The Journal of Supercomputing
https://doi.org/10.1007/s11227-020-03424-6

High-capacity ride-sharing via shortest path clustering on large road networks

Haojia Zuo[1] · Bo Cao[1] · Ying Zhao[1] · Bilong Shen[1] · Weimin Zheng[1] · Yan Huang[2]

Accepted: 30 August 2020

Abstract

Ride-sharing has been widely studied in academia and applied in mobility-on-demand systems as a means of reducing the number of cars, congestion, and pollution by sharing empty seats. Solving this problem is challenging on large-scale road networks for the following two reasons: Distance calculation on large-scale road networks is time-consuming, and multi-request allocation and route planning have been proved to be NP-hard problems. In this paper, we propose a clustering-based request matching and route planning algorithm *Roo* whose basic operations are merging requested trips on road networks. Several requested trips can be merged and served by a vehicle if their shortest paths from origins to destinations are close to each other based on spatiotemporal road network distances. The resultant routes are further refined by introducing meeting points, which can shorten the total traveling distance while keeping matched ride requests satisfied. The *Roo* algorithm has been evaluated with two real-world taxi trajectory datasets and road networks from New York City and Beijing. The results show that *Roo* can save up to 50% of mileage by 1000 vehicles serving around 7000 trip requests in New York City between 7:40 and 8:00 am with an average waiting time of 4 minutes.

Keywords Spatial data mining · Trajectory mining · Ride-sharing · Route planning

✉ Ying Zhao
yingz@tsinghua.edu.cn

Yan Huang
Yan.Huang@unt.edu

[1] Department of Computer Science and Technology, Tsinghua University, Beijing, China

[2] Department of Computer Science, University of North Texas, Denton, TX, USA

Published online: 14 September 2020

1 Introduction

Ride-sharing has been widely studied in academia and applied in mobility-on-demand (MoD) systems as a means of reducing the number of cars, congestion, and pollution by sharing empty seats [2, 7, 14, 21, 25, 27]. In particular, the ride-sharing problem with vehicles as dispatching resources has drawn people's attention lately [3, 5, 13, 23]. In this scenario, ride requests within a time window t are acquired ahead in MoD systems, and spatiotemporally distributed demands must be picked up and delivered, which requires effective request matching and route planning algorithms to dispatch vehicles to serve the demands. Javier et al. [3] termed this problem as the *on-demand high-capacity ride-sharing* problem, which deals with a large number of online ride requests and vehicles of various capacities, such as taxis, vans, and minibuses.

Solving this problem is challenging on large-scale road networks for the following two reasons: Distance calculation on large-scale road networks is time consuming, and multi-request allocation and route planning have been proved to be NP-hard problems [19]. The existing methods use approximation algorithms to solve the latter problem, such as genetic algorithms [22, 29], heuristic search [10], iterative solution [35], and kinetic tree [15]. For large-scale road networks, existing methods assume distances between nodes in a road network are pre-computed and stored [3, 15], or use clustering as a filtering step to reduce the solution space [4, 10].

However, road networks of large cities, such as New York City or Beijing, contain millions of nodes, for which pre-computing and storing pairwise distances are not practical. In addition, the filtering step may assign ride requests to wrong groups from the beginning that may lead to poor ride-sharing performance. Thanks to recent advances in indexing large-scale road networks (V-Tree [28]), shortest paths, and distances between two locations can be computed within milliseconds. In this paper, we propose a clustering-based multi-request matching and route planning algorithm `Roo` that considers spatiotemporal distances between ride requests on road networks, and produces ride-sharing routes on-demand without pre-computed distances. The proposed algorithm `Roo` is efficient enough to be applied to a large number of online ride requests and vehicles on large road networks.

The problem that `Roo` is designed to solve can be defined in a literal way as follows. **Ride requests** (or **trips**) within a time window t are collected, each containing an origin, a requested departure time, and a destination, all pre-defined by passengers. A limited number of vehicles are dispatched to serve these trips. We refer to the trajectory that a vehicle travels as a **route**, which is described as a series of road network vertices (called **stops**) passed by the route, and the shortest paths connecting them sequentially. Passengers may get on or off the vehicle at any stop on the route. The optimization objective of `Roo` is to find a set of ride-sharing routes that lead to the most total traveling distance saving.

This paper makes the following contributions:

1. A clustering-based algorithm Roo is proposed, which solves request matching and route planning by merging ride-sharing candidates through a single clustering process.
2. A route refinement algorithm based on *meeting points* is proposed to shorten the total traveling distance while keeping matched ride requests satisfied.
3. The Roo algorithm was evaluated on real-world taxi trajectory datasets and road networks from New York City and Beijing and compared with two state-of-the-art methods.

The rest of the paper is organized as follows. Section 2 reviews related works. Section 3 defines the high-capacity ride-sharing problem. Section 4 proposes the clustering-based algorithm Roo. Section 5 presents the experimental results on two real-world taxi trajectory datasets and road networks. Section 6 concludes the paper.

2 Related work

Multi-request matching and planning plays an important role in ride-sharing systems, which itself is NP-hard [19, 26, 27]. In this section, we investigate the techniques for multi-request matching and route planning that can handle large numbers of ride requests and vehicles.

Clustering the origins and destinations of ride requests is often used as a filtering step to reduce the solution space, such as in Flexi [4], B-Planner [8] and BDTA [9]. Flexi. Bastani et al. [4] proposed a flexible minibus planning algorithm (shorted as Flexi) to serve ride requests collected beforehand with a goal of minimizing the total traveling time. Flexi first adopts a clustering algorithm to group ride requests with close origins and destinations and requested times together, and then uses a greedy method to plan routes by traveling through each cluster. Chuah et al. [9] proposed a greedy approach BDTA to plan new bus routes by analyzing taxi trajectories. BDTA uses DBSCAN to cluster origins and destinations of taxi trajectories and a greedy approach to plan new bus routes. The clusters formed by BDTA are only based on the locations of origins and destinations and are used as bus stops. B-Planner [8] also uses the taxi trajectory data to generate hotspot locations, constructs the night bus station site information, and then adopts a rule-based routing and pruning method to generate night bus lines. These methods divide the request matching and route planning into two processes and only consider origins and destinations of ride requests.

More complicated trajectory clustering algorithms are also available in the literature [11, 17]. In [11], probabilistic clustering was used to cluster trajectories that allows for overlapping in cluster models and objective learning of the distance metric and the number of clusters. Lee et al. [17] considered that complex trajectories may have related segments, and proposed a partition-and-group framework to discover common sub-trajectories. Although the goal of these trajectory clustering algorithms is not for determining ride-sharing routes, the idea of identifying overlapping segments inspires the design of our proposed method Roo.

Many route planning algorithms have been developed as well. Shrivastava et al. [29] designed an method to improve public transportation by combining a genetic algorithm to initially generate feeder routes and a heuristic repair algorithm that inserts nodes into the generated feeder routes. A genetic algorithm was also used in [22], with a focus on a novel variable string length coding that allows for a simultaneous search. Vazifeh et al. [33] made efforts to address the minimum fleet problem on large real road networks, with no shared rides considered. Ride requests that can be served one after another are discovered and assigned to one vehicle to minimize the fleet. Recent studies ([32, 34]) recommended destinations when planning ride-sharing routes using users' historical trajectory data or information from social networks.

Recently, on-demand ride-sharing systems on large road networks have drawn people's attention. Due to the complexity of request matching and route planning on large road networks, existing works often focus on certain ride-sharing types. Ta et al. [31] proposed a ride-sharing algorithm on large road networks, but only considered one-to-one matching between vehicles and passengers, while we focus on high-capacity ride-sharing. Mahin and Hashem [20] introduced a ride-sharing method on large road networks, but the traveling destinations are limited to predefined points of interests (POIs). Zhu et al. [36, 37] also introduced ride-sharing path-planning strategies for public vehicle systems on large road networks, in which passengers could transfer among different public vehicles, while we focus on ride-sharing strategies for MoD systems, in which passengers and vehicles are matched. Javier et al. [3] proposed a dynamic trip-vehicle assignment approach DTVA for on-demand high-capacity ride-sharing. DTVA first computes feasible trips from a pairwise shareability graph and then assigns trips to vehicles through an integer linear programming (ILP) to minimize waiting time and delay time. DTVA plans routes by solving the traveling salesman problem with a heuristic method, which is computationally expensive. Due to this high complexity, DTVA schedules a limited number of ride requests at a time, which undermines ride-sharing potentials. Qian et al. [24] also exploited group ride graphs to represent the shareability between requests and converted an ILP problem into a heuristic graph algorithm. The main differences to DTVA are that the shareability is evaluated from an economic perspective and the lack of consideration in taking the number of vehicles as a constraint in [24]. The differences between this work and ours are in twofold. Firstly, our proposed ride-sharing algorithm is not limited to serving trips that have nearby origins and destinations, but also allows ride-sharing between trips that have overlapped shortest paths. Secondly, Qian et al. [24] assigned one taxi to a group of matched ride requests directly and did not plan the actual ride-sharing routes. In a more realistic setting, the number of vehicles needs to be considered and ride-sharing routes need to be planned.

Finally, clustering moving objects is another technique to handle a large number of movements that has been studied in the fields of weather forecasting and animal migration analysis, etc. In [18], objects are grouped into moving micro-clusters that are kept small via rectangular bounding. The moving micro-cluster algorithm also considers collisions of two micro-clusters. In [16], efficient algorithms to handle the deletion of an object from a cluster, the insertion of an object to a cluster, and

the splitting and merging of clusters are constructed by a modified Birch algorithm based on object movement dissimilarity and clustering features. However, the goal of clustering moving objects is not to generate routes, but to find the object movement patterns.

3 High-capacity ride-sharing problem

3.1 Basic definitions

Definition 1 *(Road Network)* The road network is modeled as a directed weighted graph $G = \langle V, E \rangle$, where V is the set of network nodes and E is the set of edges. Each edge $(u, v) \in E$ means there is a road connecting vertex u and vertex v directly and the distance of the road is used as its weight.

Definition 2 *(Ride Request)* Given a road network G, a ride request (or trip) is represented by $tr = \langle tid, o, o_t, d, d_t \rangle$, where tid is the request ID, $tr.o$ is the origin of the request tr on the road network, $tr.o_t$ is the requested departure time, $tr.d$ is the destination of the request tr on the road network, and $tr.d_t$ is the earliest possible arrival time by taking the shortest path from $tr.o$ to $tr.d$.

Note that we can map the spatial positions of ride requests in latitude and longitude coordinates to nodes on road networks using the method in [6].

Given two vertices u and v on $G = (V, E)$, let $\texttt{SPath}(u, v)$ denote the shortest path from u to v and $\texttt{SPDist}(u, v)$ denote the shortest path distance. Recent high-efficiency data structures such as V-Tree [28] can support shortest path queries on large-scale road networks within milliseconds. Using these data structures, given a trip tr, the shortest path from $tr.o$ to $tr.d$ can be computed and denoted as $\texttt{SPath}(tr.o, tr.d)$ with the shortest path distance $\texttt{SPDist}(tr.o, tr.d)$. Given $tr.o_t$ and an average driving speed, we can estimate time stamps for each vertex on $\texttt{SPath}(tr.o, tr.d)$ and $tr.d_t$ accordingly.

Definition 3 *(Ride-sharing)* Given a set of K vehicles $\mathcal{V}$, the capacity (i.e., maximum number of passengers each vehicle can take) C, a road network G, and a set of n ride requests $\mathcal{T} = \langle tr_1, ..., tr_n \rangle$ within a time window t, the ride-sharing problem is to match n ride requests to K vehicles and generate multiple routes $\mathcal{R}$ to optimize a cost function, where each route provides the pick-up and drop-off locations of all matched ride requests recommended by a ride-sharing algorithm.

Figure 1 shows an example of four ride requests $tr_1, \ldots, tr_4$ with their origins and destinations shown as vertices. Suppose tr_1, tr_2, tr_3 are matched to a vehicle and tr_4 remains unsatisfied. The route taken by the vehicle, which is presented as solid lines, passes the pick-up and drop-off locations (assumed to be the same as the origins and destinations) of tr_1, tr_2, tr_3. The direct edge between two vertices

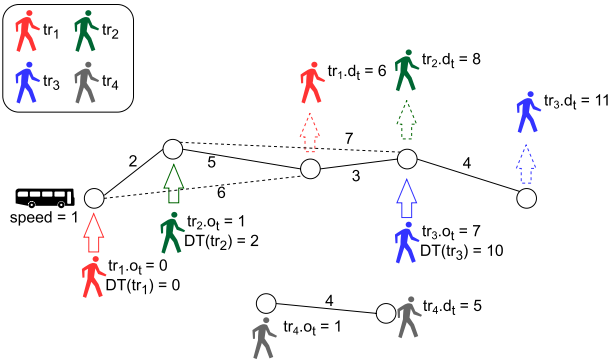


Fig. 1 An example of ride-sharing

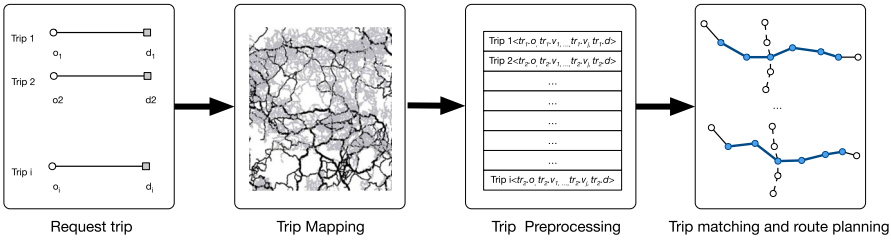


Fig. 2 Framework of high-capacity ride-sharing

in Fig. 1 represents a shortest path calculated on the road network. We also show the shortest path distance on each edge. The shortest path between the origin and destination of tr_1 and tr_2 is presented as dotted lines. The requested departure time of each trip is given at each origin. Suppose the average driving speed is 1, we can estimate the earliest possible arrival time for each trip using its requested departure time and shortest path distance. For example, tr_2 requests a ride at time 1, its shortest path distance to the destination is 7, hence its earliest possible arrival time $tr_2.d_t = 8$. Starting from $tr_1.o$, we can also estimate the actual pick-up time along the ride-sharing route for each trip, which is denoted as $DT(tr_i)$ for trip tr_i.

In this paper, we focus on the *high-capacity ride-sharing* problem, which deals with large number of passengers and vehicles with various capacity C suitable for car-pooling, shared vans, or minibuses. The framework of such ride-sharing system is shown in Fig. 2. The pre-collected ride requests (trips) are mapped onto road networks. These trips can also be pre-processed with fast road network algorithms such as V-Tree [28] to calculate distances and shortest paths on road networks. Then fast multi-request multi-vehicle matching and planning algorithms generate ride-sharing routes to optimize a cost function.

3.2 Cost function

Given a set of ride requests and a set of vehicles, ride-sharing algorithms can match some of the ride requests with available vehicles and plan ride-sharing routes, leaving the remaining ride requests unmatched. Ride-sharing algorithms may also let passengers wait or move to some meeting points [30] in order to plan more shared rides. In general, ride-sharing algorithms tend to minimize the total travel cost [1, 4, 19], waiting time for passengers [3], or maximize the matching rate [1, 3, 12, 19].

Definition 4 *(Dratio)* Given a set of ride requests $\mathcal{T}$, a set of vehicles $\mathcal{V}$, and a ride-sharing algorithm A, *Dratio* is defined as follows,

$$\text{Dratio} = \frac{D_A + D_{\hat{A}}}{D_{\text{raw}}},$$

where D_A is the total distance traveled by the ride-sharing routes $\mathcal{R}$ according to the ride-sharing algorithm A (i.e., $D_A = \sum_{r \in \mathcal{R}} length(r)$) , $D_{\hat{A}}$ is the total shortest path distance of the remaining unmatched requests, and $D_{\text{raw}} = \sum_{tr \in \mathcal{T}} \texttt{SPDist}(tr.o, tr.d)$, i.e., the total shortest path distance of all ride requests in $\mathcal{T}$ without ride-sharing.

In the example shown in Fig. 1. tr_1, tr_2, tr_3 are satisfied by a ride-sharing route, whose length is $D_A = 2 + 5 + 3 + 4 = 14$. tr_4 is not served, thus $D_{\hat{A}} = 4$. Let each of the four ride requests take its own shortest path, then we add their shortest path distances up to get the total distance $D_{\text{raw}} = 6 + 7 + 4 + 4 = 21$. Finally, we get $Dratio = \frac{14+4}{21} = \frac{6}{7}$.

Dratio [4] represents the proportion of the mileage of the ride-sharing routes computed by A to the total mileage of all ride requests in $\mathcal{T}$. The lower the value, the more the total mileage is saved. *Dratio* focuses on total mileage savings, and provides a means of comparison across different ride-sharing algorithms and ride request datasets.

Definition 5 *(Satisfied Ratio)* Given a set of ride requests $\mathcal{T}$, a set of vehicles $\mathcal{V}$, and a ride-sharing algorithm A, Satisfied Ratio is defined as follows,

$$\text{Satisfied Ratio} = \frac{|\mathcal{T}_A|}{|\mathcal{T}|},$$

where $\mathcal{T}_A$ is the set of the ride requests that can be matched by A given $\mathcal{V}$. Satisfied Ratio is the percentage of the ride requests that can be satisfied by a ride-sharing algorithm given $\mathcal{V}$.

In the example shown in Fig. 1, three out of four ride requests are satisfied. Therefore we get $Satisfied\ Ratio = \frac{3}{4}$.

Definition 6 *(Waiting Time)* Given a set of ride requests $\mathcal{T}$, a set of vehicles $\mathcal{V}$, and a ride-sharing algorithm A, *Waiting Time* is defined as follows,

$$\text{Waiting Time} = \frac{\sum_{tr \in \mathcal{T}_A} (DT_A(tr) - tr.o_t)}{|\mathcal{T}_A|},$$

where $\mathcal{T}_A$ is the set of the ride requests that can be matched by A given $\mathcal{V}$, $DT_A(tr)$ is the departure time of tr by taking the route computed by A. Waiting Time is the *average* waiting time of matched ride requests caused by taking ride-sharing routes computed by a ride-sharing algorithm given $\mathcal{V}$.

In the example shown in Fig. 1, the waiting time for tr_1, tr_2, and tr_3 is 0, 1, and 3, respectively. Hence, we have Waiting Time $= \frac{0+1+3}{3} = \frac{4}{3}$.

3.3 Problem formulation

The goal of the high-capacity ride-sharing problem is to find ride-sharing routes to minimize *Dratio* in order to save the total mileage. Given a set of n ride requests $\mathcal{T} = \langle tr_1, ..., tr_n \rangle$, K vehicles (each vehicle has a capacity of C), the problem is to find ride-sharing routes $\mathcal{R}$ to serve matched ride requests $\mathcal{T}_A$ that minimize *Dratio*.

We formalize this problem as a mixed integer programming (MIP) problem on a conceptual origin-destination (O-D) graph $G' = \langle V', E' \rangle$. V' consists of a set of origin vertices V'_o and a set of destination vertices V'_d. For each $tr_i \in \mathcal{T}$, we add a vertex i to V'_o and let $v_i = tr_i.o$, and add a vertex $i + n$ to V'_d and let $v_{i+n} = tr_i.d$. It is possible that two distinct vertices $i, j \in V'$ are mapped to the same road network node, if the two corresponding ride requests share the same origin or destination. With this vertex set V', G' is a directed complete graph with the weight of any edge $(i, j) \in E'$ (denoted as $D_{i,j}$) defined as the shortest path distance between v_i and v_j in G, i.e., $D_{i,j} = \texttt{SPDist}(v_i, v_j)$. For convenience, we use $\langle i, j \rangle \in \mathcal{T}$ to represent two vertices in V' that correspond to the origin and destination of a ride request in $\mathcal{T}$.

With this conceptual O-D graph G', we define the following decision variables:

$$y_{k,i,j} = \begin{cases} 1, \text{ if vehicle } k \text{ passes edge } (i,j), \\ 0, \text{ otherwise.} \end{cases}$$

$$\epsilon_{k,i} = \begin{cases} 1, \text{ if vehicle } k \text{ passes vertex } i, \\ 0, \text{ otherwise.} \end{cases}$$

$u_{k,i} =$ the number of passengers on vehicle k when it leaves vertex i,

$a_i =$ time at which a vehicle arrives at vertex i,

$z_{k,i}$ is an auxiliary binary variable.

In the following, M is a large enough constant, s_c is the average speed of vehicles, and WT is the max waiting time threshold. The high-capacity ride-sharing problem is thus formulated as follows.

$$\min \frac{1}{\sum_{\langle i,j \rangle \in \mathcal{T}} D_{i,j}} \left[\sum_{k=1}^{K} \sum_{(i,j) \in E'} D_{i,j} y_{k,i,j} + \sum_{\langle i,j \rangle \in \mathcal{T}} D_{i,j} (1 - \sum_{k=1}^{K} \epsilon_{k,i}) \right] \tag{1}$$

subject to

$$\sum_{k=1}^{K} \sum_{j|(i,j)\in E'} y_{k,i,j} \leq 1, \qquad \forall i \in V', \tag{2}$$

$$\sum_{k=1}^{K} \sum_{j|(j,i)\in E'} y_{k,j,i} \leq 1, \qquad \forall i \in V', \tag{3}$$

$$z_{k,i} = \sum_{j|(j,i)\in E'} y_{k,j,i} \oplus^{1} \sum_{j|(i,j)\in E'} y_{k,i,j}, \quad \forall i \in V',\ \forall k \in \mathbb{Z} : k \in [1,K], \tag{4}$$

$$z_{k,i} - M\left(2 - \sum_{k=1}^{K} \sum_{j|(j,i)\in E'} y_{k,j,i} - \sum_{k=1}^{K} \sum_{j|(i,j)\in E'} y_{k,i,j}\right) \leq 0, \quad \forall i \in V',\ \forall k \in \mathbb{Z} : k \in [1,K], \tag{5}$$

$$\sum_{l|(i,l)\in E'} y_{k,i,l} + \sum_{l|(j,l)\in E'} y_{k,j,l} \leq 1 + M \sum_{k=1}^{K} \left(\sum_{l|(l,i)\in E'} y_{k,l,i} + \sum_{l|(l,j)\in E'} y_{k,l,j} \right) \quad \forall i \in V', \forall j \in V', i \neq j, \forall k \in \mathbb{Z} : k \in [1,K], \tag{6}$$

$$\sum_{j|(i,j)\in E'} y_{k,i,j} = \epsilon_{k,i}, \qquad \forall i \in V'_o,\ \forall k \in \mathbb{Z} : k \in [1,K], \tag{7}$$

$$\sum_{j|(j,i)\in E'} y_{k,j,i} = \epsilon_{k,i}, \qquad \forall i \in V'_d,\ \forall k \in \mathbb{Z} : k \in [1,K], \tag{8}$$

$$\epsilon_{k,i} = \epsilon_{k,j}, \qquad \forall k \in \mathbb{Z} : k \in [1,K],\ \forall \langle i,j \rangle \in \mathcal{T}, \tag{9}$$

$$0 \leq \sum_{k=1}^{K} \sum_{i \in V'} \epsilon_{k,i} - \sum_{k=1}^{K} \sum_{(i,j)\in E'} y_{k,i,j} \leq K, \tag{10}$$

$$u_{k,j} \geq u_{k,i} + 1 - M(1 - y_{k,i,j}),\ \forall (i,j) \in E' \text{ and } j \in V'_o,\ \forall k \in \mathbb{Z} : k \in [1,K], \tag{11}$$

$$u_{k,j} \geq u_{k,i} - 1 - M(1 - y_{k,i,j}),\ \forall (i,j) \in E' \text{ and } j \in V'_d,\ \forall k \in \mathbb{Z} : k \in [1,K], \tag{12}$$

$$u_{k,i} \leq C, \qquad \forall i \in V',\ \forall k \in \mathbb{Z} : k \in [1,K], \tag{13}$$

$$a_j \geq a_i + \frac{1}{s_c} D_{i,j} - M\left(1 - \sum_{k=1}^{K} y_{k,i,j}\right), \qquad \forall (i,j) \in E', \tag{14}$$

$$tr_i.o_t - WT \leq a_i \leq tr_i.o_t + WT, \qquad \forall i \in V'_o, \tag{15}$$

$$a_i \leq a_j, \qquad \forall \langle i,j \rangle \in \mathcal{T}, \tag{16}$$

$$y_{k,i,j}, \epsilon_{k,i}, z_{k,i} \in \{0,1\},\ u_{k,i}, a_i \geq 0, \quad \forall i \in V',\ \forall (i,j) \in E',\ \forall k \in \mathbb{Z} : k \in [1,K]. \tag{17}$$

[1]

The objective function (1) minimizes *Dratio*. Constraints (2) and (3) ensure that there is at most one vehicle arriving at and leaving any vertex. Constraint (4) indicates that $z_{k,i}$ is the exclusive OR result of the two sums on the right-hand side which stand for whether vehicle k arrives at and leaves vertex i, respectively. Then constraint (5) further ensures that it is the same vehicle k which arrives at and leaves vertex i, unless i has zero in-degree or zero out-degree. In this way, we guarantee that any directed chain (i.e., a route) is served by a single vehicle all along. Constraint (6) ensures that two directed chains are not served by the same vehicle. Constraints (7) and (8) calculate $\epsilon_{k,i}$. Constraint (9) ensures that any ride request is served by one single vehicle, i.e., its origin and destination is on the same directed chain. Constraint (10) indicates that the number of directed chains, i.e., the number of routes, cannot exceed K. Constraints (11) and (12) ensure the number of passengers change reasonably when pick-ups and drop-offs occur, and constraint (13) guarantees the capacity is never exceeded. Constraint (14) legitimizes the arrival time when vehicles travel along their routes, and constraint (15) ensures that passengers are picked up at a proper time. Constraint (16) indicates the chronological order between the pick-up and the drop-off within any ride requests. Finally, constraint (17) sets the binary and non-negativity conditions.

Note that this MIP problem optimizes the objective function (1) globally and achieves optimal ride-sharing routes. However, solving this MIP problem is time consuming and infeasible for large number of ride requests, thus we introduce a clustering-based heuristic algorithm in the next section to find approximate solutions. In this heuristic algorithm, ride requests that share overlapped shortest paths are merged locally and the merges that lead to the most *Dratio* decrease are picked in a global fashion. In addition, in this MIP formulation passengers are picked up and dropped off at the exact origins and destinations specified in ride requests. Our proposed clustering-based algorithm provides extra flexibility allowing passengers to walk for a short distance to pick-up and drop-off locations.

[1] The XOR operation can be expressed by a set of linear inequalities. E.g. $z = x_1 \oplus x_2$ is equivalent to $z \leq x_1 + x_2,\ \ z \geq x_1 - x_2,\ \ z \geq x_2 - x_1,\ \ z \leq 2 - x_1 - x_2,\ \ \ z, x_1, x_2 \in \{0,1\}$.

Fig. 3 Two ride-sharing routes generated by Roo

Table 1 Sample trips

	T1	T2	T3	T4	T5	T6	T7
O	07:53	07:44	07:51	07:52	07:48	07:51	07:53
D	8:17	8:08	08:17	08:16	08:21	08:25	08:23

4 Clustering-based high-capacity ride-sharing algorithm Roo

We propose algorithm Roo for matching and planning routes for n requests with K vehicles based on road network distances. Roo achieves simultaneous request matching and route planning through a clustering process based on shortest paths, which is computationally affordable to deal with thousands of ride requests collected beforehand.

4.1 Motivation

We first use a sample example to illustrate the motivation of Roo.

Seven sample ride requests from the New York City taxi dataset are shown in red lines in Fig. 3, where $\langle O1, D1\rangle$, $\langle O2, D2\rangle$, $\langle O3, D3\rangle$, $\langle O4, D4\rangle$, $\langle O5, D5\rangle$, $\langle O6, D6\rangle$, and $\langle O7, D7\rangle$ are the origin and destination pairs of the seven requests. The starting and finishing times are shown in Table 1. If we only consider origins and destinations to group requests, we can easily find a route $O1, O2, O3 \rightarrow D3 \rightarrow D1, D2$ and a route $O5, O6 \rightarrow D5 \rightarrow D6$. $T4$ and $T7$ cannot be added to the shared routes, because their origins or destinations are not close enough to those of other requests. However, if we consider the shortest paths of each ride requests to find overlaps, we can extend the shared routes to be $O1, O2, O3 \rightarrow O4 \rightarrow D4 \rightarrow D3 \rightarrow D1, D2$ and $O5, O6, O7 \rightarrow D7 \rightarrow D5 \rightarrow D6$, as shown in green lines in Fig. 3. Now the shared two routes can serve all seven trip requests and save more total traveling distance.

4.2 Spatiotemporal distance on road networks

In order to check whether a ride-sharing route provides feasible pick-up and drop-off locations for a ride request, we need to define shareability in our study. The shareability can be determined by many factors, such as the difference between the requested departure time pf a ride request and the planned pick-up time offered by a route, the distance between the origin of a ride request and the pick-up location offered by a route, or other economic factors. In this study, we consider both spatial and temporal distances as the inconvenience caused by ride-sharing, and propose to use a unified measure to normalize these two.

Definition 7 *(Vertex-to-vertex Spatiotemporal Distance)* Given two vertices u and v, and their time stamps u_t and v_t, we define the spatiotemporal distance based on road network from u to v as follows,

$$\texttt{SPTDist}(u, v) = \sqrt{\left(\frac{\texttt{SPDist}(u, v)}{w_{sp}}\right)^2 + \left(\frac{u_t - v_t}{w_t}\right)^2},$$

where w_{sp} and w_t are normalization factors to balance the importance of spatial distance and temporal distance.

For example, let $w_{sp} = 250m$ and $w_t = 6min$, the condition $\texttt{SPTDist}(u, v) \leq 1$ means that the road network distance between u, v $\texttt{SPDist}(u, v)$ cannot be greater than 250 meters, and the time difference between u_t and v_t cannot be greater than 6 minutes.

Definition 8 *(Vertex-to-path Spatiotemporal Distance)* Given a vertex u with its time stamp u_t and a path p with time stamps on all vertices in p, we define the spatiotemporal distance $\texttt{SPTDist}$ from u to p as the $\texttt{SPTDist}$ from u to its nearest vertex contained in p,

$$\texttt{SPTDist}(u, p) = \min_{v_i \in p}\{\texttt{SPTDist}(u, v_i)\}.$$

Similarly, we define the $\texttt{SPTDist}$ from p to u as

$$\texttt{SPTDist}(p, u) = \min_{v_i \in p}\{\texttt{SPTDist}(v_i, u)\}.$$

4.3 Merging routes

Definition 9 *(Route r)* A route r can be planned to serve one or more request trips, which are stored in $r.trips$. Let $r.stops$ be a set of stops on r that are pick-up locations, drop-off locations, and intermediate vertices on the shortest paths of all request trips ordered by their time stamps. Let $r.o$ and $r.d$ be the first and last stop on r, respectively. Let $r.p$ be the shortest path passing through $r.stops$ with the

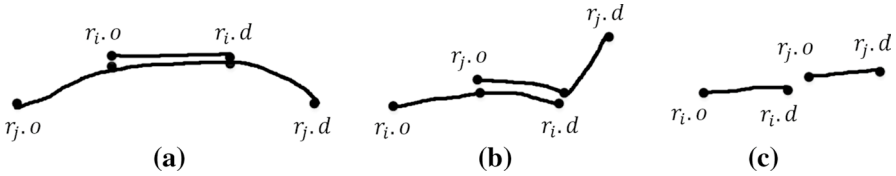


Fig. 4 Three cases of relative location between two routes

shortest path distance $r.dist$. The pick-up and drop-off location of a trip tr in $r.trips$ is determined as the stop on r that defines $\texttt{SPTDist}(tr.o, r.p)$ and $\texttt{SPTDist}(r.p, tr.d)$, respectively.

Function $\texttt{MergeRoute}(r_i, r_j)$ checks whether routes r_i and r_j can be merged together, and returns r_{cand} if it succeeds, otherwise returns NULL. We design this procedure in three steps: candidate identification, route planning, and feasibility check.

Candidate identification Given two routes r_i and r_j, if they (or parts of them) are close enough both spatially and temporally, such as in case A and B as shown in Fig. 4, we may find a path segment shared by the two routes. By merging them into one route r_{cand}, if r_{cand} can satisfy all trips in $r_i.trips$ and $r_j.trips$, we can reduce the overall traveling distance successfully.

We use a simple criterion to identify such candidates: one route needs to pass near the origin of another route to start a shared segment (*eg.,* r_i passing near $r_j.o$ in case B); and one of the two destinations needs to be close to the other route to end a shared segment.

Formally, **merge condition one** is,

$$\begin{aligned} &\min\{\texttt{SPTDist}(r_i, r_j.o), \texttt{SPTDist}(r_j, r_i.o)\} \le \theta_1 \\ \texttt{and } &\min\{\texttt{SPTDist}(r_i.d, r_j), \texttt{SPTDist}(r_j.d, r_i)\} \le \theta_1, \end{aligned} \tag{18}$$

where θ_1 is the maximum allowed SPTDist difference for merging routes.

For case C in Fig. 4, if two routes r_i and r_j can be traveled by one vehicle one by one, they can be merged to become a lengthened route r_{cand}.

Formally, **merge condition two** is,

$$\begin{aligned} &0 \le r_j.o_t - (r_i.d_t + \texttt{SPDist}(r_i.d, r_j.o)/s_c) \le \theta_2 \\ \texttt{or } &0 \le r_i.o_t - (r_j.d_t + \texttt{SPDist}(r_j.d, r_i.o)/s_c) \le \theta_2, \end{aligned} \tag{19}$$

where s_c is the average speed of a vehicle, and θ_2 is the maximum allowed waiting time for a vehicle.

The distance traveled from $r_i.d$ to $r_j.o$ is considered as operational cost, thus $r_{\text{cand}}.dist = r_i.dist + r_j.dist$. Note that this merging itself will not reduce the total incurred traveling distance if there is no limit on the number of vehicles. When the number of vehicles is limited, r_{cand} covers trips both in r_i and r_j by just one vehicle, leaving more vehicles available to cover other trips and potentially to reduce the total traveling distance.

Route planning We first determine the set of stops in r_{cand}, and then form $r_{\text{cand}}.p$ accordingly.

When r_i and r_j have a shared segment, we determine the set of stops S from r_i and r_j that are in the shared segment through a function called `GetSharedSegment()`. The stops in r_i and r_j that are not in the shared segment are copied to $r_{\text{cand}}.stops$. We adopt a simple heuristic to replace nearby stops (pairwise `SPTDist` is less than θ_1) in S with their mean as a new stop, which is done through a function called `planning(S)`.

When r_i and r_j satisfy merge condition two, we can simply make $r_{\text{cand}}.stops$ be the union of $r_i.stops$ and $r_j.stops$.

Feasibility check. Finally, we need to check whether the planned path $r_{\text{cand}}.p$ can serve all requested trips. In particular, we need to check the following **feasibility condition** to ensure the feasibility of r_{cand},

$$\begin{aligned} \forall tr \in r_{\text{cand}}.trips, \texttt{SPTDist}(tr.o, r_{\text{cand}}.p) \le \theta_1 \\ \texttt{and } \texttt{SPTDist}(r_{\text{cand}}.p, tr.d) \le \theta_1. \end{aligned} \tag{20}$$

We also check whether the route can satisfy the capacity limit in this step. From the first stop, we track the number of passengers on the vehicle and check it every time when passengers join or leave the route.

Function MergeRoute(r_i, r_j)

```
if r_i = φ or r_j = φ then
    return NULL;
if r_i and r_j satisfy merge condition one then
    rename r_i and r_j so that r_i starts first;
    r ← the one ends later in r_i and r_j;
    S ← GetSharedSegment(r_i, r_j);
    S' ← Planning(S);
    r_cand.o ← r_i.o; r_cand.d ← r.d;
    r_cand.stops ← remaining stops in r_i ∪ S' ∪ remaining stops in r_j;
    r_cand.p ← remaining path in r_i ∪ a shortest path p' passing through S' ∪
      remaining path in r_j;
    r_cand.dist ← length(p') + remaining dist of r_i and r_j;
else
    if r_i and r_j satisfy merge condition two then
        rename r_i and r_j so that r_i starts first;
        r_cand.o = r_i.o; r_cand.d = r_j.d;
        r_cand.p ← r_i.p ∪ r_j.p;
        r_cand.stops ← r_i.stops ∪ r_j.stops;
        r_cand.dist = r_i.dist + r_j.dist;
    else
        return NULL;
r_cand.trips ← r_i.trips ∪ r_j.trips;
if r_cand passes feasibility condition then
    return r_cand;
else
    return NULL;
```

4.4 Refining routes

We keep the route planning step used in `MergeRoute`(r_i, r_j) simple and fast, as `MergeRoute`(r_i, r_j) is called frequently when clustering ride requests. However, once we obtain the clustering results, i.e., ride-sharing routes, we can refine these routes using more complicated algorithms. Here we propose a route refinement algorithm `RefineRoute`(r) based on **meeting points** [30].

Definition 10 *(Meeting Point)* Given a road network G, a route r in G, and the spatial distance threshold w_{sp}, for each trip tr in $r.trips$, we define the meeting points for $tr.o$ or $tr.d$ as the set of vertices in G whose `SPDist` to $tr.o$ or $tr.d$ is less than or equal to w_{sp}, i.e., the locations in G that are within the walking distance to the requested origins or destinations of $r.trips$.

For example, the solid lines shown in Fig. 5 form a route r and the circles are origins or destinations of $r.trips$. The meeting points of each circle can be computed when receiving the ride request. They are represented as crosses in Fig. 5. If we can identify common meeting points shared by multiple origins or destinations of $r.trips$ (shown as dark dots in Fig. 5), we can use them to shorten the total traveling distance while keeping all ride requests satisfied. The shortened route is shown as dashed lines in Fig. 5.

Our proposed algorithm `RefineRoute`(r) is based on the following proposition.

Proposition 1 *Given a feasible route r with its pick-up and drop-off order, if a set of adjacent pick-up and drop-off stops share a common meeting point, replacing these stops with the meeting point in r will not cause capacity violations.*

Given a feasible route r with its pick-up and drop-off order, starting from the first pick-up stop, `RefineRoute`(r) repeats the following two steps:

(1) Find the next set of adjacent pick-up and drop-off stops that share a common meeting point;
(2) Estimate the time stamp of the meeting point, replacing these stops with the meeting point in r if such replacement passes **feasibility condition**.

4.5 Clustering-based ride-sharing algorithm `Roo`

`Roo`($\mathcal{T}, K$) takes a given set of n trip requests $\mathcal{T}$, and the number of routes to be planned K as the input, and the output is K planned routes.

Given a set of n trips $\mathcal{T}$, we first initialize the route set $\mathcal{R}$ by making each trip a route as shown in lines 2-7 of Algorithm 1. The decrease in *Dratio* Δ_{Dratio} generated by merging the two routes is calculated by `DeltaDratio`() and stored in a priority queue. Lines 9-12 initialize the Δ_{Dratio} of all mergeable trips.

We repeat the following two steps:

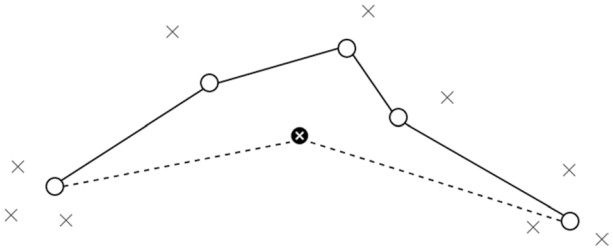


Fig. 5 Route refining based on meeting points

(1) Merge r_i and r_j that lead to the maximum decreased *Dratio* value in Q, the merged route is recorded as new r_i and updated in $\mathcal{R}$ as in lines 16 and 17;
(2) Update Q by removing Δ_{Dratio} of all merges involving r_i or r_j as in line 19, and adding Δ_{Dratio} of all merges with r_{merged} as in lines 20–22.

Algorithm 1: Roo($\mathcal{T}$, K))

```
1  R ← φ;
2  for i = 1 to n do
3  |  Route r_i;
4  |  r_i.o = tr_i.o; r_i.d = tr_i.d; r_i.p = SPath(tr_i.o, tr_i.d); r_i.dist = SPDist(tr_i.o, tr_i.d);
5  |  r_i.stops = ⟨tr_i.o, tr_i.d⟩;
6  |  r_i.trips = ⟨tr_i⟩;
7  |  R.Insert(r_i);
8  PriorityQuery Q ← φ;
9  for i = 1 to n do
10 |  for j = i + 1 to n do
11 |  |  if (r_merged = MergeRoute(r_i, r_j)) ≠ NULL then
12 |  |  |  Q.Insert(⟨DeltaDratio(r_i, r_j, r_merged), i, j⟩);
13 while Q is not empty do
14 |  ⟨key, i, j⟩ ← Q.Top();
15 |  if key > 0 then
16 |  |  r_i = MergeRoute(r_i, r_j);
17 |  |  R.Update(r_i);
18 |  |  R.Delete(r_j);
19 |  |  Q.Delete(i); Q.Delete(j);
20 |  |  for l = 1 to size(R) do
21 |  |  |  if (r_merged = MergeRoute(r_i, r_l)) ≠ NULL then
22 |  |  |  |  Q.Insert(⟨DeltaDratio(r_i, r_l, r_merged), i, l⟩);
23 sort R according to total distance of satisfied trips;
24 RefineRoute(R);
25 return top min{K, size(R)} routes in R;
```

At the end of clustering, routes are sorted according to the total distance of the trips they satisfied and refined using `RefineRoute`($\mathcal{R}$). The top min $\{K, size(\mathcal{R})\}$ routes are refined and selected as the output.

Note that adding Δ_{Dratio} of all merges with r_{merged} (as in lines 20–22) is the most costly step in `Roo`. Given n trips, the initial size of Q is determined by the number of merged routes after executing lines 9-12. In the worst case, the initial size of Q can be as large as n^2. However, the average number of trips that can be merged with a given trip in line 11 is most likely a constant, thus the initial size of Q is a linear function of n. In addition, $size(\mathcal{R})$ is bounded by n. Hence, `Roo` is a quadratic procedure with respect to the size of ride requests.

5 Experiment

5.1 Experimental environment and settings

Dataset: Road network data Two real-world road networks were used in our experiments: Beijing Urban Road Network and New York City Road Network. The Beijing Urban Road Network contains 171,078 nodes and 462,178 edges; the New York City Road Network contains 264,346 nodes and 730,100 edges.

Dataset: Ride request Two real data sets were used in our experiments: BJTaxi and NYCTaxi. BJTaxi consists of real taxi requests in Beijing on March 15, 2009, which contains 130,794 real taxi trajectories. NYCTaxi consists of real taxi requests in New York City on May 10, 2016, which contains 400,088 real taxi trajectories. We used the existing method [6] to map the locations of origins and destinations of all trips onto the road network.

Metrics Three metrics *Dratio*, *Satisfied Ratio*, and *Waiting Time* are used to evaluate various ride-sharing algorithms. *Dratio* measures the effectiveness of ride-sharing by comparing traveling distance under ride-sharing with the total traveling distance of all individual ride requests. The satisfied ratio is the percentage of ride requests that can be satisfied by a ride-sharing algorithm given k vehicles. *Waiting Time* is the difference between the real pick-up time and the requested pick-up time, averaged over all satisfied requests.

Baseline The two-stage clustering route planning algorithm [4] `Flexi` and the dynamic trip-vehicle assignment algorithm [3] `DTVA` are used as baselines. `Flexi` also minimizes *Dratio* and is stricter for merging trips by requiring both origins and destinations of ride requests are close. `DTVA` minimizes waiting time and finds ride-sharing routes by solving an integer linear programming problem on trip-vehicle assignment graphs.

Experimental environment The experiment was performed on a 64-bit Linux computer with an Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz CPU and 64 GB RAM. The algorithm `Roo` is based on C++ implementation.

Parameter settings. We summarize the default parameter settings in Table 2.

Both `Roo` and `Flexi` need parameters for calculating spatial and temporal distances. w_{sp} and w_t are set to 250m and 6min, respectively. θ_1 is set to 1, which is

Table 2 Default parameter settings

	w_{sp}	w_t	Maximum Delay time	K	Capacity	Time window (min)
Roo	250	6 min	–	1000	4	20
Flexi	250	6 min	–	1000	20	20
DTVA	–	–	6 min	1000	4	20

based on a reasonable assumption that an individual passenger can tolerate a time variation of no more than 6 minutes waiting and a walking distance of no more than 250 meters. Maximum delay time for DTVA is set to 6 minutes accordingly. θ_2 is set to 15 minutes, which means a vehicle will not wait for a customer for more than 15 minutes. The driving speed of the vehicle on each edge of the network s_c is set to 30km/h.

Capacity limits for Roo and DTVA are both set to 4 as default while capacity limit for Flexi is set to 20 as default. The reason is that the vehicle used in Roo and DTVA is typically a car while that for Flexi is typically a minibus. The number of vehicles K is 1000 for all three algorithms on both BJTaxi and NYCTaxi. Roo and Flexi are fast enough to plan all ride requests within a time window of up to 30 minutes, and we use the time window of 20 minutes as default. DTVA is more expensive and can only handle ride requests within a time window of 30 seconds in real-time. We let DTVA run multiple rounds during longer time windows as suggested in [3].

5.2 Ride-sharing results

This set of experiments compare Roo with baselines in various time periods on both datasets. We take 20 minutes as a time window, dividing one day into 72 time windows, and create 72 sets of ride requests accordingly. We restrict the number of vehicles to 1000, and the ride-sharing results in various time periods are fully examined. The experimental results are shown in Fig. 6.

On both datasets, starting from 0:00 am, we can see the *Dratio* rises for Roo as the number of ride requests drops during late nights, and suddenly drops when morning rush hour begins at 6:00 am. Flexi also shows the similar trend on NYCTaxi, but fails on BJTaxi.

Roo shows superior planning performance over Flexi on both datasets, because Roo considers overlapping ride requests and is more flexible than Flexi. For example, on the New York City dataset, Flexi achieved a *Dratio* value of 0.8961 at 18:00–18:20, while Roo achieved 0.4130. Roo saves 48% more mileage than the Flexi method. While Flexi fails on BJTaxi as the ride requests are much sparser than those in NYCTaxi, Roo can still make improvements on *Dratio*.

Roo outperformed DTVA in terms of *Dratio* and *Satisfied Ratio* on both datasets as well. Since DTVA is designed to minimize *Waiting Time*, it is not surprising to see that DTVA was able to achieve an average waiting time around 1 minute on both datasets, whereas Roo achieved between 2-5 minutes.

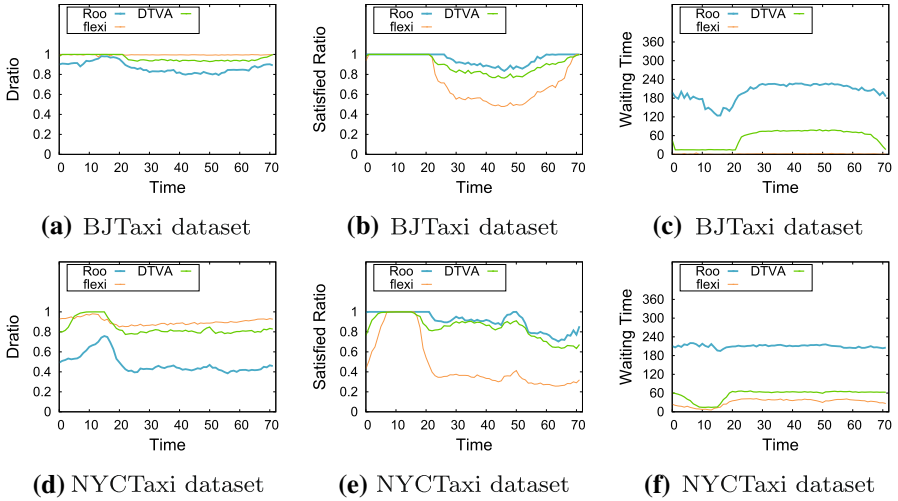


Fig. 6 Ride-sharing results in different time periods

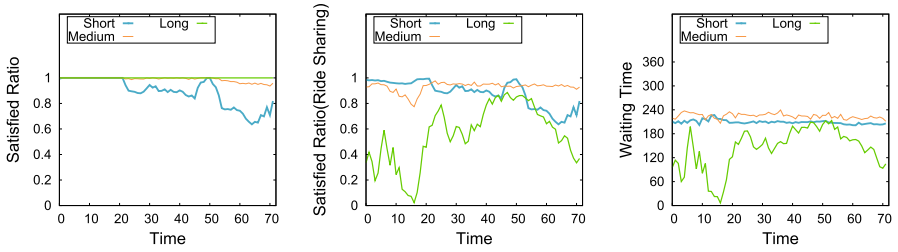
Furthermore, we break down the results of Roo shown in Fig. 6 by looking at the ride-sharing performance of ride requests with various lengths. In particular, we classified ride requests as "short", "medium", and "long" if their Manhattan distance between the origin and destination is less than 5km, between 5km to 15km, and greater than 15km, respectively. Figure 7a shows the satisfied ratio of short, medium and long ride requests. Note that long ride requests tend to have higher satisfied ratio, because Roo selects longer routes in the hope of achieving higher *Dratio* values. Figure 7b shows the ratio that ride requests are satisfied by a ride-sharing route. A ride-sharing route is a route that satisfies more than one trip. It can be seen that this ratio of short trips only drops slightly comparing with the result in Fig. 7a. It means that most short trips can take part in ride-sharing. However, this ratio of long trips drops a lot, meaning that many long trips are served individually. Figure 7c shows the average waiting time. Long trips have lower waiting time as many of them are satisfied individually.

5.3 Parameter studies

In this section, we first examine the performance of our route refinement algorithm using ride requests of the whole day. Then we conduct comprehensive studies on various parameters on the ride requests during morning rush hours 07:00–09:00 on both BJTaxi and NYCTaxi datasets.

5.3.1 Route refinement

This set of experiments examines the effectiveness of our route refinement algorithm on *Dratio*. Our route refinement algorithm can maintain the same *Satisfied Ratio* and



(a) NYCTaxi dataset **(b)** NYCTaxi dataset **(c)** NYCTaxi dataset

Fig. 7 Ride-sharing results on trips with different length ranges

keep *Waiting Time* under 5 mins, so we focus on how effective it is to shorten the total traveling distance. The experimental results are shown in Fig. 8.

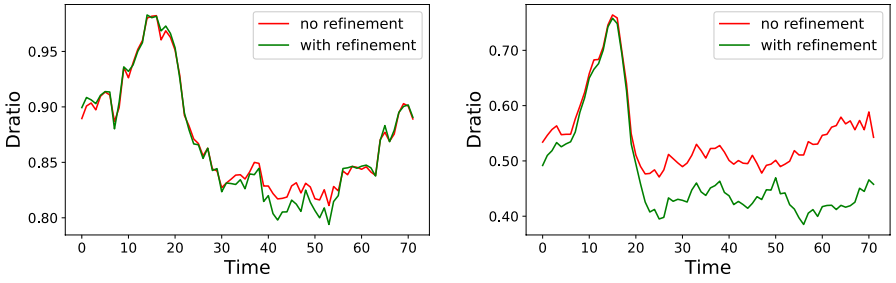
We can see that our route refinement strategy brings a higher decrease in *Dratio* on NYCTaxi than that on BJTaxi. As the ride requests on NYCTaxi are much denser, drivers have to go along winding paths in order to pick up or drop off passengers at very nearby places. In this case, traveling distances can be shortened considerably by introducing meeting points, resulting in a more obvious drop in *Dratio* compared to the case on BJTaxi. Our route refinement strategy also works more effectively on day times than late nights, which is not surprising as there are rarely ride-sharing routes with enough requests to find meeting points during late nights. Nevertheless, the *Dratio* on NYCTaxi dropped more than 10% on average during day times.

5.3.2 Varying θ_1

θ_1 determines the easiness to merge routes in `Roo`. Larger θ_1 means routes with larger difference can be merged, i.e., more shared rides will be planned given that passengers will wait for a longer time or walk for a longer distance. This set of experiments examines the effect of θ_1 and the results are shown in Fig. 9. As θ_1 increases, a route can satisfy more trips at the same time, which leads to lower *Dratio* values and higher *Satisfied Ratio* values. However, *Waiting Time* increases as well. We set θ_1 to be 1 to trade off between planning more shared rides and keeping passengers convenient.

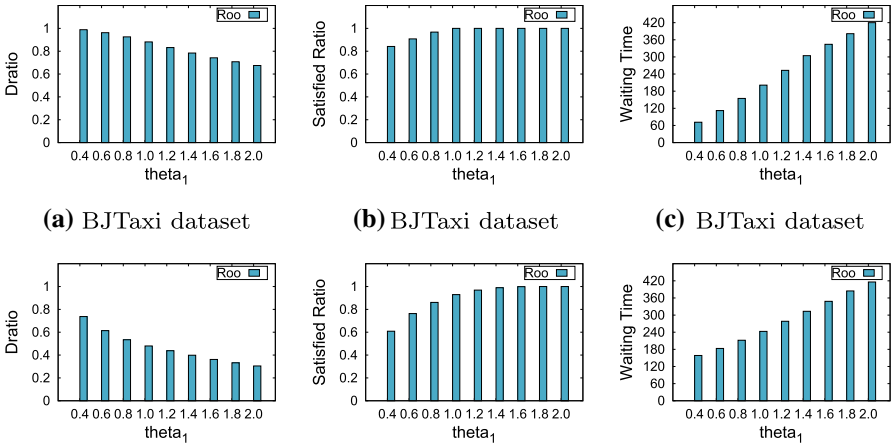
5.3.3 Varying vehicle capacity

Our experiments assume that each ride request in the dataset corresponds to only one passenger. To show the effect of different vehicle capacity limits, we set the vehicle capacity limit from 1 to 6 and unlimited on BJTaxi, and from 1 to 7 and unlimited on NYCTaxi. The ride requests are also partitioned using a time window of 20 minutes, and the three metric values produced by `Roo` and `DTVA` are averages of all the 6 time periods during morning rush hours. Due to the complexity of `DTVA`, it cannot run in real-time for NYCTaxi with the vehicle capacity limit of 7.



(a) BJTaxi dataset **(b)** NYCTaxi dataset

Fig. 8 Effectiveness of route refinement



(a) BJTaxi dataset **(b)** BJTaxi dataset **(c)** BJTaxi dataset

(d) NYCTaxi dataset **(e)** NYCTaxi dataset **(f)** NYCTaxi dataset

Fig. 9 Varying θ_1

The experimental results are shown in Fig. 10. When the capacity limit is 1, the *Dratio* value is 1, which means no ride-sharing happened. When the vehicle capacity increases, *Dratio* gradually decreases for both `Roo` and `DTVA`. When the vehicle capacity is limited to 4, the *Dratio* value of `Roo` on the New York City dataset is 0.4186, the *Dratio* value on the Beijing dataset is 0.8783. These *Dratio* values are still better than those of `Flexi` without capacity limit (0.8598 on NYCTaxi and 0.9961 on BJTaxi).

5.3.4 Varying number of vehicles

This set of experiments varies the number of vehicles K that can be used for ride-sharing. We examine the impact of the number of vehicles on *Dratio*, *Satisfied Ratio*

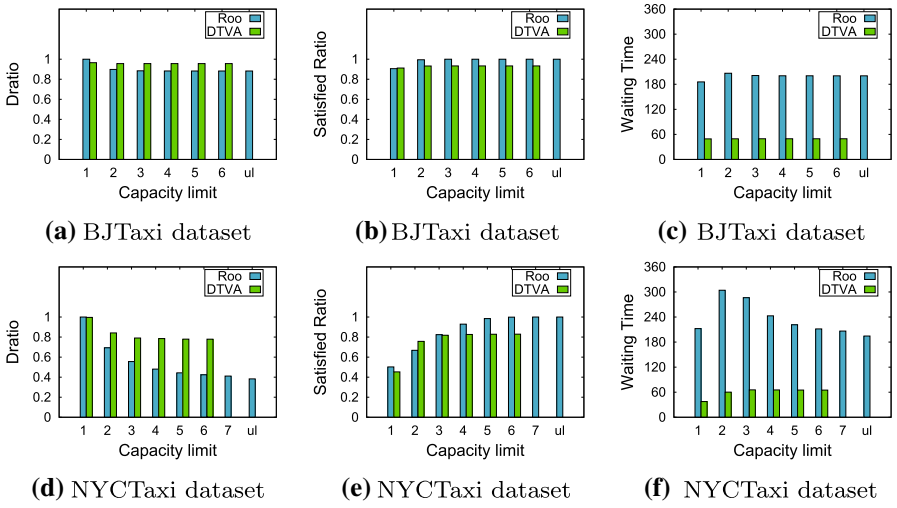


Fig. 10 Varying vehicle capacity limits

and *Waiting Time* for Roo, DTVA and Flexi. The experimental results are shown in Fig. 11.

We can see that with the same number of vehicles, Roo can reach lower *Dratio* and higher *Satisfied Ratio* than Flexi and DTVA. In order to meet the same proportion of the original ride requests, Roo needs to invest fewer vehicles. For example, to match 80% of ride requests in New York City datasets between 07:40-08:00, Roo only needs around 800 vehicles while DTVA and Flexi requires about 1100 and 2500 vehicles, respectively. It is also interesting to observe that as K increases, *Waiting Time* decreases and *Dratio* increases for DTVA. Since DTVA minimizes *Waiting Time*, it prefers vehicles to serve single customers rather than to serve shared rides when given extra vehicles.

5.3.5 Varying time window size

This set of experiments varies time window sizes and examines the ride-sharing results under different window sizes. We set the time window to 5 minutes, 10 minutes, 20 minutes, 30 minutes, and 60 minutes. The ride requests are partitioned according to different window sizes, and the averaged values of *Dratio* for all time periods during morning rush hours from 7:00 am to 9:00 am are calculated and reported in Fig. 12.

We can find that *Dratio* first decreases when increasing the window size and then has no significant changes after 20 minutes on both datasets. This is because when the time window is small, larger time window sizes can increase the potential of ride-sharing. After 20 minutes, the potential of ride-sharing is constrained by the capacity limit and the number of requests per time unit. Hence, 20 minutes

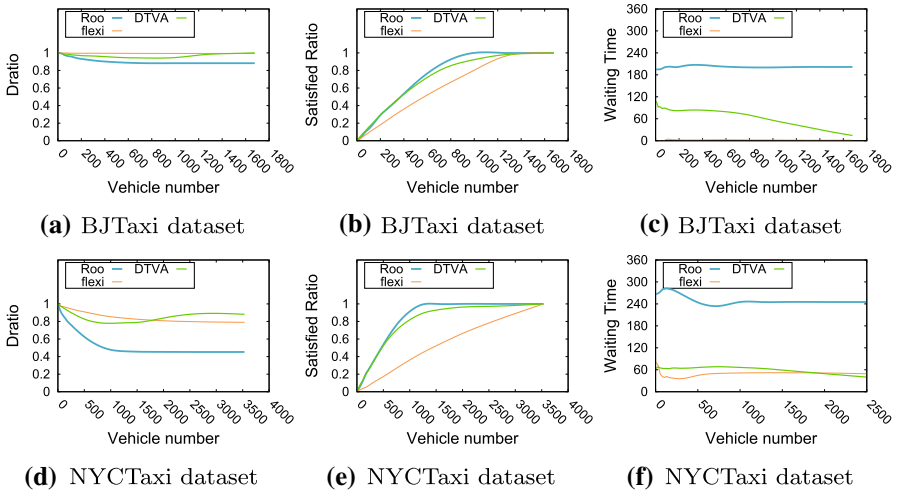


Fig. 11 Varying number of vehicles

is a reasonable choice for balancing the computing cost and the effectiveness of ride-sharing.

5.4 Running time

The computing complexity of Roo mainly comes from two parts: distance calculations on large-scale road networks, and the agglomerative clustering procedure. With the support of V-Tree [28], distance between two locations can be computed within milliseconds on large scale road networks such as in Beijing and New York City. The agglomerative clustering is a quadratic procedure of the number of ride requests and varies on datasets depending on how many ride-sharing routes can be found.

We evaluated the running time of Roo on both BJTaxi and NYCTaxi datasets. The time period 07:00–09:00 is selected to evaluate the running time on varying time window sizes, as it contains the largest number of ride requests. We recorded both total running time and initialization time (which mainly calculates pairwise distances of ride requests on road networks) of Roo. As shown in Table 3, Roo plans ride-sharing routes in minutes when the time window size is smaller than 20 minutes. Since Roo does not gain much with time window sizes larger than 20 minutes, it is beneficial and affordable for ride-sharing systems to adopt Roo if it can acquire ride requests minutes ahead in practice. We also compared the running time of Flexi and DTVA on the same datasets in Table 3. Flexi runs much faster than Roo, because Flexi does not compute distances on road networks and adopts a greedy planning strategy that is simpler than the agglomerative clustering used in Roo. DTVA is more expensive for planning routes. We show

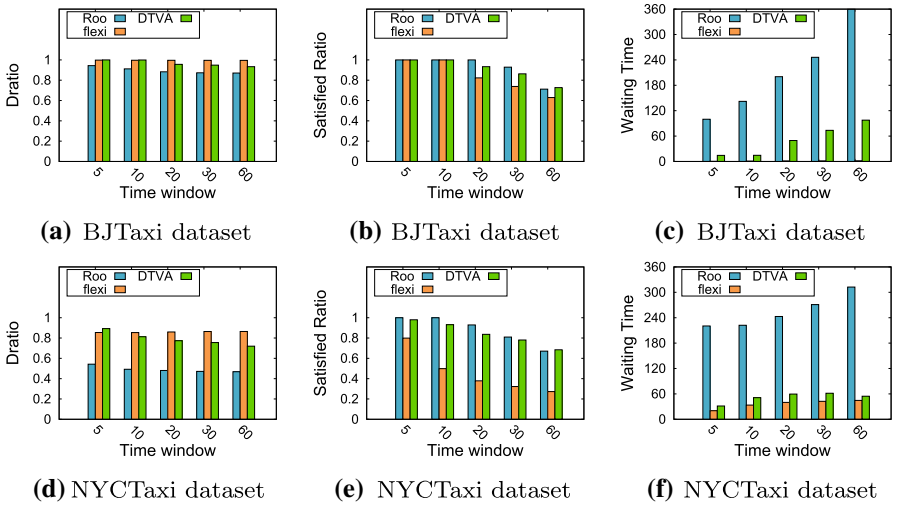


Fig. 12 Varying time window sizes

Table 3 Running time (in minutes)

Time window	5 min	10 min	20 min
BJ avg # of requests	405	810	1620
BJ `Roo` total time	0.2	0.4	0.8
BJ `Roo` initialization time	0	0.1	0.3
BJ `Flexi`	0.02	0.02	0.03
BJ `DTVA`	0.2	0.5	1.2
NYC avg # of requests	1700	3400	6800
NYC `Roo` total time	0.5	1.6	8.3
NYC `Roo` initialization time	0.2	0.6	4.0
NYC `Flexi`	0.03	0.06	0.16
NYC `DTVA`	1.1	4.0	16.6

the accumulated running time of multiple rounds of `DTVA` for each time window, where for each round `DTVA` plans ride requests collected of 30 seconds.

6 Conclusions

The problem of multi-request matching and route planning for ride-sharing systems is NP-hard. It is very challenging to effectively dispatch vehicles to satisfy large number of online ride requests on road networks on demand. In this paper, we propose a spatiotemporal similarity measurement of ride requests based on the shortest

path distance on road networks. Based on this measurement, we propose a multi-request matching and route planning algorithm Roo that merges overlapped ride requests into shared routes and refines them based on meeting points. The results show that Roo can save up to 50% of mileage by 1000 vehicles serving about 7000 trip requests in New York City between 7:40 am to 8:00 am with an average waiting time of 4 minutes.

References

1. Agatz N, Erera A, Savelsbergh M, Wang X (2011) Dynamic ride-sharing: a simulation study in metro atlanta. Procedia—social and behavioral sciences. In: Papers selected for the 19th International Symposium on Transportation and Traffic Theory 17:532–550
2. Agatz N, Erera A, Savelsbergh M, Wang X (2012) Optimization for dynamic ride-sharing: a review. Eur J Oper Res 223(2):295–303
3. Alonso-Mora J, Samaranayake S, Wallar A, Frazzoli E, Rus D (2017) On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. Proc Natl Acad Sci 114(3):462–467
4. Bastani F, Xie X, Huang Y, Powell JW (2011) A greener transportation mode: flexible routes discovery from GPS trajectory data. In: 19th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS 2011, November 1-4, 2011, Chicago, IL, USA, Proceedings, pp 405–408
5. Berbeglia G, Cordeau J-F, Laporte G (2010) Dynamic pickup and delivery problems. Eur J Oper Res 202(1):8–15
6. Cao L, Krumm J (2009) From gps traces to a routable road map. In: GIS, New York, NY, USA. ACM, pp 3–12
7. Carrion C, Levinson D (2012) Value of travel time reliability: a review of current evidence. Transp Res Part A Policy Pract 46(4):720–741
8. Chen C, Zhang D, Li N, Zhou Z-H (2014) B-planner: planning bidirectional night bus routes using large-scale taxi gps traces. IEEE Trans Intell Transp Syst 15(4):1451–1465
9. Chuah SP, Wu H, Lu Y, Yu L, Bressan S (2016) Bus routes design and optimization via taxi data analytics. In: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, ACM, pp 2417–2420
10. Fan W, Machemehl RB (2004) Optimal transit route network design problem: algorithms, implementations, and numerical results. Technical report
11. Gaffney S, Smyth P (1999) Trajectory clustering with mixtures of regression models. In: Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 15-18, 1999, pp 63–72
12. Gidofalvi G, Pedersen TB, Risch T, Zeitler E (2008) Highly scalable trip grouping for large-scale collective transportation systems. In: Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology, EDBT'08, New York, NY, USA. ACM, pp 678–689
13. Golden BL, Raghavan S, Wasil EA (2008) The vehicle routing problem: latest advances and new challenges, vol 43. Springer, Berlin
14. Hennessy DA, Wiesenthal DL (1999) Traffic congestion, driver stress, and driver aggression. Aggress Behav Off J Int Soc Res Aggress 25(6):409–423
15. Huang Y, Bastani F, Jin R, Wang XS (2014) Large scale real-time ridesharing with service guarantee on road networks. Proc VLDB Endow 7(14):2017–2028
16. Jensen CS, Lin D, Ooi BC (2007) Continuous clustering of moving objects. IEEE Trans Knowl Data Eng 19(9):1161–74
17. Lee J-G, Han J, Whang K-Y (2007) Trajectory clustering: a partition-and-group framework. In: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, ACM, pp 593–604
18. Li Y, Han J, Yang J (2004) Clustering moving objects. In: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, pp 617–622

19. Ma S, Zheng Y, Wolfson O (2013) T-share: a large-scale dynamic taxi ridesharing service. In: 2013 IEEE 29th International Conference on Data Engineering (ICDE), IEEE, pp 410–421
20. Mahin MT, Hashem T (2019) Activity-aware ridesharing group trip planning queries for flexible POIs. ACM Trans Spatial Algorithms Syst 5(3):1–41
21. Pant P, Harrison RM (2013) Estimation of the contribution of road traffic emissions to particulate matter concentrations from field measurements: a review. Atmosp Environ 77:78–97
22. Pattnaik SB, Mohan S, Tom VM (1998) Urban bus transit route network design using genetic algorithm. J Transp Eng 124(4):368–375
23. Pillac V, Gendreau M, Guéret C, Medaglia AL (2013) A review of dynamic vehicle routing problems. Eur J Oper Res 225(1):1–11
24. Qian X, Zhang W, Ukkusuri SV, Yang C (2017) Optimal assignment and incentive design in the taxi group ride problem. Transp Res Part B Methodol 103:208–226
25. Santi P, Resta G, Szell M, Sobolevsky S, Strogatz SH, Ratti C (2014) Quantifying the benefits of vehicle pooling with shareability networks. Proc Natl Acad Sci 111(37):13290–13294
26. Savelsbergh MWP (1985) Local search in routing problems with time windows. Ann Oper Res 4(1):285–305
27. Shen B, Huang Y, Zhao Y (2016) Dynamic ridesharing. SIGSPATIAL Spec 7(3):3–10
28. Shen B, Zhao Y, Li G, Zheng W, Qin Y, Yuan B, Rao Y (2017) V-tree: efficient knn search on moving objects with road-network constraints. In: 2017 IEEE 33rd International Conference on Data Engineering (ICDE), IEEE, pp 609–620
29. Shrivastava P, O'Mahony M (2007) Design of feeder route network using combined genetic algorithm and specialized repair heuristic. J Publ Transp 10(2):109–133
30. Stiglic M, Agatz N, Savelsbergh M, Gradisar M (2015) The benefits of meeting points in ride-sharing systems. Transp Res Part B Methodol 82:36–53
31. Ta N, Li G, Zhao T, Feng J, Ma H, Gong Z (2018) An efficient ride-sharing framework for maximizing shared route. IEEE Trans Knowl Data Eng 30(2):219–233
32. Tang L, Duan Z, Zhu Y, Ma J, Liu Z (2019) Recommendation for ridesharing groups through destination prediction on trajectory data. IEEE Trans Intell Transp Syst:1–14
33. Vazifeh MM, Santi P, Resta G, Strogatz SH, Ratti C (2018) Addressing the minimum fleet problem in on-demand urban mobility. Nature 557(7706):534–538
34. Wang Y, Kutadinata R, Winter S (2019) The evolutionary interaction between taxi-sharing behaviours and social networks. Transp Res Part A Policy Pract 119:170–180
35. Zhao F, Ubaka I (2004) Transit network optimization-minimizing transfers and optimizing route directness. J Publ Transp 7(1):4
36. Zhu M, Liu X-Y, Tang F, Qiu M, Shen R, Shu W, Min-You W (2016) Public vehicles for future urban transportation. IEEE Trans Intell Transp Syst 17(12):3344–3353
37. Zhu M, Liu X-Y, Wang X (2019) An online ride-sharing path-planning strategy for public vehicle systems. IEEE Trans Intell Transp Syst 20(2):616–627