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 $\{o_r, d_r, t_r^r, t_r^{pl}, t_r^p, t_r^d, t_r^*\}$, indicating its origin o_r, its destination d_r, the time of the request t_r^r, the latest acceptable pick-up time t_r^{pl} (initially given by $t_r^{pl} = t_r^r + \Omega$ with Ω the *maximum waiting* time), the pick-up time t_r^p, the expected drop off time t_r^d, and the earliest possible time at which the destination could be reached $t_r^* = t_r^r + \tau(o_r, d_r)$.

The current state of a *vehicle* v is given by a tuple $\{q_v, t_v, \mathcal{P}_v\}$, indicating its current position q_v, the current time t_v and its passengers $\mathcal{P}_v = \{p_1, \ldots, p_{n_v^{pass}}\}$. 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 ν (maximum number of passengers in a vehicle), and a *maximum delay* Δ 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, $t_r^d - t_r^*$.

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 $\mathcal{R} = \{r_1, \ldots, r_n\}$, a set of vehicles $\mathcal{V} = \{v_1, \ldots, v_m\}$ at their current state including passengers, and a function to compute travel times on the road network. Compute the optimal assignment Σ of requests to vehicles that minimizes a cost function $\mathcal{C}$ and that satisfies a set of constraints $\mathcal{Z}$, including a maximum capacity ν of passengers per vehicle.*

In particular, and not limited to, we will consider the following set of constraints $\mathcal{Z}$:

- For each request r, its maximum waiting time $t_r^p \leq t_r^{pl} \leq t_r^r + \Omega$.
- For each request or passenger r, its maximum travel delay $t_r^d \leq t_r^* + \Delta$.
- For each vehicle, the maximum number of passengers $n_v^{pass} \leq \nu$ 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 $\mathcal{R}_{ok}$ the set of requests assigned to a vehicle and $\mathcal{R}_{ko}$ the set of requests that are not served by any vehicle.

We define the cost $\mathcal{C}$ of an assignment Σ 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,

$$\mathcal{C}(\Sigma) = \sum_{v \in \mathcal{V}} \sum_{p \in \mathcal{P}_v} \delta_p + \sum_{r \in \mathcal{R}_{ok}} \delta_r + \sum_{r \in \mathcal{R}_{ko}} c_{ko}. \tag{1}$$

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 Σ 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 $\mathcal{C}$.*

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 by 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 σ_v for the vehicle that minimizes the sum of delays $\sum_{r \in \mathcal{P}_v \cup \mathcal{R}_v} (t_r^d - t_r^*)$ subject to the constraints $\mathcal{Z}$ (waiting time, delay and capacity) is given by a function

$$travel(v, \mathcal{R}_v), \tag{2}$$

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.

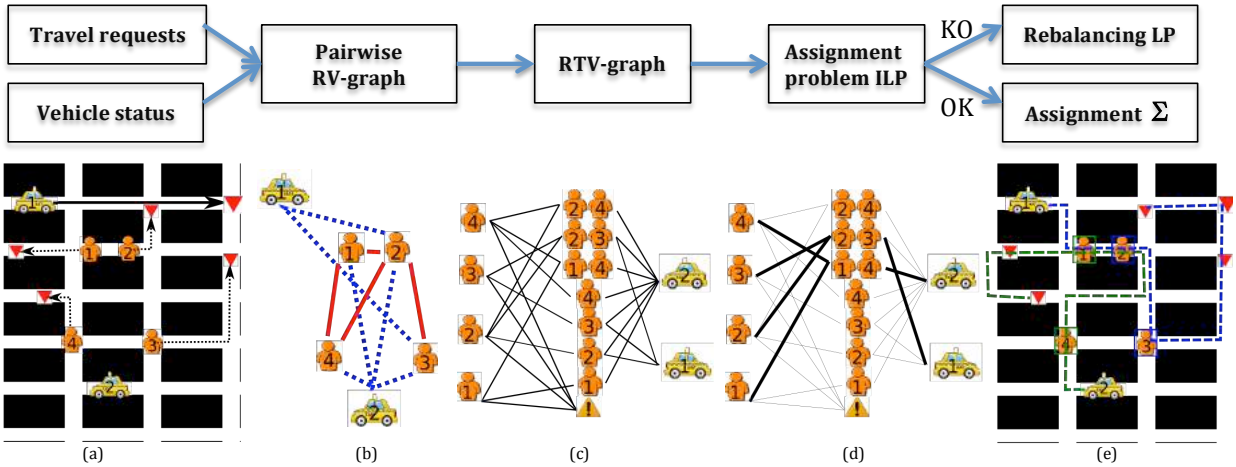


Fig. 1. Schematic overview of the proposed method for batch assignment of multiple requests to multiple vehicles of capacity ν. The method consists of several steps leading to an integer linear optimization which provides an anytime optimal assignment. (a) Example of a street network with four requests (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 $\mathcal{Z}$ of maximum waiting time and delay. A cost $\sum_{r=\{1,2\}} (t_r^d - t_r^*)$ 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 $\mathcal{Z}$. This is, if $travel(v, r)$ 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.

D. Graph of candidate trips and pick-ups (RTV-graph)

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 $\mathcal{Z}$.

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,

$$\exists\, e(r, T) \Leftrightarrow r \in T \tag{3}$$

$$\exists\, e(T, v) \Leftrightarrow travel(v, T) = \text{"valid"} \tag{4}$$

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,*

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

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,*

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

Therefore, a trip T only needs to by 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 $\mathcal{T}$ is the list of feasible trips. With each node $e(T, v)$, the cost $\mathcal{C}$ 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

$\mathcal{T} = \emptyset$
for each vehicle $v \in \mathcal{V}$ **do**
    $\mathcal{T}_k = \emptyset \ \forall k \in \{1, \ldots, \nu\}$
    [Add trips of size one]
    for $e(r, v)$ edge of RV-graph **do**
        $\mathcal{T}_1 \leftarrow T = \{r\}$; Add $e(r, T)$ and $e(T, v)$
    [Add trips of size two]
    for all $\{r_1\}, \{r_2\} \in \mathcal{T}_1$ and $e(r_1, r_2) \in$ RV-graph **do**
        if $travel(v, \{r_1, r_2\}) =$ valid **then**
            $\mathcal{T}_2 \leftarrow T = \{r_1, r_2\}$; Add $e(r_i, T)$ and $e(T, v)$
    [Add trips of size k]
    for $k \in \{3, \ldots, \nu\}$ **do**
        for all T_1, $T_2 \in \mathcal{T}_{k-1}$ with $|T_1 \cup T_2| = k$ **do**
            Denote $T_1 \cup T_2 = \{r_1, ..., r_k\}$
            if $\forall i \in \{1, ..., k\}$, $\{r_1, ..., r_k\} \setminus r_i \in \mathcal{T}_{k-1}$ **then**
                if $travel(v, T_1 \cup T_2) =$ valid **then**
                    $\mathcal{T}_k \leftarrow T = T_1 \cup T_2$
                    Add $e(r_i, T)$, $\forall r_i \in T$, and $e(T, v)$
    $\mathcal{T} \leftarrow \bigcup_{i \in \{1,...,\nu\}} \mathcal{T}_i$

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 $\mathcal{C}$ of Equation 1 is described in Algorithm 2.

Algorithm 2 Greedy assignment

$\mathcal{R}_{ok} = \emptyset$; $\mathcal{V}_{ok} = \emptyset$
for $k = \nu$; $k > 0$; $k--$ **do**
  S_k := sort $e(T, v)$ in increasing cost, $\forall T \in \mathcal{T}_k, v \in \mathcal{V}$
  while $S_k \neq \emptyset$ **do**
    pop $e(T, v) \leftarrow S_k$
    if $\forall r \in T, r \notin \mathcal{R}_{ok}$ and $v \notin \mathcal{V}_{ok}$ **then**
      $\mathcal{R}_{ok} \leftarrow \{\forall r \in T\}$; $\mathcal{V}_{ok} \leftarrow v$
      $\Sigma_{greedy} \leftarrow e(T, v)$

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

$$\mathcal{X} = \{\epsilon_{i,j},\ \chi_k;\ \ \forall e(T_i, v_j) \text{ node in RTV-graph } \forall r_k \in \mathcal{R}\}. \tag{5}$$

3) Cost: The cost function, equivalent to $\mathcal{C}(\Sigma)$ in Equation (1), is given by [1]

$$\mathcal{C}(\mathcal{X}) := \sum_{i,j \in \mathcal{E}_{TV}} c_{i,j}\epsilon_{i,j} + \sum_{k \in \{0,\dots,n\}} c_{ko}\chi_k, \tag{6}$$

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 singe trip at most,

$$\sum_{i \in \mathcal{I}_j^V} \epsilon_{i,j} \leq 1 \quad \forall v_j \in \mathcal{V}, \tag{8}$$

where $\mathcal{I}_j^V$ denotes the indexes i for which an edge $e(T_i, v_j)$ exists in the RTV-graph.

[1]We note that the cost function of Eq. 6 is equivalent to (1) only if the cost term $c_{i,j}$ of an individual assignment, of trip i to vehicle j, is given by the sum of delays for all current passengers and assigned requests of vehicle i, minus the sum of delays for all current passengers of vehicle i if it where not to take any additional requests.

Each request is assigned to a vehicle or ignored,

$$\sum_{i \in \mathcal{I}_k^R} \sum_{j \in \mathcal{I}_i^T} \epsilon_{i,j} + \chi_k = 1 \quad \forall r_k \in \mathcal{R}, \tag{9}$$

where $\mathcal{I}_k^R$ denotes the indexes i for which an edge $e(r_k, T_i)$ exists in the RTV-graph and $\mathcal{I}_i^T$ 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

Initial guess: Σ_{greedy}

$\Sigma_{optim} := \arg\min_{\mathcal{X}} \ \mathcal{C}(\mathcal{X})$ (Eq. 6)

s.t. $\sum_{i \in \mathcal{I}_j^V} \epsilon_{i,j} \leq 1 \qquad \forall v_j \in \mathcal{V}$

$\sum_{i \in \mathcal{I}_k^R} \sum_{j \in \mathcal{I}_i^T} \epsilon_{i,j} + \chi_k = 1 \quad \forall r_k \in \mathcal{R}$

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

$$m\frac{n!}{\nu!(n-\nu)!} \sim O(m\, n^{\nu}), \tag{10}$$

where m is the number of vehicles, n the number of requests and ν 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 ν 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 $\mathcal{Z}$.

Proof: For any particular assignment between a trip T (set of requests) and a vehicle v, the function $travel(T, v)$ 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 $travel(T, v)$ 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 $travel(T, v)$.

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 $travel(T, v)$, 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 he 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 $\mathcal{R}$ 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 χ_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 $\mathcal{Z}$ in Section II-B.

V. REBALANCING

Here we describe a solution for Problem 4.

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 by 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 $\mathcal{V}_{idle}$ are assigned to requests in $\mathcal{R}_{ko}$ 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

Given: the idle (empty, stopped and unassigned) vehicles $\mathcal{V}_{idle}$, and the unassigned requests $\mathcal{R}_{ko}$.
Given: the shortest travel time $\tau_{v,r}$ for vehicle $v \in \mathcal{V}_{idle}$ to pick request $r \in \mathcal{R}_{ko}$.
Variables: $\mathcal{Y} = \cup_{v \in \mathcal{V}_{idle}, r \in \mathcal{R}_{ko}} y_{v,r}$. Where $y_{v,r} \in \mathbb{R}$ indicates individual assignments.

$$\Sigma_{rebalance} := \arg\min_{\mathcal{Y}} \sum_{v \in \mathcal{V}_{idle}} \sum_{r \in \mathcal{R}_{ko}} \tau_{v,r}\ y_{v,r}$$

$$\text{s.t.} \sum_{v \in \mathcal{V}_{idle}} \sum_{r \in \mathcal{R}_{ko}} y_{v,r} = \min(|\mathcal{V}_{idle}|, |\mathcal{R}_{ko}|)$$

$$\sum_{v \in \mathcal{V}_{idle}} y_{v,r} \leq 1 \qquad \forall r \in \mathcal{R}_{ko}$$

$$\sum_{r \in \mathcal{R}_{ko}} y_{v,r} \leq 1 \qquad \forall v \in \mathcal{V}_{idle}$$

$$0 \leq y_{v,r} \leq 1 \qquad \forall y_{v,r} \in \mathcal{Y}.$$

Where $|.|$ denotes the number of elements of a set.
The solution of this Linear Program is the solution of the Integer Linear Program with $y_{v,r} \in \{0, 1\}$.

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 ω, (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 Δ of 5 minutes. The points shown represent the average over a week of data in Manhattan with about 3 million requests.

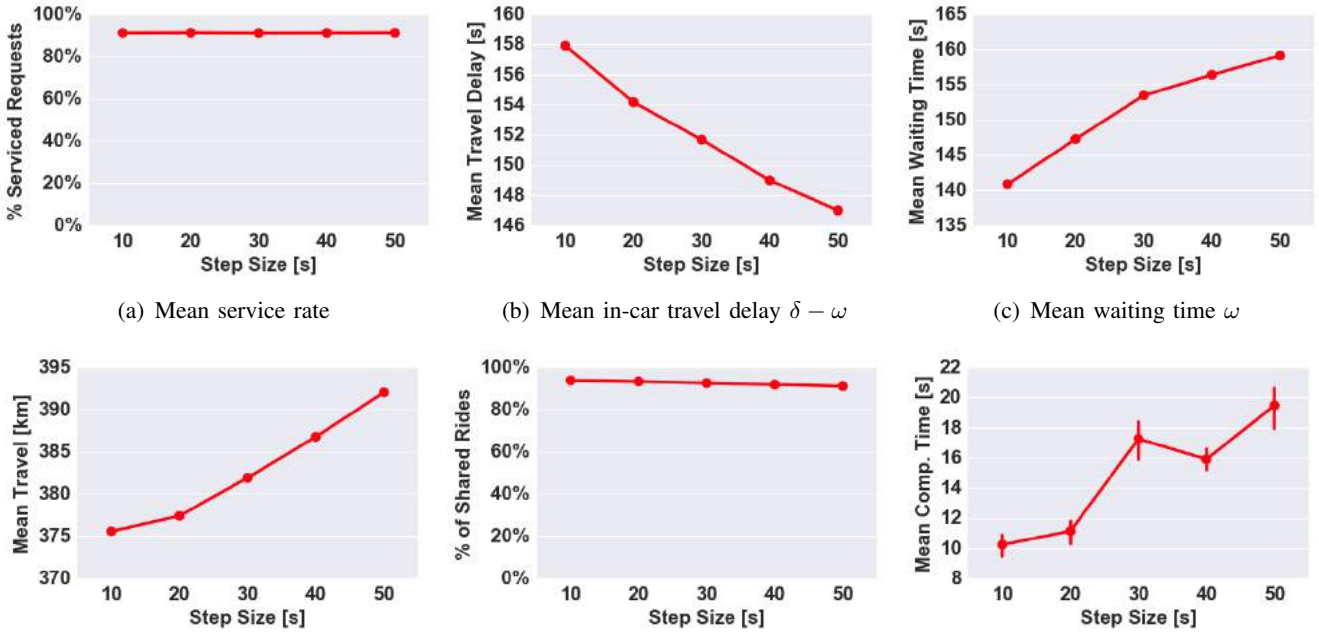
In the experimental analysis of the effects of ride sharing, shown in the results section of the main paper, we employed a time window of 30 seconds, which we considered reasonable when taking into account the computation cost of the approach and the time a person would be willing to wait for receiving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

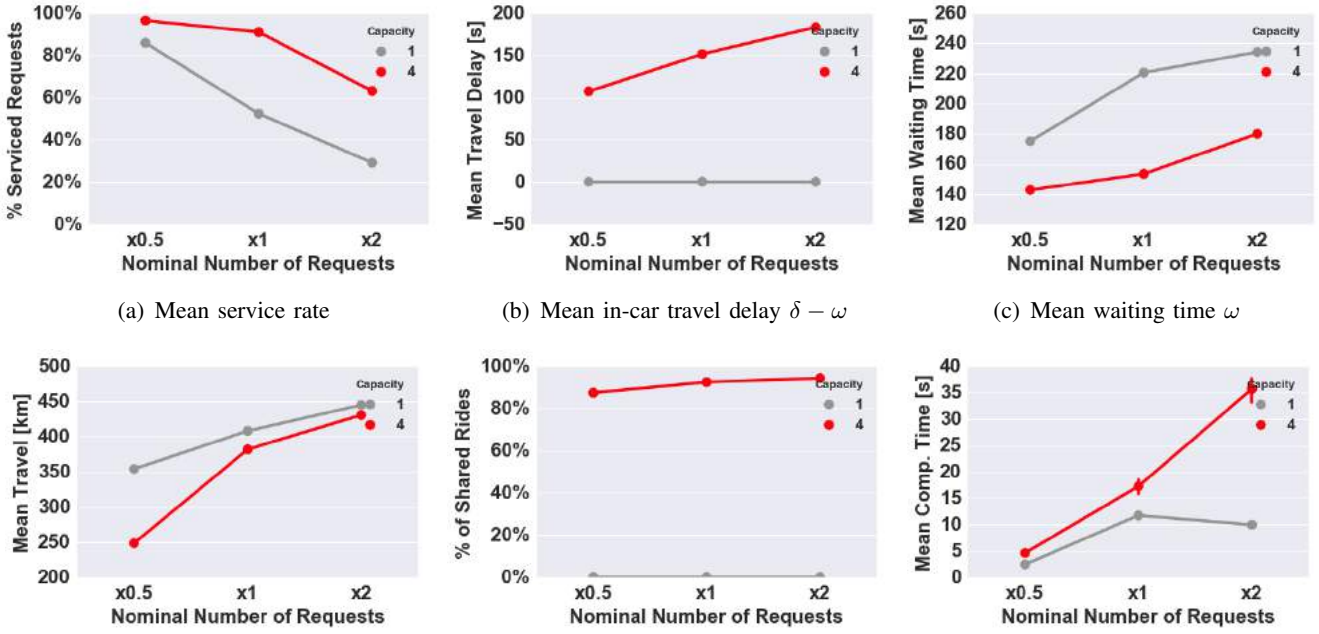
We observe that the approach is robust with the decrease and increase in the number of requests, and that, as expected that performance metrics improve with a decrease in the number of requests.



(a) Mean service rate (b) Mean in-car travel delay $\delta - \omega$ (c) Mean waiting time ω

(d) Mean distance travelled (e) Percentage of shared rides (f) Mean computational time

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 Δ of 5 minutes. The points shown represent the average over a week of data in Manhattan with about 3 million requests.



(a) Mean service rate (b) Mean in-car travel delay $\delta - \omega$ (c) Mean waiting time ω

(d) Mean distance travelled (e) Percentage of shared rides (f) Mean computational time

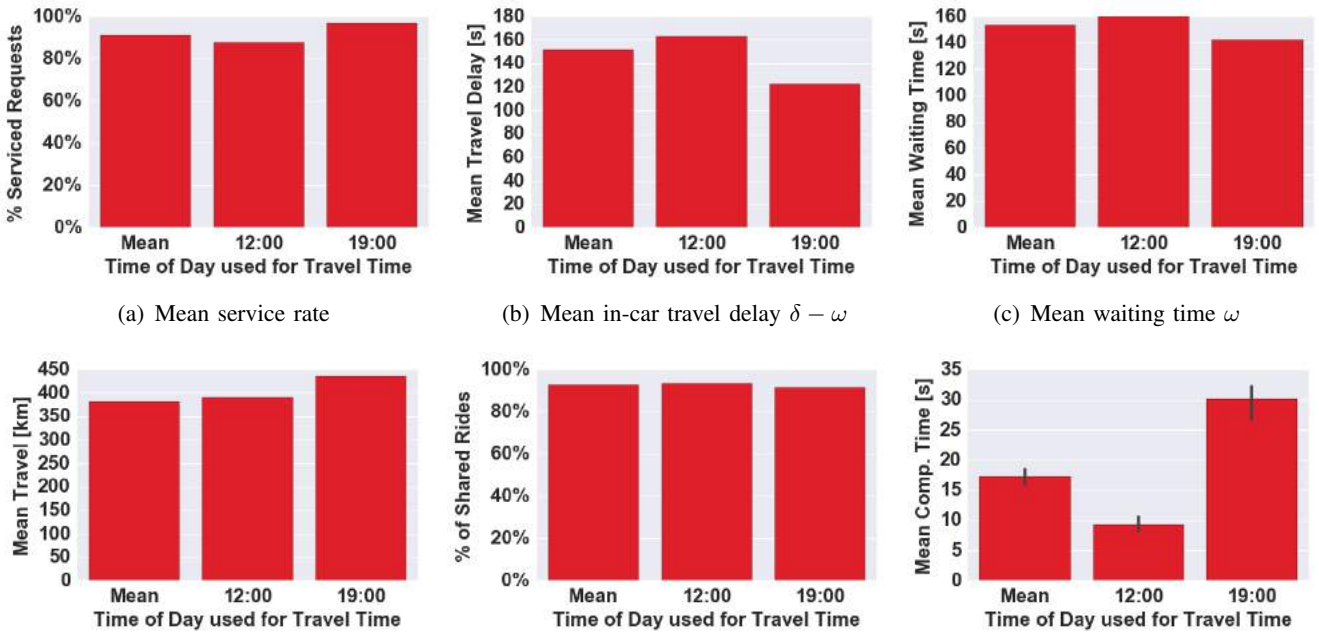
Fig. 5. Comparison of several performance metrics for varying density of requests. The nominal case [x1] is for a simulation with the real requests in Manhattan, of about 3 million per week. The case [x0.5] contains only half of the requests, about 1.5 million per week. And the case [x2] contains double 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 Δ 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 experiments 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 where 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 Δ 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 through-out the daytime. Furthermore, by using the mean daily time estimate, we smoothen possible inaccuracies in the travel time estimates.



(a) Mean service rate (b) Mean in-car travel delay $\delta - \omega$ (c) Mean waiting time ω

(d) Mean distance travelled (e) Percentage of shared rides (f) Mean computational time

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 cases where we employ a fleet of 2000 vehicles of capacity 4 and a maximum waiting time Δ 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.

Vehicles	Capacity	Waiting Time	Mean Computational Time [s]
1000	1	120	2.90436
1000	1	300	6.23247
1000	1	420	10.0162
1000	2	120	2.99213
1000	2	300	6.64713
1000	2	420	26.3971
1000	4	120	3.55018
1000	4	300	8.952
1000	4	420	27.8819
1000	10	120	3.05873
1000	10	300	11.442
1000	10	420	34.1471
2000	1	120	6.76338
2000	1	300	11.7457
2000	1	420	15.2656
2000	2	120	7.78438
2000	2	300	12.1214
2000	2	420	33.8259
2000	4	120	7.74553
2000	4	300	17.2651
2000	4	420	57.557
2000	10	120	6.23868
2000	10	300	19.4179
2000	10	420	57.0266
3000	1	120	4.71717
3000	1	300	11.938
3000	1	420	17.923
3000	2	120	4.82994
3000	2	300	12.5836
3000	2	420	31.3838
3000	4	120	5.15316
3000	4	300	21.3322
3000	4	420	51.5582
3000	10	120	5.09283
3000	10	300	24.7557
3000	10	420	60.3936

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.

Vehicles	Capacity	Ω [s]	Δ [s]	Service Rate	Mean Waiting [s]	Mean In-Car Delay [s]	Mean Passengers	Shared Rate	Mean Travel [km]
1000	1	120	240	0.262978	81.5581	0.472491	0.768697	0	396.549
1000	1	300	600	0.283784	234.689	0.472562	0.835099	0	423.753
1000	1	420	840	0.287732	320.615	0.472491	0.847364	0	428.665
1000	2	120	240	0.36998	76.3792	44.7869	1.16233	0.907206	404.992
1000	2	300	600	0.413232	219.236	134.524	1.50662	0.957515	414.216
1000	2	420	840	0.412036	325.386	171.178	1.5667	0.962714	412.298
1000	4	120	240	0.433255	73.5066	52.5421	1.37394	0.905983	406.908
1000	4 n.R.	300	600	0.373123	183.438	195.725	1.48346	0.942783	246.834
1000	4	300	600	0.580854	176.804	188.716	2.30771	0.943285	415.614
1000	4	420	840	0.603213	258.585	258.446	2.60321	0.950714	407.503
1000	10	120	240	0.440009	73.2917	52.804	1.39497	0.902917	407.288
1000	10	300	600	0.669158	157.273	197.282	2.68934	0.905023	416.094
1000	10	420	840	0.75182	203.156	287.461	3.37339	0.899895	408.589
2000	1	120	240	0.481458	77.7428	0.472356	0.700887	0	360.627
2000	1	300	600	0.524316	220.625	0.472704	0.769403	0	388.581
2000	1	420	840	0.532492	289.226	0.472188	0.782802	0	393.826
2000	2	120	240	0.645492	73.3992	37.0945	1.00548	0.87967	364.08
2000	2	300	600	0.745878	201.348	121.656	1.33669	0.955198	372.974
2000	2	420	840	0.753914	288.845	153.138	1.40076	0.96388	372.093
2000	4	120	240	0.709123	71.3443	42.9121	1.11723	0.877091	363.639
2000	4 n.R.	300	600	0.625829	173.514	191.711	1.23814	0.940455	212.985
2000	4	300	600	0.911007	153.465	151.686	1.70709	0.925342	361.019
2000	4	420	840	0.937389	197.158	197.389	1.86932	0.931588	356.274
2000	10	120	240	0.713112	71.2705	43.1219	1.12366	0.875883	363.484
2000	10	300	600	0.944974	143.459	156.101	1.78505	0.899875	354.207
2000	10	420	840	0.977252	171.2	210.808	1.98942	0.895912	342.352
3000	1	120	240	0.652067	72.4532	0.472158	0.63218	0	325.245
3000	1	300	600	0.734434	195.736	0.472759	0.71695	0	363.028
3000	1	420	840	0.744333	251.324	0.472569	0.727727	0	368.505
3000	2	120	240	0.794154	69.3053	28.0659	0.812734	0.798371	311.386
3000	2	300	600	0.929443	158.198	76.3427	1.02932	0.903125	327.535
3000	2	420	840	0.942151	191.746	87.6798	1.06026	0.913134	328.13
3000	4	120	240	0.829284	68.0709	33.162	0.857788	0.806376	302.332
3000	4 n.R.	300	600	0.797301	157.168	161.856	1.00192	0.92347	196.769
3000	4	300	600	0.969812	136.944	112.643	1.14426	0.895421	280.063
3000	4	420	840	0.979149	162.45	137.158	1.20086	0.902623	274.142
3000	10	120	240	0.830674	68.0008	33.3181	0.859327	0.806087	301.886
3000	10	300	600	0.973502	133.96	127.549	1.17848	0.885291	261.108
3000	10	420	840	0.985835	154.013	164.524	1.26653	0.887157	249.694

REFERENCES

[1] D. Delling, P. Sanders, D. Schultes, and D. Wagner. Engineering Route Planning Algorithms. 2:117–139, 2009.

[2] F. Glover and M. Laguna. *Tabu Search?* Springer, 2013.

[3] K. Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

[4] S. Ma, Y. Zheng, and O. Wolfson. T-share: A large scale dynamic taxi ridesharing service. *Proceedings of IEEE ICDE*, 2013.

[5] D. Pham and D. Karaboga. *Intelligent optimisation techniques: genetic algorithms, tabu search, simulated annealing and neural networks*. Springer Science & Business Media, 2012.

[6] P. Santi, G. Resta, M. Szell, S. Sobolvesky, S. Strogatz, and C. Ratti. Quantifying the benefits of vehicle pooling with shareability networks. *PNAS*, 2014.