities.

Our system runs in real time and is particularly suited to autonomous vehicle fleets that can continuously reroute based on real-time requests. It can also rebalance idle vehicles to areas with high demand and is general enough to be applied to other multivehicle, multitask assignment problems.

**Passenger Assignment and Vehicle Routing**

We consider a fleet $\mathcal{V}$ of $m$ vehicles of capacity $\nu$, the maximum number of passengers each vehicle can have at any given time. We address the problems of both optimally assigning online travel requests to vehicles and finding optimal routes for the vehicle fleet. Each travel request consists of the time of request, a pickup location and a drop-off location.

We propose an anytime optimal algorithm for batch assignment of a set of requests $\mathcal{R} = \{r_1, \ldots, r_n\}$ to a set of vehicles $\mathcal{V} = \{v_1, \ldots, v_m\}$, which minimizes a cost function $C$, satisfies a set of constraints $\mathcal{Z}$, and allows for multiple passengers per vehicle. A passenger is a past request that has been picked up by a vehicle and that is now on route to its destination. We denote by $\mathcal{P}_v$ the set of passengers for vehicle $v \in \mathcal{V}$. In a second step, the method also allows to rebalance the fleet of vehicles by driving idle vehicles to areas of high demand, where those vehicles are

**Fig. 1.** Schematic overview of the proposed method for batch assignment of multiple requests to multiple vehicles of capacity $\nu$. The method consists of several steps leading to an integer linear optimization that provides an anytime optimal assignment. (A) Example of a street network with four requests (orange human, origin; red triangle, destination) and two vehicles (yellow car, origin; red triangle, destination of passenger). Vehicle 1 has one passenger, and vehicle 2 is empty. (B) Pairwise shareability RV-graph of requests and vehicles. Cliques of this graph are potential trips. (C) RTV-graph of candidate trips and vehicles which can execute them. A node (yellow triangle) is added for requests that cannot be satisfied. (D) Optimal assignment given by the solution of the ILP, where vehicle 1 serves requests 2 and 3 and vehicle 2 serves requests 1 and 4. (E) Planned route for the two vehicles and their assigned requests. In this case, no rebalancing step is required because all requests and vehicles are assigned.
likely to be required in the future. A schema of the method is shown in Fig. 1.

Our formulation is flexible with respect to physical and performance-related constraints that might need to be added. In our implementation, we consider the following. (i) For each request \( r \), the waiting time \( \omega_r \), given by the difference between the pickup time \( t^r_1 \) and the request time \( t^r \), must be below a maximum waiting time \( \Omega \), for example, 2 min. (ii) For each passenger or request \( r \) the total travel delay \( \delta_r = t^r_2 - t^r \) must be lower than a maximum travel delay \( \Delta \), for example, 4 min, where \( t^r_2 \) is the drop-off time and \( t^r_2 = t^r_2 + r(o, d) \) is the earliest possible time at which the destination could be reached if the shortest path between the origin \( o \) and the destination \( d \) was followed without any waiting time. The total travel delay \( \delta_r \) includes both the in-vehicle delay and the waiting time. Finally, (iii) for each vehicle \( v \), we consider a maximum number of passengers, \( n^{pass}_v \leq \nu \), for example, capacity 10.

We define the cost \( C \) of an assignment as the sum of delays \( \delta_r \) (which includes the waiting time) over all assigned requests and passengers, plus a large constant \( c_{ko} \) for each unassigned request. Given an assignment \( \Sigma \) of requests to vehicles, we denote by \( \mathcal{R}_v \) the set of requests that have been assigned to some vehicle and \( \mathcal{R}_o \) the set of unassigned requests, due to the constraints or the fleet size. Formally,

\[
C(\Sigma) = \sum_{v \in V} \sum_{r \in \mathcal{R}_v} \delta_r + \sum_{r \in \mathcal{R}_o} \delta_r + \sum_{r \in \mathcal{R}_o} c_{ko}. \tag{1}
\]

This constrained optimization problem is solved via four steps (Fig. 1), which are: computing a pairwise request-vehicle shareability graph (RV-graph) (Fig. 1B); computing a graph of feasible trips and the vehicles that can serve them (RTV-graph) (Fig. 1C); solving an ILP to compute the best assignment of vehicles to trips (Fig. 1D); and rebalancing the remaining idle vehicles (Fig. 1E).

Given a network graph with travel times, we consider a function \( travel(v, \mathcal{R}_o) \) for single-vehicle routing. For a vehicle \( v \), with passengers \( \mathcal{P}_v \), this function returns the optimal travel route \( \sigma_v \) to satisfy requests \( \mathcal{R}_o \). This route minimizes the sum of delays \( \sum_{r \in \mathcal{R}_o} \delta_r \) subject to the constraints \( \mathcal{Z} \) (waiting time, delay, and capacity). For low-capacity vehicles, such as taxis, the optimal path can be computed via an exhaustive search. For vehicles with larger capacity, heuristic methods such as Lin–Kernighan (20), Tabu search (21), or simulated annealing (22) may be used. Fig. 2, Right shows the optimal route for a vehicle with four passengers and an additional request.

The RV-graph (Fig. 1B) represents which requests and vehicles might be pair-wise shared and builds on the idea of shareability graphs proposed by ref. 5 but also includes the vehicles at their current state. Two requests \( r_1 \) and \( r_2 \) are connected if an empty virtual vehicle starting at the origin of one of them could pick up and drop off both requests while satisfying the constraints \( \mathcal{Z} \). A cost \( \delta_{r_1} + \delta_{r_2} \) is associated to each edge \( e(r_1, r_2) \). Likewise, a request \( r \) and a vehicle \( v \) are connected if the request can be served by the vehicle while satisfying the constraints \( \mathcal{Z} \), as given by \( travel(v, r) \). The edge is denoted by \( e(r, v) \).

Next, the cliques of the RV-graph—or regions for which its induced subgraph is complete—are explored to find feasible trips and compute the RTV-graph (Fig. 1C). A trip \( T = \{t_1, \ldots, t_m\} \) is a set of \( m \) requests to be combined in one vehicle. A trip is feasible if all of the requests can be picked up and dropped off by some vehicle, while satisfying the constraints \( \mathcal{Z} \).

This step computes feasible trips. There might be several trips of varying size that can service a particular request. In addition, more than one vehicle might be able to service a trip. The assignment step will later ensure that each request and vehicle are assigned to a maximum of one trip. The RTV-graph contains two types of edges: (i) edges \( e(r, T) \), between a request \( r \) and a trip \( T \) that contains request \( r \) (i.e., \( \exists e(r, T) \Leftrightarrow r \in T \)), and (ii) edges \( e(T, v) \), between a trip \( T \) and a vehicle \( v \) that can execute the trip (i.e., \( \exists e(T, v) \Leftrightarrow travel(v, T) \) is feasible). The cost \( \sum_{r \in \mathcal{R}_o \cup T} \delta_r \) sum of delays, is associated to each edge \( e(T,v) \).

The algorithm to compute the feasible trips and edges proceeds incrementally in trip size for each vehicle, starting from the request-vehicle edges in the RV-graph (SI Appendix, Algorithm 1). For computational efficiency, we rely on the fact that a trip \( T \) only needs to be checked for feasibility if there exists a vehicle \( v \) for which all of its subtrips \( T' = T \setminus r \) (obtained by removing one request) are feasible and have been added as edges \( e(T', v) \) to the RTV-graph.

Next, we compute the optimal assignment \( \Sigma_{opt} \) of vehicles to trips. This optimization is formalized as an ILP, initialized with a greedy assignment obtained directly from the RTV-graph. To compute the greedy assignment \( \Sigma_{greedy} \), trips are assigned to vehicles iteratively in decreasing size of the trip and increasing cost (sum of travel delays). The idea is the maximize the amount of requests served while minimizing the cost (SI Appendix, Algorithm 2).

The optimization problem is formulated in Algorithm 1. A binary variable \( \epsilon_{i,j} \in \{0, 1\} \) is introduced for each edge \( e(T_i, v_j) \) between a trip \( T_i \in T \) and a vehicle \( v_j \in \mathcal{V} \) in the RTV-graph. If \( \epsilon_{i,j} = 1 \), then vehicle \( v_j \) is assigned to trip \( T_i \). We denote by \( \mathcal{E}_{RTV} \) the set of \( \{i,j\} \) indices for which an edge \( e(T_i, v_j) \) exists in the RTV-graph, i.e., the set of possible pickup trips. An additional binary variable \( \chi_k \in \{0, 1\} \) is introduced for each request \( r_k \in \mathcal{R} \). These variables are active, i.e., \( \chi_k = 1 \), if the associated request \( r_k \) can not be served by any vehicle and is ignored. The set of variables is then \( \mathcal{X} = \{\epsilon_{i,j}, \chi_k; \forall e(T_i, v_j) \text{ edge in RTV-graph and } \forall r_k \in \mathcal{R}\} \).

The cost terms \( c_{rj} \) are the sum of delays for trip \( T_i \) and vehicle \( v_j \) pickup (stored in the \( e(T_i, v_j) \) edge of the RTV-graph) and \( c_{ko} \) is a large constant to penalize ignored requests.
Two types of constraints are included. Line 3 in Algorithm 1 imposes that each vehicle is assigned to one trip at most. Line 4 in Algorithm 1 imposes that each request is assigned to a single vehicle or ignored. In these constraints, three sets appear: the set of trips that can be serviced by a vehicle, the set of trips that contain request $k$, and the set of vehicles that can service trip $i$. A request might be rematched to a different vehicle in subsequent iterations as long as its waiting time does not increase. A request might be rematched to a different vehicle in subsequent iterations as long as its waiting time does not increase. A request might be rematched to a different vehicle in subsequent iterations as long as its waiting time does not increase.

This method is well suited for online execution to assign incoming requests $r(t)$ to a fleet of vehicles for which a pool of requests $\mathcal{R}$ is maintained where (i) new requests are added as they are received and (ii) requests are removed when they are either (a) picked up by a vehicle or (b) could not be successfully matched to any vehicle within the maximum waiting time (they are ignored).

Requests are collected during a time window (e.g., 30 s), after which they are assigned in batch to the different vehicles. If a request is matched to a vehicle at any given iteration, its latest time of pickup is reduced to the expected pickup time by that vehicle and the cost $\chi_k$ of ignoring it is increased for subsequent iterations. A request might be rematched to a different vehicle in subsequent iterations as long as its waiting time does not increase and until it is picked up by some vehicle. Once a request is picked up, it remains in that vehicle and cannot be rematched—the vehicle may still pick additional passengers. In each iteration, the new assignments of requests to vehicles guarantees that the current passengers are dropped off within the maximum delay constraint.

After the assignment, due to fleet imbalances, the set $\mathcal{R}_{\text{idle}}$ of unassigned requests may not be empty, and some empty vehicles $\mathcal{V}_{\text{idle}}$ may still be unassigned to any request. These imbalances may occur when the idle vehicles are in areas far away from the area of current requests and due to the maximum waiting time and delay constraints and vehicle capacity. Under the assumptions that (i) ignored requests may wait longer and request again, (ii) it is likely that more requests occur in the same area where all requests cannot be satisfied, and (iii) there are not enough requests in the neighborhood of the idle cars, we propose the following approach to rebalance the fleet by moving only the idle vehicles.

To rebalance the vehicle fleet, after each batch assignment, the vehicles in $\mathcal{V}_{\text{idle}}$ are assigned to requests in $\mathcal{R}_{\text{idle}}$ to minimize the sum of travel times, with the constraint that either all requests or all of the vehicles are assigned. We first compute the travel time of each individual idle vehicle in $\mathcal{V}_{\text{idle}}$ to pick each ignored request in $\mathcal{R}_{\text{idle}}$ and then obtain the optimal assignment via a linear program (SI Appendix, Algorithm 4). In this approach, all requests can be satisfied, some vehicles may remain idle, saving fuel and distance traveled, which is the case at nighttime.

Complexity. The number of variables in the ILP is equal to the number of edges $e(T,v)$ in the RTV-graph plus the number of requests. In the worst case, the number of variables is of order $\mathcal{O}(mn^2)$ but only reached with complete RV- and RTV-graphs, where all vehicles can serve all requests and all requests can be combined with each other. In practice, the number of variables is orders of magnitudes lower and related to the size of the cliques in the RV-graph. The number of constraints is $n + m$.

### Algorithm 1. Optimal assignment

1. Initial guess: $\Sigma_{\text{greedy}}$
2. $\Sigma_{\text{optimal}} := \arg \min_{\Sigma} \{ \sum_{i, j \in E_T} c_{ij} x_{ij} + \sum_{k \in \{1, \ldots, n\}} \gamma_k \chi_k \}$
3. $\sum_{i \in E_T} x_{ij} \leq 1 \quad \forall v \in V$
4. $\sum_{i \in E_T} x_{ij} + \chi_k = 1 \quad \forall \chi_k \in \mathcal{R}$

### Results

We assess the performance of a MoD fleet controller using the proposed algorithm, against real data from an arbitrarily chosen representative week, from 0000 hours Sunday, May 5, 2013, to 2359 hours, Saturday May 11, 2013, from the publicly available dataset of taxi trips in Manhattan, New York City (23). This dataset contains for each day the time and location of all of the pickups and drop-offs executed by each of the 13,586 active taxis. From these data, we extract all of the requests (origin and destination within Manhattan) and consider the time of request equal to the time of pickup. We consider the complete road network of Manhattan (4,092 nodes and 9,453 edges), with the travel time on each edge (road segment) of the network given by the daily mean travel time estimate, computed using the method in ref. 5. Shortest paths and travel times between all nodes are then precomputed and stored in a lookup table.

We perform a simulation of the evolution of the taxi fleet, where vehicles are initialized at midnight at sampled positions from a historical demand distribution and continuously travel to pick up and drop off passengers to satisfy the real requests extracted from the dataset. Requests are collected during a
time window, 30 s in our experiments, after which they are assigned in batch to the different vehicles. Past requests are kept in the requests pool until picked up and can be reassigned if a better match is found before pickup. Each day contains between 382,779 (Sunday) and 460,700 (Friday) requests, and the running pool of requests contains up to 2,000 requests at any given time. The method is robust both with respect to the chosen time window and the density of demands, as shown in SI Appendix, VI. Robustness Analysis in results with a time window between 10 and 50 s, and with half/double the amount of requests (∼220,000/∼880,000 per day) in New York City.

We analyze several metrics, with different vehicle fleet sizes (m ∈ {1,000, 2,000, 3,000} vehicles), vehicle capacities (χ ∈ {1, 2, 4, 10} passengers), and maximum waiting times (Ω ∈ {120, 300, 420} s). The maximum trip delay ∆ is double the maximum waiting time and includes both the waiting time ω and the inside-vehicle travel delay. Our analysis shows that, thanks to high-capacity ride-sharing, a reduced fleet of vehicles (below 25% of the active taxis in New York City) is able to satisfy 99% of the requests, with a mean waiting time and delay of about 2.5 min. All results in this section include rebalancing of idle vehicles to unassigned requests; experimentally, we observed that the rebalancing step contributed an increase in the service rate of about 20% (SI Appendix, Table II). Movie S1 shows the evolution of the taxi fleet in New York City for a subset of experiments.

High vehicle occupancy is achieved in times of high demand, with a large number of the trips being shared. In Fig. 2, we observe that many vehicles are located in mid-Manhattan and contain three or four passengers. Fig. 3 shows that the occupancy depends on the fleet size, capacity, and the maximum waiting/delay time. Lower fleet size, larger capacity and longer waiting/delay times increase the possibilities for ride-sharing and lead to higher mean vehicle occupancy. In Fig. 4, we observe that during peak hours, a small fleet of high-capacity vehicles does indeed operate at high occupancy. For a fleet of 1,000 vehicles of capacity 10, we observe that, during peak time (1800 hours) of a Friday, 10% of the vehicles have eight or more passengers, 40% of the vehicles have six or more, 80% have three or more, and 98% have at least one passenger. For a fleet of 2,000 vehicles of capacity four, we observe that, at the same peak time, over 70% of them have at least three passengers onboard.

We observe that the value of fleets with larger passenger capacities increases with larger Ω and Δ values, as expected, because passengers are willing to incur a larger personal time penalty. High-capacity vehicles are also more important when the fleet size is smaller, because seating capacity might be a bottleneck with smaller fleets. For instance (Fig. 5A), a fleet of 1,000 vehicles with a capacity of 10 can satisfy almost 80% of the requests with Ω = 420 s, compared with below 30% for a single-rider taxi, for a net gain of over 50%. However, with a larger fleet of 3,000 vehicles and Ω = 120 s, the benefit is only about 15%. Interestingly, if longer waiting times and delays are
allowed, $\Omega = 420$ s, a fleet of 3,000 vehicles with a capacity of 2, 4, and 10 could serve 94, 98, and 99% of the demand. To achieve 98% service rate, a fleet of just 2,000 vehicles with a capacity of 10 was required, which represents a reduction of the fleet size to 15% of the active taxi fleet in New York City.

As expected, the in-car travel delay does increase with the increase in vehicle capacity (Fig. 5B). Nonetheless, that increase seems practically negligible—well below 100 s—once ride-sharing is allowed. Furthermore, the mean waiting time does in fact decrease as vehicle capacity is increased (Fig. 5C).

For a fleet size of 1,000 vehicles and $\Delta = 420$ s, high-capacity vehicles not only improved the service rate but also achieved a reduction in mean waiting time of over 100 s, which partially offsets the increased in-car delay. In particular, we observe that 3,000 vehicles with a capacity of 2 and 4 could serve 94 and 98% of the demand, with a mean waiting time of 3.2 and 2.7 min and a mean delay of 1.5 and 2.3 min, respectively. To achieve 98% service rate, with comparable waiting time (2.8 min) and delay (3.5 min), a fleet of just 2,000 vehicles with a capacity of 10 was required.

We also observed that increasing the vehicle capacity not only increases the service rate but also reduces the mean distance traveled by the vehicles in the fleet (Fig. 5D), potentially leading to a reduction in costs, congestion, and pollution. We also observe that, with our online method, about 90% of the rides were shared. The number of shared rides slightly increases with $\Delta$ and decreases with the fleet size (Fig. 5E). Finally, we note that our approach is real-time capable (Fig. 5F).

In our setup, for $\Omega \leq 300$ s, the method is executed in less than 30 s, which is the period for which requests are collected.

**Conclusion**

In this paper, we introduced a reactive anytime optimal method with scalable real-time performance for assigning passenger requests to a fleet of vehicles of varying capacity. We quantify experimentally the tradeoff between fleet size, capacity, waiting time, travel delay, and operational costs for low- and medium-capacity vehicles, such as taxis or vans in a large-scale city dataset. Under the assumption of one person per ride, we show that 98% of the taxi rides currently served by over 13,000 taxis could be served with just 3,000 taxis of capacity four. We observe that a vehicle capacity of two is sufficient for ride-sharing when a small trip delay of 2 min is imposed. If a maximum delay of 5 min or more (comparable to the time spent retrieving a car from parking) is allowed, higher-capacity vehicles (i) increase the service rate significantly, (ii) reduce the waiting time, and (iii) reduce the distance traveled by each vehicle. Our analysis shows that a ride-pooling service can provide a substantial improvement in urban transportation systems and that the system parameters such as vehicle capacity and fleet size depend on quality of service requirements and demand.

**ACKNOWLEDGMENTS.** We thank G. Resta, P. Santi, and C. Ratti for sharing the graph of Manhattan and the estimated travel times of ref. 5. This work was supported in part by the Office of Naval Research Grant N00014-12-1-1000 and the Massachusetts Institute of Technology–Singapore Alliance on Research and Technology under the Future of Urban Mobility.
This movie shows experimental results of the proposed real-time-capable, anytime-optimal route-planning and -assignment algorithm with real requests (about 440,000 per day) in Manhattan, New York City. We show four examples: (i) one day with a 1,000 vehicle fleet with a capacity of 10 passengers per vehicle; (ii) a comparison of the influence of the fleet size (1,000, 2,000, and 3,000 vehicles); (iii) a comparison of the influence of the vehicle capacity (1, 4, and 10 passengers per vehicle); and (iv) a comparison of the influence of the day of the week.

**Movie S1**

**Other Supporting Information Files**

[SI Appendix (PDF)](www.pnas.org/cgi/content/short/1611675114)
On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment

--- Supplemental Material ---

Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli and Daniela Rus

Abstract

Ride sharing services are transforming urban mobility by providing timely and convenient transportation to anybody, anywhere, anytime. These services present enormous potential for positive societal impacts with respect to pollution, energy consumption, congestion, etc. Current mathematical models, however, do not fully address the potential of ride sharing. Recently, a large-scale study highlighted some of the benefits of car pooling, but was limited to static routes with two riders per vehicle (optimally) or three (with heuristics). We present a more general mathematical model for real-time high-capacity ride sharing that (1) scales to large numbers of passengers and trips, and (2) dynamically generates optimal routes with respect to online demand and vehicle locations. The algorithm starts from a greedy assignment and improves it via a constrained optimization, quickly returning solutions of good quality and converging to the optimal assignment over time. For the first time, we quantify experimentally the trade-off between fleet size, capacity, waiting time, travel delay, and operational costs for low to medium capacity vehicles, such as taxis and van shuttles. The algorithm is validated with approximately 3 million rides extracted from the New York City taxicab public dataset. Our experimental study considers ride sharing with rider capacity of up to ten simultaneous passengers per vehicle. The algorithm applies to fleets of autonomous vehicles and also incorporates rebalancing of idling vehicles to areas of high demand. This framework is general and can be used for many real-time multi-vehicle, multi-task assignment problems.
I. PROBLEM STATEMENT

A. Definitions

Consider two positions in space, \(q_1\) and \(q_2\), which can be encoded by their latitude and longitude coordinates. A function \(\tau(q_1, q_2)\) to compute the travel time from \(q_1\) to \(q_2\) is required. When a network representation of the map is available, standard techniques for efficiently computing shortest paths can be used \([1]\). The best methods can compute shortest paths on networks with 70 million edges in less than a millisecond.

A request \(r\) is defined by a tuple \(\{o_r, d_r, t^p_r, t^{pl}_r, t^{pd}_r, t^d_r, t^*_r\}\), indicating its origin \(o_r\), its destination \(d_r\), the time of the request \(t^p_r\), the latest acceptable pick-up time \(t^{pl}_r\) (initially given by \(t^{pl}_r = t^p_r + \Omega\) with \(\Omega\) the maximum waiting time), the pick-up time \(t^p_r\), the expected drop off time \(t^d_r\), and the earliest possible time at which the destination could be reached \(t^*_r = t^p_r + \tau(o_r, d_r)\).

The current state of a vehicle \(v\) is given by a tuple \(\{q_v, t_v, P_v\}\), indicating its current position \(q_v\), the current time \(t_v\) and its passengers \(P_v = \{p_1, \ldots, p_{n^{pass}}\}\). A passenger \(p\) is a request that has been picked-up by a vehicle.

A trip \(T = \{r_1, \ldots, r_{n_T}\}\) is a set of requests that can be combined and served by a single vehicle. A trip may have one or more candidate vehicles for execution.

Consider a fixed size \(m\) of the vehicle fleet, a fixed vehicle capacity \(\nu\) (maximum number of passengers in a vehicle), and a maximum delay \(\Delta\) tolerated by a request \(r\). The delay is given by the difference between the drop-off time and the minimal time to reach the destination, \(t^d_r - t^*_r\).

B. Problem statement

In this work we propose a general framework for the multi-vehicle multi-request routing capable of addressing three main problems of ride-sharing or car-pooling.

- Compute, incrementally, an optimal assignment of a set of requests to a set of vehicles of given capacity.
- Allow for continuous operation and assignment of incoming requests to a fleet of vehicles.
- Enable rebalancing of the fleet of vehicles.

Problem 1 (Batch assignment). Consider a set of requests \(R = \{r_1, \ldots, r_n\}\), a set of vehicles \(V = \{v_1, \ldots, v_m\}\) at their current state including passengers, and a function to compute travel times on the road network. Compute the optimal assignment \(\Sigma\) of requests to vehicles that minimizes a cost function \(C\) and that satisfies a set of constraints \(Z\), including a maximum capacity \(\nu\) of passengers per vehicle.

In particular, and not limited to, we will consider the following set of constraints \(Z\):
- For each request \(r\), its maximum waiting time \(t^p_r \leq t^{pl}_r \leq t^*_r + \Omega\).
- For each request or passenger \(r\), its maximum travel delay \(t^{pd}_r \leq t^*_r + \Delta\).
- For each vehicle, the maximum number of passengers \(n^{pass}_v \leq \nu\) at any given time.

Ideally all the requests shall be assigned to a vehicle, but given the constraints, this might not always be the case. Denote by \(R_{ok}\) the set of requests assigned to a vehicle and \(R_{ko}\) the set of requests that are not served by any vehicle.

We define the cost \(C\) of an assignment \(\Sigma\) as the sum of travel delay over all passengers and assigned requests plus a large enough cost \(c_{ko}\) for each non-assigned request. Formally,

\[
C(\Sigma) = \sum_{v \in V} \sum_{p \in P_v} \delta_p + \sum_{r \in R_{ok}} \delta_r + \sum_{r \in R_{ko}} c_{ko}.
\]

The method here presented is general and admits alternative constraints and cost function.

Given that the problem at hand is NP hard, obtaining an optimal assignment can be computationally expensive. For practical applications it is required that a sub-optimal solution is returned within an allocated
runtime budget, which might be improved incrementally up to optimality. We refer to this as an anytime algorithm.

Problem 2 (Anytime property). Devise a method to solve Problem 1 this is to obtain the optimal assignment, which incrementally improves the quality of the solution. Given a limited computational budget, a sub-optimal assignment $\Sigma$ of requests to vehicles, is returned.

The proposed method of Section II addresses this two problems. Additionally, for real-time fleet management, the method shall apply to continuous discovery and assignment of incoming requests.

Problem 3 (Continuous assignment). Consider a fixed number of vehicles $m$ and incoming requests $r(t)$ with request time $t^r = t$. Assign requests to a vehicles while respecting the constraints $\mathcal{Z}$ and minimizing the cost $C$.

The proposed approach, described in Section IV is to perform batch assignment of the requests within a short time span, for example every 30 seconds, to the fleet of vehicles. Problem 2 is invoked with the predicted state of the fleet at the assignment time and the cumulated requests. Requests that have not been picked-up by a vehicle within the previous assignment round are kept in the pool for assignment.

After the assignment, due to fleet imbalances, the set $\mathcal{R}_{ko}$ of unassigned requests may not be empty, and some empty vehicles $\mathcal{V}_{idle}$ may still be unassigned to any request. This may occur when the idle vehicles are in areas far away from the area of current request, and due to the maximum waiting time and delay constraints and vehicle capacity. Under the assumptions that, (a) ignored requests may wait longer and request again, (b) it is likely that more requests occur in the same area where all requests can not be satisfied and (c) there are not enough requests in the neighborhood of the idle cars, we propose the following approach to rebalance the fleet by moving only the idle vehicles.

Problem 4 (Rebalancing). Consider Problem 3. Consider Problem 1 as a batch assignment to solve Problem 3. Compute an assignment of the idle vehicles $\mathcal{V}_{idle}$ to the ignored requests in $\mathcal{R}_{ko}$ to minimize the sum of waiting times, with the constraint that either all requests or all the vehicles are assigned.
II. Method

In this section we describe a solution to Problem 1 and Problem 2.

A. Overview

Given a set of requests $\mathcal{R} = \{r_1, \ldots, r_n\}$ and a set of vehicles $\mathcal{V} = \{v_1, \ldots, v_m\}$ at their current state including passengers, the algorithm computes an incrementally optimal solution and consists of the following three steps, shown in the high-level overview of Figure 1:

1) Pairwise request-vehicle RV graph: The first step is to compute (a) which requests can be pairwise combined, taking into account both their origin and destination, and (b) which vehicles can serve which requests individually, given their current passengers.

2) Request-trip-vehicle RTV graph: The second step is to explore the regions of the graph for which its induced subgraph is complete, or cliques, to find trips (groups of requests) that can be combined and picked-up by a vehicle, while satisfying all the constraints. A request may form part of several potential trips, and a trip might have several candidate vehicles for execution.

3) Optimal assignment: The last step is to, given the RTV graph of potential trips and pick-up vehicles, compute the optimal assignment of one trip per vehicle at most. This is converted into an Integer Linear Program ILP, which is solved incrementally.

B. Optimal route for fixed vehicle and passengers

Consider a vehicle $v$, with a set of passengers $\mathcal{P}_v$ and a small set of assigned requests $\mathcal{R}_v$, which may have different request times. The optimal travel path $\sigma_v$ for the vehicle that minimizes the sum of delays $\sum_{r \in \mathcal{P}_v \cup \mathcal{R}_v} (t_d^r - t_*)$ subject to the constraints $Z$ (waiting time, delay and capacity) is given by a function

$$\text{travel}(v, \mathcal{R}_v),$$

which returns “invalid” if no path exists. This vehicle routing problem is a computationally hard problem.

For vehicles with low capacity and fixed requests, such as taxis, the optimal path can be computed via an exhaustive search. For vehicles with larger capacity, heuristic methods such as Lin-Kernighan [3], Tabu search [2] or Simulated Annealing [5] may be used. In Figure 2 a representative vehicle with two passengers is shown, with its past path and its optimal path for picking up three additional requests and dropping off all passengers and requests. The vehicle has capacity four and it will drop-off one of the passengers before picking up the last one.
Fig. 1. Schematic overview of the proposed method for batch assignment of multiple requests to multiple vehicles of capacity $\nu$. The method consists of several steps leading to an integer linear optimization which provides an anytime optimal assignment. (a) Example of a street network with four requests (orange human = origin, red triangle = destination) and two vehicles (yellow car = origin, red triangle = destination of passenger). Vehicle 1 has one passenger and vehicle 2 is empty. (b) Pairwise shareability RV-graph of requests and vehicles. Cliques of this graph are potential trips. (c) RTV-graph of candidate trips and vehicles which can execute them. A node (yellow triangle) is added for requests that can not be satisfied. (d) Optimal assignment given by the solution of the ILP, where vehicle 1 serves requests 2 and 3 and vehicle 2 serves requests 1 and 4. (e) Planned route for the two vehicles and their assigned requests. In this case no rebalancing step is required since all requests and vehicles are assigned.

Fig. 2. Past path (yellow) and optimal path (brown) for a vehicle (yellow dot) with two passengers and scheduled pick up of three requests (star) and drop-offs (triangle).
C. Pairwise graph of vehicles and requests (RV-graph)

The first step of the method is to compute (a) which requests can be pairwise combined, and (b) which vehicles can serve which requests individually, given their current passengers. This step builds on the idea of share-ability graphs proposed by [6], but it is not limited to the requests and includes the vehicles at their current state as well.

Two requests $r_1$ and $r_2$ are connected in the graph if they can potentially be combined. This is, if a virtual vehicle starting at the origin of one of them could pick-up and drop-off both requests while satisfying the constraints $Z$ of maximum waiting time and delay. A cost $\sum_{r=\{1,2\}} (t^d_r - t^*_r)$ can be associated to each edge $e(r_1, r_2)$.

Likewise, a request $r$ and a vehicle $v$ are connected if the request can be served by the vehicle while satisfying the constraints $Z$. This is, if $\text{travel}(v, r)$ returns a valid trip that drops the current passengers of the vehicle and the picked request $r$ within the specified maximum waiting and delay times. The edge is denoted by $e(r, v)$.

Limits on the maximum number of edges per node can be imposed, trading-off optimality at the later stages. Speed-ups such as the ones proposed in T-share [4] could be employed in this stage to prune the most likely vehicles to pick up a request.

This graph, denoted $RV$-graph, gives an overview of which requests and vehicles might be shared. In Figure 3 an example of the RV-graph is shown with 90 requests and 30 vehicles.

Fig. 3. Example of a pairwise RV-graph for 90 requests (star) and 30 vehicles (circle) with edges between two requests in dotted red and between a request and a vehicle in solid green. The maximum waiting time and delay are three and six minutes in this example.
The second step of the method is to explore the regions of the RV-graph for which its induced subgraph is complete, or cliques, to find feasible trips. Recall that a trip $T$ is defined by a set of requests $T = \{r_1, \ldots, r_{n_T}\}$. A trip is feasible if the requests can be combined, picked-up and dropped-off by some vehicle, while satisfying the constraints $Z$.

A request may form part of several feasible trips of varying size, and a trip might admit several different vehicles for execution. The request-trip-vehicle RTV-graph contains edges $e(r, T)$, between a request $r$ and a trip $T$ and feasible edges $e(T, v)$, between a trip $T$ and a vehicle $v$. Namely,

\[ \exists e(r, T) \iff r \in T \]  
\[ \exists e(T, v) \iff \text{travel}(v, T) = "\text{valid}" \]

The following observations are made to efficiently compute feasible trips.

**Lemma 1 (Cliques).** A trip $T$ can be feasible only if a clique in the RV-graph exists for all requests in $T$ and some vehicle $v$. Namely, if $T$ is valid, then,

\[ \exists v \in V \text{ such that } \forall r_1, r_2 \in T, e(r_1, r_2) \text{ and } e(r_1, v) \text{ exist} \]

A stronger requirement for existence is the following, which leads to the idea of incrementally computing trips of larger size.

**Lemma 2 (Sub-feasibility).** A trip $T$ can be feasible only if there exists a vehicle $v$ for which, for all $r \in T$, the sub-trips $T' = T \setminus r$ are feasible (a sub-trip $T'$ contains all the requests of $T$ but one). Namely,

$T$ feasible $\Rightarrow \exists v \in V$ such that $\forall r \in T$, $e(T \setminus r, v)$ exists.

Therefore, a trip $T$ only needs to be checked for existence if there exists a vehicle $v$ for which all of its sub-trips $T'$ present an edge $e(T', v)$ in the RTV-graph.

The algorithm to compute the feasible trips and edges proceeds incrementally in trip size for each vehicle, as shown in Algorithm 1 where $T$ is the list of feasible trips. With each node $e(T, v)$, the cost $C$ of the trip and pick-up is stored. For each vehicle, a timeout can be set after which no more trips are explored. This leads to suboptimality of the solution, but faster computation, removing longer trips.

Note that steps 7 and 12 of the Algorithm can be efficiently implemented by employing ordered lists with respect to the request ids. This step can be parallelized among the vehicles.
Algorithm 1 Generation of RTV-graph

1: \( \mathcal{T} = \emptyset \)

2: for each vehicle \( v \in \mathcal{V} \) do

3: \( \mathcal{T}_k = \emptyset \ \forall k \in \{1, \ldots, \nu\} \)

4: [Add trips of size one]

5: for \( e(r, v) \) edge of RV-graph do

6: \( \mathcal{T}_1 \leftarrow T = \{r\}; \) Add \( e(r, T) \) and \( e(T, v) \)

7: [Add trips of size two]

8: for all \( \{r_1\}, \{r_2\} \in \mathcal{T}_1 \) and \( e(r_1, r_2) \in \text{RV-graph} \) do

9: if \( \text{travel}(v, \{r_1, r_2\}) = \text{valid} \) then

10: \( \mathcal{T}_2 \leftarrow T = \{r_1, r_2\}; \) Add \( e(r_i, T) \) and \( e(T, v) \)

11: [Add trips of size \( k \)]

12: for \( k \in \{3, \ldots, \nu\} \) do

13: for all \( T_1, T_2 \in \mathcal{T}_{k-1} \) with \( |T_1 \cup T_2| = k \) do

14: Denote \( T_1 \cup T_2 = \{r_1, \ldots, r_k\} \)

15: if \( \forall i \in \{1, \ldots, k\}, \{r_1, \ldots, r_k\} \setminus r_i \in \mathcal{T}_{k-1} \) then

16: if \( \text{travel}(v, T_1 \cup T_2) = \text{valid} \) then

17: \( \mathcal{T}_k \leftarrow T = T_1 \cup T_2 \)

18: Add \( e(r_i, T) \), \( \forall r_i \in T \), and \( e(T, v) \)

19: \( \mathcal{T} \leftarrow \bigcup_{i \in \{1, \ldots, \nu\}} \mathcal{T}_i \)
E. Optimal assignment of trips to vehicles

The last step of the method is to, given the RTV-graph of potential trips and pick-up vehicles, compute the optimal assignment of vehicles to trips. This is formalized as an Integer Linear Program (ILP).

First, a greedy solution is computed, which serves as initial point for the ILP optimization.

1) Greedy assignment: Trips are assigned to vehicles iteratively in decreasing size of the trip and increasing cost. The idea is the maximize the amount of requests served while minimizing cost. The method to greedily maximize the cost function \( C \) of Equation 1 is described in Algorithm 2.

Algorithm 2 Greedy assignment

1: \( R_{ok} = \emptyset; \ V_{ok} = \emptyset \)
2: for \( k = \nu; k > 0; k \leftarrow \) do
3: \( S_k := \) sort \( e(T, v) \) in increasing cost, \( \forall T \in \mathcal{T}_k, v \in \mathcal{V} \)
4: while \( S_k \neq \emptyset \) do
5: \( \) pop \( e(T, v) \leftarrow S_k \)
6: if \( \forall r \in T, r \notin R_{ok} \) and \( v \notin V_{ok} \) then
7: \( R_{ok} \leftarrow \{ \forall r \in T \}; \ V_{ok} \leftarrow v \)
8: \( \sum_{\text{greedy}} \leftarrow e(T, v) \)

In the following, the optimization method to optimally assign trips to vehicles is described.

2) Variables: A binary variable \( \epsilon_{i,j} \in \{0, 1\} \) is introduced for each edge \( e(T_i, v_j) \) between a trip \( T_i \in \mathcal{T} \) and a vehicle \( v_j \in \mathcal{V} \) in the RTV-graph. If \( \epsilon_{i,j} = 1 \) then vehicle \( v_j \) is assigned to trip \( T_i \). Denote by \( \mathcal{E}_{TV} \) the set of \( \{i, j\} \) indexes for which an edge \( e(T_i, v_j) \) exists in the RTV-graph.

An additional binary variable \( \chi_k \in \{0, 1\} \) is introduced for each request \( r_k \in \mathcal{R} \). These variables are active, \( \chi_k = 1 \), if the associated request \( r_k \) can not be served by any vehicle and is ignored.

Denote the set of variables
\[
\mathcal{X} = \{ \epsilon_{i,j}, \chi_k; \text{ node in RTV-graph } \forall r_k \in \mathcal{R} \}. \tag{5}
\]

3) Cost: The cost function, equivalent to \( C(\Sigma) \) in Equation 1 is given by
\[
C(\mathcal{X}) := \sum_{i,j \in \mathcal{E}_{TV}} c_{i,j} \epsilon_{i,j} + \sum_{k \in \{0, \ldots, n\}} c_{ko} \chi_k, \tag{6}
\]
where the individual costs are given by the sum of delays,
\[
c_{i,j} = \sum_{r \in T_i} (t_r^d - t_r^*) \text{, as returned by } travel(v_j, T_i), \tag{7}
\]
and \( c_{ko} \) is a large enough constant to penalize ignored requests.

4) Constraints: Two types of constraints are included, as follows.
Each vehicle is assigned to a single trip at most,
\[
\sum_{i \in \mathcal{T}_j^V} \epsilon_{i,j} \leq 1 \ \forall v_j \in \mathcal{V}, \tag{8}
\]
where \( \mathcal{T}_j^V \) denotes the indexes \( i \) for which an edge \( e(T_i, v_j) \) exists in the RTV-graph.

Each request is assigned to a vehicle or ignored,
\[
\sum_{i \in \mathcal{I}_k^R} \sum_{j \in \mathcal{I}_j^T} \epsilon_{i,j} + \chi_k = 1 \ \forall r_k \in \mathcal{R}, \tag{9}
\]
where \( \mathcal{I}_k^R \) denotes the indexes \( i \) for which an edge \( e(r_k, T_i) \) exists in the RTV-graph and \( \mathcal{I}_j^T \) denotes the indexes \( j \) for which an edge \( e(T_i, v_j) \) exists in the RTV-graph. This is, the trips of which the request forms part and the vehicles that can service each of those trips.
5) Assignment: The optimal assignment is found by solving an ILP optimization defined by the aforementioned variables, cost and constraints, as shown in Algorithm 3.

Starting from the greedy assignment, the ILP can be solved with state of the art solvers via branch and bound with heuristics and duality gap checking. These algorithms can be parallelized and return a suboptimal solution if stopped before convergence.

**Algorithm 3 Optimal assignment**

1: Initial guess: $\Sigma^{\text{greedy}}$
2: $\Sigma^{\text{optimal}} := \arg \min \mathcal{C}(\mathcal{X})$ (Eq. 6)
3: s.t. $\sum_{i \in T'_v} \epsilon_{i,j} \leq 1$ $\forall j \in \mathcal{V}$
4: $\sum_{i \in T'_r} \sum_{j \in T'_i} \epsilon_{i,j} + \chi_k = 1$ $\forall r_k \in \mathcal{R}$
III. THEORETICAL GUARANTEES

A. Complexity

The number of binary variables in the ILP is equal to the number of edges $e(T,v)$ in the RTV graph plus the number of requests. In the worst case, where all possible trip combinations are explored and feasible, it is

$$m \frac{n!}{\nu!(n-\nu)!} \sim O(m n^\nu),$$

where $m$ is the number of vehicles, $n$ the number of requests and $\nu$ the maximum capacity of the vehicles. Nonetheless, this value would only be reached in a complete RV-graph where all vehicles can serve all requests and all requests can be combined with each other. This is, where all possible trip combinations up to capacity $\nu$ are feasible. In practice, the number of variables is orders of magnitude lower and at most the number of cliques in the RV-graph.

The number of constraints is $n + m$, the number of requests plus the number of vehicles.

B. Anytime optimality and correctness

**Theorem 1** (Optimality). The method is optimal, given enough computational time.

If all the steps of the method are executed until all possible trips and assignments are explored, the ride-vehicle assignment method guarantees optimality of the assignment and routes, given the constraints $Z$.

**Proof:** For any particular assignment between a trip $T$ (set of requests) and a vehicle $v$, the function $\text{travel}(T,v)$ returns the optimal route, which can be computed exhaustively.

Consider the case where algorithms to compute the RV-graph and the RTV-graph are executed up to completion, this is, until all possible trips of size lower or equal than the vehicle capacity are explored. At that point, all combinations of riders and vehicles which satisfy the constraints have been explored exhaustively and have been added in the RTV-graph. At this stage, all valid trip combinations are included in the RTV-graph.

Next, the ILP assignment contains binary variables for all possible trip-vehicle assignments. This problem is solved via branch and bound, which, given enough time, is guaranteed to return the optimal assignment, although it may have to explore all possible assignments.

In the worst case, the algorithm can be seen as an exhaustive search which returns the optimal assignment and routes. The advantage of the method is that, by decoupling the computation of valid routes and trips from the assignment, good solutions can be found efficiently.

**Theorem 2** (Correctness). The returned assignment and vehicle routes respect the constraints.

**Proof:** By construction, the function $\text{travel}(T,v)$ returns a valid route (if one exists) that satisfies the set of constraints. In our case, this is, the vehicle capacity, maximum waiting time and maximum delay are respected. Therefore, in the RTV-graph, all included edges $e(T,v)$ verify that a valid route exists, as returned by $\text{travel}(T,v)$.

The greedy assignment provides a feasible assignment of trips to vehicles from the RTV-graph by construction. This is, each request is assigned at most to one vehicle. And the ILP optimization refines this solution, while respecting the constraints. In the worst case, the ILP returns the greedy assignment.

These steps guarantee that the assignment and routes are valid.

**Theorem 3** (Anytime optimality). The method is anytime optimal. The method is able to return a correct solution quickly. Given additional computational resources, the solution is refined and the method returns the best solution found up to that time.
Proof: The RTV-graph can be initialized with all valid trips with a single request per vehicle. If additional computational resources are available, additional trips (of increasing number of requests) can be explored and added to the RTV-graph.

A greedy assignment of trips to vehicles is computed quickly. The ILP, which returns an optimized assignment, is then initialized from this greedy assignment and as time progresses explores additional branches. If quitted at any time, it returns the best solution found up to that time. If run to completion, it returns the optimal assignment, for the edges of the RTV-graph.

C. Heuristics for real-time execution

In practice, a time-out can be set both for the amount of time spent generating candidate trips for each vehicle, and for the amount of time spent exploring the branches of the ILP. Alternatively, one may set a limit on the number of vehicles considered per request, the number of candidate trips per vehicle or the optimality gap of the ILP. These trade-off optimality for tractability and depend on the available resources.

To achieve real-time performance we employ a set of timeouts. If allowed to progress further, the method would eventually find the optimal assignment.

We implemented the function \(\text{travel}(T,v)\), which computes the optimal route for given trip \(T\) and vehicle \(v\), as follows. If the number of passengers and requests is less or equal than four, we perform an exhaustive search to compute the optimal route which satisfies the constraints. If the number of passengers is greater than four, for each additional request we only check the routes that maintain the order of the current passengers in the vehicle.

In the computation of the RV-graph one may set limits on the number of edges. In particular, we compute the complete graph and, for each request, we keep a maximum of 30 links with candidate vehicles, in particular those of lowest trip cost. Speed-ups such as the ones proposed in T-share [4] could be employed in this stage to prune the most likely vehicles to pick up a request.

In the computation of the RTV-graph we specify a maximum amount of time, per vehicle, to explore potential trips and add edges to the graph. In particular, we used a timeout of 0.2 seconds per vehicle. This leads to sub-optimality of the solution, but faster computation, removing longer trips.

The ILP can be solved with state of the art solvers. In particular, we employed Mosek and set both an optimality gap of 0.1% and a maximum run time of 15 seconds. This solvers employ heuristics in the exploration of the branches of the problem.

D. Parallelization

All steps of the proposed method can be parallelized. We parallelized the computation of the RV-graph among the requests. The computation of the RTV-graph can be performed in parallel as well, distributing the computation among the vehicles. Finally, the ILP optimization is parallelized in most state of the art solvers.
IV. CONTINUOUS APPLICATION WITH INCOMING REQUESTS

Here we describe a solution for Problem 3. Consider a fixed vehicle fleet of size $m$ and incoming requests $r(t)$ with request time $t^r = t$. A pool of requests $R$ is maintained where:

- new requests are added as they are received
- requests are removed when they are (a) picked up by a vehicle (they became passengers) or (b) they could not be successfully matched and picked-up by any vehicle within the maximum waiting time (they are ignored).

At a certain framerate (in our experiments we used 30s) a batch assignment of the requests to the fleet of vehicles is computed by resolving Problem 2 with the predicted state of the fleet at the assignment time (including passengers).

If a request is matched to a vehicle at any given iteration, its latest pick-up time is reduced to the expected pick-up time by that vehicle and the cost $\chi_k$ of ignoring it is increased for subsequent iterations. A request might be rematched to a different vehicle in subsequent iterations as long as its waiting time decreases and until it is picked up. Once a request is picked-up it remains in that vehicle and can not be rematched - the vehicle may still pick additional passengers.

In each iteration, the new assignment of requests to vehicles guarantees that the current passengers are dropped-off within the maximum delay constraint, included in $Z$ in Section II-B.
V. REBALANCING

Here we describe a solution for Problem 4.

After the assignment, due to fleet imbalances, the set $R_{ko}$ of unassigned requests may not be empty, and some empty vehicles $V_{idle}$ may still be unassigned to any request. This may occur when the idle vehicles are in areas far away from the area of current request, and due to the maximum waiting time and delay constraints and vehicle capacity. Under the assumptions that, (a) ignored requests may wait longer and request again, (b) it is likely that more requests occur in the same area where all requests can not be satisfied and (c) there are not enough requests in the neighborhood of the idle cars, we propose the following approach to rebalance the fleet by moving only the idle vehicles.

To rebalance the vehicle fleet, after each batch assignment, the vehicles in $V_{idle}$ are assigned to requests in $R_{ko}$ to minimize the sum of travel times, with the constraint that either all requests or all the vehicles are assigned. For this, we first compute the travel time $\tau_{v,r}$ of each individual request $r$ - vehicle $v$ pickup and then obtain the optimum of the assignment via a fast Linear Program described in Algorithm 4. In this approach, if all requests can be satisfied some vehicles may remain idle, saving fuel and distance travelled, which is the case at night time.

**Algorithm 4 Rebalancing**

1: Given: the idle (empty, stopped and unassigned) vehicles $V_{idle}$, and the unassigned requests $R_{ko}$.
2: Given: the shortest travel time $\tau_{v,r}$ for vehicle $v \in V_{idle}$ to pick request $r \in R_{ko}$.
3: Variables: $Y = \cup_{v \in V_{idle}, r \in R_{ko}} y_{v,r}$. Where $y_{v,r} \in \mathbb{R}$ indicates individual assignments.
4: $\Sigma_{rebalance} := \arg\min_{Y} \sum_{v \in V_{idle}} \sum_{r \in R_{ko}} \tau_{v,r} y_{v,r} \quad \text{s.t.} \quad \sum_{v \in V_{idle}} \sum_{r \in R_{ko}} y_{v,r} = \min(|V_{idle}|, |R_{ko}|) \quad 0 \leq y_{v,r} \leq 1 \forall y_{v,r} \in Y.$
5: The solution of this Linear Program is also a solution of the Integer Linear Program with $y_{v,r} \in \{0, 1\}$. Where $|.|$ denotes the number of elements of a set.
VI. Robustness analysis

In this section we present results to confirm the robustness of the proposed method with respect to the length of the time window and the number - or density - of requests. We also present results highlighting the influence of travel time and congestion.

In each figure we analyze (a) service rate (percentage of requests serviced), (b) average in car delay \( \delta - \omega \), (c) average waiting time \( \omega \), (d) average distance traveled by each vehicle during a single day, (e) percentage of shared rides (number of passengers who shared a ride, divided by the total number of picked-up passengers) and (f) average computational time for a 30 seconds iteration of the method, in a 24 core 2.5GHz machine, including computation of the RV-graph, computation of the RTV-graph, ILP assignment, rebalancing and data writing (higher levels of parallelization would drastically reduce this computational time). The parameters employed in the simulation are specified in (SI Appendix, Sec. III.C).

A. Interval length

In Figure 4 we show robustness results with respect to the interval length, this is, the period of time for which requests are aggregated before a new assignment to the fleet of vehicles. We compare different interval sizes of 10 s, 20 s, 30 s, 40 s and 50 seconds. Results are shown for a nominal case where we employ a fleet of 2000 vehicles of capacity 4 and a maximum waiting time \( \Delta \) of 5 minutes. The points shown represent the average over a week of data in Manhattan with about 3 million requests.

In the experimental analysis of the effects of ride sharing, shown in the results section of the main paper, we employed a time window of 30 seconds, which we considered reasonable when taking into account the computation cost of the approach and the time a person would be willing to wait for receiving an assignment.

In Figure 4 we observe that the method is robust with respect to the time interval: the service rate and percentage of shared rides is mostly unchanged, the mean in-car travel delay slightly decreases with larger time intervals (better assignments are achieved), while both the mean waiting time and mean traveled distance by each vehicle do increase slightly with larger time intervals (the user have to wait longer to receive an assignment). The computational time of the method does increase with the size of the interval, since more requests are jointly assigned.

B. Density of demand

In Figure 5 we show robustness results with respect to the density of demand, this is, the number of travel requests. We compare three different densities, the nominal one of Manhattan with about 3 million requests per week, half of the demand (x0.5) and double the demand (x2). To obtain half of the requests (x0.5) we sorted all the requests of each day by time and removed every odd line, this leads to about 1.5 million requests per week, or 200,000 per day. To obtain double the requests (x2), we cumulated the requests of the same day (e.g. Monday) for two consecutive weeks, this leads to about 6 million requests per week, or 850,000 per day. Results are shown for two nominal cases where we employ (i) a fleet of 2000 vehicles of capacity 4 and a maximum waiting time \( \Delta \) of 5 minutes, and (ii) the same fleet but of capacity 1 (standard taxis). The points shown represent the average over a week of data in Manhattan.

We observe that the approach is robust with the decrease and increase in the number of requests, and that, as expected that performance metrics improve with a decrease in the number of requests.
Fig. 4. Comparison of several performance metrics for varying interval size for the method, 10, 20, 30, 40 and 50 seconds. Results are shown for a nominal case where we employ a fleet of 2000 vehicles of capacity 4 and a maximum waiting time $\Delta$ of 5 minutes. The points shown represent the average over a week of data in Manhattan with about 3 million requests.

Fig. 5. Comparison of several performance metrics for varying density of requests. The nominal case [x1] is for a simulation with the real requests in Manhattan, of about 3 million per week. The case [x0.5] contains only half of the requests, about 1.5 million per week. And the case [x2] contains double the number of requests, about 6 million per week, or about 800,000 per day. Results are shown for two nominal cases where we employ (i) a fleet of 2000 vehicles of capacity 4 and a maximum waiting time $\Delta$ of 5 minutes, and (ii) the same fleet but of capacity 1 (standard taxis). The points shown represent the average over a week of data in Manhattan.
C. Influence of congestion

In Figure 6, we show results highlighting the influence of travel time and congestion. In this experiment, we compare three different travel time estimates: (i) the mean daily travel time, (ii) the estimated travel times for 12:00, and (iii) the estimated travel times for 19:00. Estimated travel times for each edge in the road network of Manhattan were obtained from [6]. Results are shown for a nominal case where we employ a fleet of 2000 vehicles of capacity 4 and a maximum waiting time $\Delta$ of 5 minutes. The points shown represent the average over a week of data in Manhattan with about 3 million requests.

We highlight two observations. First, when using the travel times for 12:00 (lower estimated travel speed), the service rate is reduced by about 5% and the mean waiting time and travel delay slightly increase. This is expected due to the lower travel speed during congested hours. Second, when using the travel times for 19:00 (higher estimated travel speed), the service rate is increased by about 5% and the mean waiting time and travel delay decrease considerably (20 and 10 seconds respectively). This is also expected due to the higher travel speed and correlates with the longer distance traveled by the vehicles.

We observe that the mean travel time estimate provides a good estimate of the performance metrics, closer to the ones from the "rush hour" time than to those from "low hour" time. This is in line with the observation that travel speed in Manhattan remains pretty constant throughout the daytime. Furthermore, by using the mean daily time estimate, we smoothen possible inaccuracies in the travel time estimates.

Fig. 6. Comparison of several performance metrics when using three different travel time estimates: mean daily travel time, travel time at 12:00 and travel time at 19:00. Results are shown for a nominal case where we employ a fleet of 2000 vehicles of capacity 4 and a maximum waiting time $\Delta$ of 5 minutes. The points shown represent the average over a week of data in Manhattan.
### VII. Data

**TABLE I**
Mean computational time for a 30 seconds batch assignment of requests to vehicles in a 24 core 2.5GHz PC.

<table>
<thead>
<tr>
<th>Vehicles</th>
<th>Capacity</th>
<th>Waiting Time</th>
<th>Mean Computational Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1</td>
<td>120</td>
<td>2.90436</td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
<td>300</td>
<td>6.23247</td>
</tr>
<tr>
<td>1000</td>
<td>2</td>
<td>120</td>
<td>2.99213</td>
</tr>
<tr>
<td>1000</td>
<td>2</td>
<td>300</td>
<td>6.64713</td>
</tr>
<tr>
<td>1000</td>
<td>2</td>
<td>420</td>
<td>26.3971</td>
</tr>
<tr>
<td>1000</td>
<td>4</td>
<td>120</td>
<td>3.55018</td>
</tr>
<tr>
<td>1000</td>
<td>4</td>
<td>300</td>
<td>8.952</td>
</tr>
<tr>
<td>1000</td>
<td>4</td>
<td>420</td>
<td>27.8819</td>
</tr>
<tr>
<td>1000</td>
<td>10</td>
<td>120</td>
<td>3.05873</td>
</tr>
<tr>
<td>1000</td>
<td>10</td>
<td>300</td>
<td>11.442</td>
</tr>
<tr>
<td>1000</td>
<td>10</td>
<td>420</td>
<td>34.1471</td>
</tr>
<tr>
<td>2000</td>
<td>1</td>
<td>120</td>
<td>6.76338</td>
</tr>
<tr>
<td>2000</td>
<td>1</td>
<td>300</td>
<td>11.7457</td>
</tr>
<tr>
<td>2000</td>
<td>1</td>
<td>420</td>
<td>15.2656</td>
</tr>
<tr>
<td>2000</td>
<td>2</td>
<td>120</td>
<td>7.78438</td>
</tr>
<tr>
<td>2000</td>
<td>2</td>
<td>300</td>
<td>12.1214</td>
</tr>
<tr>
<td>2000</td>
<td>2</td>
<td>420</td>
<td>33.8259</td>
</tr>
<tr>
<td>2000</td>
<td>4</td>
<td>120</td>
<td>7.74553</td>
</tr>
<tr>
<td>2000</td>
<td>4</td>
<td>300</td>
<td>17.2651</td>
</tr>
<tr>
<td>2000</td>
<td>4</td>
<td>420</td>
<td>57.557</td>
</tr>
<tr>
<td>2000</td>
<td>10</td>
<td>120</td>
<td>6.23868</td>
</tr>
<tr>
<td>2000</td>
<td>10</td>
<td>300</td>
<td>19.4179</td>
</tr>
<tr>
<td>2000</td>
<td>10</td>
<td>420</td>
<td>57.0266</td>
</tr>
<tr>
<td>3000</td>
<td>1</td>
<td>120</td>
<td>4.71717</td>
</tr>
<tr>
<td>3000</td>
<td>1</td>
<td>300</td>
<td>11.938</td>
</tr>
<tr>
<td>3000</td>
<td>1</td>
<td>420</td>
<td>17.923</td>
</tr>
<tr>
<td>3000</td>
<td>2</td>
<td>120</td>
<td>4.82994</td>
</tr>
<tr>
<td>3000</td>
<td>2</td>
<td>300</td>
<td>12.5836</td>
</tr>
<tr>
<td>3000</td>
<td>2</td>
<td>420</td>
<td>31.3838</td>
</tr>
<tr>
<td>3000</td>
<td>4</td>
<td>120</td>
<td>5.15316</td>
</tr>
<tr>
<td>3000</td>
<td>4</td>
<td>300</td>
<td>21.3322</td>
</tr>
<tr>
<td>3000</td>
<td>4</td>
<td>420</td>
<td>51.5582</td>
</tr>
<tr>
<td>3000</td>
<td>10</td>
<td>120</td>
<td>5.09283</td>
</tr>
<tr>
<td>3000</td>
<td>10</td>
<td>300</td>
<td>24.7557</td>
</tr>
<tr>
<td>3000</td>
<td>10</td>
<td>420</td>
<td>60.3936</td>
</tr>
</tbody>
</table>
Table II

Average values of several performance metrics for ride sharing over a whole week with over 3 million requests. All experiments include rebalancing except for those indicated with (N.R.) next to the vehicle capacity. Ω is the maximum waiting time and Δ the maxim delay, including both in-car delay and waiting time. Service and shared rates are in per one.

<table>
<thead>
<tr>
<th>Vehicles</th>
<th>Capacity</th>
<th>Ω [s]</th>
<th>Δ [s]</th>
<th>Service Rate</th>
<th>Mean Waiting [s]</th>
<th>Mean In-Car Delay [s]</th>
<th>Mean Passengers</th>
<th>Shared Rate</th>
<th>Mean Travel [km]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1</td>
<td>120</td>
<td>240</td>
<td>0.262978</td>
<td>81.5581</td>
<td>0</td>
<td>0.768697</td>
<td>0</td>
<td>182.027</td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
<td>300</td>
<td>600</td>
<td>0.283784</td>
<td>234.689</td>
<td>0</td>
<td>0.835099</td>
<td>0</td>
<td>203.598</td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
<td>420</td>
<td>840</td>
<td>0.287732</td>
<td>320.615</td>
<td>0</td>
<td>0.847364</td>
<td>0</td>
<td>207.773</td>
</tr>
<tr>
<td>1000</td>
<td>2</td>
<td>120</td>
<td>240</td>
<td>0.369998</td>
<td>76.3792</td>
<td>44.7869</td>
<td>1.16233</td>
<td>0.907206</td>
<td>185.032</td>
</tr>
<tr>
<td>1000</td>
<td>2</td>
<td>300</td>
<td>600</td>
<td>0.413232</td>
<td>219.236</td>
<td>134.524</td>
<td>1.50662</td>
<td>0.957515</td>
<td>196.476</td>
</tr>
<tr>
<td>1000</td>
<td>2</td>
<td>420</td>
<td>840</td>
<td>0.412036</td>
<td>325.386</td>
<td>171.178</td>
<td>1.5667</td>
<td>0.962714</td>
<td>196.663</td>
</tr>
<tr>
<td>1000</td>
<td>4</td>
<td>120</td>
<td>240</td>
<td>0.433255</td>
<td>73.5066</td>
<td>52.5421</td>
<td>1.37394</td>
<td>0.905983</td>
<td>185.513</td>
</tr>
<tr>
<td>1000</td>
<td>4</td>
<td>300</td>
<td>600</td>
<td>0.580854</td>
<td>176.804</td>
<td>134.524</td>
<td>1.48346</td>
<td>0.942783</td>
<td>193.919</td>
</tr>
<tr>
<td>1000</td>
<td>4</td>
<td>420</td>
<td>840</td>
<td>0.603213</td>
<td>258.585</td>
<td>258.446</td>
<td>1.5667</td>
<td>0.962714</td>
<td>196.663</td>
</tr>
<tr>
<td>1000</td>
<td>4</td>
<td>n.R.</td>
<td>300</td>
<td>600</td>
<td>0.373123</td>
<td>183.438</td>
<td>195.725</td>
<td>0.899895</td>
<td>117.326</td>
</tr>
<tr>
<td>2000</td>
<td>1</td>
<td>120</td>
<td>240</td>
<td>0.481458</td>
<td>77.7428</td>
<td>0</td>
<td>0.700887</td>
<td>0</td>
<td>154.727</td>
</tr>
<tr>
<td>2000</td>
<td>1</td>
<td>300</td>
<td>600</td>
<td>0.524316</td>
<td>220.625</td>
<td>0</td>
<td>0.769403</td>
<td>0</td>
<td>175.176</td>
</tr>
<tr>
<td>2000</td>
<td>1</td>
<td>420</td>
<td>840</td>
<td>0.532492</td>
<td>289.226</td>
<td>0</td>
<td>0.782802</td>
<td>0</td>
<td>179.546</td>
</tr>
<tr>
<td>2000</td>
<td>2</td>
<td>120</td>
<td>240</td>
<td>0.645492</td>
<td>73.3992</td>
<td>37.0945</td>
<td>1.00548</td>
<td>0.87967</td>
<td>155.225</td>
</tr>
<tr>
<td>2000</td>
<td>2</td>
<td>300</td>
<td>600</td>
<td>0.745878</td>
<td>201.348</td>
<td>121.656</td>
<td>1.33669</td>
<td>0.95198</td>
<td>164.91</td>
</tr>
<tr>
<td>2000</td>
<td>2</td>
<td>420</td>
<td>840</td>
<td>0.753914</td>
<td>288.845</td>
<td>153.138</td>
<td>1.40076</td>
<td>0.96388</td>
<td>165.58</td>
</tr>
<tr>
<td>2000</td>
<td>4</td>
<td>120</td>
<td>240</td>
<td>0.709123</td>
<td>71.3443</td>
<td>42.9121</td>
<td>1.11723</td>
<td>0.877091</td>
<td>154.697</td>
</tr>
<tr>
<td>2000</td>
<td>4</td>
<td>n.R.</td>
<td>300</td>
<td>600</td>
<td>0.625829</td>
<td>173.514</td>
<td>191.711</td>
<td>1.23814</td>
<td>101.356</td>
</tr>
<tr>
<td>2000</td>
<td>4</td>
<td>300</td>
<td>600</td>
<td>0.911007</td>
<td>153.465</td>
<td>151.686</td>
<td>1.70709</td>
<td>0.925342</td>
<td>157.391</td>
</tr>
<tr>
<td>2000</td>
<td>4</td>
<td>420</td>
<td>840</td>
<td>0.937389</td>
<td>197.158</td>
<td>197.389</td>
<td>1.86932</td>
<td>0.931588</td>
<td>155.629</td>
</tr>
<tr>
<td>3000</td>
<td>1</td>
<td>120</td>
<td>240</td>
<td>0.652067</td>
<td>72.4532</td>
<td>0</td>
<td>0.63218</td>
<td>0</td>
<td>133.714</td>
</tr>
<tr>
<td>3000</td>
<td>1</td>
<td>300</td>
<td>600</td>
<td>0.734434</td>
<td>195.736</td>
<td>0</td>
<td>0.71695</td>
<td>0</td>
<td>156.49</td>
</tr>
<tr>
<td>3000</td>
<td>1</td>
<td>420</td>
<td>840</td>
<td>0.744333</td>
<td>251.324</td>
<td>0</td>
<td>0.727727</td>
<td>0</td>
<td>160.412</td>
</tr>
<tr>
<td>3000</td>
<td>2</td>
<td>120</td>
<td>240</td>
<td>0.794154</td>
<td>69.3053</td>
<td>28.0659</td>
<td>0.812734</td>
<td>0.798371</td>
<td>127.736</td>
</tr>
<tr>
<td>3000</td>
<td>2</td>
<td>300</td>
<td>600</td>
<td>0.929443</td>
<td>158.198</td>
<td>76.3427</td>
<td>1.02932</td>
<td>0.903125</td>
<td>139.298</td>
</tr>
<tr>
<td>3000</td>
<td>2</td>
<td>420</td>
<td>840</td>
<td>0.942151</td>
<td>191.746</td>
<td>87.6798</td>
<td>1.06026</td>
<td>0.913134</td>
<td>140.334</td>
</tr>
<tr>
<td>3000</td>
<td>4</td>
<td>120</td>
<td>240</td>
<td>0.829284</td>
<td>68.0709</td>
<td>33.162</td>
<td>0.857788</td>
<td>0.806376</td>
<td>124.355</td>
</tr>
<tr>
<td>3000</td>
<td>4</td>
<td>n.R.</td>
<td>300</td>
<td>600</td>
<td>0.797301</td>
<td>157.168</td>
<td>161.856</td>
<td>1.00192</td>
<td>93.0818</td>
</tr>
<tr>
<td>3000</td>
<td>4</td>
<td>300</td>
<td>600</td>
<td>0.969812</td>
<td>136.944</td>
<td>112.643</td>
<td>1.14426</td>
<td>0.895421</td>
<td>119.019</td>
</tr>
<tr>
<td>3000</td>
<td>4</td>
<td>420</td>
<td>840</td>
<td>0.979149</td>
<td>162.45</td>
<td>137.158</td>
<td>1.20086</td>
<td>0.902623</td>
<td>117.996</td>
</tr>
</tbody>
</table>
VIII. ADDITIONAL EXPERIMENTAL FIGURES

Fig. 7. Occupancy rates, 1000 vehicles, capacity 1, $\Delta = 120$ s

Fig. 8. Occupancy rates, 1000 vehicles, capacity 1, $\Delta = 300$ s

Fig. 9. Occupancy rates, 1000 vehicles, capacity 1, $\Delta = 420$ s
Fig. 10. Occupancy rates, 1000 vehicles, capacity 2, $\Delta = 120$ s

Fig. 11. Occupancy rates, 1000 vehicles, capacity 2, $\Delta = 300$ s

Fig. 12. Occupancy rates, 1000 vehicles, capacity 2, $\Delta = 420$ s
Fig. 13. Occupancy rates, 1000 vehicles, capacity 4, $\Delta = 120$ s

Fig. 14. Occupancy rates, 1000 vehicles, capacity 4, $\Delta = 300$ s

Fig. 15. Occupancy rates, 1000 vehicles, capacity 4, $\Delta = 420$ s
Fig. 16. Occupancy rates, 1000 vehicles, capacity 10, $\Delta = 120$ s

Fig. 17. Occupancy rates, 1000 vehicles, capacity 10, $\Delta = 300$ s

Fig. 18. Occupancy rates, 1000 vehicles, capacity 10, $\Delta = 420$ s
Fig. 19. Occupancy rates, 2000 vehicles, capacity 1, $\Delta = 120$ s

Fig. 20. Occupancy rates, 2000 vehicles, capacity 1, $\Delta = 300$ s

Fig. 21. Occupancy rates, 2000 vehicles, capacity 1, $\Delta = 420$ s
Fig. 22. Occupancy rates, 2000 vehicles, capacity 2, $\Delta = 120$ s

Fig. 23. Occupancy rates, 2000 vehicles, capacity 2, $\Delta = 300$ s

Fig. 24. Occupancy rates, 2000 vehicles, capacity 2, $\Delta = 420$ s
Fig. 25. Occupancy rates, 2000 vehicles, capacity 4, $\Delta = 120$ s

Fig. 26. Occupancy rates, 2000 vehicles, capacity 4, $\Delta = 300$ s

Fig. 27. Occupancy rates, 2000 vehicles, capacity 4, $\Delta = 420$ s
Fig. 28. Occupancy rates, 2000 vehicles, capacity 10, \( \Delta = 120 \) s

Fig. 29. Occupancy rates, 2000 vehicles, capacity 10, \( \Delta = 300 \) s

Fig. 30. Occupancy rates, 2000 vehicles, capacity 10, \( \Delta = 420 \) s
Fig. 31. Occupancy rates, 3000 vehicles, capacity 1, $\Delta = 120$ s

Fig. 32. Occupancy rates, 3000 vehicles, capacity 1, $\Delta = 300$ s

Fig. 33. Occupancy rates, 3000 vehicles, capacity 1, $\Delta = 420$ s
Fig. 34. Occupancy rates, 3000 vehicles, capacity 2, $\Delta = 120$ s

Fig. 35. Occupancy rates, 3000 vehicles, capacity 2, $\Delta = 300$ s

Fig. 36. Occupancy rates, 3000 vehicles, capacity 2, $\Delta = 420$ s
Fig. 37. Occupancy rates, 3000 vehicles, capacity 4, $\Delta = 120$ s

Fig. 38. Occupancy rates, 3000 vehicles, capacity 4, $\Delta = 300$ s

Fig. 39. Occupancy rates, 3000 vehicles, capacity 4, $\Delta = 420$ s
Fig. 40. Occupancy rates, 3000 vehicles, capacity 10, $\Delta = 120$ s

Fig. 41. Occupancy rates, 3000 vehicles, capacity 10, $\Delta = 300$ s

Fig. 42. Occupancy rates, 3000 vehicles, capacity 10, $\Delta = 420$ s
REFERENCES


