A Distributed Algorithm for Operating Large-Scale Ridesourcing Systems

Ruolin Zhang[1], Neda Masoud[1]
[1] Civil and Environmental Engineering, University of Michigan, Ann Arbor
Corresponding Author – Email: nmasoud@umich.edu

Abstract

With ridesourcing services gaining popularity in the past few years, there has been growing interest in algorithms that could enable real-time operation of these systems. As ridesourcing systems rely on independent entities to build the supply and demand sides of the market, they have been shown to operate more successfully in metropolitan areas where there is a high level of demand for rides as well as a high number of drivers, and a large volume of trips occurring within a geographically constrained region. Despite the suitable ecosystem that metropolitan areas offer for ridesourcing operations, there is a lack of methods that can provide high-quality matching solutions in real-time. To fill this gap, this paper introduces a framework that allows for solving the large-scale matching problems by means of solving smaller problems in a distributed fashion. The proposed methodology is based on constructing approximately-uniform clusters of trip requests, where vehicle tours form cluster centers. Using the New York Taxi dataset, we compare the performance of the proposed methodology against three benchmark methods to showcase its advantages in terms of solution quality and solution time.

1. Introduction

In recent years, population and economic growth have led to the formation of traffic jams in metropolitan areas, with direct influence on pollution of exhaust emissions and increasing travel time and cost. Single-occupancy vehicles are a major source for generating carbon dioxide emissions (Hensher 2008)–a problem that is exacerbated due to congestion. However, scaling-up the infrastructure to meet the growing demand is constrained and costly. Therefore, seeking solutions to increase the utilization rate of the existing transportation infrastructure has been the focus of extensive research in the past decade.

A number of alternative modes of transportation have been introduced to expand the utilization rate of the existing transportation infrastructure. Public transportation is a traditional means to reduce the number of single-occupancy vehicles. Public transportation systems, such as buses and rail systems, are generally regulated on a fixed schedule and operated on established routes, and charge a posted fee for each trip. Although having a fixed schedule and route could lead to offering more reliable services, the limited operational flexibility of public transportation services leads to more constrained coverage, both spatially and temporally. This has led to a growing interest in shared mobility options, which introduce more flexibility and comfort compared to fixed public transportation options, but offer discounted prices compared to taxis and other private modes of transportation.

Technological advancements such as GPS-enabled smart personal devices, online payment systems and big data together with a global quest for environmentally-friendly and cost-efficient mobility options have led to the emergence of a significant number of internet-based companies around the globe that offer ride-sharing and ridesourcing services to satisfy on-demand requests. Examples ((Chan and Shaheen 2012)) include Flinc (Flinc 2011 (accessed Dec. 15, 2020)), Ville Fluide (Fluide 2011 (accessed Dec. 15, 2020)), Carticipate (Carticipate 2008 (accessed Dec. 15, 2020)), Uber (Uber 2009 (accessed Dec. 15, 2020)), and Lyft (Lyft 2012 (accessed Dec. 15, 2020)). Benefits of ride-sharing consist of saving travel cost and possibly travel time for

drivers and riders, alleviating traffic congestion, conserving fuel, and mitigating air pollution. According to the National Household Travel Survey (NHTS), which is the authoritative source reporting on the travel behavior of the American public, the average light vehicle occupancy (the number of travelers per vehicle trip) is relatively low–1.67 in 2017, unchanged from 2009. Therefore, ride-sharing services have great potential for development. As such, devising algorithmic tools for real-time matching of drivers and riders in a ride-sharing system, or ridesourcing with pooling, also known as the ride-matching problem, is an important and timely topic of research.

This paper introduces a methodology for efficiently solving the one-to-many ride-matching problem, in which a driver can carry multiple passengers at once or in sequence. More specifically, we introduce a clustering method to decompose the problem into multiple sub-problems such that sub-problems can be optimized independently of each other and in parallel. The proposed method guarantees that the sizes of the sub-problems remain approximately uniform, since the computational complexity of the ride-matching problem grows exponentially with the size of the problem. We use the New York City Taxi dataset (NYC.gov (accessed Dec. 15, 2020)) to perform numerical experiments. To evaluate the performance of our proposed methodology, we compare the results with the optimal solution as well as three partitioning methods from the literature, namely point-based, balanced point-based, and trip-based partitioning. We also conduct sensitivity analysis to test the impact of the degree of uniformity in the size of sub-problems and the number of clusters on the computation time and the objective function.

2. Literature Review

In this section we will review the relevant studies from the graph partitioning and ridesharing literature.

2.1. Graph Partitioning

When modeling problems in different application domains, researchers often use graphs as abstractions (Buluc et al. 2016). Splitting a graph into smaller sub-graphs is one of the basic algorithmic operations that allows for solving a large-scale problem by means of solving smaller problems that correspond to sub-graphs, in a distributed fashion. In the past decade, graph partitioning has gained increasingly higher popularity due to the emergence of larger problem instances in various application domains. Applications of graph partitioning in practice can be found in parallel processing (Chu and Cheng 2011; Boman, Devine, and Rajamanickam 2013), complex networks (Fortunato 2010), transportation networks (Luxen and Schieferdecker 2015; Kieritz et al. 2010), and image processing (Grady and Schwartz 2006; Peng, Zhang, and Zhang 2013), among others.

A graph can be represented by a set of vertices and weighted edges. A graph partitioning problem seeks to partition vertices and/or edges into different sub-graphs. In a number of application domains, balancing constraints are also imposed during graph portioning to ensure that all clusters have (approximately) equal weights, number of edges, or number of vertices. An imbalance parameter ε can be used to impose the balancing constraint.

Graph partitioning can be formulated to optimize different objective functions, which reflect different objectives of graph partitioning for different application domains. The most prominent objective function is to minimize the total cut, typically quantified by the total weight of removed edges from the original graph. It has been shown that the problem of dividing a graph into k clusters with approximately equal size to minimize an objective function is NP-complete (Grady and Schwartz 2006). (Andreev and Racke 2006) shows that on a general graph, a perfectly balanced partitioning ($\varepsilon = 0$) has no constant-factor approximation. If $\varepsilon \in (0,1]$, a $O(\log^2 n)$ factor approximation can be achieved.

There are a large number of methods to solve the graph partitioning problem. They can be divided into two groups, namely, global algorithms, and local algorithms (Buluc et al. 2016). Global algorithms are methods that apply to the entire graph and directly obtain the solution. This family of algorithms could include exact methods (Delling and Werneck 2012; Masoud and Jayakrishnan 2017b; Karisch, Rendl, and Clausen 2000; Regue, Masoud, and Recker 2016; Masoud et al. 2017; Lloret-Batlle, Masoud, and Nam 2017) and heuristic algorithms

(Kernighan and Lin 1970; Nam et al. 2018; Zhang, Tafreshian, and Masoud 2020; Masoud and Jayakrishnan 2017c). These algorithms are usually used for smaller graphs. Many of these methods are limited to bi-partitioning but can also be applied to k-clustering by recursion (Buluc et al. 2016). Local algorithms, on the other hand, are based on a starting solution, where this starting solution is iteratively improved. Examples of this family of algorithms include local search (Kernighan and Lin 1970; Fiduccia and Mattheyses 1982) and flow-based improvement (Sanders and Schulz 2013; Akhremtsev, Sanders, and Schulz 2017).

Many applications in the transportation domain can be modeled using graphs, rendering graph partitioning an important tool, especially due to the higher penetration rate of shared mobility as well as emergence of connectivity that lead to higher complexity of transportation problems. In this paper, we propose a methodology that utilizes approximately-uniform graph partitioning/clustering for the real-time ride-matching problem. We formulate the problem as a clustering problem that assigns trips to clusters such that a total cost is minimized, where we use vehicle tours as cluster representatives, and impose uniformity constraints on clusters.

2.2. Dynamic Ride-sharing

According to the 2018 Global Traffic Scorecard, Americans lost an average of 97 hours a year due to traffic congestion, costing nearly $87 billion in 2018, an average of $1,348 per driver. Increasing the utilization rate of vehicles can be an effective way to reduce vehicle-miles-traveled (VMT) and improve traffic congestion.

Ride-sharing has garnered plenty of attention in recent years due to its effectiveness in utilizing empty car seats (Furuhata et al. 2013). A ride-sharing system aims to bring together participants with compatible routes and time schedules to share a vehicle. Here, we focus on a dynamic ride-sharing system, where requests to participate in the system can be made at any point in time. Dynamic ride-sharing concentrates on single, non-recurring trips, differentiating it from conventional carpooling (Baldacci and Mingozzi 2004; Li, Liu, and Zhang 2018), which focuses on recurring trips. In such a system, participants can request a trip as riders or provide rides as drivers. Participants input their requests, including the origins and destinations of their trips and their trip timelines, and the operator of the system makes arrangements to match drivers with riders on a short notice or even en-route. In a dynamic ride-sharing system, typically a central ride-matching problem is solved periodically.

Ride-matching problems can have different objectives, such as minimizing system-wide VMT (Tafreshian et al. 2021), maximizing the operator's profit (Jafari et al. 2016), and maximizing the number of matched participants (Masoud and Jayakrishnan 2017a), among others. When matching a driver with a rider, several constraints must be considered. Many studies let a rider or driver provide their earliest departure and latest arrival times, constructing a time window that constrains the matches (Agatz et al. 2012). In addition to travel time, a number of other constraints may be imposed to satisfy a participant's needs and preferences (Ghoseiri 2012).

A ride-sharing system can adopt one of several possible strategies when matching riders with drivers: (1) a single rider matched with a single driver, (2) a single driver matched to multiple riders (i.e., pooling), (3) a single rider matched with multiple drivers (i.e., multi-hop matching), and (4) multiple rider, multiple driver arrangements (i.e., pooling in a multi-hop system) (Agatz et al. 2012; Furuhata et al. 2013). Typically, ride-matching algorithms are developed assuming that rider and driver roles are fixed. However, a number of studies have considered the more general but complex scenario where a portion of participants are flexible and can take any role that is assigned to them by the system (Agatz et al. 2011; Tafreshian and Masoud 2020b). A review of ride-matching algorithms for different system configurations is shown in Table **[table:1]**. A comprehensive review of ride-matching methods can be found in (Tafreshian, Masoud, and Yin 2020).

The simplest form of ride-matching involves matching a single rider with precisely a single driver, also known as one-to-one matching. (Agatz et al. 2011) formulates the one-to-one ride-matching problem as a maximum weight bipartite matching problem. They use the optimization-based approaches with a rolling horizon strategy to solve the ride-matching problem. (Najmi, Rey, and Rashidi 2017) proposes a clustering heuristic algorithm to solve the one-to-one matching problem. More complicated forms of the ride-matching problems aim to increase the number of riders on board beyond a single rider to take advantage of empty seats

in a vehicle. (Herbawi and Weber 2012) proposes a genetic and insertion heuristic algorithm to solve the ride-marching problem in which a driver can serve more than one rider, also known as the one-to-many ride-matching problem. (Di Febbraro, Gattorna, and Sacco 2013) formulates the one-to-many ride-matching problem as a mixed integer-linear programming problem that can be solved by commercial optimization engines. (Stiglic et al. 2015) models the system as a maximum weight bipartite matching problem, where the number of stops for a rider is limited to two. (Alonso-Mora et al. 2017) presents a scalable mathematical formulation of the one-to-many problem, where within multiple steps vehicles are matched with groups of passengers. Through numerical experiments, they demonstrate that when re-purposed for ridesharing, only a fraction of taxis in the New York Taxi dataset can serve almost the entire demand for rides.

The one-to-many matching problem, also known as the pooling problem, arises in systems similar to ridesharing, such as taxi sharing and pooled variants of ridesourcing. To serve on-demand requests by shared taxis, (Ma, Yu, and Wolfson 2013) first prunes the search space by identifying candidate taxis that can potentially serve a request, and then uses a scheduling algorithm to find a match that imposes the least added distance traveled. (Santos and Xavier 2015) proposes a greedy randomized adaptive search procedure to operate a ridesharing/taxi sharing system with on-demand request, and demonstrates using numerical experiments that pooling could reduce the cost of trips by about 30% compared to private rides. (Zhan et al. 2021) proposes an artificial bee colony algorithm for serving requests in a pooled ridesourcing system, where the objective is to maximize the number of served ride requests while at the same time minimizing travel time and cost ratios. Their numerical experiments demonstrate that pooling in ridesourcing systems can lead to substantial cost savings with minimal increase in travel time.

A number of studies allow a rider to transfer between multiple vehicles to complete his/her trip, giving rise to multi-hop systems. (Herbawi and Weber 2011) proposes an evolutionary multi-objective route planning algorithm to solve a many-to-one ride-matching problem in which riders can transfer between different drivers, but a driver can transport a single passenger. (Drews and Luxen 2013) provides a solution to a ride-matching problem with an arbitrary number of transfers that respect the users' personal preferences using a graph searching algorithm. Many-to-many matching problems are the most complex problems that allow riders to transfer between different drivers and, at the same time, drivers to have more than one riders on board. (Cortés, Matamala, and Contardo 2010) models the many-to-many ride-matching problem and solve relatively small instances of the problem using a branch-and-cut method. (Ghoseiri 2012) proposes a spatial, temporal, and hierarchical decomposition solution strategy to solve the many-to-many ride-matching problem, formulated by a mixed-integer problem. However, the number of transfers is limited. (Masoud and Jayakrishnan 2017a) models the multi-hop ride-matching problem as a binary program and propose a decomposition algorithm to solve this problem. A comprehensive review of ride-matching problems for different system configurations, including ride-sharing, Ridesourcing, taxi-sharing, and demand-responsive services, can be found in (Tafreshian, Masoud, and Yin 2020; Wang and Yang 2019; Chakraborty et al. 2020). For a comprehensive literature review on the dial-a-ride problem, see (Cordeau and Laporte 2003, 2007; Ho et al. 2018).

In major metropolitan areas where thousands of ride requests arrive dynamically, centralized ride-matching methods may fall short in providing high-quality solutions in real-time. Consequently, several attempts have been made in the literature to develop decentralized optimization schemes. One approach to decentralize the decision-making process is adopting agent-based models. (Winter and Nittel 2006) considers the transportation network as a mobile geo-sensor network of agents that interact locally by short-range communication and heuristic way-finding strategies. (Mes, Heijden, and Harten 2007) introduces an agent-based approach where intelligent vehicle agents schedule their routes. (Nourinejad and Roorda 2016) sets up a single-shot first-price Vickrey auction where the virtual driver–passenger agents are matched.

Another approach to decentralized decision making is dividing the problem into multiple smaller sub-problems that can be solved independently of each other. (Pelzer et al. 2015) partitions the road network into distinct sub-networks that define the search space for ride matches. Recently, (Najmi, Rey, and Rashidi 2017) proposed a framework that embeds a network clustering heuristic within an on-demand ride-sharing system. To address participants whose origins and destinations fall in different clusters, they solve an additional matching problem. We refer to this method as "point-based clustering".

Table 1: A review of ride-matching methods. (If the study focuses on dynamic ride-matching, the number listed under the No. of participants column is the number of participants during a one-hour horizon.)

Study	Dynamic	Configuration	No. of participants	Computation time (s)	Solution Methodology
(Agatz et al. 2011)	Yes	single rider, single driver	4,933	78	Max-weight bipartite matching with rolling horizon strategy
(Najmi, Rey, and Rashidi 2017)	Yes	single rider, single driver	15,412	80.21	Clustering heuristic algorithm with rolling horizon strategy
(Herbawi and Weber 2012)	Yes	single driver, multiple riders	744	100	Genetic and insertion heuristic algorithms
(Stiglic et al. 2015)	No	single driver, multiple riders	2,849	150	Bipartite matching formulation with meeting points
(Herbawi and Weber 2011)	Yes	multiple drivers, single rider	250	1.476	Evolutionary multi-objective route planning problem solved by the NSGA-II algorithm
(Masoud and Jayakrishnan 2017a)	No	multiple drivers, multiple riders	400	6	An exact decomposition algorithm
(Nourinejad and Roorda 2016)	No	all but multiple drivers, multiple riders	0-2000		Decentralized (dynamic auction-based multi-agent) optimization algorithms
(Pelzer et al. 2015)	Yes	single rider, single driver	4,500		A partition-based match-making algorithm
(Tafreshian and Masoud 2020a)	Yes	single rider, single driver	22,000	36.32	A graph partitioning methodology
This paper	Yes	single driver, multiple riders	20,000	42.31	Approximately-uniform clustering with tours as cluster representative

In another recent work, (Tafreshian and Masoud 2020a) proposed a trip-based graph partitioning method for dynamic ridesharing systems. In contrast to (Najmi, Rey, and Rashidi 2017), they used complete trips, rather than trip ends, to form clusters, leading to partitions that may be geographically overlapping. Furthermore, they form clusters that are approximately uniform in size to reduce solution time. In Section 7, we use three benchmark methods to evaluate the performance of our proposed methodology: point-based clustering, balanced point-based clustering in which uniformity constraints are imposed when forming clusters in point-based clustering, and trip-based clustering.

2.3. Our Contributions

In this paper, we propose a framework to solve the one-to-many ride-matching problem that arises in ride-sharing and Ridesourcing systems in a distributed fashion. Our proposed method forms approximately-uniform clusters of trips, and assigns drivers to trip clusters proportional to the cluster sizes. Different from the existing literature that uses points or trips as cluster centers during partitioning, we use *vehicle tours* as cluster representatives to capture the fact that ride requests can be pooled and served by a single vehicle even when they do not share the same origin, destination, or time window. This allows for obtaining higher-quality solutions for one-to-many ridesharing systems or ridesourcing systems with pooling. The proposed method decomposes the original problem into smaller sub-problems that are approximately uniform in size within a threshold of ε. The value of ε captures the trade-off between the complexity of solving the sub-problems and the lost performance due to partitioning: While setting $\varepsilon = 0$ ensures all sub-problems are uniform in size and therefore the solution time is minimized, to obtain such uniform partitioning more potential matches are ignored, thereby affecting the solution quality. Finally, we compare the results of our proposed methodology with three benchmark methods, namely point-based, balanced point-based, and trip-based partitioning, as well as the optimal solution.

As such, the contributions of this paper can be summarized as follows. This is the first study to propose an approximately-uniform clustering method with tours as cluster representatives, which as demonstrated in the numerical experiments section, outperforms existing clustering approaches. We develop an iterative procedure based on Lloyd's algorithm (Lloyd 1982) to find approximately-uniform clusters, and show that this clustering approach has favorable properties, such as monotonically converging to a local optimal solution in a finite number of steps.

3. Problem statement

Consider a dynamic ride-sharing system that matches drivers with riders in a region over time. Let us divide the study area into a set of stations $S = \{s_1, s_2, \cdots, s_m\}$, where drivers and riders start or end their trips. Furthermore, we divide the time horizon, e.g., an hour, into a series of short time intervals, e.g., 1 min. After this discretization in time and space, a travel time matrix T can be used to retrieve the shortest path travel time between any pair of stations.

Let us consider a set of drivers D and a set of riders R in this system, and introduce set $P = R \cup D$ to include all participants. A rider $r \in R$ registers her trip, including her origin station $s_r^{\mathcal{O}}$, destination station $s_r^{\mathcal{D}}$, her earliest departure time t_r^{ED}, and her latest arrival time t_r^{LA}. A driver $d \in D$ registers her origin station $s_d^{\mathcal{O}}$ and her earliest departure time t_d^{ED}. After serving a rider, the driver's time and location will be updated to the rider's drop-off time and location, respectively. Without loss of generality, here we assume that drivers are available for the entirety of the study time horizon.

To accommodate the inherent uncertainty present in a dynamic ride-sharing system, we adopt a rolling horizon strategy, in which the system operator solves the ride-matching problem periodically, at evenly-spaced points in time to which we refer as re-optimization times. The time period between two consecutive re-optimization times is a *re-optimization period*.

Table 2: Table of notations

Sets	
$S = \{s\}$	set of stations where drivers and riders start or finish their trips
$D = \{1, \cdots, \mathcal{D}\}$	set of drivers
$R = \{1, \cdots, \mathcal{R}\}$	set of riders/trips
$P = R \cup D$	set of participants p
R^c	set of riders whose trips do not lie on tours
$K = \{1, \cdots, \mathcal{K}\}$	set of clusters
T	set of time intervals t
L	set of links of $l = (t_i, s_i, t_j, s_j) \in T \times S \times T \times S \in L$
L_p	set of links that are accessible by participant p
L_{rd}	set of links on which driver d can serve rider r, where $L_{rd} = L_r \cup L_d$
$\mathcal{L}$	set of links where $\ell(i,j) \in \mathcal{L}$ indicates rider j can be served after rider i
Indices	
r, i, j	indices for rider/trip
d	index for driver/vehicle
p	index for participant (driver or rider)
k	index for cluster/tour
$\ell = (i,j)$	link ℓ exists if rider r_j can be served following rider r_i
Parameters	
$s_p^{\mathcal{O}}$	origin station of participant p
$s_p^{\mathcal{D}}$	destination station of participant p
t_p^{ED}	earliest departure time of participant p
t_p^{LA}	latest arrival time of participant p
cap_d	the capacity of (the vehicle of) driver d (i.e., the maximum number of riders allowed on board at one time)
Functions	
$T(s_i, s_j)$	shortest-path travel time between station i and station j
$c(r,k)$	the primary cost between rider trip r and tour k
$d(r,k)$	the secondary cost between rider trip r and tour k
$\gamma(d,k)$	the cost between driver d and tour k
Decision Vaablesri	
ω_{rd}	binary decision variable that holds the value 1 if rider r is matched with driver d, and the value 0 otherwise
x_d^l	binary decision variable that holds value 1 if driver d travels on link l, and value 0 otherwise
y_{rd}^l	binary decision variable that holds value 1 if rider r is served by driver d on link l, and value 0 otherwise
v_{ij}	binary decision variable that holds the value 1 if link ℓ_{ij} is selected as a part of the tour u, and value 0 otherwise
f_{rk}	a binary variable that holds the value 1 if trip r is assigned to cluster k, and value 0 otherwise
z_{dk}	binary decision variable that holds the value 1 if driver d is assigned to cluster k, and the value 0 otherwise

At each re-optimization time n, we formulate a ride-matching problem that consists of all announced trips that have not yet expired or finalized. An announced trip is considered expired if its latest departure time, i.e., $t_r^{LA} - T(s_r^O, s_r^D)$, occurs before the end of the current re-optimization period. In Fig. **1**, for example, all trips whose announcement times are before re-optimization time n and latest departure times are after re-optimization time $n+1$ will be considered in the ride-matching problem in the specified re-optimization period. (Note that we require the latest departure time of a trip to be higher than $n+1$ to account for solution time.) An announced trip is considered finalized if it has been previously matched in the ride-matching problem. Drivers who are matched previously can always be part of the new ride-matching problem after accounting for their previous assignments, i.e., the new origin station and earliest departure time of a driver who is transporting a passenger will be set to the drop off location and time of their onboard passenger, respectively. The objective of the ride-matching problem is to maximize the number of served riders. Since the dynamic ridesharing system solves ride-matching problems that are structurally similar across re-optimization periods, in rest of this paper we focus our discussion on the optimization problem in a single re-optimization period.

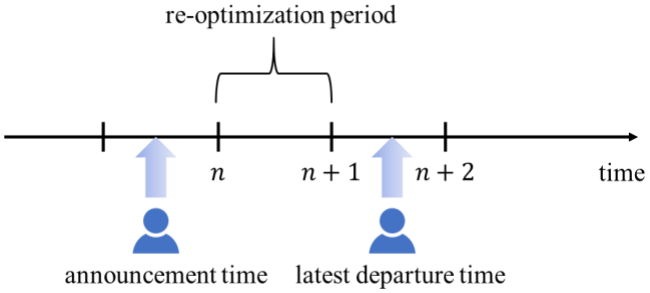


Figure 1: The rolling horizon implementation

4. The Ride-matching Problem

In this paper, we consider a one-to-many ride-matching problem in which a driver can serve multiple riders. This ride-matching problem may arise in ride-sharing or ridesourcing systems with pooling and can be formulated as in model (A.1) in Appendix I. This optimization problem is an NP-hard problem, and cannot be readily adopted in a dynamic system. Hence, in this paper we introduce a solution methodology that decomposes the original matching problem into multiple smaller sup-problems that will be solved independently of each other, using model (A.1).

5. Illustrative Example

We use a small instance of a ridesourcing problem to clearly demonstrate each step of our proposed method for a single static optimization problem. This example includes 4 drivers and 20 riders. We use data form the New York City Taxi dataset (NYC.gov (accessed Dec. 15, 2020)) to construct this example. Due to the small size of the problem, we only consider a single optimization period. Table 8 in Appendix II provides information on the 24 participants in this example, including their roles (rider/driver), their origin and destination stations, and their earliest departure and latest arrival times. Tables **9** and 10 in Appendix III provide shortest path travel times between all pick-up and drop-off stations in this example. We solved the ride-matching problem in model (A.1) for this small example using the CPLEX solver in AMPL. The optimal matching results are demonstrated in Table **3**. In the rest of the paper, we apply the algorithms in each section to this example, and eventually compare the outcome of the proposed methodology with that of the optimal solution.

Table 3: Optimal solution of the illustrative example. The itinerary of each driver is indicated as a sequence of tuples. A tuple (t, s) indicates that the driver visits station t at time interval s.

Driver	Served riders	Vehicle itinerary
d_1	r_1, r_6, r_9, r_{20}	(19:00,162)→(19:15,103)→(19:22,97)→(19:23,97)→(19:30,56)→(19:47,50)→(19:54, 121)→(19:59,125)→(20:06,128)
d_2	r_4, r_{11}, r_{18}	(19:12,95)→(19:25,52)→(19:35,124)→(19:36,124)→(19:43,76)→(19:54,58)→(19:56, 58)→(20:04,159)
d_3	r_2, r_7, r_{14}	(19:00,161)→(19:13,80)→(19:25,33)→ (19:33,60)→(19:45,78)→(19:47,78)→(19:58,161)
d_4	r_3, r_{10}, r_{16}	(19:07,80)→(19:16,129)→(19:32,98)→(19:34,98)→(19:40,97)→ (19:50,98)→(19:52,98)→ (20:00,130)

6. Methodology

The matching problem in (A.1) can be solved quickly for the instance presented in section 5 using commercial solvers, because of the problem's relatively small size. However, solving the optimization problem in model (A.1) for larger instances of the ride-matching problem can be computationally prohibitive for real-time implementations. As such, in this paper, we propose a framework to produce high-quality solutions in near real-time, depicted in Figure 2.

This framework includes a clustering algorithm, which we call ϵ-uniform tour-based clustering. This clustering algorithm includes a tour-forming problem, in which a tour, i.e., sequence of trips, is formed in each cluster to represent the trips in the cluster, and an assignment step, in which trips are assigned to clusters based on their proximity to the tours representing the clusters and subject to a uniformity constraint. By iteratively solving these two problems, the clustering algorithm groups trips into multiple clusters that can be optimized independently of each other. After the clusters of trips are formed, in section 6.2 we present an optimization-based algorithm to assign drivers to clusters of trips. Finally, the matching problem presented in Appendix A can be solved independently for each cluster.

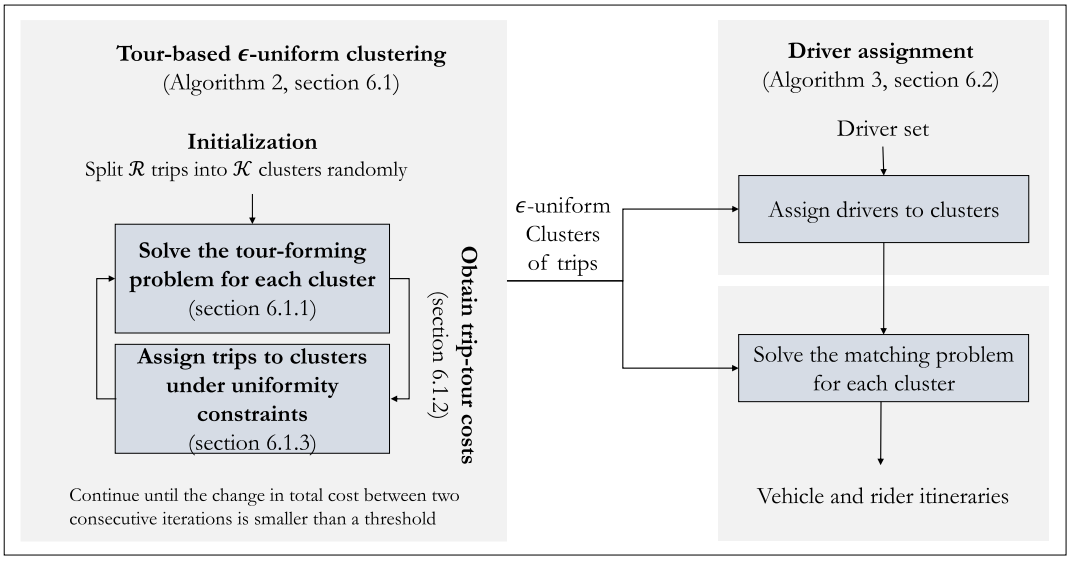


Figure 2: The general flow of the proposed framework.

6.1. ε-uniform tour-based Clustering

The proposed ε-uniform tour-based clustering algorithm iterates between the following two steps until convergence: (1) partitioning trip requests into approximately-uniform clusters so as to minimize the intra-cluster distances, and (2) finding the best representative for each cluster. In sections 6.1.1, 6.1.2, and 6.1.3, we describe different components of the proposed clustering approach and demonstrate each component using the illustrative example presented in Section 5. In section 6.1.4, we combine the components to present the clustering method. Section 6.1.5 describes the properties of the clustering approach.

6.1.1 The Tour Forming Problem

This problem can be represented by a graph $G = (R, \mathcal{L})$, where R is the set of ride requests and $\mathcal{L}$ is the link set. A link $\ell_{ij} \in \mathcal{L}$ between riders i and j exists in graph G if rider j can always be served following rider i. This condition can be mathematically expressed in inequality (1). This inequality ensures that a driver who drops off rider i at her latest arrival time still has enough time to transport rider j to her destination within this rider's requested time window. Furthermore, we introduce two nodes, $\mathcal{O}$ and $\mathcal{D}$, such that there is an outgoing link from node $\mathcal{O}$ to all other nodes and an incoming link from all nodes to node $\mathcal{D}$.

$$t_i^{LA} + T\left(s_i^{\mathcal{D}}, s_j^{\mathcal{O}}\right) \leq t_j^{LA} - T\left(s_j^{\mathcal{O}}, s_j^{\mathcal{D}}\right) \tag{1}$$

Under this setting, we seek to find the longest tour, i.e., the tour that contains the greatest number of trips. This tour-finding problem can be formulated as a longest path problem, as presented in model (1). The decision variable v_{ij} is a binary variable that takes the value 1 if link ℓ_{ij} is selected as a part of the tour, and the value 0 otherwise. The objective function in (2a) maximizes the number of trips that lie on the selected tour. Constraints (2b) and (2c) ensure that the tour begins at $\mathcal{O}$ and ends at $\mathcal{D}$, respectively. Constraint (2d) is the flow balance constraint.

$$\max \quad w = \sum_{i,j \in R} v_{ij} \tag{2a}$$

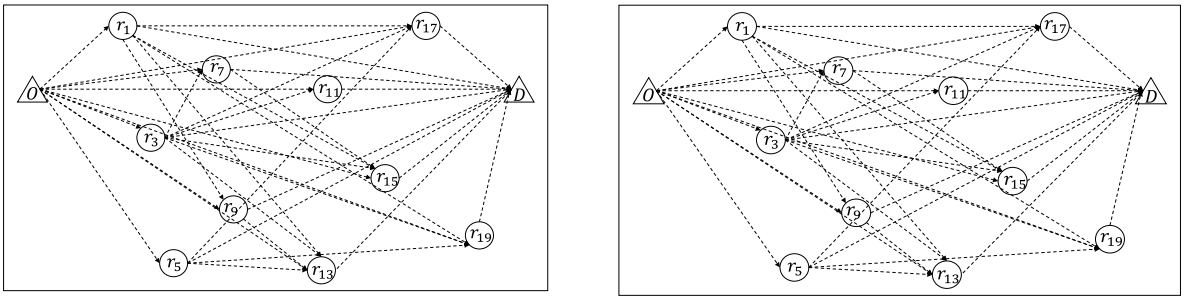
$$\text{s.t.} \quad \sum_{j \in R} v_{\mathcal{O}j} = 1 \tag{2b}$$

$$\sum_{i \in R} v_{i\mathcal{D}} = 1 \tag{2c}$$

$$\sum_{i \in R} v_{ir} - \sum_{i \in R} v_{ri} = 0 \qquad \forall\ r \in R \tag{2d}$$

$$v_{ij} \in \{0,1\} \qquad \forall\ i,j \in R \tag{2e}$$

To clearly demonstrate the tour-forming problem, we apply it to the illustrative example in section **5**. We randomly split the 20 rider trips in this problem into two clusters of size 10, and solve the tour-forming problem on the graph $\mathcal{G} = (R, \mathcal{L})$ for each cluster, as demonstrated in Figure 3. Each graph has 12 nodes, including 10 nodes associated with riders, node $\mathcal{O}$, and node $\mathcal{D}$. The dashed arrows in this figure represent the link set $\mathcal{L}$, obtained based on inequality (1). The longest path problem in model (1) can be solved efficiently using polynomial time algorithms, such as the Network Simplex algorithm. After solving the longest path problem, we obtain the following tours (i.e., sequences of stations): $[80 \rightarrow 129 \rightarrow 33 \rightarrow 60 \rightarrow 124 \rightarrow 156]$ for cluster 1, and $[161 \rightarrow 80 \rightarrow 98 \rightarrow 97 \rightarrow 78 \rightarrow 161]$ for cluster 2. These tours are demonstrated in Figure 3.



(a) Optimal tour in graph $\mathcal{G}$ for cluster 1

(b) Optimal tour in graph $\mathcal{G}$ for cluster 2

Figure 3: Tours associated with the two clusters in the illustrative example

6.1.2 The Cost Function

Once tours are formed to represent each cluster, we need to define a measure of distance between a trip and a tour. This measure will be later used to assign trips to tours so as to minimize the sum of the intra-cluster distances under uniformity constraints. Let us define the cost function c, whose value $c(r,k)$ denotes the distance between rider r's trip and tour k, as presented in Equation (3). We denote this cost as the *primary cost*, and define it as follows. If tour k contains trip r, then the value of the cost function is zero; otherwise, if trip r is not on tour k, then the value of cost function is set to M–a large positive number. Note that this cost function is selected to guarantee the convergence of the clustering algorithm, as will be discussed later in section 6.1.5.

$$c(r,k) = \begin{cases} 0 & \text{if trip } r \text{ is on the tour } k \\ M & \text{if trip } r \text{ is not on the tour } k \end{cases} \tag{3}$$

When we assign trips to tours based on this primary cost, there might be many trips that do not readily lie on any tour. In this case, a tie-breaking rule is needed. Here, we use a secondary objective function to break the ties, to which we refer as the *secondary cost*. Since tours and trips are both sequences of stations, we use an algorithm inspired by dynamic time warping (DTW) to measure the cost of assigning a trip to a tour. DTW is a method developed for measuring the similarity between two sequences by finding an optimal match between their elements (Senin 2008). Consider a tour k that has a sequence of m stations $\mathcal{K}_k = [s_k(1), s_k(2), \ldots, s_k(m)]$, i.e., $m/2$ sequential trips. Let us represent a trip r as a sequence of stations, denoted by $\mathcal{R}_r = \left[s_r(1) = s_r^{\mathcal{O}}, s_r(2) = s_r^{\mathcal{D}}\right]$. We measure the distance between each pair of stations i and j by the shortest-path travel time between them, denoted by $T(i,j)$.

The distance between a trip and a tour can be calculated as the smallest total distance between the origin and destination of the trip (i.e., $s_r(1)$ and $s_r(2)$) from any station on the tour, under the condition that the station on the tour that is matched to the trip destination should appear after the station matched to the trip origin. Mathematically, this condition can be specified as $q+1 \le s_r(2) \le m$ when $s_r(1) = q$. (Note that here we use equality to indicate the matching of two stations.) It is easy to see that one can enumerate all the possible matchings between the two sequences $[s_r(1), s_r(2)]$ and $[s_k(1), s_k(2), \ldots, s_k(m)]$ to find the matching that provides the smallest distance. Algorithm 1 lays out the details of measuring this distance without having to enumerate all the possibilities, rendering this step more computationally efficient. This algorithm starts by defining a $2 \times m$ matrix D, corresponding to the size of the two sequences. The first row of this matrix is the shortest path travel time between the trip origin and the stations on the tour. In the second row of this matrix, all distances are set to infinity, except for the distance between the trip destination and the last station on the tour. The distances between other stations and the trip destination are calculated recursively using Equation (5). Finally, Equation (6) evaluates the total distances between the trip ends and their best matched stations on the tour, and finds the smallest of these distances as the secondary cost.

Algorithm 1: Obtaining the dissimilarity between a trip and a tour

Input: A trip $[s_r(1), s_r(2)]$
A tour $[s_k(1), s_k(2), \ldots, s_k(m)]$
Shortest-path travel time matrix T_t

Output: The distance $d(r,k)$ between trip r and tour k

Step 1: Assume an initial distance matrix D of size $2 \times m$ as follows

$$D = \begin{bmatrix} T\big(s_r(1), s_k(1)\big) & \cdots & T\big(s_r(1), s_k(m-1)\big) & T\big(s_r(1), s_k(m)\big) \\ \infty & \cdots & \infty & T\big(s_r(2), s_k(m)\big) \end{bmatrix} \quad (4)$$

Step 2 Update the distance matrix D

for $q = m-1, \cdots, 1$ **do**

$$D[2,q] = \min\{\, T\big(s_r(2), s_k(q)\big)\,,\; D[2,q+1] \,\} \quad (5)$$

Step 3 Calculate the distance between trip r and tour k

$$d(r,k) = \min_q\{\, D[1,q] + D[2,q] \,\} \quad (6)$$

Using these two cost functions, we calculate the primary and secondary costs, $c(r,k)$ and $d(r,k)$, between trips and tours. In our illustrative example, the primary costs between trips $\mathcal{R}_3$, $\mathcal{R}_7$, and $\mathcal{R}_{15}$ and tour $\mathcal{K}_1$, and between trips $\mathcal{R}_2$, $\mathcal{R}_{10}$, and $\mathcal{R}_{14}$ and tour $\mathcal{K}_2$, are zero. The primary costs between other trips and the two tours are M. The primary and secondary costs of all trips in the illustrative example are displayed in Table **2** in Appendix D.

6.1.3 The Two-Step Trip Assignment Process

In this section, we discuss assignment of trips to clusters under a uniformity constraint. The output of this step is a revised set of clusters. In a conventional clustering problem, objects are allocated to clusters so as to minimize sum of intra-cluster costs. Here, our end goal is not to cluster trips, but to solve the optimization problem in model (A.1) for each cluster in near real-time. Since the computational complexity of the optimization problem in each cluster depends on the number of trips in that cluster, clustering instances in which cluster sizes are highly non-uniform are not of interest, since larger clusters would create a computational bottleneck. As such, we strive to form clusters that are approximately uniform in their number of trips.

In this paper, we consider a two-step assignment process. The first assignment problem seeks to allocate trips to clusters so as to minimize the total primary cost, as indicated in the objective function (7a). Here, the decision variable f_{rk} takes value 1 if trip r is assigned to cluster k, and value 0 otherwise. Constraint (7b) ensures that each trip is allocated to a single cluster. The assignment problem in model (7) has a trivial solution wherein trips that lie on a tours, i.e., trips for which $c(r,k) = 0$, would be allocated to the cluster represented by that tour.

$$\min \quad z_p = \sum_{r \in R} \sum_{k \in K} c\,(r,k)\; f_{rk} \quad (7a)$$

$$\text{s.t.} \quad \sum_{k \in K} f_{rk} = 1 \qquad \forall r \in R \quad (7b)$$

$$f_{rk} \in \{0,1\} \qquad \forall r \in R,\; \forall k \in K \quad (7c)$$

After this initial assignment, we solve a second assignment problem to allocate trips that do not readily lie on a tour. This allocation problem can be formulated as an ε-uniform assignment problem in model (8). Let us

define the set R^c to include ride requests that do not readily lie on a tour, i.e., trips for which $c(r,k) = M$. The objective function in (8a) minimizes sum of the secondary costs between trips and their associated cluster representatives. Constraint (8b) ensures that each trip is assigned to a single cluster. constraint (8c) ensures that the difference between the number of trips in set R^c in any two clusters is at most $\varepsilon|R^c|$. The parameter ε is an imbalance parameter, whose value affects the size clusters. $\varepsilon = 0$ ensures that all clusters have the exact same size, while $\varepsilon = \mathcal{K} - 1$ imposes no constraint on cluster sizes. Note that the ε-uniform assignment problem was first proposed in (Tafreshian and Masoud 2020a) for a peer-to-peer ridesharing system. Model (8) is based on the work in (Tafreshian and Masoud 2020a), customized for a ridesourcing system with one-to-many matching.

$$\min \quad z_s = \sum_{r \in R^c} \sum_{k \in K} d\,(r,k)\; f_{rk} \tag{8a}$$

$$\text{s.t.} \quad \sum_{k \in K} f_{rk} = 1 \qquad \forall r \in R^c \tag{8b}$$

$$\sum_{r \in R^c} f_{rk} \leq \frac{|R^c|}{\mathcal{K}}(1+\varepsilon) \qquad \forall k \in K \tag{8c}$$

$$f_{rk} \in \{0,1\} \qquad \forall r \in R^c,\ \forall k \in K \tag{8d}$$

Figure 5 shows the outcome of this two-step assignment process for the illustrative example. In this figure, trips that lie on tours all have primary cost of zero. Other trips are allocated to tours so as to minimize the secondary cost under the uniformity constraint for cluster sizes.

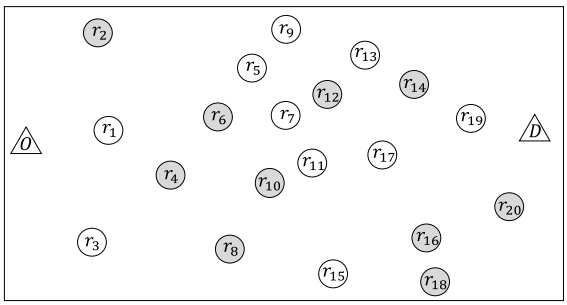


Figure 4: Assignment in iteration 1 of the illustrative example. The trips on tours have primary cost of zero. Other trips are allocated to clusters based on their secondary costs, as outlined in Table 2 in Appendix D.

6.1.4 The ε-uniform tour-based Clustering Algorithm

In the previous sections, we outlined different components of the ε-uniform tour-based clustering algorithm, i.e., the tour-forming problem, the assignment problems, and the cost measures to quantify the distance between a tour and a trip.

The ε-uniform tour-based clustering algorithm is described in Algorithm 2. The inputs to this algorithm are the set of trips, the number of clusters, and the uniformity parameter, and the maximum number of iterations. The algorithm starts (step 0) by randomly assigning $\mathcal{R}$ trips into $\mathcal{K}$ clusters. In step 1, a tour is formed using the set of trips in each cluster. In step 2, distances between trips and tours are obtained, allowing for computing the new total intra-cluster distances. In the assignment step (step 3), first, the assignment solution based on the primary costs is obtained. Next, the ε-uniform assignment problem is solved, where trips that do not readily lie on tours are assigned to clusters based on their proximity to tours as well as the uniformity constraint applied to cluster sizes. The objective function value corresponding to the optimal assignment based on the primary

costs provides the total sum of intra-cluster distances. Step 4 assesses the termination criteria. If the total intra-cluster distance obtained from assignments in two consecutive iterations (obtained in steps 2 and 3) remains the same or the maximum number of iterations is reached, then the algorithm terminates, providing a local optimal solution. Otherwise, the iteration counter α is increased by 1, and steps 1 through 3 are repeated. Since this process provides a local optimal solution, we repeat it for a total of itr times, and report the final set of clusters, and their associated tours, that provide the lowest overall intra-cluster distance.

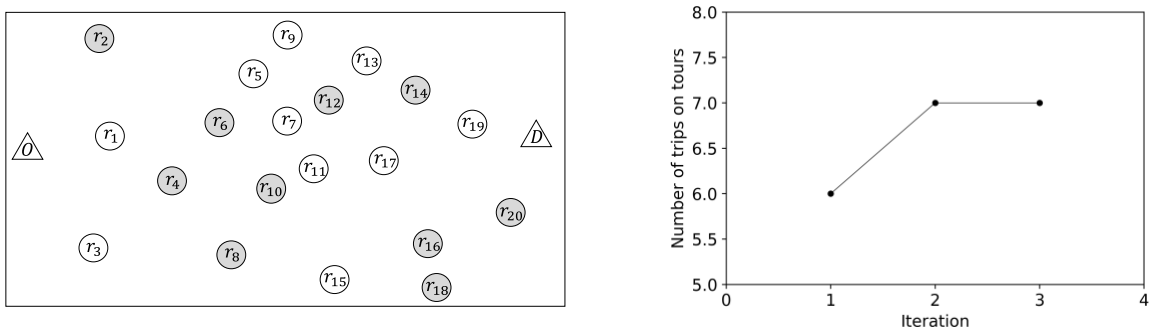
The final clustering results for the illustrative example are shown in Figure 5. The solution consists of two tours, one including three trips and the other one including four trips. As Figure 7 demonstrates, convergence is obtained in only three steps.

Algorithm 2: The ε-uniform tour-based clustering Algorithm

Input: Set of trips, R
Number of clusters, $\mathcal{K}$
Uniformity parameter, ε
Max number of iterations, $\alpha_{\max}$

Output: clusters of trips and their corresponding tours, T^*

for $a = 1, \ldots,$ *itr* **do**
  Step 0: Initialization
  Obtain $f_{rk}^*(0)$ by randomly dividing $\mathcal{R}$ trips into $\mathcal{K}$ clusters
  $\alpha \leftarrow 1$
  Step 1 Tour formation
  Choose a random sample of trips in each cluster (where clusters are determined by $f_{rk}^*(\alpha - 1)$) to form new tours by solving the optimization problem in model (3) for each cluster. Let w^* be the optimal value of the objective function.
  $v_{ij}^*(\alpha) \leftarrow$ argmax w^*
  Step 2 Cost update
  Calculate the new primary and secondary costs, $c(n,k)$ and $d(n,k)$, respectively, using Equation (3) and Algorithm 1, and based on $f_{rk}^*(\alpha - 1)$ and $v_{ij}^*(\alpha)$
  Calculate $h_\alpha = \sum_{k=1}^{\mathcal{K}} \sum_{r=1}^{\mathcal{R}} c\,(r,k)$
  Step 3 Assignment
  Assign $\mathcal{R}$ trips to $\mathcal{K}$ clusters. Set z^* as the number of trips on tours, and find z_s^* by solving the optimization problems in model (8).
  $h_{\alpha+1} \leftarrow z^*$
  $f_{rk}^*(\alpha) \leftarrow$ argmin z_s^*
  Step 4 Termination criteria
  if $h_\alpha = h_{\alpha+1}$ or $\alpha = \alpha_{\max}$ **then**
    $C_a = h_\alpha$
    $T_a = \{f_{rk}^*(\alpha), v_{ij}^*(\alpha)\}$
    Terminate
  else
    $\alpha \leftarrow \alpha + 1$
    Go to **Step 1**
  $T^* \leftarrow T_a$ corresponding to the solution with the minimum C_a



(a) Tours and assignments (b) Convergence of Algorithm 2

Figure 5: Final assignment for the illustrative example

6.1.5 Properties of the ε-uniform tour-based Clustering Algorithm

In this section, we first prove that the objective function of the ε-uniform tour-based clustering problem decreases with iterations in Algorithm 2. Next, we prove the convergence of the algorithm.

Proposition 1. *The objective function in model (7) decreases with iterations of Algorithm 2.*

Proof. In step 1 of Algorithm 2, the optimization problem in model (2) seeks to find the longest tour within each cluster. It is easy to see that finding the tour with maximum length is equivalent to solving an optimization problem that finds the min-cost tour, where the cost function is described in Equation (3). In step 3 of Algorithm 2, the first step of the assignment is completed under the same cost function. Therefore, it is easy to see that each of these two main steps in Algorithm 2 attempt to minimize the same cost function, and the results follow.

Proposition 2. *Algorithm 2 Converges in a finite number of steps.*

Proof. There are possibly many but finite number of ways to assign $\mathcal{R}$ trips to $\mathcal{K}$ clusters. Furthermore, in Proposition **1** we showed that the objective function decreases from one iteration to the next (otherwise, we stop). As such, there are a finite number of ways in which clusters could change, and the algorithm is designed such that no solution is visited twice (unless at convergence, at which point we stop). The results follow.

6.2. Driver Assignment

After clusters of trips are formed, we need to allocate drivers to these clusters. Algorithm 3 details the steps of this procedure. In the first step of this algorithm, the cost of assigning a driver d to a cluster k is calculated. This cost is based on the time distance between the origin location of driver d to the pick-up location of the first trip on tour k that can be served by d, and the number of remaining trips on the tour. In general, the higher the time distance, the higher the cost of allocating the driver to the cluster. Adversely, the higher the number of the remaining trips on tour k, the more effective driver d can be in serving the cluster, and therefore the cost would be lower.

In the second step of Algorithm 3, we solve the bipartite matching problem outlined in model (9) to allocate the set of drivers to clusters. The decision variable z_{dk} takes the value 1 if driver d is assigned to cluster k, and the value 0 otherwise. The objective function in (9) minimizes the total driver-cluster assignment cost. Constraint (9b) ensures that the proportion of drivers allocated to a cluster is approximately equal to the proportion of trips in that cluster. Constraint (9c) ensures that each driver is allocated to exactly one cluster. Constraint (9d) imposes the binary condition on the decision variable z_{dk}. Note that the constraint coefficient matrix in model (9) has a totally unimodular structure, relaxing this optimization problem to a linear program that can be easily solved in real-time. The assignment of drivers to clusters is demonstrated in Table **1**. The optimization problem in model (9) allocates drivers 1 and 3 to cluster 1, and drivers 2 and 4 to cluster 2.

Algorithm 3: Assignment of drivers to clusters

Input: $\mathcal{K}$ tours, T^*

Set of drivers, D, with their origins, $s_d^{\mathcal{O}}$ and earliest departure times, t_d^{ED}

Output: Driver assignment to clusters, $z_{dk}^*, \forall d \in D, k \in K$

$n_k \leftarrow$ number of trips in cluster k under T^*

Step 1 Average cost of driver-tour assignment

for $d \in D$ **do**

  for $k \in K$ **do**

    Find the first trip in tour k that can be served by d based on t_d^{ED}.

    Denote this trip as r. Denote the number of trips after this trip on the tour as n_{rem}

    $\gamma(d,k) = \frac{T(s_d^{\mathcal{O}}, s_r^{\mathcal{O}})}{n_{\text{rem}}+1}$

Step 2 The driver assignment problem

Solve the optimization problem in (9) to obtain the optimal driver-cluster assignment, z_{dk}^*

$$\min \quad \sum_{d\in D}\sum_{k\in K} \gamma(d,k)\ z_{dk} \tag{9a}$$

$$\sum_{d\in D} z_{dk} \geq \lfloor \frac{\sum_{n\in R} f_{rk}^*}{\mathcal{R}}\ \mathcal{D} \rfloor \qquad \forall k \in K \tag{9b}$$

$$\sum_{k\in K} z_{dk} = 1 \qquad \forall d \in D \tag{9c}$$

$$z_{dk} \in \{0,1\} \qquad \forall d \in D,\ \forall k \in K \tag{9d}$$

Table 4: The final ride-matching results

	Driver	Matched with	Number of riders being served
Cluster 1	d_1	r_1, r_6, r_9	3
	d_3	r_2, r_{10}, r_{14}	3
Cluster 2	d_2	r_4, r_{16}	2
	d_4	r_3, r_7, r_{12}, r_{20}	4

7. Numerical Experiments

In this section, we use the New York City Taxi dataset (NYC.gov (accessed Dec. 15, 2020)) to demonstrate the performance of the ε-uniform tour-based clustering method.

7.1. Dataset

The New York City Taxi and Limousine Commission (NYC TLC), in partnership with the NYC Department of Information Technology and Telecommunications (DOITT), has published millions of trip records from yellow medallion taxis and green SHLs for several years. These records include attributes such as pick-up and drop-off dates, times, and locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts.

The data used in this study belongs to the evening peak hour (19:00-20:00) of Feb 19th in 2016. We select trips that are geographically concentrated in the Manhattan area, thereby creating ride-matching problems that are large-scale due to the high number of trips as well as the high spatiotemporal proximity between them.

7.2. Simulation Settings

In this study, we adopt a rolling-horizon approach with a re-optimization period of 1 minute, indicating that the ride-matching optimization problem will be solved every 1 minute starting from 18:59pm. We assume that all trips will be completed on their shortest travel time paths, and obtain the travel times for every re-optimization period from the Google Maps API. In this study, we set the ratio of riders to drivers to 4, i.e., the total number of riders is 4 times the number of drivers. We generate the earliest departure times of drivers uniformly from the window 18:59 to 20:00. We assume that riders request a ride a few minutes ahead of their earliest departure times. Since the dataset does not contain the times when ride requests are issued, we generate a uniform random number from 0 to 30 for each rider, and subtract it from their earliest departure time to obtain their ride-request time. Doing this allows us to have a mixture of pre-arranged and on-the-fly trip requests. We assume that the study area consists of 184 pre-defined stations, denoted by S, where participants start/end their trips. Stations are distributed in the network so as to make sure that there is at least one station within the walking distance (< 0.15 miles) of a typical trip's origin/destination. (Fig. **8**). We set the capacity of each vehicle to four.

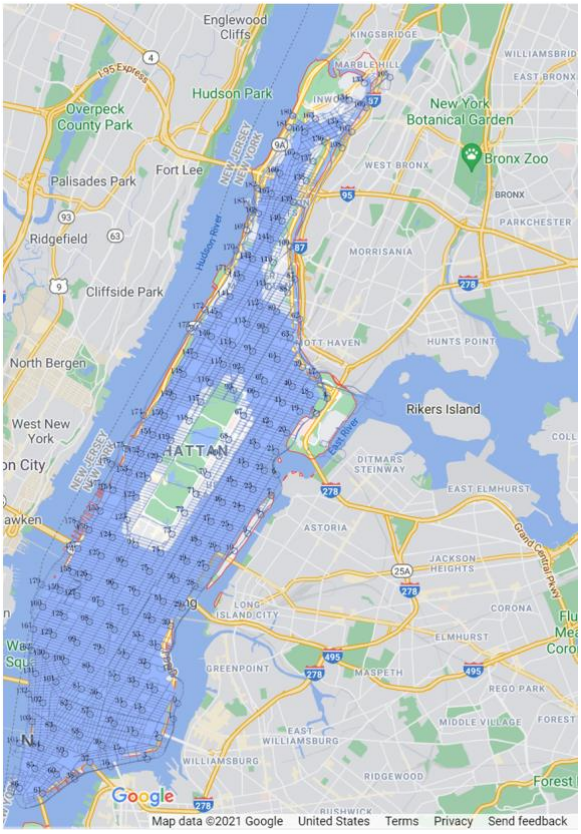


Figure 6: The pre-defined stations distribution of the Manhattan area.

7.3. Results

In this section, we first define a base experimental setting and conduct extensive numerical analysis to draw a comparison between our proposed method, three benchmarks from the literature, and the optimal solution. Next, we observe the convergence properties of Algorithm 2, and study the impact of sample size in this

algorithm. Finally, we conduct sensitivity analysis over some of the critical parameters of our method. For all experiments conducted in this section, we consider a planning horizon of one hour, with sixty one-minute re-optimization periods.

7.3.1 Base Experiment

In our base experimental setting, we set the total number of ride requests to 2000, and the sample size in Algorithm 2 to 150. We set the value of ε to 0.1, and the maximum number of iterations to 10.

Figure 7 demonstrates the clustering results, where for a given trip request, both trip ends are colored based on the clusters to which the trip is assigned. It is interesting to note that the clusters in Figure 7 have a high level of spatial overlap.

(a) Two clusters. (b) Three clusters. (c) Four clusters.

Figure 7: ε-uniform tour-based clustering results. Trip ends are colored based on their assigned clusters.

In order to objectively assess the performance of the proposed method, we compare it against three benchmark methods form the literature, including point-based clustering (Najmi, Rey, and Rashidi 2017), balanced point-based clustering (point-based clustering with uniform clusters), ε-uniform trip-based clustering (Tafreshian and Masoud 2020a), and the exact mathematical formulation presented in model (A.1), solved to optimality. These comparisons allow us to position our proposed method in terms of solution quality and the required computational effort.

In the point-based method, trip ends (i.e., origin and destination locations) of all trips (i.e., both rider and driver trips) are the objects in clustering. Once the clusters are obtained, an optimization problem can be solved independently for each cluster. Trips whose origin and destination locations fall in different clusters will not be served. The balanced point-based clustering is similar to point-based clustering, except that balance constraints are imposed on the number of objects within clusters. In trip-based clustering, each trip (whether it is a rider or a driver trip) is considered an object in clustering. Once clustering is concluded, the rider and driver trips within each cluster are matched independent of other clusters.

Tables 5, 6 and 7 summarize the results. The value of k in these tables indicates the number of clusters. The reported computation time is the total time spent on solving the ride-matching problem, and the time for clustering where applicable, over 60 re-optimization periods, each of duration 1 min. For the ε-uniform tour-based method, the computation time is provided in more detail, breaking the total time into the clustering and optimization times. The clustering time consists of the time required for generating the network, computing the costs, and the assignment steps. The optimization solution time is the sum of the time spent on model construction and obtaining a solution from a commercial solver. The numbers in parenthesis are the average computation times across all re-optimization periods. The optimal solution finds a match for 60.40% of the ride requests. We normalize the optimal matching rate to 100%, and report the *solution quality* of all other methods as their matching rate divided by the optimal matching rate.

Table 5: Comparison of matching rate and solution time between the proposed ϵ-uniform tour-based method, the optimal solution, and the point-based clustering benchmark. The matching rate of the optimal solution is normalized to 100%, and the matching rates obtained by other methods are reported in terms of the percentage of the optimal matching rate. The value of K refers to the number of clusters.

Optimal			Point-based			Tour-based				
							Computation Time (Sec)			
k	Computation Time (Sec)	Solution quality	k	Computation Time (Sec)	Solution quality	k	Total	Graph Partitioning	Optimization Problem	Solution quality
1	1834 (30.56)	100%	2	973 (16.22)	82.62%	2	366 (6.1)	146	220	94.95%
			3	942 (15.70)	67.30%	3	283 (4.72)	163	120	77.15%
			4	933 (13.88)	60.60%	4	197 (3.28)	151	46	65.98%

Table 6: Comparison of matching rate and solution time between the proposed ϵ-uniform tour-based method, the optimal solution, and the balanced point-based clustering benchmark. The matching rate of the optimal solution is normalized to 100%, and the matching rates obtained by other methods are reported in terms of the percentage of the optimal matching rate. The value of K refers to the number of clusters.

Optimal			Balanced Point-based			Tour-based				
							Computation Time (Sec)			
k	Computation Time (Sec)	Solution quality	k	Computation Time (Sec)	Solution quality	k	Total	Graph Partitioning	Optimization Problem	Solution quality
1	1834 (30.56)	100%	2	343 (5.72)	75.32%	2	366 (6.1)	146	220	94.95%
			3	251 (4.18)	62.43%	3	283 (4.72)	163	120	77.15%
			4	173 (2.88)	54.37%	4	197 (3.28)	151	46	65.98%

Table 7: Comparison of matching rate and solution time between the proposed ϵ-uniform tour-based method, the optimal solution, and the ϵ-uniform trip-based clustering benchmark. The matching rate of the optimal solution is normalized to 100%, and the matching rates obtained by other methods are reported in terms of the percentage of the optimal matching rate. The value of K refers to the number of clusters.

Optimal			Trip-based			Tour-based				
							Computation Time (Sec)			
k	Computation Time (Sec)	Solution quality	k	Computation Time (Sec)	Solution quality	k	Total	Graph Partitioning	Optimization Problem	Solution quality
1	1834 (30.56)	100%	2	350 (5.8)	83.61%	2	366 (6.1)	146	220	94.95%
			3	279 (4.65)	68.54%	3	283 (4.72)	163	120	77.15%
			4	185 (3.08)	59.85%	4	197 (3.28)	151	46	65.98%

Table 5 compares the ε-uniform tour-based clustering method with point-based clustering and the optimal solution. This table demonstrates that the gap in solution quality by the two methods decreases as we increase the number of clusters. This is due to the fact that when forming clusters, both methods disregard potential matches between objects that are assigned to different clusters. The higher solution times for the point-based method can be explained by the fact that this method does not seek to construct uniform clusters, implying that some clusters may have a significantly higher number of trips, and thereby higher optimization times. This imbalance in the size of clusters under point-based clustering is demonstrated in Figure 8. The lower solution quality of the point-based method is due to the fact that in this method clusters are constructed based on trip ends, rather than whole trips. Therefore, since after clustering the matching optimization problems are solved independently for each cluster, trips whose trip ends lie in different clusters are not served. This is a drawback that is addressed by the trip-based clustering method in which an entire trip is considered as a clustering object.

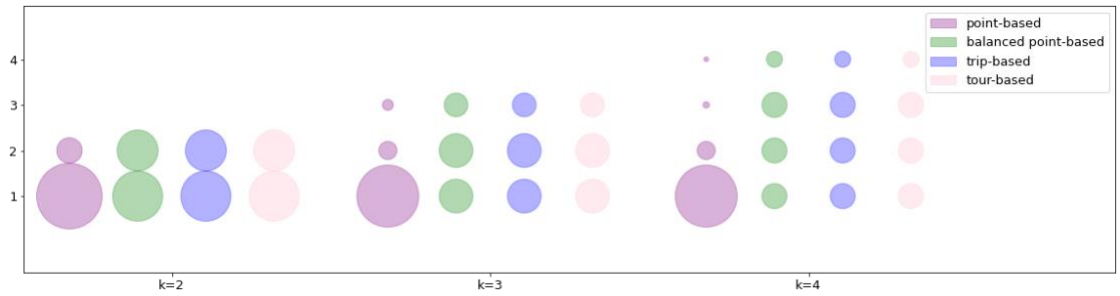


Figure 8: Cluster sizes for point-based, balanced point-based, trip-based and tour-based methods.

Table 6 compares the ε-uniform tour-based clustering method with balanced point-based clustering, in which a uniformity constraint is incorporated in the clustering algorithm, and the optimal solution. This table suggests that when we seek to balance cluster sizes in point-based clustering, the computation time decreases significantly. However, unsurprisingly, the solution quality of the balanced point-based clustering is worse than that of the unbalanced point-based clustering and the tour-based method. This table suggests that with $k = 2$, the solution quality of the tour-based method is higher that of the balanced point-based method by about 20%. This is due to the fact that when forming uniform clusters, the system has a lower level of flexibility to assign trips to clusters, thereby resulting in lower quality solutions.

Finally, Table 7 compares the tour-based and trip-based clustering approaches. The tour-based method outperforms the solution quality of the trip-based method by about 11% under two clusters. The reason behind this improvement in solution quality can be attributed to the fact that in tour-based clustering trips that are close to any trip on the representative tour are assigned to that cluster. As a result, not only are trips that are spatio-temporally close assigned to the same cluster, as is the case in trip-based clustering, but also trips assigned to a cluster can be served sequentially by a single vehicle. This makes it more likely for the matching problem to make a more effective use of the vehicles in serving ride requests.

The gap between these solutions is reduced to 6% under 4 clusters for the same reasons discussed above. The difference in solution times of these two methods is not statistically significant, as both methods strive to generate clusters that are balanced in size within a threshed. The main difference between the balanced point-based, trip-based, and tour-based approaches is what they consider as a unit of analysis: the point-based method considers a trip end as a unit of modeling, while the trip-based method considers an entire trip, and the tour-based method considers a tour–a sequence of trips–as a cluster representative. The tour-based method provides the highest quality solution because the cluster representatives can more closely capture the ultimate product of the matching problem, which in a ridesourcing setting is a set of vehicle tours.

7.3.2 Algorithm Properties

In this section, we first study the convergence rate of our base experimental setting. Next, we investigate the impact of sample size on the quality of solutions.

Convergence Properties

Figure 9 demonstrates the convergence of the objective function in model (8) in our base experiment, using a randomly-selected instance. This figure clearly shows that the objective function increases monotonically, which is equivalent to the cost function decreasing monotonically. As demonstrated in this figure, the objective function typically converges in a few iterations.

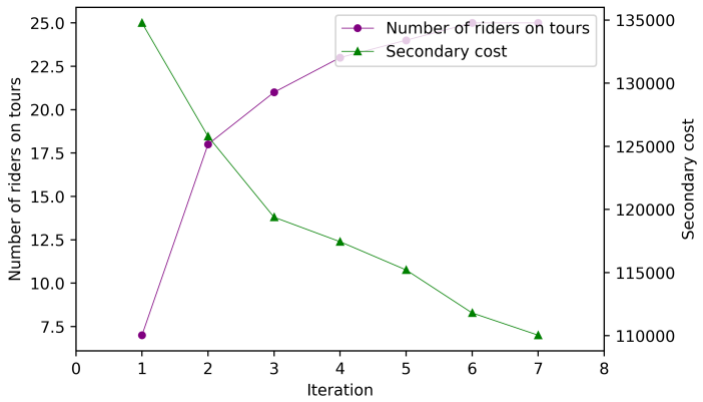
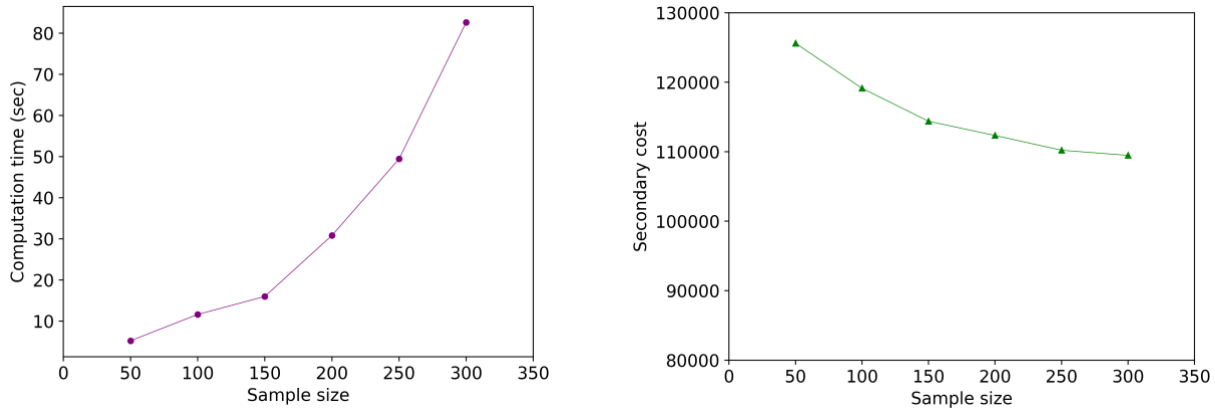


Figure 9: Primary and secondary costs per iteration.

Impact of Sample Size on Solution Quality

Figure 10 shows the impact of sample size in Algorithm 2 on the computation time and the secondary cost in the tour forming problem. Figure 10(a) displays that the computation time increases super-linearly with sample size. Figure 10(b) demonstrates that the secondary cost decreases with sample size. This is due to the fact that increasing the sample size increases the likelihood of obtaining tours that better represent their corresponding clusters, resulting in a smaller secondary cost for the entire set of trips. However, there is a critical sample size beyond which the reduction in cost becomes negligible. In our experimental setting, this sample size is about 150 trips, which is the number utilized in our experiments.



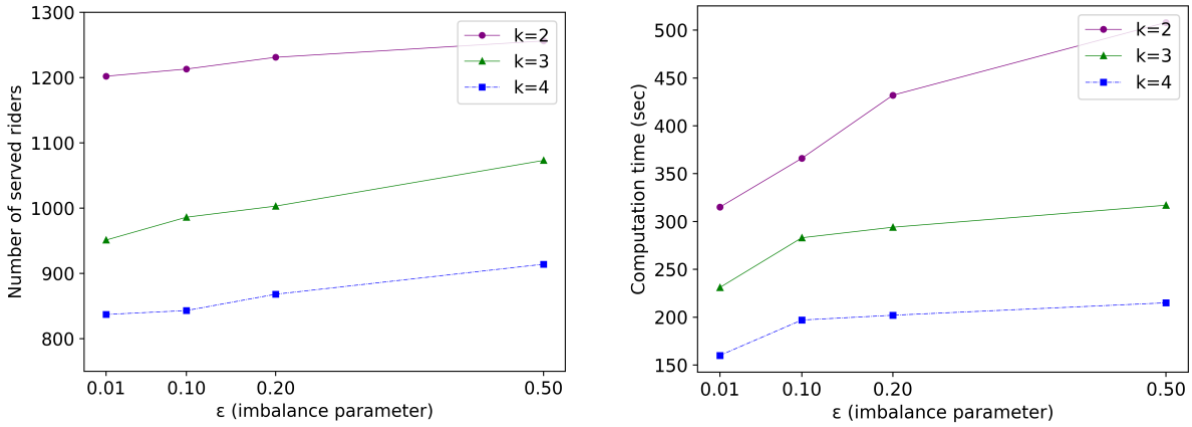
(a) Computation time (b) Total cost

Figure 10: The computation time and the secondary cost of the tour forming problem with different sample sizes.

7.3.3 Sensitivity Analysis

The Imbalance Parameter

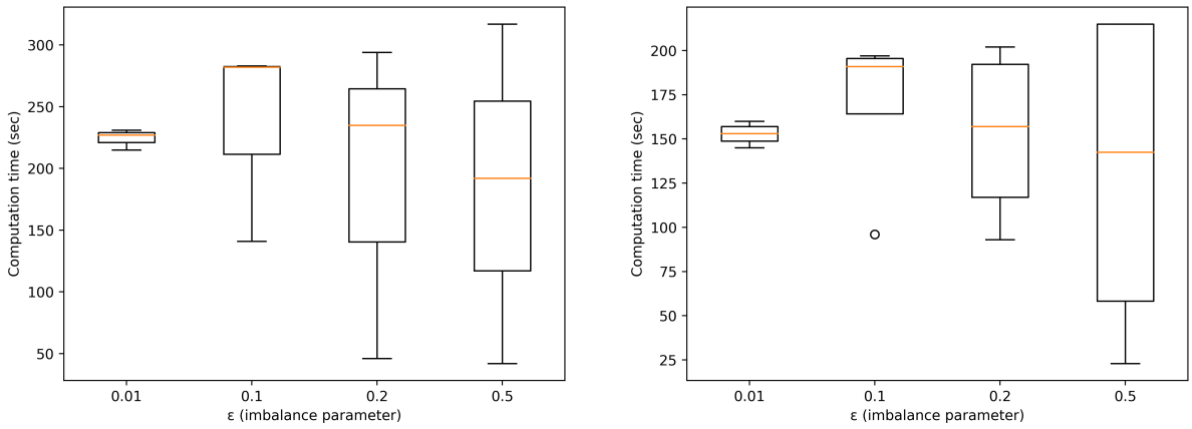
Figure 11 shows the impact of changing the value of ε on the number of served trips as well as the computation time of the ε-uniform tour-based clustering method under different numbers of clusters. Figure 11 (a) displays that as the value of ε increases, the number of served riders increases for all values of K. This is because a higher value for ε provides a higher level of flexibility to assign trips to clusters, thereby resulting in higher quality solutions. Figure 11 (b) demonstrates that the computation time increases with ε over all values of K. This is due to the fact that the increased level of flexibility that accompanies a higher ε value results in less uniform cluster sizes. As such, some clusters will be larger than others, increasing the overall solution time.



(a) Number of riders being served. (b) Computation time.

Figure 11: Number of served riders and computation time of the proposed algorithm with different values of ε

Figure 12 provides a more detailed view of the change in computation time for different values of ε. Figures 12(a) and 12(b) demonstrate the distribution of computation time under 3 and 4 clusters, respectively. These figures indicate that the cluster sizes become less uniform as we increase ε, resulting in the larger clusters taking longer to optimize.

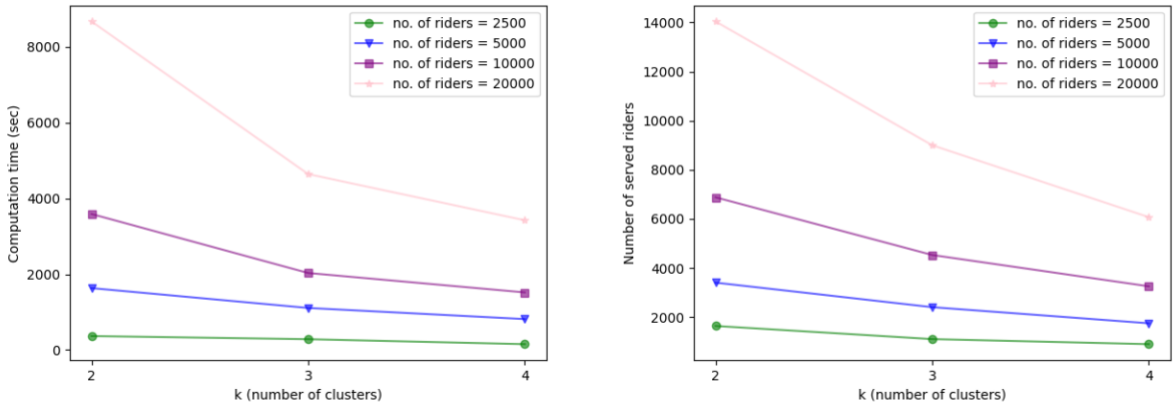


(a) Number of clusters = 3. (b) Number of clusters = 4.

Figure 12: Distribution of computation time

Number of System Participants

Figure 13 displays the influence of the size of participants on the computation time and the number of served riders under different numbers of clusters and $\varepsilon = 0.1$. Note that $k = 1$ provides the optimal solution. Figure 13 (a) demonstrates that the computation time decreases with the number of clusters regardless of the number of participants. Still, the rate of reduction in solution time decreases as the number of clusters becomes larger. This is because the larger number of clusters indicates that there are fewer participants in each cluster, which results in less computation time. However, once the number of clusters reaches a threshold, the reduction in solution time as we increase the number of clusters becomes small, indicating that there is a critical threshold for k where having more clusters does not help with reducing solution time any further, but decreases system throughput (Figure 13 (b)). Additionally, Figure 13 (a) shows that once the number of clusters is over a critical threshold, the computation time is not substantially affected by the number of participants anymore.



(a) Time

(b) Number of served riders

Figure 13: Number of served riders and computation time under different numbers of clusters

Figure 13 (b) shows that increasing the number of clusters reduces the matching rate; however, this reduction is smaller when the base number of participants is lower. This is due to the fact that with higher trip density produced by a higher number of participants, clustering leads to removing a higher number of potential matches from the feasible region of the solution, leading to higher loss in system throughput. This figure also shows that the rate of reduction in system throughput decreases between $k = 3$ and $k = 2$, compared to $k = 2$ and $k = 1$. Figure 13 shows that, regardless of the number of participants, there is a a critical value for k, where increasing the number of clusters beyond this value does not reduce the solution time any further, but reduces system throughput. For our experimental setting, this critical value is $k = 2$.

8. Conclusion

In this paper we devise a framework to solve the ride-matching problem that arises in dynamic ridesourcing systems in a distributed fashion. The methodology is based on clustering, where ride requests are grouped into a number of clusters so as to (1) maximize the intra-cluster similarity between trips within a cluster, and (2) guarantee cluster sizes to be uniform within a threshold. The proposed clustering approach accounts for the fact the ultimate goal is to form vehicle tours in each cluster, thereby using tours, i.e., sequences of trips, as cluster representative. We devise what we call the ε-uniform tour-based algorithm to assign trips to clusters,

and prove its convergence. Next, we optimally assign drivers to clusters, and solve the ride-matching problem for clusters independently of each other.

Using the New York City taxi dataset, we conduct extensive numerical experiments to analyze the performance of the proposed methodology and compare it against three state-of-the-art benchmarks, namely point-based, balanced point-based, and trip-based clustering, as well as the optimal solution. First, we demonstrate that the proposed methodology has favorable convergence properties, providing solutions in a few iterations. Secondly, we demonstrate the importance of forming approximately-uniform clusters, and showcase the resulting trade-offs between solution time and quality. Finally, we show that our proposed methodology could result in a statistically significant increase in the matching rate compared to the benchmarks, where this improvement decreases with the number of clusters.

Acknowledgment

The work described in this paper was supported by NSF award 2046372.

Appendix A

we consider a one-to-many ride-matching problem in which a driver can serve multiple riders. In such systems, a trip can be denoted by a link, $l = (t_i, s_i, t_j, s_j) \in T \times S \times T \times S$, where T is an ordered set of time intervals during the study time horizon. Due to the large size of the transportation network and the number of time intervals in the study horizon, solving such a ride-matching problem can be computationally prohibitive. Therefore, we adopt the pre-processing procedure proposed by (Masoud and Jayakrishnan 2017a) to reduce the size of the link sets.

The rationale of the pre-processing procedure is that the spatiotemporal constraints enforced by travel time windows of participants limit their access to members of the link set L. This pre-processing procedure starts by forming an ellipse for each participant, where the foci of the ellipse are set to the participant's origin and destination stations, the distance between the foci is the Euclidean distance between the participant's origin and destination stations, and the distance between the vertices of the ellipse is set to an upper-bound on the distance that the participant can travel within their travel time window. This ellipse defines a reduced graph, where any link with at least one station outside the ellipse will be infeasible for the participants as it will violate at least one of the spatio-temporal constraints.

This pre-processing procedure provides $L_d \subset L$ and $L_r \subset L$ as the set of links accessible to driver d and rider r, respectively. Furthermore, we define $L_{rd} = L_r \cap L_d$. Finally, $T_r \subset T$ and $T_d \subset T$ are sets of time intervals within the time window of rider r and driver d, respectively. The ride-matching problem can be modeled on a graph $G = (S, L)$, where S is the set of pick-up and drop-off stations, and L is the set of links.

This problem can be mathematically formulated as an integer programming model in (A.1). In this model, the decision variable ω_{rd} is a binary variable that takes on the value 1 if rider r is matched with driver d, and the value 0 otherwise. There are two additional sets of binary variables : (1) x_d^l which takes the value 1 if driver d travels on link l, and the value 0 otherwise; and (2) y_{rd}^l which takes the value 1 if rider r is transported by driver d on link l, and the value 0 otherwise.

The objective function in (A.1a) maximizes the total number of served riders. Constraints (A.1b) and (A.1c) ensure that drivers start their trips from their origin stations and end their trips at their destination stations, respectively. Constraint (A.1d) guarantees the flow conservation of vehicles. Constraints (A.1e) and (A.1f) ensure that served riders depart from their origin stations and arrive at their destination stations within their specified time windows. Constraint (A.1g) guarantees the flow conservation of riders. Constraint (A.1h) states that riders can be matched with only one driver. Constraint (A.1i) limits the capacities of the vehicles.

$$\max \quad \sum_{r\in R}\sum_{d\in D}\omega_{rd} \tag{A.1a}$$

$$\text{s.t.} \quad \sum_{\substack{l\in L_d:\\ s_i=s_d^O;t_i,t_j\in T_d}} x_d^l - \sum_{\substack{l\in L_d:\\ s_j=s_d^O;t_i,t_j\in T_d}} x_d^l = 1 \qquad \forall\ d\in D; \tag{A.1b}$$

$$\sum_{\substack{l\in L_d:\\ s_j=s_d^D;t_i,t_j\in T_d}} x_d^l - \sum_{\substack{l\in L_d:\\ s_i=s_d^D;t_i,t_j\in T_d}} x_d^l = 1 \qquad \forall\ d\in D; \tag{A.1c}$$

$$\sum_{\substack{t_i,s_i\\ l=(t_i,s_i,t,s)\in L_d}} x_d^l - \sum_{\substack{t_i,s_i\\ l=(t,s,t_i,s_i)\in L_d}} x_d^l = 0 \qquad \forall\ d\in D, \forall\ t\in T_d, \forall s\in S\backslash\{s_d^O\cup s_d^D\}; \tag{A.1d}$$

$$\sum_{\substack{l\in L_{rd}:s_i=s_r^O;\\ t_i,t_j\in T_r}} y_{rd}^l - \sum_{\substack{l\in L_{rd}:s_j=s_r^O;\\ t_i,t_j\in T_r}} y_{rd}^l = \omega_{rd} \qquad \forall\ r\in R, \forall\ d\in D; \tag{A.1e}$$

$$\sum_{\substack{l\in L_{rd}:s_j=s_r^D;\\ t_i,t_j\in T_r}} y_{rd}^l - \sum_{\substack{l\in L_{rd}:s_i=s_r^D;\\ t_i,t_j\in T_r}} y_{rd}^l = \omega_{rd} \qquad \forall\ r\in R, \forall\ d\in D; \tag{A.1f}$$

$$\sum_{d\in D}\sum_{\substack{t_i,s_i\\ l=(t_i,s_i,t,s)\in L_{rd}}} y_{rd}^l - \sum_{d\in D}\sum_{\substack{t_i,s_i\\ l=(t_i,s_i,t,s)\in L_{rd}}} y_{rd}^l = 0 \qquad \forall\ r\in R, \forall\ t\in T_r, \forall s\in S\backslash\{s_r^O\cup s_r^D\}; \tag{A.1g}$$

$$\sum_{d\in D}\omega_{rd}\le 1 \qquad \forall\ r\in R \tag{A.1h}$$

$$\sum_{r\in R} y_{rd}^l \le \mathrm{cap}_d \qquad \forall\ d\in D, \forall l\in L_d(2) \tag{A.1i}$$

Appendix B

Table 8: Information of participants in the illustrative example

Participant	Role	Origin station	Destination station	Earliest departure time	Latest arrival time	Shortest path travel time
r_1	rider	162	103	19:00	19:19	15
r_2	rider	161	80	19:00	19:15	13
r_3	rider	80	129	19:07	19:16	9
r_4	rider	95	52	19:12	19:30	13
r_5	rider	97	56	19:16	19:36	16
r_6	rider	97	121	19:23	19:55	30
r_7	rider	33	60	19:25	19:39	8
r_8	rider	50	35	19:28	19:53	20
r_9	rider	56	50	19:30	19:56	17
r_{10}	rider	98	97	19:34	19:41	6
r_{11}	rider	124	76	19:36	19:45	7
r_{12}	rider	26	71	19:38	19:50	9
r_{13}	rider	57	131	19:42	19:52	7
r_{14}	rider	78	161	19:47	20:02	11

r_{15}	rider	124	156	19:50	19:54	3
r_{16}	rider	98	130	19:52	20:05	8
r_{17}	rider	49	124	19:53	20:03	10
r_{18}	rider	58	159	19:56	20:06	8
r_{19}	rider	128	100	19:58	20:10	9
r_{20}	rider	125	128	19:59	20:15	13
d_1	driver	98	NA	18:50	NA	NA
d_2	driver	57	NA	19:00	NA	NA
d_3	driver	31	NA	18:50	NA	NA
d_4	driver	78	NA	19:00	NA	NA

Appendix C

Table 9: Shortest path travel time for illustrative example (I)

Origin Station	Destination Station															
	26	31	33	35	49	50	52	56	57	58	60	71	76	78	80	95
26	0	25	16	29	7	11	20	29	33	38	37	9	13	19	23	13
31	10	0	11	18	11	9	6	16	21	26	36	19	10	7	11	20
33	14	4	0	9	15	13	10	7	12	17	8	23	14	11	10	24
35	18	8	4	0	19	17	14	5	5	10	21	27	18	15	14	28
49	7	21	28	31	0	4	14	26	30	35	46	9	10	16	20	10
50	6	17	26	20	12	0	10	23	27	32	43	11	7	13	17	11
52	11	7	18	25	11	9	0	18	23	28	28	20	10	9	13	20
56	18	8	4	7	19	17	12	0	5	10	3	27	18	33	39	28
57	21	11	7	3	22	20	15	3	0	5	16	30	21	16	12	31
58	24	14	10	6	25	23	18	6	3	0	11	33	24	19	15	34
60	34	24	20	16	35	33	28	16	13	10	0	43	34	29	25	43
71	13	24	24	35	10	14	19	28	32	37	38	0	12	18	22	12
76	17	12	19	23	13	11	7	16	20	25	36	22	0	6	10	10
78	16	6	13	17	17	15	7	10	14	19	30	25	11	0	4	21
80	22	12	9	13	23	21	13	6	10	15	26	31	22	13	0	32
95	15	18	15	29	11	9	13	22	26	31	18	20	6	12	16	0
97	22	12	19	23	22	20	11	16	20	25	36	31	11	6	10	12
98	19	9	16	20	20	18	10	13	17	22	33	28	14	3	7	18
100	26	16	12	15	27	25	17	8	12	17	28	35	26	17	4	29
103	32	22	18	14	33	31	26	14	11	10	18	41	32	25	18	32
121	16	27	34	38	13	17	22	31	35	40	47	7	15	21	25	12
124	17	22	29	33	14	13	17	26	30	35	45	16	7	16	20	4
125	18	21	28	32	14	12	16	25	29	34	42	21	9	15	19	3
128	22	12	19	23	23	21	13	16	20	25	31	31	17	6	10	21

129	28	16	9	23	25	22	18	18	21	21	26	27	15	9	9	17
130	31	21	17	20	32	30	22	13	17	21	26	40	27	16	9	27
131	28	18	14	17	29	27	22	10	14	18	23	37	28	21	12	30
156	22	25	32	36	18	16	20	29	33	38	40	15	13	19	23	7
159	28	18	25	29	26	24	19	22	26	31	36	30	21	12	16	15
161	29	19	21	24	30	28	20	17	21	26	30	38	24	13	13	24
162	32	22	12	23	33	31	23	16	20	24	26	38	27	16	12	23

Table 10: Shortest path travel time for illustrative example (II)

Origin Station	Destination Station														
	97	98	100	103	121	124	125	128	129	130	131	156	159	161	162
26	21	22	26	36	13	11	14	25	28	28	31	16	23	30	30
31	16	10	14	24	23	21	21	13	16	16	19	26	19	18	20
33	20	14	13	23	27	25	25	17	19	15	18	30	23	22	19
35	24	18	17	16	31	29	29	21	23	19	22	34	27	26	23
49	18	19	23	33	13	11	14	22	25	25	28	16	23	27	29
50	15	16	20	30	15	13	16	19	22	22	25	18	21	24	26
52	13	12	16	26	24	22	21	15	13	18	21	17	21	20	22
56	22	16	12	16	31	18	29	19	18	14	17	14	25	38	18
57	25	19	15	11	34	32	32	22	21	17	7	37	28	13	20
58	28	22	18	6	37	35	35	25	21	18	15	38	27	18	15
60	4	32	28	15	47	15	40	30	26	23	20	43	32	23	20
71	20	21	25	35	4	10	13	24	27	27	30	12	20	26	25
76	8	9	13	23	26	16	11	16	15	15	18	18	14	17	19
78	9	3	7	17	29	27	22	6	9	9	12	27	12	11	13
80	22	16	3	13	35	33	29	15	9	5	8	32	21	12	9
95	18	21	19	29	21	6	5	14	17	19	22	11	13	19	21
97	0	3	13	23	30	18	13	6	9	11	14	20	12	31	14
98	6	0	10	20	32	24	19	3	6	8	11	24	9	8	11
100	21	15	0	10	39	31	26	12	6	2	5	29	18	9	6
103	28	22	15	0	44	34	29	19	15	12	9	11	21	12	9
121	20	23	28	36	0	8	11	21	24	24	27	8	16	22	21
124	12	15	23	33	15	0	3	16	19	21	24	3	12	19	19
125	11	14	22	31	20	5	0	13	16	18	21	9	9	16	16
128	9	3	9	18	33	23	18	0	3	5	8	21	6	5	8
129	9	6	6	15	24	19	16	3	0	4	9	17	9	3	6
130	19	13	5	13	39	29	24	10	4	0	3	27	16	7	4
131	14	18	9	10	41	32	27	15	9	5	0	30	19	10	7
156	15	17	22	29	12	6	4	14	17	17	20	0	8	15	14

159	15	9	15	24	27	17	12	6	9	11	14	15	0	7	10
161	16	10	9	18	36	26	21	7	3	5	8	24	9	0	4
162	19	13	8	15	35	25	20	10	6	3	6	23	12	16	0

Appendix D

Table 11: The costs between trips and tours for the illustrative example

Rider	Preliminary Cost		Secondary Cost	
	tour k_1	tour k_2	tour k_1	tour k_2
r_1	M	M	21	28
r_2	M	0		
r_3	0	M		
r_4	M	M	21	28
r_5	M	M	19	23
r_6	M	M	26	20
r_7	0	M		
r_8	M	M	31	28
r_9	M	M	32	30
r_{10}	M	0		
r_{11}	M	M	20	16
r_{12}	M	M	21	37
r_{13}	M	M	23	21
r_{14}	M	0		
r_{15}	0	M		
r_{16}	M	M	10	13
r_{17}	M	M	30	11
r_{18}	M	M	25	30
r_{19}	M	M	14	9
r_{20}	M	M	17	19

Reference

Agatz, Niels, Alan L. Erera, Martin W. P. Savelsbergh, and Wang Xing. 2011. "Dynamic Ride-Sharing: A Simulation Study in Metro Atlanta." *Transportation Research Part B Methodological*, 1450–64.

Agatz, Niels, Alan Erera, Martin Savelsbergh, and Xing Wang. 2012. "Optimization for Dynamic Ride-Sharing: A Review." *European Journal of Operational Research*, 295–303.

Akhremtsev, Yaroslav, Peter Sanders, and Christian Schulz. 2017. "High-Quality Shared-Memory Graph Partitioning." *IEEE Transactions on Parallel and Distributed Systems*.

Alonso-Mora, Javier, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. 2017. "On-Demand High-Capacity Ride-Sharing via Dynamic Trip-Vehicle Assignment." *Proceedings of the National Academy of Sciences* 114 (3): 462–67.

Andreev, Konstantin, and Harald Racke. 2006. "Balanced Graph Partitioning." *Theory of Computing Systems*, 929–39.

Baldacci, Roberto, and Maniezzo Aristide Mingozzi. 2004. "An Exact Method for the Car Pooling Problem Based on Lagrangean Column Generation." *Operations Research*, 422–39.

Boman, E. G., K. D. Devine, and S. Rajamanickam. 2013. "Scalable Matrix Computations on Large Scale-Free Graphs Using 2d Graph Partitioning." In *SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 1–12.

Buluc, Aydin, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. 2016. *Recent Advances in Graph Partitioning*. Springer International Publishing.

Carticipate. 2008 (accessed Dec. 15, 2020). "The United States." **https://www.carticipate.com/**.

Chakraborty, Jayita, Debapratim Pandit, Felix Chan, and Jianhong Xia. 2020. "A Review of Ride-Matching Strategies for Ridesourcing and Other Similar Services." *Transport Reviews*, 1–22.

Chan, Nelson D, and Susan A Shaheen. 2012. "Ridesharing in North America: Past, Present, and Future." *Transport Reviews*, 93–112.

Chu, Shumo, and James Cheng. 2011. "Triangle Listing in Massive Networks and Its Applications." In *Acm Sigkdd International Conference on Knowledge Discovery & Data Mining*.

Cordeau, Jean-François, and Gilbert Laporte. 2003. "The Dial-a-Ride Problem (DARP): Variants, Modeling Issues and Algorithms." *Quarterly Journal of the Belgian, French and Italian Operations Research Societies* 1 (2): 89–101.

———. 2007. "The Dial-a-Ride Problem: Models and Algorithms." *Annals of Operations Research* 153 (1): 29–46.

Cortés, Cristián E., Martín Matamala, and Claudio Contardo. 2010. "The Pickup and Delivery Problem with Transfers: Formulation and a Branch-and-Cut Solution Method." *European Journal of Operational Research*, 711–24.

Delling, Daniel, and Renato Werneck. 2012. "Better Bounds for Graph Bisection." *Algorithms–ESA 2012*, 407–18.

Di Febbraro, A., E. Gattorna, and N. Sacco. 2013. "Optimization of Dynamic Ridesharing Systems." *Transportation Research Record Journal of the Transportation Research Board*, 44–50.

Drews, F., and D. Luxen. 2013. "Multi-Hop Ride Sharing." *Proceedings of the 6th Annual Symposium on Combinatorial Search, SoCS 2013*, 71–79.

Fiduccia, C, and R Mattheyses. 1982. "A Linear-Time Heuristic for Improving Network Partitions." *Proceedings of the 19th Design Automation Conference*, 175–81.

Flinc. 2011 (accessed Dec. 15, 2020). "Germany." **https://flinc.org/**.

Fluide, Ville. 2011 (accessed Dec. 15, 2020). "France." **http://www.villefluide.fr/**.

Fortunato, Santo. 2010. "Community Detection in Graphs." *Physics Reports*, 75–174.

Furuhata, Masabumi, Maged Dessouky, Fernando Ordó?Ez, Marc Etienne Brunet, Xiaoqing Wang, and Sven Koenig. 2013. "Ridesharing: The State-of-the-Art and Future Directions." *Transportation Research Part B Methodological*, 28–46.

Ghoseiri, Keivan. 2012. "Dynamic Rideshare Optimized Matching Problem." *Dissertations & Theses - Gradworks*.

Grady, Leo, and Eric L Schwartz. 2006. "Isoperimetric Graph Partitioning for Image Segmentation." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 469–75.

Hensher, David A. 2008. "Climate Change, Enhanced Greenhouse Gas Emissions and Passenger Transport–What Can We Do to Make a Difference?" *Transportation Research Part D: Transport and Environment*, 95–111.

Herbawi, Wesam, and Michael Weber. 2011. "Evolutionary Multiobjective Route Planning in Dynamic Multi-Hop Ridesharing." In *Evolutionary Computation in Combinatorial Optimization*, 84–95. Springer Berlin Heidelberg.

———. 2012. "A Genetic and Insertion Heuristic Algorithm for Solving the Dynamic Ridematching Problem with Time Windows." *GECCO'12 - Proceedings of the 14th International Conference on Genetic and Evolutionary Computation*.

Ho, Sin C, Wai Yuen Szeto, Yong-Hong Kuo, Janny MY Leung, Matthew Petering, and Terence WH Tou. 2018. "A Survey of Dial-a-Ride Problems: Literature Review and Recent Developments." *Transportation Research Part B: Methodological* 111: 395–421.

Jafari, E., T. Rambha, A. Khani, and S. D. Boyles. 2016. "The for-Profit Dial-a-Ride Problem on Dynamic Networks." In *Transportation Research Board 96th Annual Meeting*.

Karisch, Stefan E, Franz Rendl, and Jens Clausen. 2000. "Solving Graph Bisection Problems with Semidefinite Programming." *INFORMS Journal on Computing*, 177–91.

Kernighan, Brian W, and Shen Lin. 1970. "An Efficient Heuristic Procedure for Partitioning Graphs." *The Bell System Technical Journal*, 291–307.

Kieritz, Tim, Dennis Luxen, Peter Sanders, and Christian Vetter. 2010. "Distributed Time-Dependent Contraction Hierarchies." *Experimental Algorithms*, 83–93.

Li, Ruimin, Zhiyong Liu, and Ruibo Zhang. 2018. "Studying the Benefits of Carpooling in an Urban Area Using Automatic Vehicle Identification Data." *Transportation Research Part C: Emerging Technologies*, 367–80.

Lloret-Batlle, Roger, Neda Masoud, and Daisik Nam. 2017. "Peer-to-Peer Ridesharing with Ride-Back on High-Occupancy-Vehicle Lanes: Toward a Practical Alternative Mode for Daily Commuting." *Transportation Research Record* 2668 (1): 21–28.

Lloyd, S. P. 1982. "Least Squares Quantization in PCM." *IEEE Trans* 28 (2): 129–37.

Luxen, Dennis, and Dennis Schieferdecker. 2015. "Candidate Sets for Alternative Routes in Road Networks." *Journal of Experimental Algorithmics (JEA)*, 1–28.

Lyft. 2012 (accessed Dec. 15, 2020). "United States." **https://www.lyft.com/**.

Ma, S., Z. Yu, and O. Wolfson. 2013. "T-Share: A Large-Scale Dynamic Taxi Ridesharing Service." In *IEEE International Conference on Data Engineering*.

Masoud, Neda, and R Jayakrishnan. 2017a. "A Decomposition Algorithm to Solve the Multi-Hop Peer-to-Peer Ride-Matching Problem." *Transportation Research Part B Methodological*, 1–29.

———. 2017b. "A Real-Time Algorithm to Solve the Peer-to-Peer Ride-Matching Problem in a Flexible Ridesharing System." *Transportation Research Part B: Methodological* 106: 218–36.

———. 2017c. "Autonomous or Driver-Less Vehicles: Implementation Strategies and Operational Concerns." *Transportation Research Part E: Logistics and Transportation Review* 108: 179–94.

Masoud, Neda, Daisik Nam, Jiangbo Yu, and R. Jayakrishnan. 2017. "Promoting Peer-to-Peer Ridesharing Services as Transit System Feeders." *Transportation Research Record: Journal of the Transportation Research Board*, 74–83.

Mes, Martijn, Matthieu Van Der Heijden, and Aart Van Harten. 2007. "Comparison of Agent-Based Scheduling to Look-Ahead Heuristics for Real-Time Transportation Problems." *European Journal of Operational Research*, 59–75.

Najmi, Ali, David Rey, and Taha H Rashidi. 2017. "Novel Dynamic Formulations for Real-Time Ride-Sharing Systems." *Transportation Research Part E: Logistics and Transportation Review* 108: 122–40.

Nam, Daisik, Dingtong Yang, Sunghi An, Jiangbo Gabriel Yu, R. Jayakrishnan, and Neda Masoud. 2018. "Designing a Transit-Feeder System Using Multiple Sustainable Modes: Peer-to-Peer (P2p) Ridesharing, Bike Sharing, and Walking." *Transportation Research Record Journal of the Transportation Research Board*.

Nourinejad, Mehdi, and Matthew J. Roorda. 2016. "Agent Based Model for Dynamic Ridesharing." *Transportation Research Part C*, 117–32.

NYC.gov. (accessed Dec. 15, 2020). "The New York City Taxi Dataset." **https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page**.

Pelzer, Dominik, Jiajian Xiao, Daniel Zehe, Michael H Lees, Alois C Knoll, and Heiko Aydt. 2015. "A Partition-Based Match Making Algorithm for Dynamic Ridesharing." *IEEE Transactions on Intelligent Transportation Systems*, 2587–98.

Peng, Bo, Lei Zhang, and David Zhang. 2013. "A Survey of Graph Theoretical Approaches to Image Segmentation." *Pattern Recognition*, 1020–38.

Regue, Robert, Neda Masoud, and Will Recker. 2016. "Car2work: Shared Mobility Concept to Connect Commuters with Workplaces." *Transportation Research Record: Journal of the Transportation Research Board* 2542 (January): 102–10. **https://doi.org/10.3141/2542-12**.

Sanders, Peter, and Christian Schulz. 2013. *Engineering Multilevel Graph Partitioning Algorithms*. Springer Berlin Heidelberg.

Santos, Douglas O, and Eduardo C Xavier. 2015. "Taxi and Ride Sharing: A Dynamic Dial-a-Ride Problem with Money as an Incentive." *Expert Systems with Applications* 42 (19): 6728–37.

Senin, Pavel. 2008. "Dynamic Time Warping Algorithm Review." *Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA*, 40.

Stiglic, Mitja, Niels Agatz, Martin Savelsbergh, and Mirko Gradisar. 2015. "The Benefits of Meeting Points in Ride-Sharing Systems." *Transportation Research Part B*.

Tafreshian, Amirmahdi, Mojtaba Abdolmaleki, Neda Masoud, and Huizhu Wang. 2021. "Proactive Shuttle Dispatching in Large-Scale Dynamic Dial-a-Ride Systems." *Transportation Research Part B: Methodological* 150: 227–59.

Tafreshian, Amirmahdi, and Neda Masoud. 2020a. "Trip-Based Graph Partitioning in Dynamic Ridesharing." *Transportation Research Part C: Emerging Technologies*, 532–53.

———. 2020b. "Using Subsidies to Stabilize Peer-to-Peer Ridesharing Markets with Role Assignment." *Transportation Research Part C: Emerging Technologies* 120.

Tafreshian, Amirmahdi, Neda Masoud, and Yafeng Yin. 2020. "Frontiers in Service Science: Ride Matching for Peer-to-Peer Ride Sharing: A Review and Future Directions." *Service Science* 12 (2-3): 44–60.

Uber. 2009 (accessed Dec. 15, 2020). "The United States." **https://www.uber.com/**.

Wang, Hai, and Hai Yang. 2019. "Ridesourcing Systems: A Framework and Review." *Transportation Research Part B: Methodological* 129: 122–55.

Winter, Stephan, and Silvia Nittel. 2006. "Ad Hoc Shared-ride Trip Planning by Mobile Geosensor Networks." *International Journal of Geographical Information Science*, 899–916.

Zhan, Xingbin, WY Szeto, CS Shui, and Xiqun Michael Chen. 2021. "A Modified Artificial Bee Colony Algorithm for the Dynamic Ride-Hailing Sharing Problem." *Transportation Research Part E: Logistics and Transportation Review* 150: 102124.

Zhang, Zhenhao, Amirmahdi Tafreshian, and Neda Masoud. 2020. "Modular Transit: Using Autonomy and Modularity to Improve Performance in Public Transportation." *Transportation Research Part E: Logistics and Transportation Review* 141: 102033.

: 9101–7.