Improving Transportation Efficiency with Ridesharing

by

Jiajian Leo Liang

M.Sc., Simon Fraser University, 2016
B.Sc., Simon Fraser University, 2013

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Summer 2023

Declaration of Committee

Name:	**Jiajian Leo Liang**
Degree:	**Doctor of Philosophy**
Thesis title:	**Improving Transportation Efficiency with Ridesharing**
Committee:	**Chair:** Igor Shinkar Assistant Professor, Computing Science
	Qianping Gu Supervisor Professor, Computing Science
	Pavol Hell Committee Member Professor, Computing Science
	Andrei Bulatov Examiner Professor, Computing Science
	Guohui Lin External Examiner Professor, Computing Science University of Alberta

Abstract

Due to growing population, there is a higher demand for public transit. On the other hand, public transportation systems in many cities are not able to handle all the increasing demand due to their slow development. This and the inconvenient of transit push people to use personal vehicles. Mobility-on-demand (MoD) systems, such as Uber and Lyft, have become popular around the globe for their convenience. Despite their convenience, the current use of MoD has created a negative effect on traffic, such as increasing traffic congestion. One way to improve transportation efficiency is to promote ridesharing among MoD systems since it is a promising way to increase the occupancy rate of personal vehicles and reduce traffic congestion. We study two ridesharing minimization problems: given a set of individuals, (1) assign the minimum number of individuals as drivers to serve all individuals, and (2) minimize the total travel distance of the assigned drivers to serve all individuals. We show that even restricted variants of these two problems are NP-hard (and NP-hard to approximate within a constant factor). We propose exact and approximation algorithms for restricted variants of these two problems. We also study a ridesharing maximization problem: given a set of drivers and a set of passengers, maximize the number of passengers assigned to drivers such that the total profit of drivers reaches a specified target. This problem focuses on the profitability for adopting ridesharing in practice. We give an exact and two approximation algorithms for two variants of this problem. Based on a real-world ridesharing dataset in Chicago City, profit model of Uber and practical scenarios, we create datasets for an extensive computational study on our model and algorithms. We study another optimization problem that focuses on the integration of public transit with ridesharing, which can increase transit ridership. Specifically, given a set of drivers and a set of transit riders, we assign riders to drivers such that the number of riders with shorter transit travel time (with the use of ridesharing) is maximized. We show that this problem is NP-hard, and we present an exact algorithm approach and several polynomial-time constant approximation algorithms. Based on real-world ridesharing and transit datasets in Chicago City, we conducted an extensive computational study to showcase the potential of integrating public transit with ridesharing.

Keywords: Ridesharing; Multimodal transportation; Combinatorial optimization; Approximation algorithms; Algorithmic analysis; Computational study

Acknowledgements

It has been a long journey. I thank everyone who has provided guidance, advice and help along the way, including those in my personal life.

Firstly, I would like to express my gratitude to my supervisor Prof. Qianping Gu for his guidance and support. His research knowledge has been invaluable to me in the development of my research skill. I am grateful that we have worked on and published a number of papers together. His thoughtful ideas and encouragements have been instrumental in the completion of my thesis. I am grateful for the opportunity he provided to me to work with Prof. Guochuan Zhang (from Zhejiang University) who is very professional and knowledgeable. I would like to thank Prof. Guochuan Zhang for his contribution. I would also like to thank my graduate committee member Prof. Pavol Hell for his advice and encouragement.

I would also thank the members of my examination committee, Prof. Qianping Gu, Prof. Pavol Hell, Prof. Andrei Bulatov and Prof. GuoHui Lin (from University of Alberta) for their time and feedback on my thesis.

I am also grateful to my colleagues and friends who have supported me along the way. Research discussions with Dr. Songhua Li (from City University of Hong Kong) have been valuable.

Finally, I want to thank my parents.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

As the population grows in urban areas, commuting between and within cities (districts) can be time-consuming and resource demanding. Due to growing passenger demand, the number of vehicles on the road for both public and private transportation has increased to handle the demand. Current public transportation systems in many cities may not be able to handle all the demand due to their slow development, which can cause greater inconvenience for transit users, such as longer waiting time, more transfers and/or imbalanced transit ridership. Personal vehicles and mobility-on-demand (**MoD**) systems are major transportation modes for many people.

According to studies in [23, 98, 100], personal vehicles were the main transportation mode in Canada and the United States in recent years and in more than 200 European cities between 2001 and 2011. Almost 73% of total work commute is by car as a driver in Canada in recent years [100]. In Europe 2017 [32], the transport sector accounted for 27% of total greenhouse gas emissions; and of these 27% gas emissions, 31.55% (8.52% total) were from passenger cars. In the US, the growth rates of population and transit ridership are usually align. However, the growth rate of population has been higher than that of transit ridership since 2016 [12] (9% higher in 2019 and substantial higher in 2020 due to COVID). The occupancy rate of personal vehicles in the U.S. was 1.6 persons per vehicle in 2011 [41, 96] and decreased to 1.5 persons per vehicle in 2017 [23], which can be a major cause for congestion and pollution.

MoD systems, such as Uber, Lyft and DiDi, have become popular around the globe for their convenience. Drivers and passengers can be matched based on real-time ridesharing requests (arrange a ride for passengers in a personal vehicle). MoD system operators and drivers participated in such systems are mostly motivated by profit in practice. Solely focusing on profit and market share from MoD systems and drivers may have increased congestion and CO_2 emissions as MoD systems become increasingly popular in many cities (due to the increase of low-occupancy vehicles on the road) [31, 52, 108]. This, coupled with the saturated personal vehicle usage (with low occupancy rate) in Europe and North America, worsens the traffic congestion and emissions [81].

On the other hand, there is an urgency to reduce traffic congestion and greenhouse gas emissions; and from the above, we can see that there is a need to improve transportation efficiency. This is the main motivation of this dissertation.

1.1 Research overview and contributions

Ridesharing is a promising way to increase the occupancy rate of personal vehicles, improve transportation efficiency, and reduce traffic congestion and emissions [7, 39, 107]. It is estimated that commuting to work by ridesharing in Dublin, Ireland, can reduce 12,674 tons of CO_2 emissions per year [22], and taxi-ridesharing in Beijing can reduce 120 million liters of gasoline annually [78]. As mentioned above, majority of work commute is by car. With the increasing popularity in ridesharing/ridehailing service, there may be potential to integrate private and public transportation systems, as suggested by some studies (e.g., [5, 58, 64, 80, 87, 102]). From the research report of [34], it is recommended that public transit agencies should build on mobility innovations to allow public-private engagement in ridesharing because the use of shared modes increases the likelihood of using public transit. A similar finding, reported in [116], indicates that the use of ridesharing may be positively associated with public transit ridership. We study the *ridesharing problem* and the problem of integrating public transit with ridesharing (called the *multimodal transportation with ridesharing problem*). There are several optimization goals in these problems. In this thesis, we focus on the following key optimization problems for improving transportation efficiency:

- Minimize the number of drivers to serve all passengers; minimize the travel distance of vehicles to serve all passengers.
- Maximize the number of served passengers by a given set of drivers.

Most previous works on solving the ridesharing problem used an Integer Programming (IP) or a Mixed Integer Programming (MIP) formulation and solved it by exact methods (for small and/or restricted instances) or heuristics. Only recently, there are studies that provide large-scale numerical experiments. It is similar for the multimodal transportation with ridesharing problem; and since it is a more recent problem, there are less results compared to the ridesharing problem. There are still missing approximation algorithms and NP-hardness complexity analysis for both problems. In addition, large-scale realistic experiments are still lacking. Hence, we propose a number of different fast and efficient algorithms, including exact and approximation algorithms and heuristics, for different variants of both problems, along with extensive large-scale numerical experiments. Here, a *fast algorithm* means that its actual running time/computational time is competitive among other algorithms for the same problem in our experiments, and *an efficient algorithm* means that it is a polynomial-time algorithm in the size of the input. In addition, a rigorous analysis of NP-hardness complexity is introduced, especially for the ridesharing problem, which is a

complete different focus compared to most previous works on the ridesharing problem. A high level description of each of these optimization problems is given below, along with a brief discussion and our contributions.

1.1.1 Ridesharing problem

In general, a *ridesharing problem* instance consists of a set of trips (ridesharing offers and requests) in a road network, where each trip has an individual and some requirements as input parameters; common requirements of a trip include an origin, a destination, a capacity (the number of seats in the vehicle for passengers), a detour limit, preferred paths, number of stops and time constraints [2, 39, 104]. The individual in each trip can be a driver (providing service) or a passenger (to be served). When an individual can be assigned as either a driver or a passenger, the individual is called *flexible*. When an individual is specified as a driver or a passenger in the input, the individual is called *fixed*. Most previous studies focus on the problems that all individuals are fixed [2, 39, 85, 104]. We consider both flexible and fixed individuals for different ridesharing optimization problems.

Ridesharing minimization problems.

We study the following optimization problems:

Problem 1. Given a set of flexible individuals, assign a subset individuals to be drivers to serve all individuals (pick-up at their origins and delivered to their destinations) subject to the requirements such that the number of assigned drivers is minimized.

Problem 2. Given a set of flexible individuals, assign a subset individuals to be drivers to serve all individuals subject to the requirements such that the total travel distance of the assigned drivers is minimized.

Our contributions for the ridesharing minimization problems are summarized as follows:

1. We analyze the relations between the time complexity of the ridesharing problem and its parameters (requirements) using the model recently introduced in [43, 68]. In particular, we associate each common input parameter with a restriction (labeled as **Conditions** *C1* to *C5*) as follows:

   C1. The individuals of all trips have the same destination or have the same origin.

   C2. The individual of each trip can only serve others who are on the individual's preferred path (without any detour).

   C3. There is only one preferred path from the individual of each trip.

   C4. The number of stops each individual is willing to make to pick-up and/or drop-off passengers is at least the input capacity from the individual of each trip.

   C5. The individuals of all trips have the same arrival time and departure time.

2. It is shown in [43, 68] that if one of Conditions C2, C3 and a weaker version of C1 (all trips have the same destination) is not satisfied, both minimization problems Problem 1 and Problem 2 are NP-hard. Based on the new model, we extend the results to if one of Conditions C1-C5 is not satisfied, both Problem 1 and Problem 2 are NP-hard. We further show that [47, 48] if one of Conditions C2-C5 is not satisfied, not only are the minimization problems NP-hard, but they are also NP-hard to approximate within a constant factor.

3. When all five Conditions are satisfied, we give two efficient exact algorithms for Problem 1 and Problem 2 [44, 46]. When Conditions C1-C3 and C5 are satisfied (Condition C4 is not), we give three $\frac{\lambda+2}{2}$-approximations algorithms for Problem 1 [47, 48], where λ is the maximum input capacity parameter among all trips.

Ridesharing maximization problem.

We study the following optimization problem on fixed individuals:

RPC Problem. Given a set of drivers and a set of passengers, assign passengers to drivers to maximize the number of passengers served subject to the trip requirements, including the total profit of drivers. The maximization problem is called the *Ridesharing with Profit Constraint (RPC)* problem.

The RPC problem can be formulated as the maximum set packing problem that is subject to a weight constraint, which can be seen as a more complex variant of the maximum (weight) set packing problem. We show that the RPC problem is NP-hard due to the NP-hardness of the maximum weight set packing problem [40, 62]. Our contributions for the RPC problem are:

1. We give a polynomial-time exact algorithm framework (including two practical implementations of the algorithm) and a $\frac{1}{2}$-approximation algorithm for a special case that each vehicle serves at most one passenger.

2. We give a fast $\frac{2}{3\lambda}$-approximation algorithm for the case that each vehicle serves at most $\lambda \geq 2$ passengers while only positive-profit assignments of passenger to drivers are allowed.

3. Based on a real-world ridesharing dataset in Chicago City, profit model of Uber and practical scenarios, we create datasets for an extensive computational study on our model and algorithms.

Importance of our results.

We have started a line of research that explores the hardness of the general ridesharing problem. The extend of the complexity analysis has not been done for the ridesharing

problem. We create a novel algorithmic analysis model for the ridesharing problem. Based on this model, we obtain complexity analysis results which provide us an insight into how difficult the ridesharing problem is. This research direction allows us to design exact and approximation algorithms for variants of the ridesharing problem, which are lacking for ridesharing and related problems in the literature. From the NP-hardness results and the exact algorithms, we can clearly draw a boundary between NP-hard and polynomial-time solvable cases. Our algorithms' novelty comes from the fact that most of them are discrete algorithms using graph theory, unlike most algorithms in previous studies in ridesharing and related optimization problems. Although the use of graph theory for ridesharing problem is not new, only recently such an approach has attracted attention.

Our solution approach for the RPC problem are developed by adopting the graph matching approach proposed by [95]. Such an approach is practical for a more general setting, such as the RPC problem and general ridesharing. The RPC problem is more complicated than the maximum weight set packing (MWSP) problem; and to the best of our knowledge, the RPC problem has not been studied before. Having said that, one of our approximation algorithms for RPC is an incremental work of a well known approximation algorithm for the MWSP problem [24]. Nonetheless, it can be a foundation for future work.

1.1.2 Multimodal transportation problem.

Travel planning that uses different transportation modes, such as bus, biking and private vehicles, are considered as *multimodal transportation* or *multimodal route planning* [14, 58]. Multimodal transportation, specifically integrating public transit system with ridesharing, is effective to improve the transportation efficiency. We focus on the *multimodal transportation with ridesharing (MTR)* problem which bridges the public transit and private transportation systems on fixed individuals. In the MTR problem, a set of drivers (who can provide ridesharing service) and a set of passengers (public transit users who want to use both transit and ridesharing) are given, submitted to the integrated transportation system (**ITS**). Each driver and each passenger have trip requirements related to locations and time constraints. The ITS tries to compute an acceptable route for as many passengers as possible. For a passengers, an *acceptable route* is a public transit and ridesharing combined route providing a commuting time that is faster than using only public transit, where the trip requirements of both passenger and driver are satisfied. We consider the following optimization goal of the MTR problem:

MTR maximization problem. Given a set of drivers and a set of passengers, match passengers with drivers to maximize the number of passengers, each of them is assigned an acceptable route.

For the MTR maximization problem, we obtain the following results [42]:

1. We give an exact algorithm approach (an ILP formulation based on a hypergraph representation) for integrating public transit and ridesharing.

2. We show that the MTR maximization problem is NP-hard and give an LP-rounding based $(1 - \frac{1}{e})$-approximation algorithm and a discrete $\frac{1}{2}$-approximation algorithm for the problem, where e is the Euler's number.

3. Based on a real-world ridesharing dataset and public transit data in Chicago City, we create datasets for an extensive computational study to evaluate the effectiveness of integrating public transit system with ridesharing.

Importance of our results.

To the best of our knowledge, no approximation algorithms have been proposed for any problem related to the integration of public transit and ridesharing. Approximation algorithms are important because they can guarantee the solution quality, and in this case, a certain amount of passengers are guaranteed to receive ridesharing service.

There are drawbacks in previous computational studies. In some previous experiments, the transit data (including the transit network and schedule, usually for a particular transportation mode) comes from a real-world dataset, but the ridesharing dataset is either small-scale or not real-world data. Some experiments have medium- to large-scale ridesharing dataset, but the road network/transit network/transit schedule datasets are artificial. Another caveat of some previous experiments is that they pre-compute certain data required in the experiments. For example, shortest paths (distances) between the drivers and passengers are pre-computed. This hides the realistic aspect of the application as such information is not known beforehand and computational time is substantially reduced. On the other hand, our experiment uses large-scale real-world ridesharing dataset. All computation is done in real-time, except computation related to transit schedule since it is known in advance. Admittedly, we only use a simplified transit network in our simulation, but it is created from a real-world transit network dataset.

1.2 Thesis outline

The rest of the thesis is organized as follows. In Chapter 2, common definitions and notations are introduced. A detailed problem definition of ridesharing is given, along with the theme of our general approach for solving the ridesharing problems. Chapter 3 is dedicated to the ridesharing problem: Including a review on ridesharing in Section 3.1, algorithmic analysis model and NP-hardness results in Section 3.2, and algorithm results for Problem 1 and Problem 2 in the remainder of Chapter 3. In Chapter 4, we first introduce the RPC problem formally. We explain our model for the problem in Section 4.1 and review related work in Section 4.2. Proposed algorithms for two variants of the RPC problem are presented

in Section 4.3 and Section 4.4, followed by an extensive computational study in Section 4.5. In Chapter 5, we first introduce the MTR problem, and then we review related work in Section 5.1. An exact algorithm approach for the MTR problem is given in Section 5.2. We then show that the MTR problem is NP-hard and give several polynomial-time approximation algorithms for the MTR problem in Section 5.3. An extensive computational study for the MTR problem is presented in Section 5.4. The final chapter concludes the thesis by providing a summary of the main contributions and possible future work.

Chapter 2

Preliminaries

First, we introduce some general notations. The sets of real numbers and integers are denoted by $\mathbb{R}$ and $\mathbb{Z}$, respectively. The sets $\mathbb{R}_{\geq 0}$ and $\mathbb{Z}_{\geq 0}$ represent non-negative real numbers and integers, respectively. For two integers a and b in $\mathbb{Z}$, $[a, b]$ denotes the range of integers from a to b inclusive (for clarity, we also use $[a, \ldots, b]$). Given two sets A and B, $A \times B = \{(a, b) \mid a \in A, b \in B\}$ denotes the cross product of the two sets A and B. For a set A, 2^A denotes the power set of the set A. Since most of our models for the ridesharing problem and multimodal transportation with ridesharing problem use graph theory. We introduce some graph theory notations.

2.1 Basic graph theory

An undirected graph $G(V, E)$ consists of a finite non-empty set V of elements called *vertices* (or *nodes*) and a set E of 2-element subsets of V called *edges* (or *arcs*). We usually use G to denote a graph $G(V, E)$. The sets $V(G)$ and $E(G)$ are called the vertex set and the edge set of G, respectively. The cardinality of $V(G)$, denoted by $|V(G)|$, is the number of vertices in G. The cardinality of $E(G)$, denoted by $|E(G)|$, is the number of edges in G. We may use V and E to denote $V(G)$ and $E(G)$, respectively, if graph G is clear from the context. For a pair of vertices $u, v \in V(G)$, the edge e between u and v is represented as $e = \{u, v\}$, and we also call u and v endpoints (end vertices) of edge e. For any pair of vertices $u, v \in V(G)$, if $e = \{u, v\}$ is an edge in $E(G)$, then u and v are said to be *adjacent* to each other, and they are *neighbors*. The vertices u and v are said to be *incident* to e and vice versa. Two edges $e, e' \in E(G)$ are *incident* if they share a common vertex of $V(G)$. The degree of a vertex $v \in V(G)$, denoted by $\deg_G(v)$, is the number of neighbors of v, namely, the number of vertices in G adjacent to v. A vertex v is *isolated* if $\deg_G(v) = 0$.

A directed graph (*digraph* for short) is similar to a graph G defined above. Formally, a digraph $G(V, E)$ is a finite non-empty set $V(G)$ of vertices and a set $E(G)$ of ordered pairs of members of $V(G)$; $E(G)$ is the set of directed edges of G. For a pair of vertices u, v in a digraph G, the directed edge e from u to v is represented as $e = (u, v)$. An edge $e = (u, v)$ is

also called an outgoing-edge (out-edge) of u and an incoming-edge (in-edge) of v. Vertices u and v are called the *tail* and *head* of e, respectively, and they are *adjacent* to each other. The *outdegree*, denoted by $\text{outdeg}_G(v)$, of a vertex $v \in V(G)$ is the number of edges in G leaving v. The *indegree*, denoted by $\text{indeg}_G(v)$, of a vertex $v \in V(G)$ is the number of edges entering v. We may use $\deg(v)$ (instead of $\deg_G(v)$) if graph G is clear from the context; and similar for outdegree and indegree. For a vertex $v \in V(G)$, v is called a *source* vertex if $\text{indeg}_G(v) = 0$ and $\text{outdeg}_G(v) \geq 1$ and a *sink* vertex if $\text{indeg}_G(v) \geq 1$ and $\text{outdeg}_G(v) = 0$.

A *bipartite graph* $G(U, V, E)$ is a graph in which the vertices of G are partitioned into two disjoint sets U and V such that for any edge e of G, one end vertex of e is in U and the other end vertex of e is in V. A graph G' is a *subgraph* of another graph G if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$.

A *walk* W in a graph (digraph) G is a subgraph of G and an alternating sequence between vertices and edges, that is, $W = v_1, e_1, v_2, e_2, \ldots, e_{b-1}, v_b$, where $b \geq 1$ and $e_i = \{v_i, v_{i+1}\}$ ($e_i = (v_i, v_{i+1})$ for digraphs) for $1 \leq i \leq b-1$. A *path* $P = v_1, e_1, v_2, \ldots, e_{b-1}, v_b$ in a graph (digraph) G is a walk with distant vertices. For simplicity, we denote a walk/path by using vertices only. Namely, for a path $P = v_1, e_1, v_2, \ldots, e_{b-1}, v_b$, we use $P = \{v_1, v_2, \ldots, v_b\}$ to represent the path P and $P = (v_1, \ldots, v_b)$ for digraphs. We say that $P = \{v_1, \ldots, v_b\}$ is a path between vertices v_1 and v_b and a path from vertex v_1 to vertex v_b for digraphs. The length of a walk W is the number of edges in W, denoted by the cardinality $|W|$ of W; and similarly for $|P|$ of a path P. A walk is *closed* if it starts and ends at the same vertex. A closed walk is also called a *cycle*. A *simple cycle* is a closed walk with distinct vertices and edges. In this dissertation, we are only concerned with simple cycles, and hence, we just simply call a simple cycle a cycle. A cycle is even (odd) if it has even (odd) length. A graph G is *connected* if there is a path between every pair of vertices in G. A *component* of a graph G is a connected subgraph of G. A graph without any cycle is called *acyclic* and a directed graph without any cycle is called a directed acyclic graph (DAG). A *tree* is a connected acyclic graph $G(V, E)$ where $|E(G)| = |V(G)| - 1$. An arbitrary acyclic graph is called a *forest*, which is a set of vertex-disjoint trees.

A hypergraph can be undirected or directed, but we only consider undirected hypergraphs in this dissertation with the following hypergraph definition: A hypergraph $H(V, E)$ consists of a finite non-empty set V of vertices and a finite family E of non-empty subsets of V such that each $e \in E(H)$ contains at least two vertices. Each $e \in E(H)$ is called a *hyperedge*, and denoted by $v \in e$ is an element v belongs to e. The number of elements in a hyperedge e is the size of e and is denoted by $|e|$. Note that our definition of a hypergraph H allows a hyperedge $e \in E(H)$ to be a subset of another hyperedge $e' \in E(H)$. All the definitions for undirected graphs stated above also applied to undirected hypergraphs.

An edge-weighted (hyper)graph G is a (hyper)graph where each edge of G is associated with a real number, called *weight*. The weight of an edge $e = \{u, v\} \in E(G)$ (or directed edge $e = (u, v)$) is denoted by $w(e) = w(u, v)$. For a set $E' \subseteq E(G)$, the weight of E' is

denoted by $w(E') = \sum_{e \in E'} w(e)$. We denote an edge-weighted (hyper)graph by $G(V, E, w)$, where $V(G)$ is the vertex set of G, $E(G)$ is the edge set of G, and w is a real value function on $E(G)$, namely, $w : E \to \mathbb{R}$. A *weighted* graph usually refers to an edge-weighted graph, unless otherwise stated. An integer-weighted (hyper)graph $G(V, E, w)$ is a (hyper)graph with integer edge weight, namely, $w : E \to \mathbb{N}$. The weight of a path $P = \{v_1, \ldots, v_b\}$ or $P = (v_1, \ldots, v_b)$ is denoted by

$$w(P) = \sum_{i=1}^{b-1} w(v_i, v_{i+1}).$$

We also use $\text{dist}(P)$ to denote the weight $w(P)$ of a path P. For a path P in an unweighted graph G, $\text{dist}(P) = w(P) = |P|$. A path $P = (u, \ldots, v)$ between two vertices u and v in a (weighted) graph G is a *shortest path* if $w(P)$ is smallest among all paths between u to v (or from u to v for digraphs); and further, the shortest distance between u and v (or from u to v for digraphs) is denoted by $\text{dist}(u, v)$.

2.2 Matching and set packing

A matching in a (hyper)graph $G(V, E)$ is a set $M \subseteq E(G)$ of vertex-disjoint edges, that is, every pair of edges in M do not share a common vertex. Let M be a matching in a (hyper)graph G. An edge that belongs to a matching M is called a *matched* edge; otherwise, it is called an *unmatched* edge with respect to M. A vertex v is called *matched* if v is incident to a matched edge. Otherwise, it is an *unmatched* vertex with respect to M. The size $|M|$ of a matching M is the number of edges in M.

Definition 2.1. A matching M in a (hyper)graph G is *maximal* if $M \nsubseteq M'$ for any matching $M' \neq M$ in G. A matching M in G is *maximum* if $|M| \geq |M'|$ for every matching M' in G. A *perfect* matching is a matching in which all vertices of G are matched.

Definition 2.2. The weight of a matching M in an edge-weighted (hyper)graph $G(V, E, w)$ is denoted by $w(M) = \sum_{e \in M} w(e)$. A maximum weighted matching M in a weighted G is a matching with weight $w(M) \geq w(M')$ for every matching M' in G.

A concept related matching is set packing. Given a universe $\mathcal{U}$ and a family $\mathcal{S}$ of subsets of $\mathcal{U}$, a *packing* is a subfamily $\mathcal{C} \subseteq \mathcal{S}$ of sets such that all sets in $\mathcal{C}$ are pairwise disjoint. The size $|\mathcal{C}|$ of a packing $\mathcal{C}$ is the number of sets in $\mathcal{C}$. When every subset of $\mathcal{S}$ is given a real weight, a packing becomes a *weighted* set packing.

Definition 2.3. A packing $\mathcal{C}$ is *maximum* if $|\mathcal{C}| \geq |\mathcal{C}'|$ for every packing $\mathcal{C}'$ in $\mathcal{S}$. A weighted packing $\mathcal{C}$ is *maximum* if $w(\mathcal{C}) \geq w(\mathcal{C}')$ for every weighted packing $\mathcal{C}'$ in $\mathcal{S}$.

2.3 General ridesharing problem definition

We give a formal description on the ridesharing problems studied. In the *ridesharing problems*, we have a centralized ridesharing system (CRS) responsible for assigning ridesharing arrangement - match between individual drivers and passengers. The system receives a set $\mathcal{A} = D \cup R \cup DR$ of ridesharing trip announcements, where $D \cap R = \emptyset$, $D \cap DR = \emptyset$ and $R \cap DR = \emptyset$. Each trip announcement has an individual and some requirements as input parameters describing the itinerary of the individual. Each individual of a trip specifies whether he/she is a driver, passenger, or flexible to perform either role. D is the set of *driver trips*, R is the set of *passenger trips* (both of these trips are also referred to as *fixed trips*) and DR is the set of *flexible trips*. A driver is assumed to operate a vehicle and provide ridesharing service, whereas a passenger is assumed to receive ridesharing service only and agree to share a vehicle with others. For brevity, we usually call a driver trip just a *driver* and a passenger trip a *passenger*. Each trip announcement in $\mathcal{A}$ is indexed by an integer label i, so an integer labeled trip $i \in \mathcal{A}$ may be referred to as a driver, a passenger or a flexible trip. For clarity, a driver trip $i \in D$ is also denoted as η_i and a passenger trip $i \in R$ is also denoted as r_i. The individual of each flexible trip i is assigned as a driver or a passenger in any solution, and when this happens, we just call i a driver η_i or a passenger r_i accordingly.

In addition to a vehicle and individual, each trip i of $\mathcal{A}$ has an origin o_i, a destination d_i, a capacity λ_i of the vehicle, a limit z_i (optional) on the detour distance/time from the preferred path to provide ridesharing service, a set $\mathcal{P}$ of (optional) preferred paths (e.g., shortest paths) to reach the destination, a limit δ_i (optional) on the number of stops a driver wants to make to pick-up/drop-off passengers, an earliest departure time α_i, and a latest arrival time β_i. The parameters $(o_i, d_i, \lambda_i, z_i, \mathcal{P}_i, \delta_i, \alpha_i, \beta_i)$ are summarized in Table 2.1. All parameters related to real world locations are represented in a road network. Specifically, the road network is modeled as a digraph $N(V, E, w)$ with edge-weight function $w : E \to \mathbb{R}$, where the edge weight represents the distance from one endpoint to the other endpoint (tail to head).

Notation	Definition
o_i	The origin (start location) of i (a vertex in N)
d_i	The destination of i (a vertex in N)
λ_i	The number of seats (capacity) of i available for passengers
z_i	The detour limit i willing to make for offering services
$\mathcal{P}_i$	The set of preferred paths of i from o_i to d_i in N
δ_i	The maximum number of stops i willing to make to pick-up/drop-off passengers
α_i	The earliest departure time of i
β_i	The latest arrival time of i

Table 2.1: Parameters for a trip $i \in \mathcal{A}$; $\lambda_i, z_i, \mathcal{P}_i, \delta_i$ for i in $D \cup DR$ only.

We define a serve relation between a trip $i \in (D \cup DR)$ and a set $J \subseteq (R \cup DR)$ of trips with $i \notin J$, which is a central theme for solving the ridesharing problems we consider. The set $\sigma(i) = \{i\} \cup J$ is a *feasible serve relation* if the route (a path in road network N) used by trip i to serve all trips of $\sigma(i)$ satisfies the requirements/constraints specified by the parameters of the trips collectively as listed below:

1. Ridesharing route constraint: there is a path $P(i, J) = (o_i, \ldots, d_i)$ in N (starts at o_i and ends at d_i) visiting the origin o_j (for pick-up) the destination d_j (for drop-off) for each trip $r_j \in J$ subject to the origin o_j visited before the destination d_j.

2. Capacity constraint: the number of trips i can serve, $0 \leq |J| \leq \lambda_i$.

3. Detour constraint: $\text{dist}(P(i, J)) \leq z_i + \text{dist}(P)$ for some path $P \in \mathcal{P}_i$ (if $\mathcal{P}_i = \emptyset$, P is a shortest path from o_i to d_i in N computed by the system).

4. Stop frequency constraint: the number of stops i needs to make to pick-up or drop-off (or both) passengers is at most δ_i (this can be different from λ_i if some origins/destinations of J have the same location).

5. Travel time constraint: each trip $j \in \sigma(i)$ (driver/passenger) departs from o_j no earlier than α_j and arrives at d_j no later than β_j.

We say that driver η_i *serves* itself and passengers of J for a feasible serve relation $\sigma(i) = \{i\} \cup J$ in a solution. Note that J can be empty, namely, $\sigma(i) = \{i\}$, and in this case, η_i serves itself only. There is a re-take passenger model in previous studies: After a driver η_i serves (picks-up, transports and drops-off) a passenger, η_i's available vehicle capacity is increased by one and can be used to serve other passengers; as a result, driver η_i may serve more than λ_i passengers. In this thesis, we study the ridesharing problems without the re-take passenger model, so η_i can serve at most λ_i passengers. We represent the feasible serve relation between every pair of trips in $\mathcal{A}$ by a digraph $G_R(V, E)$, we call ***serve relation graph***:

- Each trip i is represented by a vertex in $V(G_R)$. There is an edge (i, j) in $E(G_R)$ if trip i can serve trip j, that is, if $\sigma(i) = \{i, j\}$ is a feasible serve relation.

Chapter 3

Ridesharing Minimization Problems

In this chapter, we study two ridesharing minimization problems, formally defined as follows:

Ridesharing Problem One (denoted as **RSOne**) Given an instance $(N, \mathcal{A})$ of RSOne where $\mathcal{A} = DR$ is the set of ridesharing trip announcements and the parameters of trips of $\mathcal{A}$ are defined on the road network N, minimize the number of individuals in DR assigned as drivers to serve all trips of $\mathcal{A}$.

Ridesharing Problem Two (denoted as **RSTwo**) Given an instance $(N, \mathcal{A})$ of RSTwo where $\mathcal{A} = DR$ is the set of ridesharing trip announcements and the parameters of trips of $\mathcal{A}$ are defined on the road network N, minimize the total travel distance of individuals in DR assigned as drivers to serve all trips of $\mathcal{A}$.

RSOne and RSTwo are special cases of more general minimization problems RSOne* and RSTwo*. We obtained NP-hardness results for both RSOne and RSTwo, and these results apply to the more general problems RSOne* and RSTwo*, respectively. RSOne* and RSTwo* are defined below:

RSOne* Given an instance $(N, \mathcal{A})$ of RSOne where $\mathcal{A} = D \cup R \cup DR$ is the set of ridesharing trip announcements and the parameters of trips of $\mathcal{A}$ are defined on the road network N, minimize the number of individuals in $D \cup DR$ assigned as drivers to serve all trips of $\mathcal{A}$, assuming all trips can be served.

RSTwo* Given an instance $(N, \mathcal{A})$ of RSTwo where $\mathcal{A} = D \cup R \cup DR$ is the set of ridesharing trip announcements and the parameters of trips of $\mathcal{A}$ are defined on the road network N, minimize the total travel distance of individuals in $D \cup DR$ assigned as drivers to serve all trips of $\mathcal{A}$, assuming all trips can be served.

We denote a partial solution for an instance $(N, \mathcal{A})$ of RSOne or RSTwo by (S, σ), where $S \subseteq DR$ is the set of trips assigned as drivers and σ is a mapping $\sigma : S \to 2^{DR}$ such that:

1. for each $\eta_i \in S$, all trips in $\sigma(i)$ can be served by η_i,

2. for each pair $\eta_i, \eta_j \in S$ with $\eta_i \neq \eta_j$, $\sigma(i) \cap \sigma(j) = \emptyset$, and

3. $\bigcup_{\eta_i \in S} \sigma(i) \subseteq DR$.

For a subset $S' \subseteq S$, let $\sigma(S') = \cup_{\eta_i \in S'} \sigma(i)$. For a partial solution (S, σ), if $\sigma(S') = DR$, (S, σ) is called a solution. Given a (partial) solution (S, σ) of an instance $(N, \mathcal{A})$, for every driver $\eta_i \in S$, let $\text{dist}(i)$ be the travel distance of i for serving $\sigma(i)$ and $\text{dist}(S) = \sum_{\eta_i \in S} \text{dist}(i)$ be the total travel distance of S. For a (partial) solution (S, σ), we sometimes use S to denote a (partial) solution when σ is clear from the context or not related to the discussion.

The ridesharing problems (RSOne and RSTwo) are complex because each trip of $\mathcal{A}$ has many parameters. We want to first investigate the following question:

> *Is there any variant of the ridesharing problem that can be solved in polynomial time?*

The second question we want to investigate is:

> *Do there exist fast/efficient (practical enough) approximation algorithms for some variants of the ridesharing problem?*

To answer the first question, we analyze the relations between the time complexity of the ridesharing problems and their parameters; and this is one of our major contributions, presented in Section 3.2. Before presenting our results, we first discuss some ridesharing related work that is on the methodology side in general.

3.1 Related work

Most results in the literature consider the ridesharing problem in which the centralized ridesharing system (CRS) receives a set of driver trips and a set of passenger trips, without flexible trips (please see reviews [2, 85, 104]). When the set of drivers is given, the ridesharing problem is very similar to the Dial-a-Ride problem (DARP) [28]. In DARP, the set of drivers are part of the service (e.g., taxi drivers), so they have less restrictions regarding to the ridesharing route and travel time constraints. In contrast, a driver in ridesharing can only provide service to passengers who have similar route and time schedules as the drivers'. Further, the set of passengers (customers) given in DARP must be served, whereas not all passengers in the ridesharing problem are required to be served. There are less results for the ridesharing problem when the CRS receives a set of flexible trips. In real-life or practical setting, flexible trips mostly imply carpooling.

There are different objectives (or optimization goals). Most objectives in the ridesharing problem can be classified into two main categories: *operational* objectives and *quality-related* objectives [85]. Operational objectives are usually global optimizing goals that the CRS should achieve as a whole. For example,

- Maximize the number of trips served.
- Minimize the number of vehicles (drivers) needed to serve all trips.
- Minimize the total travel time of used vehicles.
- Minimize the total travel distance of used vehicles.

Quality-related objectives focus on the performance from the individual (passenger/driver) perspective, such as under the first-come first-serve policy. For example:

- Minimize the waiting time of the passengers.
- Minimize the travel time of passengers' trips.
- Maximize the cost saving of the passengers and drivers (or profit of the drivers).

For further literature reviews on ridesharing, readers are referred to [2, 39, 81, 85, 104, 111].

3.1.1 Static and dynamic ridesharing

Ridesharing is categorized into *static* and *dynamic* ridesharing. In static ridesharing, all ridesharing trip announcements are known in advance prior to the execution of a matching process, and ridesharing arrangements are computed in advance such that once the ridesharing arrangements are settled, no further change will be made. In dynamic ridesharing, each trip announcement arrives online, and the ridesharing arrangements are made in real time and should be offered to users quickly. Driver and passenger trip announcements leave the system when a ride-share arrangement has been planned and accepted, or when trip announcements do not receive an offer within their time constraints. In general, there are four ways to handle dynamic ridesharing.

1. A dynamic ridesharing instance can be viewed as a sequence of static ridesharing instances (computing a solution for each static instance for a fixed time interval). This is our method of choice, which is common in the literature as mentioned later. Most algorithms for static ridesharing can be extended to dynamic ridesharing. In this case, the performances of the algorithms designed for static ridesharing are only guaranteed for the static instance occurs in each time interval (the overall performances are not guaranteed). However, it is a good approximation for dynamic ridesharing and can be re-optimized as discussed in the second way below.

2. To overcome the above challenge, one can use *rolling horizon* (e.g., [1, 86, 89]). The current solution is first computed based on known information, but decisions are not finalized until a predefined deadline is reached. Then, the current solution is re-computed at regularly-spaced time points if there are new trip announcements and before the deadline has reached. As a result, each static ridesharing instance is re-optimized with some degrees of freedom, such as how frequent it should be

re-optimized. An extension of this approach is that while a driver is delivering the assigned passengers, such a driver can still accept incoming new passengers as long as the constraints of the current driver and passengers are not violated.

3. The third approach is to anticipate future trip announcements and compute a solution based on the know information and estimated trip announcements. The uncertainty of trip announcements and travel time are dealt with by using *stochastic combinatorial optimization* or *multi-scenario approach* in general [85, 92].

4. The fourth approach is to design online algorithms. Passengers are matched with drivers when requests enter into the CRS in sequence. The online decisions of such ridesharing arrangements need to decide between reactiveness and decision quality. A problem related to ridesharing is the (online) *k-taxi/k-server* problem [35]. Sometimes it is called trip-vehicle assignment problem for the offline/static version. There are approximation algorithms (with reasonable competitive ratios) for both problems (e.g., [26, 75]).

3.1.2 Computational complexity of the ridesharing problem

With re-taking passengers, the ridesharing problem (as well as DARP) is a generalization of the vehicle routing problem (VRP) [19]. The vehicle routing problem is NP-hard even for single vehicle routing problem since it is a generalization of the travelling salesman problem (TSP) [66]. DARP and VRP are route-finding centric (which is to find a route to visit every location), whereas the ridesharing problem focus on the assignment of passengers to drivers. Exact solutions for the ridesharing problem exist, but only for simplified ridesharing problems or small instances (e.g., [13, 27]). In our earlier work on the ridesharing problem [43, 68], we started a line of research that provides insight into the complexity of the ridesharing problem relating to its parameters. The details are presented in Section 3.2. In short, most simplified variants of the ridesharing problem remain NP-hard. A notable exception is when the vehicle capacity is one and without re-taking passengers, as described in the next section.

3.1.3 Single driver, single passenger arrangements

The simplest form of this type of ridesharing is single driver - single passenger arrangements with fixed-role trips $\mathcal{A} = D \cup R$. This means that a driver serves at most one passenger at a time, that is, $\lambda_i \leq 1$ for every trip $\eta_i \in D$. An optimal solution can be found in polynomial time, using graph matching, for the ridesharing problem with maximizing the number of passengers served. The feasible serve relation of all trips in $\mathcal{A}$ can be represented by a weighted bipartite graph $G(U, V, E, w)$, where $U(G) = D$ and $V(G) = R$. There is an edge $e = \{i, j\} \in E(G)$ if driver η_i can serve passenger r_j; and the weight $w(e) \geq 0$ represents the distance travel for driver η_i to serve r_j. Finding a maximum cardinality bipartite matching in

G solves the ridesharing problem and can be done in polynomial time (e.g., [54]). Further, we can find a matching that maximizes the number of passengers served with minimum travel distance of the drivers. Specifically, among all maximum cardinality matchings in G, find the one with minimum weight. This can be done by finding a minimum cost maximum flow on G, with a source node connected to each of $U(G)$ and a sink node connected from each of $V(G)$. Finding a minimum cost maximum flow can be done in polynomial time [3].

If $\mathcal{A}$ contains flexible trips ($DR \neq \emptyset$), it becomes slightly harder. Agatz et al. [1] considered both cases (with and without flexible trips) and solved them by a matching approach. The graph $G(V, E, w)$ representing the feasible serve relation of all trips become a general graph instead of a bipartite graph since each trip in DR can be served by or serve others. A maximum cardinality matching in the general graph G solves the problem of maximizing the number of passengers served, which can done in polynomial time [83]. Similarly if $\mathcal{A} = DR$, a maximum cardinality matching in G solves the ridesharing problem RSOne. To solve the ridesharing problem RSTwo, a weighted graph $G'(V, E)$ constructed based on G is required. The construction of G' is detailed in ([68], Section 4.2). The construction always ensures that G' emits a perfect matching; and a minimum weight perfect matching solves RSTwo.

Lloret-Batlle et al. [73] considered the problem of passenger with guaranteed ride-back ridesharing, namely, a passenger r_i is considered in a successful assignment only if morning and evening rideshare matches of r_i are found. The idea is basically mirroring the weighted bipartite graph $G(U, V, E, w)$ and connecting the two parts by the passenger vertex set $V(G)$. Then find a (minimum cost) maximum flow on the modified graph.

One advantage of this matching approach is that approximation algorithms for the ridesharing problem can be directly derived from existing bipartite matching approximation algorithms. Online algorithms for bipartite graph matching also already exist.

3.1.4 Single driver, multiple passengers arrangements

This is the most commonly studied variant since it is more practical, and our research also focuses on this variant. A general approach for solving the ridesharing problem in the literature usually uses an Integer Programming (IP) or a Mixed Integer Programming (MIP) formulation and solves it by an exact method or heuristics. The biggest obstacle in using IP/MIP for the ridesharing problem is the large number of constraints/variables required. Thus, our methods focus more on discrete algorithms.

Flexible trips

Tamannaei and Irandoost [105] considered a variant of the home-to-work carpooling variant; all trips have the same destination (a work place). Their model simultaneously minimizes the costs of travel times, the vehicle use, and the vehicle delays (an example of multi-criteria

objective function). They gave a MIP exact formulation and proposed a refined branch-and-bound method to solve it. Armant and Brown [9] considered a similar variant, except many trips share a same origin or destination. They gave a MIP exact formulation and used a heuristic to solve it.

Kutiel and Rawitz [65] studied a special case of the ridesharing problem, called the *maximum carpool matching problem* (MCMP). An instance of the maximum carpool matching problem consists of a digraph $\vec{G}(V, E)$ and a capacity function $c : V \to \mathbb{Z}_{\geq 0}$, where the vertices of V represent the individuals and there is an edge $(u, v) \in E$ if v can serve u (the edge direction is inverse of the serve relation graph described in Section 2.3). Every $v \in V$ is a flexible-role trip, namely, v can be assigned as a driver or passenger. The goal of the MCMP is to find a set of drivers $S \subseteq V$ to serve all V such that the number of passengers is maximized. It was shown that the MCMP is NP-hard [50]. A $\frac{1}{2}$-approximation algorithm is proposed in [65]. However, such an algorithm cannot apply directly to the ridesharing minimization problem studied in this thesis. We discuss more about this in Section 3.4 in which we show how to modify the $\frac{1}{2}$-approximation algorithm so that it can apply to ridesharing problem RSOne.

Fixed trips

Baldacci et al. [13] considered a home-to-work commute variant in which all trips share the same destination; and the carpool problem considered is similar to DARP studied in [27], except the drivers are located in different locations initially. Two integer programming formulations (three-index variable and set-partitioning formulations) are given in [13], and a column generation approach is proposed. Cordeau [27] proposed a three-index MIP formulation for DARP based on a complete digraph $G_c(V, E)$. As an example what three-index means in the context of ridesharing, we give a brief discussion of the MIP formulation. The vertex set $V(G_c)$ is partitioned into three subsets $P = \{1, \ldots, n\}, D = \{n + 1, \ldots, 2n\}$ and $\{0, 2n+1\}$, where P and D represent the origins and destinations of the passengers, respectively, and $\{0, 2n + 1\}$ represent the origin and destination depots (the locations where the vehicles/drivers should departure and arrive), respectively. Each passenger i is associated with an origin node i and a destination node $n + i$. Each edge (i, j) is usually associated with a cost (travel cost and/or travel time). Let K be the set of vehicles (and drivers). For each edge $(i, j) \in E(G_c)$ and vehicle $k \in K$, the three-index binary variable $x_{ij}^k = 1$ in the formulation means vehicle k travels from node i to node j. Thus, the idea is to have a set of vehicles $K' \subseteq K$ to traverse from 0 to $2n + 1$ to cover all vertices $P \cup D$ exactly once such that the paths of vehicles k and k' are vertex disjoint in G_c for every pair k and k' in K'. Some optimization goal should be achieved (usually minimize the overall costs associated with the paths). Of course, the constraints in the MIP formulation also ensure that the traversal of each path satisfies the time requirements of the passengers. The same kind of mathematical formulation for DARP can be applied to the ridesharing problem;

however, as can be seen, the number of variables is large, and the number of constraints is even larger for the ridesharing problem. Ropke et al. [93] gave a two-index formulation of the same problem and showed that it has a better performance than the three-index formulation. Liu et al. [71] proposed a branch-and-cut algorithm to solve another variant of DARP, which consists of multiple trips and request types and a heterogeneous fleet of vehicles with configurable capacity and manpower planning.

Herbawi and Weber [53] gave an MIP formulation for a variant of dynamic ridesharing. Their method divides the day into a set of time periods, which is an example of viewing the dynamic ridesharing instances as a sequence of static ridesharing instances. Their MIP formulation actually has a multi-criteria objective function that tries to minimize the total travel distance and time of the vehicles and the total travel time of passengers' trips and maximizes the number of matched passengers. Multiple objectives are simultaneously achieved by optimizing a linear combination of them. Other examples of multi-criteria objective function can be found in [13, 15]. Di Febbraro et al. [33] considered a variant of dynamic ridesharing that optimizes the desired departure time of the users (a quality-related objective) while satisfying as many requests as possible. They gave a MIP formulation and proposed a strategy for re-optimizing the solution of the MIP using rolling horizon: the arrival of a new use triggers this optimization for handling new trip requests.

Stiglic et al. [101] considered the ridesharing problem where a set of pick-up and a set of drop-off meeting points are given. Passenger can be picked-up at either their origins or the pick-up meeting points and can be dropped-off at either their destinations or drop-off meeting points. A weighted graph is constructed to represent all feasible matches. A *feasible match* consists of a driver, a set of passengers and an optional meeting point such that the set of passengers can be served by the driver using the optional meeting point. An Integer Linear Programming (ILP) is given in [101], based on the feasible matches, which resembles a graph matching problem in a hypergraph. This approach is almost identical to approach in [5] (RTV-graph).

Taxi-sharing also falls under this form of ridesharing and has attracted attention in the literature. Qian et al. [90] proposed a taxi group ride (TGR) system. Passengers, who have similar origins, destinations and travel time, are grouped together. If a group of passengers is assigned a taxi driver for service, all passengers of that group gather at a predefined location (within walking distance of their origins). Then, they are picked-up by the assigned driver and dropped-off at a predefined location. The TGR model is somewhat similar to the meeting points idea suggested by Stiglic et al. [101].

Hosni et al. [55] gave a three-index MIP formulation for the static shared-taxi problem and briefly compared to [27]. IN [55], Lagrangian relaxation is applied to the MIP formulation to get subproblems, each of which is a single-taxi problem that can be solved more efficiently coupled with their heuristic to correct each solution to the subproblem. Huang et al. [59] proposed a branch-and-bound algorithm and an MIP formulation for solving large-

scale real-time ridesharing. The use case of ridesharing is focused on taxi-sharing in [59], and the objective is to minimize the travel time of the drivers. They also proposed a kinetic tree algorithm to build a dynamic scheduling for real-time trip requests. The kinetic tree is a data structure that maintains all the valid trip schedules with respect to each driver and can be used to speed-up computation.

Santos and Xavier [97] also suggested dividing the day into time periods, after which a deterministic (static) instance of the problem can be generated and solved by a greedy randomized adaptive search procedure (GRASP). The basic idea of their GRASP is that after a randomized greedy solution has been found, a local search is performed to improve the solution. Jung et al. [61] applied hybrid-simulated annealing (HSA) to dynamically assign passenger requests to shared taxis. Their HSA algorithm follows the skeleton of standard simulated annealing (SA). As a different kind of heuristics, Ma et al. [78] proposed a taxi-sharing system that uses a mobile-cloud architecture. It is a spatio-temporal indexing structure that searches and schedules taxis to fulfill real-time rideshare requests (similar to *nearest vehicle dispatch* in [61]). The taxi-sharing system first uses a search method, based on a spatio-temporal index, to find candidate taxis that can serve the requesting passenger. Then, the taxi with the shortest detour is selected through a scheduling process.

Alonso-Mora et al. [5] proposed an exact algorithm using matching from *RV graph*, proposed by Santi et al. [95], to solve a taxi-sharing problem given a set of drivers (taxi drivers). Each vertex in the RV graph represents a trip request, and there is an edge between $\lambda \geq 1$ trip requests if all λ trips can be served together. The RV graph becomes a hypergraph when $\lambda \geq 3$. In [5], the authors expended the idea of RV graph to construct another graph, called RTV-graph: a taxi vehicle is also represented by a vertex v and the trips can be served together are represented by a vertex T, and there is an edge (T, v) in the RTV graph if v can serve all of T. After the RTV-graph is constructed, they gave an ILP based on the RTV-graph, in which maximum user waiting times and maximum additional delays due to sharing a ride are considered. This approach is almost identical to the matching approach in [101].

The RV/RTV-graph is an extension to our feasible serve relation model in the sense that the RV graph can model multiple trips (passengers) being served by a driver, represented by using a single edge. We also use this ILP and matching approach for the ridesharing problem variant studied in Chapter 4, except our graph representation uses less number of vertices. This matching approach is better than traditional IP/MIP approach due the number of constraints in the ILP is significantly reduced. Most papers mentioned above include numerical experiments to evaluate their methods and algorithms. However, most of them can only handle small instances (less than 250 trips or use small graphs) in reasonable time (within 15 minutes). The ones that can handle larger instances (close to 1000 trips) use heuristics, but the performances of such heuristics are much weaker, compared to the exact methods proposed in their respective papers. In some papers, the data required in

their experiments are pre-computed, such as shortest paths and their distances. Such pre-computation hides the realistic aspect of the application. We use a more realistic setup for our large-scale numerical experiments, presented in Chapter 4.

Multihop ridesharing

There is another variant of ridesharing, called multihop ridesharing. However, it is not very common in practice. In this variant, drivers usually have limited number of stops for pick-ups and drop-offs. A passenger can transfer between vehicles during his/her entire trip. In other words, multiple drivers can be assigned to a passenger r_i and each of the drivers serve a portion of r_i's trip.

3.2 NP-hardness results

Because previous works mostly focused on empirical studies of the ridesharing problem, we started a line of research that explores the hardness of the ridesharing problem. In particular, we introduced an algorithmic analysis for understanding the hardness/time complexity of the ridesharing problem related to its parameters of origin, destination, vehicle capacity, detour distance limit, and preferred paths [68]. The model was further elaborated in [43]. In this thesis, we extend the analysis model to include parameters of maximum number of stops, departure time and arrival time [47, 48]. More precisely, the complexity analysis on the ridesharing problem is based on the following conditions for the parameters (briefly introduced in Chapter 1):

C1. All trips have the same destination or all trips have the same origin, that is, $d_i = d_j$ for every pair of trip $i, j \in \mathcal{A}$ or $o_i = o_j$ for every pair of trip $i, j \in \mathcal{A}$.

C2. The individual of each trip can only serve others who are on the individual's preferred path (without any detour), that is, $z_i = 0$ for every $i \in \mathcal{A}$.

C3. There is only one preferred path $\mathcal{P}_i = P_i$ for each trip, that is $|\mathcal{P}_i| \leq 1$ for every $i \in \mathcal{A}$. (note: if $\mathcal{P}_i = \emptyset$, a shortest path from o_i to d_i is computed by the CRS.)

C4. Each individual is willing to make at least $\delta_i \geq \lambda_i$ stops to either pick-up or drop-off passengers (or both), that is, $\delta_i \geq \lambda_i$ for pick-ups and/or drop-offs for every $i \in \mathcal{A}$.

C5. All trips have the same earliest departure time and same latest arrival time, that is, for every $i \in \mathcal{A}$, $\alpha_i = \alpha$ and $\beta_i = \beta$ for some time constants $\alpha < \beta$.

Let us denote C1 as **location condition**, C2 as **detour condition**, C3 as **preferred path condition**, C4 as **stop frequency condition**, and C5 as **travel time condition**. Our main complexity analysis results in this thesis are: Given an instance $(N, \mathcal{A})$ of the ridesharing problem RSOne or RSTwo, (1) if $(N, \mathcal{A})$ does not satisfy any one of the five

conditions, both RSOne and RSTwo are NP-hard; and (2) if $(N, \mathcal{A})$ does not satisfy one of Conditions C2-C5, both RSOne and RSTwo are NP-hard to approximate within a constant factor. The analysis is based on a reduction from the 3-partition problem. The decision problem of 3-partition is that given a set $A = \{a_1, a_2, ..., a_{3l}\}$ of $3l$ positive integers, where $l \geq 2, \sum_{i=1}^{3l} a_i = lK$ and $K/4 < a_i < K/2$, whether A can be partitioned into l disjoint subsets $A_1, A_2, \ldots, A_l$ such that each subset has three elements of A and the sum of integers in each subset is K.

3.2.1 NP-hardness results for C4 and C5

We first present the NP-hardness results for the stop frequency condition, that is, when C1-C3 and C5 are satisfied but C4 is not. Then, we show the NP-hardness results for the travel time condition, that is, when C1-C4 are satisfied but C5 is not. For both cases, we assume all trips have the same destination but not the same origin (since it is symmetric to prove the case that all trips have the same origin, and we show it is the case for C2 and C3 in Subsection 3.2.3). The NP-hard analysis will provide a base for proving the inapproximability of problems RSOne and RSTwo. The NP-hardness proof is a reduction from the 3-partition problem.

NP-hardness result for the stop frequency condition.

Given a 3-partition instance $A = \{a_1, ..., a_{3l}\}$, construct a ridesharing problem instance $(N, \mathcal{A})$ as follows (also see Figure 3.1).

- The road network is the weighted graph $N(V, E, w)$ with $V(N) = \{\chi, I, u_1, \ldots, u_{3l}, v_1, \ldots, v_l\}$ and $E(G)$ having edges (u_i, v_1) and (v_1, u_i) for $1 \leq i \leq 3l$, edges (v_i, v_{i+1}) and (v_{i+1}, v_i) for $1 \leq i \leq l-1$, (v_l, χ) and (χ, v_l). Each edge $(u, v) \in E(N)$ has weight of one, representing the travel distance from u to v. It takes $l + 1$ units of distance travelling from u_i to χ for $1 \leq i \leq 3l$.
- $\mathcal{A} = \{1, \ldots, 3l + lK\}$ has $3l + lK$ trips. Let α and β be valid constants representing time such that $\alpha < \beta$.
  - Each trip i, $1 \leq i \leq 3l$, has origin $o_i = u_i$, destination $d_i = \chi$, capacity $\lambda_i = a_i$, detour distance limit $z_i = 0$, stop limit $\delta_i = 1$, departure time $\alpha_i = \alpha$ and arrival time $\beta_i = \beta$, and each trip i has a preferred path $(u_i, v_1, v_2, \ldots, v_l, \chi)$ in N.
  - Each trip i, $3l+1 \leq i \leq 3l+lK$, has origin $o_i = v_j$, $j = \lceil \frac{i-3l}{K} \rceil$, destination $d_i = \chi$, capacity $\lambda_i = 0$, detour distance limit $z_i = 0$, stop limit $\delta_i = 0$, departure time $\alpha_i = \alpha$, arrival time $\beta_i = \beta$ and a unique preferred path $(v_j, v_{j+1}, v_{j+2}, \ldots, v_l, \chi)$ in N ($j = \lceil \frac{i-3l}{K} \rceil$).

Lemma 3.1. *Any solution for the instance $(N, \mathcal{A})$ has every trip i, $1 \leq i \leq 3l$, as a driver and total travel distance at least $3l \cdot (l + 1)$.*

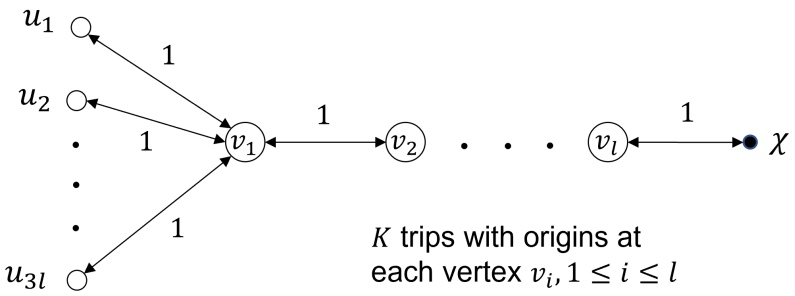


Figure 3.1: Ridesharing instance $(N, \mathcal{A})$ satisfying Conditions C1-C3 and C5, and all trips of $\mathcal{A}$ have the same destination.

Proof. Trip i, $1 \le i \le 3l$, must be assigned as a driver in any solution since no other trip $j \ne i$ can serve i for every trip $j \in \mathcal{A}$ due to detour $z_j = 0$. Let S be the set of $3l$ trips assigned as drivers η_i with $1 \le i \le 3l$. The total travel distance of the drivers in S is $3l \cdot (l+1)$. For each trip j with $3l+1 \le j \le 3l+lK$, the total travel distance of drivers in S and trip j is $3l \cdot (l+1)$ if j is served by a driver in S; otherwise, it is at least $3l \cdot (l+1) + \lceil (j-3l)/K \rceil$ (assuming j is a flexible-role trip and assigned as a driver). Hence, a solution with trips i with $1 \le i \le 3l$ assigned as drivers has total travel distance $3l \cdot (l+1)$, and any additional trip j for $3l+1 \le j \le 3l+lK$ assigned as a driver has total travel distance greater than $3l \cdot (l+1)$. □

Theorem 3.1. *The ridesharing problem RSOne is NP-hard when Conditions C1-C3 and C5 are satisfied, but the stop frequency condition C4 is not.*

Proof. We prove the theorem by showing that an instance $A = \{a_1, \ldots, a_{3l}\}$ of the 3-partition problem has a solution if and only if the ridesharing problem instance $(N, \mathcal{A})$ has a solution of $3l$ drivers.

Assume that instance A has a solution $A_1, \ldots, A_l$ where the sum of elements in each A_j, $1 \le j \le l$, is K. For each $A_j = \{a_{j_1}, a_{j_2}, a_{j_3}\}$, $1 \le j \le l$, assign the three trips whose $\lambda_i j_1 = a_{j_1}$, $\lambda_i j_2 = a_{j_2}$ and $\lambda_i j_3 = a_{j_3}$ as drivers to serve the K trips with origins at vertex v_j. Hence, we have a solution of $3l$ drivers for $(N, \mathcal{A})$.

Assume that $(N, \mathcal{A})$ has a solution of $3l$ drivers. By Lemma 3.1, every trip i, $1 \le i \le 3l$, is a driver in the solution. Then, each trip j for $3l+1 \le j \le 3l+lK$ must be assigned as a passenger in the solution, total of lK passengers. Since $\sum_{1 \le i \le 3l} a_i = lK$, each driver η_i, $1 \le i \le 3l$, serves exactly $\lambda_i = a_i$ passengers. Since $a_i < K/2$ for every $a_i \in A$, at least three drivers are required to serve the K passengers with origins at each vertex v_j, $1 \le j \le 3l$. Due to $\delta_i = 1$ for $1 \le i \le 3l$, each driver η_i can only serve passengers with the same origin. Therefore, the solution of $3l$ drivers has exactly three drivers j_1, j_2, j_3 to serve the K passengers with origins at each vertex v_j, $1 \le j \le l$, implying $a_{j_1} + a_{j_2} + a_{j_3} = K$. Let $A_j = \{a_{j_1}, a_{j_2}, a_{j_3}\}$, $1 \le j \le l$, we get a solution for the 3-partition instance.

The size of $(N, \mathcal{A})$ is polynomial in l. It takes a polynomial time to convert a solution of $(N, \mathcal{A})$ to a solution of the 3-partition instance and vice versa. □

Theorem 3.2. *The ridesharing problem RSTwo is NP-hard when Conditions C1-C3 and C5 are satisfied, but the stop frequency condition C4 is not.*

Proof. Let d_{sum} be the sum of travel distances of all trips i with $1 \leq i \leq 3l$. Then the total travel distances of drivers in any solution for $(N, \mathcal{A})$ is at least $d_{sum} = 3l \cdot (l + 1)$ by Lemma 3.1. We show that an instance $A = \{a_1, ..., a_{3l}\}$ of the 3-partition problem has a solution if and only if instance $(N, \mathcal{A})$ has a solution with travel distance d_{sum}.

Assume that the 3-partition instance has a solution. Then there is a solution of $3l$ drivers for $(N, \mathcal{A})$ as shown in the proof of Theorem 3.1. The total travel distance of the $3l$ drivers in this solution is d_{sum}.

Assume that $(N, \mathcal{A})$ has a solution with total travel distance d_{sum}. Trips i with $1 \leq i \leq 3l$ must be drivers by Lemma 3.1. From this, there is a solution for the 3-partition instance as shown in the proof of Theorem 3.1. □

NP-hardness result for the travel time condition.

Assume that Conditions C1-C4 are satisfied but C5 is not, that is, trips can have arbitrary departure time and arrival time. The NP-hardness proof is also a reduction from the 3-partition problem, which is similar to Theorem 3.1. Given a 3-partition minimization problem instance, construct a ridesharing instance $(N, \mathcal{A})$ with N shown in Figure 3.1. The only differences are the values of α_i, β_i and δ_i.

- $\mathcal{A} = \{1, \ldots, 3l + lK\}$ has $3l + lK$ trips. Let α be a valid constant representing a particular time of the day. Let ϵ be the time unit required for travelling from one endpoint to another endpoint of any edge in $E(N)$.
- For trips i, $1 \leq i \leq 3l$, origin $o_i = u_i$, destination $d_i = \chi$, capacity $\lambda_i = a_i$, detour distance limit $z_i = 0$, stop limit $\delta_i = \lambda_i$, departure time $\alpha_i = \alpha$, arrival time $\beta_i = \alpha + 2l\epsilon$. Each trip i has a preferred path $(u_i, v_1, v_2, \ldots, v_l, \chi)$ in N.
- For trips i, $3l+1 \leq i \leq 3l+lK$, origin $o_i = v_j$, $j = \lceil \frac{i-3l}{K} \rceil$, destination $d_i = \chi$, capacity $\lambda_i = 0$, detour distance limit $z_i = 0$, stop limit $\delta_i = \lambda_i$. Each trip i has a unique preferred path $(v_j, v_{j+1}, v_{j+2}, \ldots, v_l, \chi\}$ in N, $\alpha_i = \alpha + l\epsilon$ and $\beta_i = \alpha + (2l - j + 1) \cdot \epsilon$, where $j = \lceil \frac{i-3l}{K} \rceil$.

Note that every trip $i \in \mathcal{A}$ has the same travel distance from o_i to d_i as the previous construction in Subsection 3.2.1. Since they have the same graph N, Lemma 3.1 also holds for this ridesharing instance $(N, \mathcal{A})$.

Lemma 3.2. *In any solution (σ, S) for the instance $(N, \mathcal{A})$, all passengers of $\sigma(\eta_i) \setminus \{\eta_i\}$ served by driver $\eta_i \in S$ must have the same origin v_j, for some $j \in [1, \ldots, 3l]$.*

Proof. By Lemma 3.1, every trip i, $1 \leq i \leq 3l$, is assigned as a driver in any solution. Thus, only trips with origin v_j, $1 \leq j \leq l$, can be assigned as passengers. Let j be a trip with origin v_j assigned as a passenger served by a driver $\eta_i \in S$. The travel time from v_j to χ is $\epsilon \cdot (l-j+1)$. Since $\beta_j = \alpha + (2l-j+1) \cdot \epsilon$, trip j must be picked-up no later than time $\alpha + l\epsilon$. Otherwise, j cannot arrive at $d_j = \chi$ by time β_j. From this and the fact that $\alpha_j = \alpha + l\epsilon$, trip j must be picked-up at time $\alpha + l\epsilon$ exactly. The travel time from o_i to $o_j = v_j$ is $j\epsilon \leq l\epsilon$. Driver η_i can arrive at χ (after delivering r_j) no later than time $\alpha + 2l\epsilon = \beta_i$ if η_i leaves o_i at $\alpha = \alpha_i$.

Let j_1 and j_2 be two trips with $o_{j_1} = v_{j_1}$, $o_{j_2} = v_{j_2}$ and $j_1 < j_2$. Then any driver η_i with $1 \leq i \leq 3l$ can serve only one of j_1 and j_2 due to the following reasons. Suppose η_i picks-up j_1 first. As explained above, j_1 must be picked-up at time $\alpha + l\epsilon$ exactly. By the time η_i reaches v_{j_2} after picking-up j_1, it will pass time $\alpha + l\epsilon$. From this and the above, η_i can no longer serve j_2. Otherwise, j_2 will not arrive d_{h_2} on time. Suppose η_i picks-up j_2 first. When η_i reaches v_{j_1} by going back, it will also pass time $\alpha + l\epsilon$. Hence, η_i cannot serve j_1 in this case. Therefore, if η_i decides to serve a trip with origin v_j, the only other trips η_i can serve must also have origin v_j, $j \in [1, \ldots, 3l]$. □

Corollary 3.1. *For any trip $i \in \mathcal{A}$, $1 \leq i \leq 3l$, i will only make at most one stop, effectively making $\delta_i \leq 1$.*

Proof. By Lemma 3.1, every trip i, $1 \leq i \leq 3l$, is assigned as a driver in any solution for $(N, \mathcal{A})$. If η_i does not serve any other passenger, η_i does not make any stop. If η_i serves at least one passenger, η_i makes exactly one stop by Lemma 3.2. □

Theorem 3.3. *The ridesharing problem RSOne is NP-hard when Conditions C1-C4 are satisfied, but the travel time condition C5 is not.*

Proof. We prove the theorem by showing that an instance $A = \{a_1, ..., a_{3l}\}$ of the 3-partition problem has a solution if and only if the ridesharing problem instance $(N, \mathcal{A})$ has a solution of $3l$ drivers.

Assume that instance A has a solution $A_1, \ldots, A_l$ where the sum of elements in each A_j, $1 \leq j \leq l$, is K. For each $A_j = \{a_{j_1}, a_{j_2}, a_{j_3}\}$, $1 \leq j \leq l$, assign the three trips whose $\lambda_i j_1 = a_{j_1}$, $\lambda_i j_2 = a_{j_2}$ and $\lambda_i j_3 = a_{j_3}$ as drivers to serve the K trips with origins at vertex v_j. Hence, we have a solution of $3l$ drivers for $(N, \mathcal{A})$.

Assume that $(N, \mathcal{A})$ has a solution of $3l$ drivers. By Lemma 3.1, every trip i, $1 \leq i \leq 3l$, is assigned as a driver in any solution for $(N, \mathcal{A})$. Similarly, each driver η_i serves exactly $\lambda_i = a_i$ passengers, and at least three drivers are required to serve the K passengers with origins at each vertex v_j, $1 \leq j \leq l$. By Corollary 3.1, each driver η_i, $1 \leq i \leq 3l$, makes at most one stop. Then as shown in the proof of Theorem 3.1, we get a solution for the 3-partition problem instance. □

With a similar argument to that of Theorem 3.2, we have the following lemma.

Theorem 3.4. *The ridesharing problem RSTwo is NP-hard when Conditions C1-C4 are satisfied, but the travel time Condition C5 is not.*

3.2.2 Inapproximability results for each of Conditions C2-C5

Based on the NP-hardness results in the above, Subsection 3.2.1, we extent our reduction to further show that it is NP-hard to approximate both ridesharing problems RSOne and RSTwo within a constant factor if C4 or C5 is not satisfied. We then use a similar analysis approach to show that it is NP-hard to approximate both ridesharing problems RSOne and RSTwo within a constant factor if the detour condition C2 or the preferred path condition C3 is not satisfied. It has been shown that in [43, 68] when either C2 or C3 is not satisfied, both RSOne and RSTwo are NP-hard.

Inapproximability for the stop frequency condition C4.

Let $(N, \mathcal{A})$ be the ridesharing problem instance constructed based on a given 3-partition instance A as described above for Theorem 3.1 in Subsection 3.2.1 (Figure 3.1). We apply the following modification to $(N, \mathcal{A})$ to get a new ridesharing instance $(N, \mathcal{A}')$:

- **Modification:** For every trip i, $1 \leq i \leq 3l$, we multiply λ_i with lK, that is, $\lambda_i = a_i \cdot lK$, where l and K are given in instance A. Instead of K trips, there are now lK^2 trips with origins at vertex v_j for $1 \leq j \leq l$, and all such trips have the same destination, capacity, detour, stop limit, earlier departure time, latest arrival time, and preferred path as before.

The size of $(N, \mathcal{A}')$ is polynomial in l and K. Note that Theorem 3.1 and Theorem 3.2 hold for $(N, \mathcal{A}')$ and $\sum_{i=1}^{3l} \lambda_i = lK \sum_{i=1}^{3l} a_i = (lK)^2$.

Lemma 3.3. *Let $(N, \mathcal{A}')$ be a ridesharing problem instance constructed above from a 3-partition problem instance $A = \{a_1, \ldots, a_{3l}\}$. The 3-partition problem instance A has a solution if and only if the ridesharing problem instance $(N, \mathcal{A}')$ has a solution (σ, S) such that $3l \leq |S| < 3l + lK$, where S is the set of drivers.*

Proof. Assume that instance A has a solution $A_1, \ldots, A_l$ where the sum of elements in each A_j, $1 \leq j \leq l$, is K. For each $A_j = \{a_{j_1}, a_{j_2}, a_{j_3}\}$, $1 \leq j \leq l$, assign the three trips whose $\lambda_i j_1 = a_{j_1} \cdot lK$, $\lambda_i j_2 = a_{j_2} \cdot lK$ and $\lambda_i j_3 = a_{j_3} \cdot lK$ as drivers to serve the lK^2 trips with origins at vertex v_j. Hence, we have a solution of $3l$ drivers for $(N, \mathcal{A}')$.

Assume that $(N, \mathcal{A}')$ has a solution with $3l \leq |S| < 3l + lK$ drivers. Let $\mathcal{A}'(1, 3l)$ be the set of trips in $\mathcal{A}'$ with labels from 1 to $3l$. By Lemma 3.1, every trip $i \in \mathcal{A}'(1, 3l)$ is a driver in S. Since $a_i < K/2$ for every $a_i \in A$, $\lambda_i < lK \cdot K/2$ for every trip $i \in \mathcal{A}'(1, 3l)$. From this, it requires at least three drivers in $\mathcal{A}'(1, 3l)$ to serve the lK^2 trips with origins at each vertex v_j, $1 \leq j \leq l$. For every trip $i \in \mathcal{A}'(1, 3l)$, driver η_i can only serve passengers with the same origin due to $\delta_i = 1$. There are two cases: (1) $|S| = 3l$ and (2) $3l < |S| < 3l + lK$.

(1) It follows from the proof of Theorem 3.1 that every three drivers j_1, j_2, j_3 of the $3l$ drivers serve exactly lK^2 passengers with origins at vertex v_j,$1 \le j \le l$. Then similarly, let $A_j = \{a_{j_1}, a_{j_2}, a_{j_3}\}$, $1 \le j \le l$, we get a solution for the 3-partition problem instance.

(2) For every vertex v_j in N, let X_j be the set of trips with origin at v_j not served by drivers in $\mathcal{A}'(1, 3l)$. Since $|S| < 3l+lK$ and all of $\mathcal{A}'(1, 3l)$ are drivers in S, at most $lK-1$ trips in $\mathcal{A}' \setminus \mathcal{A}'(1, 3l)$ can be assigned as drivers in S. Thus, $0 \le |X_j| < lK$ for any v_j. For every trip $i \in \mathcal{A}'(1, 3l)$, $\lambda_i = a_i \cdot lK$ is a multiple of lK. Hence, the sum of capacity for any subset of trips in $\mathcal{A}'(1, 3l)$ is also a multiple of lK, and further, $\lambda_i + \lambda_i i' = (a_i + a_{i'}) \cdot lK < lK \cdot (K-1)$ for every $i, i' \in \mathcal{A}'(1, 3l)$ because $a_i < K/2$ and $a_{i'} < K/2$. From these, $|X_j| < lK$ and each trip in X_j can serve only itself, there are 3 drivers $\{j_1, j_2, j_3\} \subseteq \mathcal{A}'(1, 3l)$ to serve all lK^2 trips with origin at v_j. Note that $\lambda_i j_1 + \lambda_i j_2 + \lambda_i j_3 \ge lK^2$ as each λ_i is a multiple of lK and $|X_j| < lK$. Since $\lambda_i j_1 + \lambda_i j_2 + \lambda_i j_3 \ge lK^2$ for every $1 \le j \le l$ and $\sum_{1 \le i \le 3l} \lambda_i = (lK)^2$, $\lambda_i j_1 + \lambda_i j_2 + \lambda_i j_3 = lK^2$ for every j. Thus, we get a solution with $A_j = \{a_{j_1}, a_{j_2}, a_{j_3}\}$, $1 \le j \le l$, for the 3-partition problem.

It takes a polynomial time to convert a solution of $(N, \mathcal{A}')$ to a solution of the 3-partition instance and vice versa. □

Theorem 3.5. *Let $(N, \mathcal{A}')$ be the ridesharing instance stated above based on a 3-partition instance. Approximating the minimum number of drivers for $(N, \mathcal{A}')$ within a constant factor is NP-hard. This implies the ridesharing problem RSOne cannot be approximated within a constant factor (unless $P = NP$) when Conditions C1-C3 and C5 are satisfied, but the stop frequency condition C4 is not.*

Proof. Assume that there is a polynomial-time c-approximation algorithm C for instance $(N, \mathcal{A}')$ for any constant $c > 1$. This means that C will output a solution (σ_C, S_C) for $(N, \mathcal{A}')$ such that $\mathrm{OPT}(\mathcal{A}') \le |S_C| \le c \cdot \mathrm{OPT}(\mathcal{A}')$, where $\mathrm{OPT}(\mathcal{A}')$ is the minimum number of drivers for $(N, \mathcal{A}')$. When the 3-partition instance is a "No" instance, the optimal value for $(N, \mathcal{A}')$ is $\mathrm{OPT}(R') \ge 3l + lK$ by Lemma 3.3. Hence, algorithm C must output a value $|S_C| \ge 3l + lK$. When the 3-partition instance is a "Yes" instance, the optimal value for $(N, \mathcal{A}')$ is $\mathrm{OPT}(\mathcal{A}') = 3l$. For any constant $c > 1$, taking K such that $c < K/3 + 1$. The output $|S_C|$ from algorithm C on $(N, \mathcal{A}')$ is $3l \le |S_C| \le 3lc < 3l + lK$ for a 3-partition "Yes" instance. Therefore, by running the c-approximation algorithm C on any ridesharing instance $(N, \mathcal{A}')$ and checking the output value $|S_C|$ of C, we can answer the 3-partition problem in polynomial time, which contradicts that the 3-partition problem is NP-hard unless $P = NP$. □

Theorem 3.6. *The ridesharing problem RSTwo cannot be approximated within a constant factor (unless $P = NP$) when Conditions C1-C3 and C5 are satisfied, but the stop frequency condition C4 is not.*

Proof. Let $(N, \mathcal{A}')$ be the ridesharing problem instance stated above (same as in Theorem 3.5), based on a given 3-partition instance $A = \{a_1, ..., a_{3l}\}$. Let $d_{sum}(S)$ be the sum of travel distances for a set S of drivers. Let $\mathcal{A}'(1, 3l)$ be the set of trips in $\mathcal{A}'$ with labels from 1 to $3l$. By Lemma 3.1, all of $\mathcal{A}'(1, 3l)$ must be drivers in any solution for $(N, \mathcal{A}')$ and $d_{sum}(\mathcal{A}'(1, 3l)) = 3l \cdot (l + 1)$. Assume that there is a polynomial-time c-approximation algorithm C for the ridesharing problem $(N, \mathcal{A}')$ for any constant $c > 1$. This means that C will output a solution (σ_C, S_C) for $(N, \mathcal{A}')$ such that $\text{OPT}(\mathcal{A}') \leq d_{sum}(S_C) \leq c \cdot \text{OPT}(\mathcal{A}')$, where $\text{OPT}(\mathcal{A}')$ is the minimum total travel distance of drivers for $(N, \mathcal{A}')$. By Lemma 3.3, when the 3-partition instance is a "No" instance, the number of drivers in any solution for $(N, \mathcal{A}')$ is at least $3l + lK$. Further, lK drivers of the $3l + lK$ drivers must have origins at some vertices v_j, $1 \leq j \leq l$, so $d_{sun}(S_C) \geq 3l \cdot (l + 1) + lK$. When the 3-partition instance is a "Yes" instance, the optimal value for $(N, \mathcal{A}')$ is $\text{OPT}(\mathcal{A}') = 3l \cdot (l + 1)$. For any constant $c > 1$, taking K and l such that $c < \frac{K}{3(l+1)} + 1$. The output $d(S_C)$ from algorithm C on $(N, \mathcal{A}')$ is $3l \cdot (l + 1) \leq d_{sum}(S_C) \leq 3l \cdot (l + 1) \cdot c < 3l \cdot (l + 1) + lK$ for a 3-partition "Yes" instance. Therefore, by running the c-approximation algorithm C on any ridesharing instance $(N, \mathcal{A}')$ and checking the output value $d(S_C)$ of C, we can answer the 3-partition problem in polynomial time, which contradicts that the 3-partition problem is NP-hard unless $P = NP$. □

Inapproximability for the travel time condition C5.

It is NP-hard to approximate both ridesharing problems RSOne and RSTwo within a constant factor when Conditions C1-C4 are satisfied, but the travel time condition C5 is not. The proofs are identical to the inapproximability proof of Theorem 3.5 and Theorem 3.6 for each problem, respectively.

Let $(N, \mathcal{A})$ be the ridesharing problem instance constructed based on a given 3-partition instance as described in the Subsection 3.2.1 (NP-hardness result for the travel time condition). Then apply the Modification, described in the above Subsection 3.2.2 (Inapproximability for the stop frequency condition), to $(N, \mathcal{A})$ to get a ridesharing instance $(N, \mathcal{A}')$. Then Corollary 3.1 and Lemma 3.3 can be applied to $(N, \mathcal{A}')$. From this, the analyses of Theorem 3.5 and Theorem 3.6 can be applied to $(N, \mathcal{A}')$, and we have the following theorems.

Theorem 3.7. *The ridesharing problem RSOne cannot be approximated within a constant factor (unless $P = NP$) when Conditions C1-C4 are satisfied, but the travel time condition C5 is not.*

Theorem 3.8. *The ridesharing problem RSTwo cannot be approximated within a constant factor (unless $P = NP$) when Conditions C1-C4 are satisfied, but the travel time condition C5 is not.*

Note that the above results also apply to the more general problems RSOne* and RSTwo*.

Inapproximability for the detour condition C2.

The proof uses a similar method as described in Subsection 3.2.2 (Inapproximability for the stop frequency condition C4). Similarly, the inapproximate results also apply to RSOne* and RSTwo*. Given a 3-partition instance $A = \{a_1, \ldots, a_{3l}\}$, the ridesharing instance $(N, \mathcal{A})$ is constructed as follows (see Figure 3.2):

- The road network is the weighted graph $N(V, E, w)$ with $V(N) = \{\chi, I, v_1, \ldots, v_{3l}, u_1, \ldots, u_l\}$ and $E(N)$ having edges (χ, I) and (I, χ) with $w(\chi, I) = w(I, \chi) = lK$, edges (v_i, I) and (v_i, I) with $w(v_i, I) = w(v_i, I) = a_i$ for $1 \leq i \leq 3l$, and edges (u_i, I) and (u_i, I) with $w(u_i, I) = w(u_i, I) = lK$ for $1 \leq i \leq l$.

- $\mathcal{A} = \{1, 2, \ldots, l+3l^2K\}$ has $l+3l^2K$ trips. Let α and β be valid constants representing time such that $\alpha < \beta$.

  - Each trip i, $1 \leq i \leq l$, has origin $o_i = u_i$, destination $d_i = \chi$, capacity $\lambda_i = 3lK$, detour distance limit $z_i = 2K$, a unique preferred path $P_i = (u_i, I, \chi)$ in N, stop limit $\delta_i = \lambda_i$, departure time $\alpha_i = \alpha$ and arrival time $\beta_i = \beta$, and each trip i has a preferred path (u_i, I, χ) in N.
  - Each trip i, $l+1 \leq i \leq l+3l^2K$, has origin $o_i = v_j$, $j = \lceil \frac{i-l}{lK} \rceil$, destination $d_i = \chi$, capacity $\lambda_i = 0$, detour distance limit $z_i = 0$, stop limit $\delta_i = \lambda_i$, departure time $\alpha_i = \alpha$, arrival time $\beta_i = \beta$ and a unique preferred path $P_i = (v_j, I, \chi)$ in N $(j = \lceil \frac{i-l}{lK} \rceil)$.

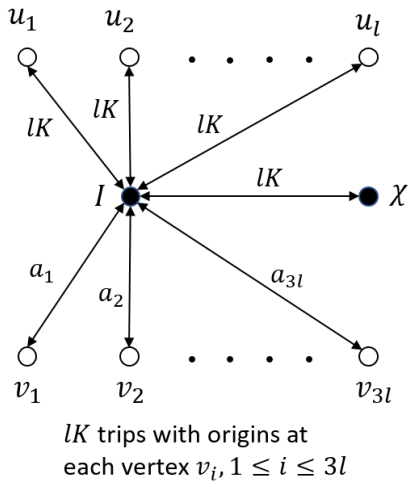


Figure 3.2: Ridesharing instance $(N, \mathcal{A})$ satisfying Conditions C1 and C3-C5.

Lemma 3.4. *Any solution for the instance $(N, \mathcal{A})$ has every trip i with $1 \leq i \leq l$ as a driver and total travel distance at least $2lK \cdot (l+1)$.*

Proof. For any trip i with origin at v_j, $1 \leq j \leq 3l$, regardless if trip i is a passenger trip or a flexible-role trip, i cannot serve any other trip since it would require detour distance larger than one, which is greater than z_i. For any trip i, $1 \leq i \leq l$, regardless if trip i is a driver trip or a flexible-role trip, i cannot serve any trip $j \neq i$ for $1 \leq j \leq l$ since it would require detour distance at least $2lK > z_i = 2K$ as $l \geq 2$.

Let S be the set of l trips assigned as drivers η_i with $1 \leq i \leq l$. The total travel distance of the drivers in S is $2l^2K$. For each trip j with $3l+1 \leq j \leq l+3l^2K$, the total travel distance of drivers in S and trip j is $2l^2K + 2a_j$ if j is served by a driver in S; otherwise, it is at least $2lK + a_j + lK$ (assuming j is a flexible-role trip and assigned as a driver). Since $a_j < lK$, the minimum total travel distance of any solution is to have every j, $3l+1 \leq j \leq l+3l^2K$, assigned as a passenger and served by S. The total travel distance of S is $2l^2K + \sum_{1 \leq j \leq 3l} 2a_j = 2l^2K + 2lK = 2lK \cdot (l+1)$. □

Lemma 3.5. *Let $(N, \mathcal{A})$ be a ridesharing problem instance constructed above from a 3-partition problem instance $A = \{a_1, \ldots, a_{3l}\}$. The 3-partition problem instance A has a solution if and only if the ridesharing problem instance $(N, \mathcal{A})$ has a solution (σ, S) such that $l \leq |S| < l + lK$, where S is the set of drivers.*

Proof. Assume that instance A has a solution $A_1, \ldots, A_l$ where the sum of elements in each A_j, $1 \leq j \leq l$, is K. For each $A_i = \{a_{i_1}, a_{i_2}, a_{i_3}\}$, $1 \leq i \leq l$, we say trips with origins at vertex v_j ($1 \leq j \leq 3l$) *correspond to* A_i if the edge $\{v_j, I\}$ has weight a_{i_1}, a_{i_2}, or a_{i_3}. By the definition of 3-partition instance, one can uniquely identify the corresponding trips of A_i. Then for each A_i, there are exactly $3lK$ corresponding trips with origins at three different vertices v_j. Recall that every trip i with origin at u_i has detour distance limit $z_i = 2K$ and capacity $\lambda_i = 3lK$. We assign a trip i with origin at u_i as a driver to serve the corresponding trips of A_i for $1 \leq i \leq l$. It requires exactly $2K$ detour distance for i to serve the $3lK$ corresponding trips of A_i. Hence, we have a solution of l drivers for $(N, \mathcal{A})$.

Assume that $(N, \mathcal{A})$ has a solution with $l \leq |S| < l + lK$ drivers. Let $\mathcal{A}(1, l)$ be the set of trips in $\mathcal{A}$ with labels from 1 to l. By Lemma 3.4, every trip $i \in \mathcal{A}(1, l)$ is a driver in S (trips with origin at u_i are assigned as drivers). For every vertex v_j, let X_j be the set of trips with origin at v_j not served by drivers in $\mathcal{A}(1, l)$. Since $|S| < 3l + lK$ and all of $\mathcal{A}(1, 3l)$ are drivers in S, at most $lK - 1$ trips in $X_j \subset (\mathcal{A}' \setminus \mathcal{A}(1, 3l))$ can be assigned as drivers in S. Thus, $0 \leq |X_j| < lK$ for any v_j. From this and there are lK trips with origin at each vertex v_j, every driver in $\mathcal{A}(1, l)$ must detour to some vertex v_j ($1 \leq j \leq 3l$) to pick-up some passengers. In other words, every vertex v_j for $1 \leq j \leq 3l$ must have been visited by at least one driver in $\mathcal{A}(1, l)$. Assume for contradiction that some driver $\eta_i \in \mathcal{A}(1, l)$ has detoured from preferred path P_i less than $2K$ (i detours to at least one vertex and at most three vertices because $K/4 < a_j < K/2$ for $1 \leq j \leq 3l$). Then from the fact that the sum of

elements in A is $\sum_{1 \leq j \leq 3l} a_j = lK$, some driver $i' \in S$ must has detoured from preferred path $P_{i'}$ greater than $2K$ so that all vertices can be visited. This is a contradiction to $z_{i'} = 2K$. Hence, every driver in $\mathcal{A}(1, l)$ has detoured exactly $2K$. For each driver i, $1 \leq i \leq l$, let A_i be the subset of the three integers of A corresponding to the detour distance travelled by i (one way). Then $A_1, \ldots, A_l$ is a solution for the 3-partition problem instance.

It takes a polynomial time to convert a solution of $(N, \mathcal{A})$ to a solution of the 3-partition instance and vice versa. □

With Lemma 3.5, the analyses of Theorem 3.5 and Theorem 3.6 can be applied to $(N, \mathcal{A})$, and we have the following theorems.

Theorem 3.9. *The ridesharing problem RSOne cannot be approximated within a constant factor (unless $P = NP$) when Conditions C1 and C3-C5 are satisfied, but the detour condition C2 is not.*

Theorem 3.10. *The ridesharing problem RSTwo cannot be approximated within a constant factor (unless $P = NP$) when Conditions C1 and C3-C5 are satisfied, but the detour condition C2 is not.*

Inapproximability for the preferred path condition C3.

The NP-hardness proof for this case is also a reduction from the 3-partition problem. Given a 3-partition instance $A = \{a_1, \ldots, a_{3l}\}$, the ridesharing instance $(N, \mathcal{A})$ is constructed as follows (see Figure 3.3):

- The road network graph $N(V, E, w)$ is the same as the graph N for Lemma 3.9 in Subsection 3.2.3.
- $\mathcal{A} = \{1, 2, \ldots, 3l + (lK)^2\}$ has $3l + (lK)^2$ trips. Let α and β be valid constants representing time such that $\alpha < \beta$.
  - Each trip i, $1 \leq i \leq 3l$, has origin $o_i = u_i$, destination $d_i = \chi$, capacity $\lambda_i = a_i \cdot lK$, detour distance limit $z_i = 0$, stop limit $\delta_i = \lambda_i$, departure time $\alpha_i = \alpha$ and arrival time $\beta_i = \beta$; and each trip i has l preferred paths (u_i, I, v_j, χ) in N for $1 \leq j \leq l$. Each trip i is either a flexible-role or driver-role trip.
  - Each trip i, $3l + 1 \leq i \leq 3l + (lK)^2$ has origin $o_i = v_j$, $j = \lceil (i - 3l)/lK^2 \rceil$, destination $d_i = \chi$, capacity $\lambda_i = 0$, detour distance limit $z_i = 0$, a unique preferred path (v_j, χ) in N, stop limit $\delta_i = \lambda_i$, departure time $\alpha_i = \alpha$ and arrival time $\beta_i = \beta$. Each trip i is either a flexible-role or passenger-role trip.

Lemma 3.6. *Any solution for the instance $(N, \mathcal{A})$ has every trip i, $1 \leq i \leq 3l$, as a driver and total travel distance at least $9l$.*

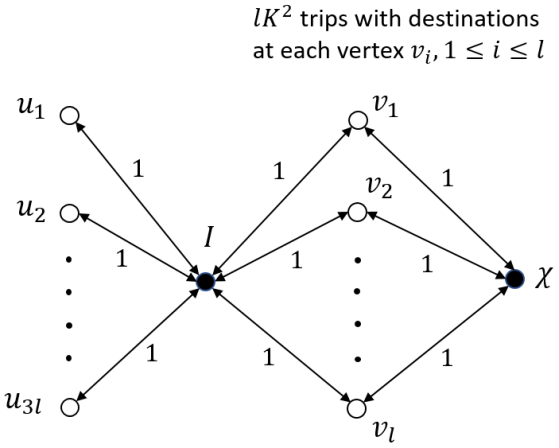


Figure 3.3: Ridesharing instance $(N, \mathcal{A})$ satisfying Conditions C1,C2,C4 and C5.

Proof. Trip i, $1 \le i \le 3l$, must be assigned as a driver in any solution since no other trip $j \ne i$ can serve i for every trip $j \in \mathcal{A}$ due to detour $z_j = 0$. Let S be the set of $3l$ trips assigned as drivers η_i with $1 \le i \le 3l$. The total travel distance of the drivers in S is $9l$. For each trip j with $3l+1 \le j \le 3l+(lK)^2$, the total travel distance of drivers in S and trip j is $9l$ if j is served by a driver in S; otherwise, it is at least $9l+1$ (assuming j is a flexible-role trip and assigned as a driver). Hence, a solution with trips i with $1 \le i \le 3l$ assigned as drivers has total travel distance $9l$, and any additional trip j for $3l+1 \le j \le 3l+(lK)^2$ assigned as a driver has total travel distance greater than $9l$. □

With Lemma 3.6 and similar analyses of Theorem 3.21 and Lemma 3.3, we have the following lemma.

Lemma 3.7. *Let $(N, \mathcal{A})$ be a ridesharing problem instance constructed above from a 3-partition problem instance $A = \{a_1, \ldots, a_{3l}\}$. The 3-partition problem instance A has a solution if and only if the ridesharing problem instance $(N, \mathcal{A})$ has a solution (σ, S) such that $3l \le |S| < 3l + lK$, where S is the set of drivers.*

From Lemma 3.7, the analyses of Theorem 3.5 and Theorem 3.6 can be applied to $(N, \mathcal{A})$, and we have the following two theorems.

Theorem 3.11. *The ridesharing problem RSOne cannot be approximated within a constant factor (unless $P = NP$) when Conditions C1,C2,C4 and C5 are satisfied, but the preferred path condition C3 is not.*

Theorem 3.12. *The ridesharing problem RSTwo cannot be approximated within a constant factor (unless $P = NP$) when Conditions C1,C2,C4 and C5 are satisfied, but the preferred path condition C3 is not.*

3.2.3 Extending previous NP-hardness results.

First, the NP-hardness results from [43, 68] are re-stated in the following six theorems.

Theorem 3.13. *The ridesharing problem RSOne (minimizing the number of drivers) is NP-hard when Condition C2-C5 are satisfied but the location condition C1 is not.*

Theorem 3.14. *The ridesharing problem RSTwo (minimizing the total travel distance of drivers) is NP-hard when C2-C5 are satisfied but the location condition C1 is not.*

Theorem 3.15. *The ridesharing problem RSOne (minimizing the number of drivers) is NP-hard when all trips have the same destination and Conditions C3-C5 are satisfied but the detour condition C2 is not.*

Theorem 3.16. *The ridesharing problem RSTwo (minimizing the total travel distance of drivers) is NP-hard when all trips have the same destination and Conditions C3-C5 are satisfied but the detour condition C2 is not.*

Theorem 3.17. *The ridesharing problem RSOne (minimizing the number of drivers) is NP-hard when all trips have the same destination and Conditions C2,C4,C5 are satisfied but the preferred path condition C3 is not.*

Theorem 3.18. *The ridesharing problem RSTwo (minimizing the total travel distance of drivers) is NP-hard when all trips have the same destination and Conditions C2,C4,C5 are satisfied but the preferred path condition C3 is not.*

We show that each of Theorem 3.15, Theorem 3.16, Theorem 3.17 and Theorem 3.18 also applies to all trips have the same origin (instead of all trips have the same destination). The proofs are similar to the ones in [43, 68], but for completeness we show them in detailed proofs. In addition, all six theorems above also apply to RSOne* and RSTwo*, the more general version (namely, $\mathcal{A} = D \cup R \cup DR$). The analysis is also based on a reduction from the 3-partition problem.

NP-hardness result for the detour condition.

First, we extend Theorem 3.15 and Theorem 3.16. Given a 3-partition problem instance $A = \{a_1, a_2, ..., a_{3l}\}$, we construct a ridesharing instance $(N, \mathcal{A})$ as follows (an example is given in Figure 3.4):

- The road network is the weighted graph $N(V, E, w)$ with $V(N) = \{\chi, I, v_1, \ldots, v_{4l}\}$ and $E(N)$ having edges (χ, I) and (I, χ) with $w(\chi, I) = w(I, \chi) = lK$, edges (v_i, I) and (v_i, I) with $w(v_i, I) = w(v_i, I) = a_i$ for $1 \leq i \leq 3l$, and edges (v_i, I) and (v_i, I) with $w(v_i, I) = w(v_i, I) = lK$ for $3l + 1 \leq i \leq 4l$.
- $\mathcal{A} = \{1, 2, \ldots, 4l\}$ has $4l$ trips.

– Each trip i, $1 \leq i \leq 4l$, has origin $o_i = \chi$ (a vertex in N), destination $d_i = v_i$ (a vertex in N), a unique preferred path $P_i = (\chi, I, v_i)$ in N, stop limit $\delta_i = \lambda_i$, departure time $\alpha_i = \alpha$ and arrival time $\beta_i = \beta$ for some time constants $\alpha < \beta$.
– Each trip i, $1 \leq i \leq 3l$, has capacity $\lambda_i = 1$, detour distance limit $0 < z_i \leq 1$.
– Each trip i, $3l + 1 \leq i \leq 4l$, has capacity $\lambda_i = 3$, detour distance limit $d_i = 2K$.

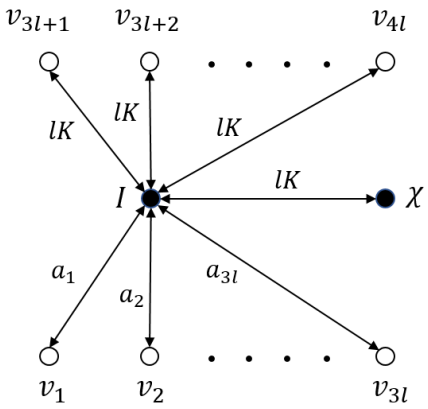


Figure 3.4: Ridesharing instance $(N, \mathcal{A})$ satisfying Conditions C3-C5, and all trips of $\mathcal{A}$ have the same origin.

The analysis of Lemma 3.4 also holds for this ridesharing instance $(N, \mathcal{A})$ (just different indexing) since they have the same graph N, which is re-stated as the following lemma.

Lemma 3.8. *Any solution for the instance $(N, \mathcal{A})$ has every trip i, $3l+1 \leq i \leq 4l$, assigned as a driver and total travel distance at least $2lK \cdot (l + 1)$.*

Theorem 3.19. *The ridesharing problem RSOne is NP-hard when Conditions C1 and C3-C5 are satisfied, but the detour condition C2 is not.*

Proof. By Theorem 3.15, the theorem holds for all trips have the same destination. We show it also holds for all trips have the same origin by proving that an instance $A = \{a_1, \ldots, a_{3l}\}$ of the 3-partition problem has a solution if and only if the above constructed ridesharing problem instance $(N, \mathcal{A})$ has a solution of l drivers.

Assume that the 3-partition instance has a solution $A_1, \ldots, A_l$ where the sum of elements in each A_j, $1 \leq j \leq l$, is K. For each $A_i = \{a_{i_1}, a_{i_2}, a_{i_3}\}$, $1 \leq i \leq l$, we say trips with origins at vertex v_j $(1 \leq j \leq 3l)$ *correspond to* A_i if the edge (v_j, I) has weight a_{i_1}, a_{i_2}, or a_{i_3}. We assign each trip i with $3l+1 \leq i \leq 4l$ to serve one set A_j for $j = i-3l$. Since $\lambda_i = \delta_i = 3$ for $3l+1 \leq i \leq 4l$ and the sum of A_j is K, driver η_i can serve the three passengers corresponding to A_j with total detour distance $2K$. Hence, we have a solution of l drivers for $(N, \mathcal{A})$.

Assume that $(N, \mathcal{A})$ has a solution of l drivers. By Lemma 3.8, every trip i with $3l+1 \leq i \leq 4l$ must be a driver η_i in the solution. Then, each trip j for $1 \leq j \leq 3l$ must be a served passenger r_j in the solution. From $\lambda_i = 3$ for $3l + 1 \leq i \leq 4l$, each driver η_i can serve at

most 3 passengers. From this and there are $3l$ passengers, each driver η_i must serve exactly 3 passengers in the solution. By the construction of N, every driver η_i must detour from the preferred path P_i to serve the passengers assigned to η_i. Assume, for contrary, that some driver η_i uses a route with a detour smaller than $2K$ to serve the passengers assigned to η_i. Then from the fact that the sum of elements in A is lK, some driver i' must have a detour greater than $2K$, a contradiction to $z_{i'} = 2K$. Hence, the actual detour from the preferred path P_i for each driver η_i is exactly $2K$. For each driver η_i with $3l + 1 \leq i \leq 4l$, let A_j, $j = i - 3l$, be the subset of the three integers of A corresponding to the passengers served by η_i. Then $A_1, \ldots A_l$ is a solution for the 3-partition problem instance.

The size of $(N, \mathcal{A})$ is linear in l. It takes a linear time to convert a solution of $(N, \mathcal{A})$ to a solution of the 3-partition instance and vice versa. □

Theorem 3.20. *The ridesharing problem RSTwo is NP-hard when Conditions C1 and C3-C5 are satisfied, but the detour condition C2 is not.*

Proof. By Theorem 3.16, the theorem holds for all trips have the same destination. We show the other half by proving that an instance $A = \{a_1, \ldots, a_{3l}\}$ of the 3-partition problem has a solution if and only if the above constructed ridesharing problem instance $(N, \mathcal{A})$ has a solution with $2lK(l + 1)$ total travel distance.

Assume that the 3-partition instance has a solution. Then there is a solution of l drivers for $(N, \mathcal{A})$ as shown in the proof of Theorem 3.19. The total travel distance of this solution is $2lK(l + 1)$ as shown in the proof of Lemma 3.8.

Assume that $(N, \mathcal{A})$ has a solution with total travel distance $2lK \cdot (l + 1)$. As shown in the proof of Lemma 3.8, trips i with $3l + 1 \leq i \leq 4l$ are the assigned drivers in the solution. From this, there is a solution for the 3-partition instance as shown in the proof of Theorem 3.19. □

NP-hardness result for the preferred path condition.

Next, we extend Theorem 3.17 and Theorem 3.18. Given a 3-partition problem instance $A = \{a_1, a_2, ..., a_{3l}\}$, we construct a ridesharing instance $(N, \mathcal{A})$ as follows (an example is given in Figure 3.5):

- The road network is the weighted graph $N(V, E, w)$ with $V(N) = \{\chi, I, u_1, \ldots, u_{3l}, v_1, \ldots, v_l\}$ and $E(N)$ having edges (χ, v_i), (v_i, χ), (v_i, I) and (v_i, I) for $1 \leq i \leq l$, and edges (u_i, I) and (u_i, I) for $1 \leq i \leq 3l$. Each edge of $E(N)$ has weight of one.

- $\mathcal{A} = \{1, 2, \ldots, 3l + lK\}$ has $3l + lK$ trips. Let α and β be valid constants representing time such that $\alpha < \beta$.

  - Each trip i, $1 \leq i \leq 3l$, has origin $o_i = \chi$ (a vertex in N), destination $d_i = u_i$ (a vertex in N), capacity $\lambda_i = a_i$, detour distance limit $z_i = 0$, stop limit $\delta_i = \lambda_i$,

departure time $\alpha_i = \alpha$ and arrival time $\beta_i = \beta$; and each trip i has l preferred paths (χ, v_j, I, u_i) in N for $1 \leq j \leq l$.

– Each trip i, $3l+1 \leq i \leq 3l+lK$ has origin $o_i = \chi$, destination $d_i = v_j$, $j = \lceil \frac{i-3l}{K} \rceil$, capacity $\lambda_i = 0$, detour distance limit $z_i = 0$, a unique preferred path (χ, v_j) in N, stop limit $\delta_i = \lambda_i$, departure time $\alpha_i = \alpha$ and arrival time $\beta_i = \beta$.

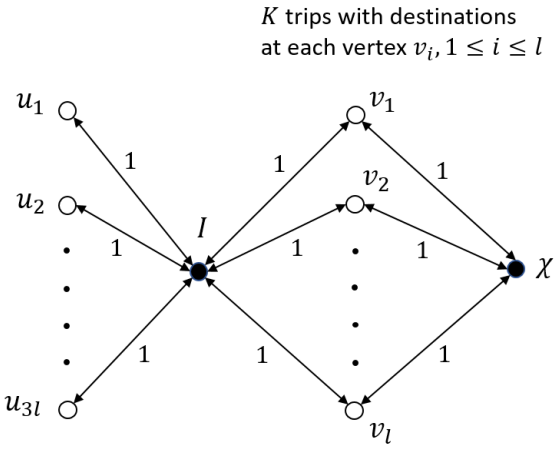


Figure 3.5: Ridesharing instance $(N, \mathcal{A})$ satisfying Conditions C1, C2, C4 and C5, and all trips of $\mathcal{A}$ have the same origin.

The analysis of Lemma 3.6 also holds for this ridesharing instance $(N, \mathcal{A})$ (just different indexing) since they have the same graph N, which is re-stated as the following lemma.

Lemma 3.9. *Any solution for the instance $(N, \mathcal{A})$ has every trip i, $1 \leq i \leq 3l$, as a driver and total travel distance at least $9l$.*

Theorem 3.21. *The ridesharing problem RSOne is NP-hard when Conditions C1, C2, C4 and C5 are satisfied, but the preferred path condition C3 is not.*

Proof. By Theorem 3.17, the theorem holds for all trips have the same destination. We show it also holds for all trips have the same origin by proving that an instance $A = \{a_1, \ldots, a_{3l}\}$ of the 3-partition problem has a solution if and only if the above constructed ridesharing problem instance $(N, \mathcal{A})$ has a solution of $3l$ drivers.

Assume that the 3-partition instance has a solution $A_1, \ldots, A_l$ where the sum of elements in each A_j, $1 \leq j \leq l$, is K. For each $A_j = \{a_{j_1}, a_{j_2}, a_{j_3}\}$, $1 \leq j \leq l$, we assign the three trips whose $\lambda_i j_1 = a_{j_1}$, $\lambda_i j_2 = a_{j_2}$ and $\lambda_i j_3 = a_{j_3}$ as drivers to serve the K trips with destinations at vertex v_j. Hence, we have a solution of $3l$ drivers for $(N, \mathcal{A})$.

Assume that $(N, \mathcal{A})$ has a solution of $3l$ drivers. By Lemma 3.9, every trip i, $1 \leq i \leq 3l$, is assigned as a driver in the solution. Then, each trip j for $3l + 1 \leq j \leq 3l + lK$ must be assigned as a passenger in the solution, total of lK passengers. Since $\sum_{1 \leq i \leq 3l} a_i = lK$, each driver i, $1 \leq i \leq 3l$, serves exactly $\lambda_i = a_i$ passengers. Since $a_i < K/2$ for every

$a_i \in A$, at least three drivers are required to serve the K passengers with destinations at each vertexv_j, $1 \leq j \leq 3l$. Therefore, the solution of $3l$ drivers has exactly three drivers j_1, j_2, j_3 to serve the K passengers with destinations at each vertex v_j, $1 \leq j \leq l$, implying $a_{j_1} + a_{j_2} + a_{j_3} = K$. Let $A_j = \{a_{j_1}, a_{j_2}, a_{j_3}\}$, $1 \leq j \leq l$, we get a solution for the 3-partition problem instance.

The size of $(N, \mathcal{A})$ is polynomial in l. It takes a polynomial time to convert a solution of $(N, \mathcal{A})$ to a solution of the 3-partition instance and vice versa. □

Theorem 3.22. *The ridesharing problem RSTwo is NP-hard when Conditions C1, C2, C4 and C5 are satisfied, but the preferred path condition C3 is not.*

Proof. By Theorem 3.18, the theorem holds for all trips have the same destination. We show the other half by proving that an instance $A = \{a_1, \ldots, a_{3l}\}$ of the 3-partition problem has a solution if and only if the above constructed ridesharing problem instance $(N, \mathcal{A})$ has a solution with $9l$ total travel distance.

Assume that the 3-partition instance has a solution. Then there is a solution of $3l$ drivers for $(N, \mathcal{A})$ as shown in the proof of Theorem 3.21. The total travel distance of this solution is $9l$ by Lemma 3.9.

Assume that $(N, \mathcal{A})$ has a solution with total travel distance $9l$. As shown in the proof of Lemma 3.9, trips i with $1 \leq i \leq 3l$ must be assigned as drivers. From this, there is a solution for the 3-partition instance as shown in the proof of Theorem 3.21. □

NP-hardness result for the location condition.

The proofs of Theorem 3.13 and Theorem 3.14 presented in [43, 68] are based on a reduction from the Interval Scheduling with Machine Availability Problem (ISMAP) [63]. Similar to the above construction of ridesharing instances, for the instance $(N, \mathcal{A})$ constructed from ISMAP, the trips of $\mathcal{A}$ that are corresponding to machines can be from either D or DR, and the trips of $\mathcal{A}$ that are corresponding to jobs can be from either R or DR. Thus, Theorem 3.13 and Theorem 3.14 can be applied to RSOne* and RSTwo*.

3.3 Polynomial-time solvable problem variants with capacity larger than one

As shown in Section 3.2, if one of Conditions C1-C5 is not satisfied, both ridesharing problems RSOne and RSTwo are NP-hard. In [43, 68], a polynomial-time exact algorithm is proposed for RSOne when Conditions C1-C5 are satisfied and the preferred paths of all trips in $\mathcal{A} = DR$ lie on a single path of the road network N (i.e., the graph induced by the preferred paths of all trips is a path in N). We gave a refined proof for that polynomial-time exact algorithm in [45]. In this section, we present two polynomial-time algorithms for a more general case of RSOne and RSTwo than the one in [43, 68].

(1) Given an instance of RSOne and an instance of RSTwo such that both instances satisfy all of the Conditions C1-C5 and transitive serve relation, we give a polynomial-time dynamic algorithm that can solve each instance.

(2) Given an instance of RSOne satisfying all of the Conditions C1-C5 and transitive serve relation, we give a polynomial-time greedy algorithm with a running time better than the dynamic algorithm in (1) for the instance.

In this section, we assume a ridesharing instance $(N, \mathcal{A})$ always satisfy all of Conditions C1-C5, unless stated otherwise. We introduce transitive serve relation and show how to compute the serve relation graph in the next subsection.

3.3.1 Transitive serve relation

Recall that the feasible serve relation is defined between a trip i and a set J of trips. The serve relation graph $G_R(V, E)$ represents the feasible serve relations between every pair of trips in $\mathcal{A}$:

- Each trip i is represented by a vertex in $V(G_R)$. There is an edge (i, j) in $E(G_R)$ if trip i can serve trip j.

A serve relation is *transitive* if trip i can serve trip j and j can serve trip k imply i can serve k for any triple of trips i, j, k in $\mathcal{A}$. For a general instance $(N, \mathcal{A})$, the serve relation may not be transitive. However, transitive serve relation typically occurs when the road network N has a unique shortest path between any pair of vertices and each trip uses the shortest path from the origin to destination as the preferred path, or the preferred paths are computed by the centralized ridesharing system (CRS). Since the preferred path condition is satisfied for $\mathcal{A}$, trip i can serve trip j implies that P_j is a subpath of P_i, so the serve relation is transitive (assuming unique shortest path in N). Note that the NP-hardness results presented in Section 3.2 remain true for problem instances satisfying the transitive serve relation.

Serve relation graph.

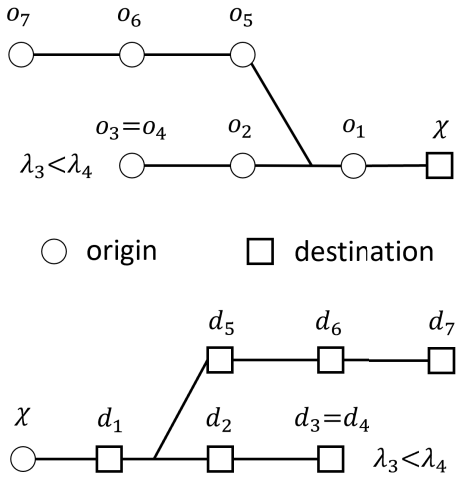
To deal with any trip $i \in \mathcal{A}$ that has zero vehicle capacity ($\lambda_i = 0$), we introduce a *pseudo serve relation* in an instance $(N, \mathcal{A})$: trip i can pseudo serve trip j if

- $\lambda_i > 0$ and i can serve j or
- $\lambda_i = 0$, $o_i = o_j$ and $d_i = d_j$ or
- $\lambda_i = 0$ and P_j is a subpath of P_i.

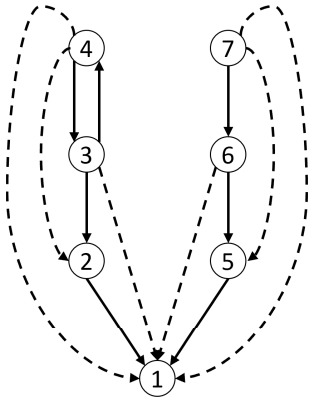
The pseudo serve relation becomes the serve relation when $\lambda_i > 0$ for every $i \in \mathcal{A}$. The pseudo serve relation can be expressed by a digraph $G_R(V, E)$ such that the digraph has $\mathcal{A}$

as the vertex set and there is an edge (i, j) in $E(G_R)$ if trip i can pseudo serve trip j. For two trips i and j with $o_i = o_j$ and $d_i = d_j$, trip i can pseudo serve trip j and vice versa, that is, both edges (i, j) and (j, i) are in the digraph initially. As shown later, we can restrict that only one of i and j can pseudo serve the other to simplify our discussion (make the digraph acyclic): if $\lambda_i \geq \lambda_j$ then we remove edge (j, i); otherwise, we remove (i, j) from the digraph.

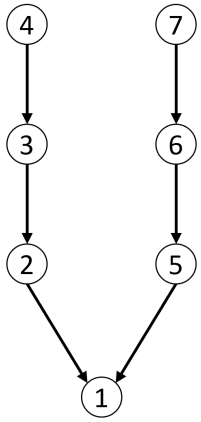
An edge (i, j) in $G_R(V, E)$ is called a *short cut* if after removing (i, j) from $G_R(V, E)$, there is still a path from i to j in $G_R(V, E)$. We remove all short cuts from $G_R(V, E)$ to get a digraph $\vec{G}(V, E)$ (called *simplified serve relation graph*) to express the pseudo serve relation in $\mathcal{A}$, namely it remains that trip i can pseudo serve trip j if there is a path from i to j in $\vec{G}$. A trip i corresponds to the vertex i in $\vec{G}$ is called a *source* (resp. *sink*) if there is no edge (j, i) (resp. (i, j)) in $\vec{G}$. A connected component of $\vec{G}$ is called a tree if the underlying graph of $\vec{G}$ is a tree. When Conditions C2 and C3 are satisfied, we assume that if trip i can pseudo serve trip j then P_j is a subpath of P_i. This makes the pseudo serve relation *transitive*. When all of Conditions C1-C5 are satisfied, every connected component of $\vec{G}$ is a tree, and we call the component an *inverse tree* as it has one unique sink and at least one source (we show this is indeed the case in Lemma 3.11). In Figure 3.6 is an example of a serve relation graph $G_R(V, E)$ and its simplified serve relation graph $\vec{G}(V, E)$ based on two symmetric instances $(N, \mathcal{A})$.



(a) Two instances $(N, \mathcal{A})$ satisfying Conditions C1-C5 with a common destination χ and a common origin χ shown at the top and bottom, respectively.



(b) The serve relation graph $G_R(V, E)$ of both instances $(N, \mathcal{A})$ shown in (a) are the same.



(c) The simplified serve relation graph $\vec{G}(V, E)$ of $G_R(V, E)$.

Figure 3.6: A serve relation graph $G_R(V, E)$ and its simplified serve relation graph $\vec{G}(V, E)$.

For any two trips i and j that can pseudo serve each other, it creates a cycle (i, j) and (j, i) in the serve relation graph $G_R(V, E)$. Based on Lemma 3.10, we break this cycle by removing one of the edges (i, j) and (j, i) to simplify the graph.

Lemma 3.10. *Let $(N, \mathcal{A})$ be an instance of either problem RSOne or RSTwo. Let i and j be two trips in $\mathcal{A}$ such that $o_i = o_j$, $d_i = d_j$ and $\lambda_i \geq \lambda_j$. Then an optimal solution for $(N, \mathcal{A})$ obtained on the $G_R(V, E)$ with edge (j, i) removed is an optimal solution on the $G_R(V, E)$ containing both (i, j) and (j, i).*

Proof. Note that the serve relation graph $G_R(V, E)$ of instance $(N, \mathcal{A})$ must contain both edges (i, j) and (j, i). Let $G'_R(V, E)$ be the serve relation graph of $G_R(V, E)$ with edge (j, i) removed. Let (S, σ) be an optimal solution computed based on $G_R(V, E)$ and (S', σ') be an optimal solution computed based on $G'_R(V, E)$ for the instance $(N, \mathcal{A})$, which is either problem RSOne or RSTwo. We first show that $\text{dist}(S') \leq \text{dist}(S)$, that is, S' is also an optimal solution based on G_R when $(N, \mathcal{A})$ is an instance of RSTwo. Assume for contradiction that $\text{dist}(S') > \text{dist}(S')$. If $i \notin \sigma(j)$ where $j \in S$, then (S, σ) is also an optimal solution based on G'_R, contradicting that S' is an optimal solution based on G'_R. Suppose that $i \in \sigma(j)$. Let (S_1, σ_1) be defined as: $S_1 = (S \setminus \{j\}) \cup \{i\}$, $\sigma_1(i) = \sigma(j)$ (this can be done because $\lambda_i \geq \lambda_j$ and each trip served by j can be served by i), and for every $\eta_y \in S \setminus \{i\}$, $\sigma_1(y) = \sigma(y)$. Then (S_1, σ_1) is a solution based on G'_R with $\text{dist}(S_1) = \text{dist}(S) < \text{dist}(S')$, contradicting that S' is an optimal solution based on G'_R. The above also holds for $|S'| \leq |S|$ following an identical analysis. □

Let T be a component of $\vec{G}$ for a ridesharing instance $(N, \mathcal{A})$. For trips i and j in T, the notations and definitions in Table 3.1 will be used for the rest of this section. Notice that all trips of D_i are in the path from i to the sink of T. As shown in Figure 3.6, whether all trips have the same destination or all trips have the same origin, the serve relation graph has the same structure. Thus, we restrict our discussions for the case that all of trips have the same destination. For the case that all trips have the same origin, the preprocessing and algorithms are similar.

Notation	**Definition**
i is a *parent* of j	If edge (i, j) is in T
i is an *ancestor* of j	If there is a path from i to j in T
i is a *child* of j	If edge (j, i) is in T (each trip has at most one child)
i is a *descendant* of j	If there is a path from j to i in T
A_i	The set of ancestors of i and i ($i \in A_i$)
D_i	The set of descendants of i and i ($i \in D_i$)

Table 3.1: Definitions for ancestors and descendants of trips in component T.

Given a ridesharing instance $(N, \mathcal{A})$, for any two connected components T_1 and T_2 in the pseudo serve relation digraph $\vec{G}$, any trip in T_1 cannot pseudo serve any trip in T_2 and vice versa. Let (S_1, σ_1) and (S_2, σ_2) be optimal solutions for trips in T_1 and trips in T_2, respectively, for either RSOne or RSTwo. Let $S = S_1 \cup S_2$ and $\sigma(i) = \sigma_1(i)$ for $\eta_i \in S_1$ and $\sigma(i) = \sigma_2(i)$ for $\eta_i \in S_2$. Then (S, σ) is an optimal solution for trips in $V(T_1) \cup B(T_2)$ for each of RSOne or RSTwo. Hence, we assume that $\vec{G}$ has one connected component T

($V(T) = \mathcal{A}$ and $\vec{G} = T$) because if $\vec{G}$ has more than one connected component, we can solve the ridesharing problems for each component independently.

3.3.2 Preprocessing

We describe how to actually construct the simplified serve relation graph $\vec{G}$ from an instance $(N, \mathcal{A})$. The construction of $\vec{G}$ is given in Algorithm 1 (Serve Relation Graph).

Algorithm 1 Serve Relation Graph

Input: A ridesharing instance $(N, \mathcal{A})$ satisfying Conditions C1-C3.
Output: A digraph $\vec{G}(V, E)$ for the pseudo serve relation for all trips in $\mathcal{A}$.
$V(\vec{G}) = \mathcal{A} = \{1, \ldots, l\}$;
Partition $\mathcal{A}$ into $r \leq l$ groups $G_1, \ldots, G_r$ s.t. trips i, j are in a same group if $o_i = o_j$;
For each group G_i, let i_1 be the trip in G_i with the largest capacity;
Let $\mathcal{A}' = \{i_1 \in G_i \mid 1 \leq i \leq r\}$;
Compute the subgraph $N_{\mathcal{A}'}$ of N induced by the vertex set $\cup_{i \in \mathcal{A}'} P_i$;
Let ST1 and ST2 be stacks, and χ (a vertex in N) be the common destination of trips in $\mathcal{A}$;
Perform a depth first search (DFS) on $N_{\mathcal{A}'}$ starting from χ using ST1;
    When a vertex u in $N_{\mathcal{A}'}$ is pushed into ST1 in DFS:
        if u is the origin o_i of some trip $i \in \mathcal{A}'$ **then**
            if ST2 has the top element o_j **then** create edge (i, j) in $E(\vec{G})$;
            push o_i into ST2;
        end if
    When a vertex u is removed from ST1 in DFS:
        if u is the origin o_i of some trip $i \in \mathcal{A}'$ **then** remove o_i from ST2;
For each group G_i with $|G_i| > 1$, let $i_1, \ldots, i_b$ be the trips in G_i with $\lambda_{i_1} \geq \ldots \geq \lambda_{i_b}$:
    replace trip i_1 in $\vec{G}$ by the chain $(i_1, i_2) \ldots (i_{b-1}, i_b)$;
    replace edge (i_1, v) by (i_b, v) if $(i_1, v) \in E(\vec{G})$;
    any $(u, i_1) \in E(\vec{G})$ remains unchanged;

Lemma 3.11. *Given a ridesharing instance $(N, \mathcal{A})$ of size M, it takes $O(M)$ time to construct $\vec{G}$. For any pair of trips $i, j \in \mathcal{A}$, if i can pseudo serve j then there is path from i to j in $\vec{G}$. Any connected component of $\vec{G}$ is an inverse tree.*

Proof. It takes $O(M)$ time to partition trips into groups. It is easy to see that each edge of the induced subgraph $N_{A'}$ is traversed $O(1)$ times. The replacement of i_1 in $\vec{G}$ takes $O(M)$ time for all groups. Therefore, the total time for constructing $\vec{G}$ is $O(M)$.

For trips $i, j \in \mathcal{A}$, if i can serve j then path P_j is a subpath of P_i, implying the origin o_j is in path P_i. Algorithm 1 creates a path from i to j in $\vec{G}$. Let $i, j, k \in \mathcal{A}$ such that i can server j and k. Then both P_j and P_k are subpaths of P_i. Since $d_i = d_j = d_k = \chi$, either P_k is a subpath of P_j or P_j is a subpath of P_k. Hence, j is either an ancestor or a descendant of k in $\vec{G}$, implying every connected component of $\vec{G}$ is an inverse tree. □

From Lemma 3.10 and Lemma 3.11, we only need to work on $\vec{G}$ for computing solutions to the ridesharing problems RSOne and RSTwo.

Labelling trips in the simplified serve relation graph.

Given an instance $(N, \mathcal{A})$ of the ridesharing problem, where $\mathcal{A} = \{1, \ldots, l\}$, and the digraph $\vec{G}$ for the pseudo serve relation in $\mathcal{A}$, let T be the connected component (an inverse tree) of $\vec{G}$. We rearrange the labels of the trips in T by Algorithm 2 (Label Inverse Tree). In the rest of this section, each trip in T is expressed by the label assigned in Algorithm 2. Based on this labelling, trip l is a source in T, trip 1 is the sink in T, and if trip i can pseudo serve trip j then $i > j$ for every $i \neq j$.

Algorithm 2 Label Inverse Tree

```
1: Input: An inverse tree T of G⃗ with l trips.
2: Output: A distinct integer label i, 1 ≤ i ≤ l, for each trip in T.
3: Let ST be a stack and push the sink of T into ST;
4: Set i = l and mark every edge in T un-visited;
5: while ST ≠ ∅ do
6:     let u be the trip at the top of ST;
7:     if there is an un-visited edge (v, u) in T then
8:         push v into ST and mark (v, u) visited;
9:     else
10:        remove u from ST; assign u integer label i; i = i − 1;
11:    end if
12: end while
```

The following notations will be used for the remainder of this section.

- For $\mathcal{A} = \{1, \ldots, l\}$ and $1 \leq i \leq j \leq l$, $T(i, j) = \{i, i+1, \ldots, j\}$.
- For a set S of drivers, $S(i, j) = S \cap T(i, j)$.
- For a trip $i \in \mathcal{A}$, v_i is the ancestor of i in T with the largest label.

Notice that $A_i = T(i, v_i)$ and v_i is a source in T. $T(i, v_i)$ is called a *branch* of T. A trip i is called a *merge point* if i has at least two parents in T. A branch is *simple* if it does not have any merge point. A branch $T(i, v_i)$ is *maximal* if the child of i is a merge point or $i = 1$ is the sink. For a (partial) solution (S, σ) of $(N, \mathcal{A})$ and $1 \leq i \leq j \leq l$, S is called a solution of $T(i, j)$ if $T(i, j) \subseteq \sigma(S)$.

3.3.3 Dynamic programming algorithm

We give a dynamic programming algorithm for both RSOne and RSTwo using the simplified serve relation graph T. We first find solutions for simple branches of T, and expand these solutions to solutions of branches consisting of simple branches, and expand solutions of branches to solutions of larger branches. We process trips in the decreasing order of their labels. This processing order guarantees that at most l expansions is enough. To make the number of solutions in each expansion small, we introduce a dominating relation between the solutions and only keep the non-dominated solutions for each expansion.

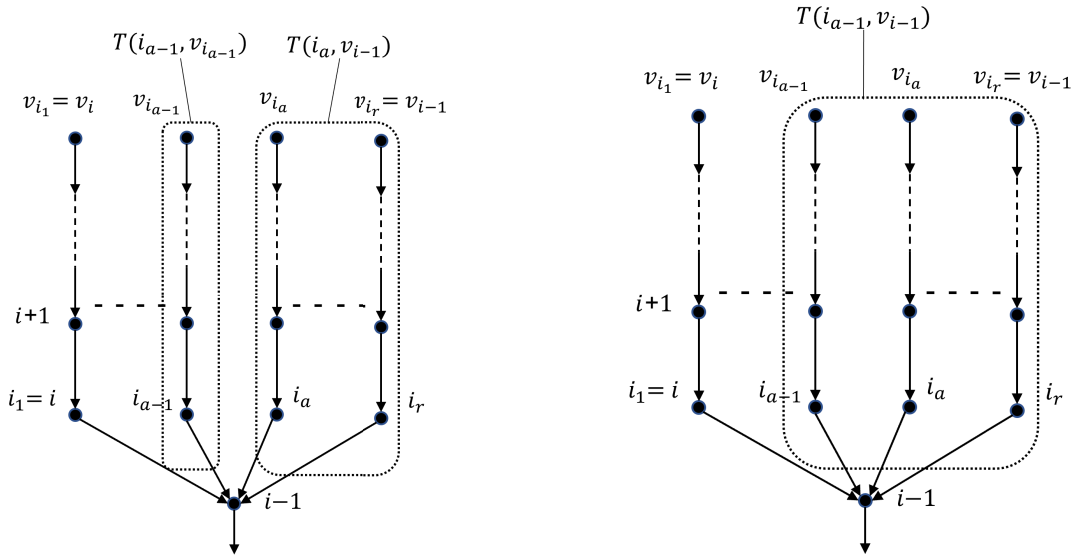
For a solution (S, σ) of $T(i, j)$, let $\text{dist}(S) = \sum_{\eta_i \in S} \text{dist}(P_i)$ be the *cost* of S, where P_i is the preferred path of trip i. For two solutions (S, σ) and (S', σ') of $T(i, j)$, S *dominates* S' if $|\sigma(S)| \geq |\sigma(S')|$ and $\text{dist}(S) \leq \text{dist}(S')$. Two solutions are *non-dominating* if none of them dominates the other. A set $\mathcal{X}$ of solutions is *non-dominating* if every pair of solutions in $\mathcal{X}$ is non-dominating. For $T(i, j)$, $1 \leq i \leq j \leq l$, $\mathcal{X}(i, j)$ denotes a set of (partial) solutions $\{(S_1, \sigma_1), \ldots, (S_b, \sigma_b)\}$, each is a solution of $T(i, j)$ computed by the algorithm (at the end of iteration$i$).

Algorithm description.

There are three major functions in our algorithm:

- (Compute solutions) Process each trip i from l to 1. If i is a source vertex, a set $\mathcal{X}(i, i)$ consisting of one solution $S = \{\eta_i\}$ is computed. When a non-source trip i is processed, a set $\mathcal{X}(i + 1, v_{i+1})$ of solutions for $T(i + 1, v_{i+1})$ has been computed. For each $S \in \mathcal{X}(i + 1, v_{i+1})$, we compute a solution (S', σ') with $S' = S \cup \{\eta_i\}$ to find a set $\mathcal{X}(i, v_i)$ of solutions for $T(i, v_i)$. Our algorithm makes $\sigma'(i)$ to serve as many trips in D_i that are not in $\sigma(S)$ and are closest to i as possible. More formally, we define $N(i, c, S)$ to be the set of c trips in D_i such that $N(i, c, S) \subseteq (D_i \setminus \sigma(S))$ and for any trip u in $N(i, c, S)$ and any trip v in $D_i \setminus (\sigma(S) \cup N(i, c, S))$, $\text{dist}(i, u) < \text{dist}(i, v)$. Let $c = \min\{\lambda_i + 1, |D_i \setminus \sigma(S)|\}$. In our algorithm, $\sigma'(i) = N(i, c, S)$.
- (Merge solutions) Just before a merge point $i - 1$ is processed by the algorithm, a set $\mathcal{X}(i, v_i)$ of solutions for each maximal simple branch $T(i, v_i)$ are found, and we want to merge all these solutions. For a merge point $i-1$, let $i_1, \ldots, i_r$ be the parents of $i - 1$ such that $i_a < i_b$ if $a < b$ (see Figure 3.7a). Notice that $v_{i_r} = v_{i-1}$ and $i_1 = i$. After trip i is processed, all of $\mathcal{X}(i_a, v_{i_a})$, $1 \leq a \leq r$, have been computed. For $a = r, r - 1, \ldots, 2$, we merge $\mathcal{X}(i_{a-1}, v_{i_{a-1}})$ and $\mathcal{X}(i_a, v_{i-1})$ into a set $\mathcal{X}(i_{a-1}, v_{i-1})$ of solutions for $T(i_{a-1}, v_{i-1})$. The merge is realized by including $S'' = S \cup S'$ in $\mathcal{X}(i_{a-1}, v_{i-1})$ for every $S \in \mathcal{X}(i_{a-1}, v_{i_{a-1}})$ and every $S' \in \mathcal{X}(i_a, v_{i-1})$ for $1 < a \leq r$ (see Figure 3.7b). By processing trips and merging solutions, we find a set $\mathcal{X}(i, v_i)$ of solutions for each maximal branch $T(i, v_i)$ and finally a set $\mathcal{X}(1, l)$ of solutions for $\mathcal{A}$.
- (Remove dominated solutions) When we compute a set of solutions, we remove each solution in the set that is dominated by another one in the same set, according to the dominating relation definition.

The pseudo code of our algorithm is given in Algorithm 3 (Find Minimum Cost Solution). To find an exact solution for an instance $(N, \mathcal{A})$ of RSTwo, execute Algorithm 3 with the dominating relation definition stated above. To find an exact solution for an instance $(N, \mathcal{A})$ of RSOne, execute Algorithm 3 with the dominating relation definition changed as: For two solutions (S, σ) and (S', σ') of $T(i, j)$, S *dominates* S' if $|\sigma(S)| \geq |\sigma(S')|$. Below is a simple example showing how Algorithm 3 works.



(a) Before merging $\mathcal{X}(i_{a-1}, v_{i_{a-1}})$ and $\mathcal{X}(i_a, v_{i-1})$

(b) After merging $\mathcal{X}(i_{a-1}, v_{i_{a-1}})$ and $\mathcal{X}(i_a, v_{i-1})$

Figure 3.7: Merge $\mathcal{X}(i_{a-1}, v_{i_{a-1}})$ (solutions of $T(i_{a-1}, v_{i_{a-1}})$) and $\mathcal{X}(i_a, v_{i-1})$ (solutions of $T(i_a, v_{i-1})$) into $\mathcal{X}(i_{a-1}v_{i-1})$ (solutions of $T(i_{a-1}, v_{i-1})$) for some $1 < a \leq r$.

- Let $\mathcal{A} = \{i \mid 1 \leq i \leq 7\}$ be the set of trips in Figure 3.6a (top instance $(N, \mathcal{A})$). Assume that the capacity λ_i and travel distance $\text{dist}(P_i)$ are as follows:

	Trip 1	Trip 2	Trip 3	Trip 4	Trip 5	Trip 6	Trip 7
λ_i	1	0	1	1	0	1	1
$\text{dist}(P_i)$	2	4	4.5	5	2.5	4	5

  Assume w.l.o.g. that the algorithm processes branch $T(5,7)$ first and branch $T(2,4)$ next for the simplified serve relation graph $\vec{G}(V, E)$ in Figure 3.6c

- For $i = 7$ (source), $\mathcal{X}(7,7)$ has one solution $(S = \{7\}, \sigma(7) = \{7,6\})\}$ denoted by $(\{7\}; \{7,6\})$.
- For $i = 6$ (non-source), $\mathcal{X}(6, v_6) = \mathcal{X}(6,7)$, solution $(\{7\}; \{7,6\})$ of $\mathcal{X}(7, v_7) = \mathcal{X}(7,7)$ is included in $\mathcal{X}(6,7)$. Then solution $(\{7,6\}; \{7,1\}, \{6,5\})$ is included in $\mathcal{X}(6,7)$. Since the two solutions of $\mathcal{X}(6,7)$ are non-dominating, they are kept in $\mathcal{X}(6,7)$.
- For $i = 5$ (non-source), $\mathcal{X}(5, v_5) = \mathcal{X}(5,7)$. Solutions of $\mathcal{X}(6,7)$ are included in $\mathcal{X}(5,7)$. Then solutions $(\{7,5\}; \{7,6\}, \{5\})$ and $(\{7,6,5\}; \{7,1\}, \{6\}, \{5\})$ are included in $\mathcal{X}(5,7)$. Next, solutions $(\{7\}; \{7,6\})$ (not a solution of $T(5,7)$) and $(\{7,6,5\}; \{7,1\}, \{6\}, \{5\})$ (dominated by the solution $(\{7,6\}; \{7,1\}, \{6,5\})$) are removed from $\mathcal{X}(5,7)$. So $\mathcal{X}(5,7)$ has two solutions $(\{7,5\}; \{7,6\}, \{5\})$ and $(\{7,6\}; \{7,1\}, \{6,5\})$.
- By processing branch $T(2,4)$, the algorithm computes $\mathcal{X}(2,4)$ which has two solutions $(\{4,2\}; \{4,3\}, \{2\})$ and $(\{4,3\}; \{4,1\}, \{3,2\})$.

- The solutions of $\mathcal{X}(2,4)$ and solutions of $\mathcal{X}(5,7)$ are merged to get solutions of $\mathcal{X}(2,7)$. During the merge, four driver sets $\{4,2\}\cup\{7,5\}, \{4,2\}\cup\{7,6\}, \{4,3\}\cup\{7,5\}$ and $\{4,3\}\cup\{7,6\}$ are computed. Solutions with driver sets $\{4,2,7,6\}$ and $\{4,3,7,6\}$ are dominated and removed; and $\mathcal{X}(2,7)$ has two solutions $(\{4,2,7,5\};\{4,3\},\{2\},\{7,6\},\{5\})$ and $(\{4,3,7,5\};\{4,1\},\{3,2\},\{7,6\},\{5\})$.
- Finally, for $i=1$, $\mathcal{X}(1,7)$ has one solution $(\{4,3,7,5\};\{4,1\},\{3,2\},\{7,6\},\{5\})$ which is an optimal solution for $(N,\mathcal{A})$.

Algorithm 3 Find Minimum Cost Solution

Input: An inverse tree T of $\vec{G}$ with l trips.
Output: A solution (S,σ) for an instance $(N,\mathcal{A})$ of either RSOne or RSTwo.
for $i=l$ to 1 **do** // process every trip of T, l is a source
  if i is a source of T **then** // process a source trip
    $S=\{\eta_i\}$; $c=\min\{\lambda_i+1, |D_i\setminus\sigma(S)|\}$; $\sigma(i)=N(i,c,S)$; $\mathcal{X}(i,i)=\{(S,\sigma)\}$;
  else // process a non-source trip
    if i is a merge point **then** $v=v_i$; **else** $v=v_i+1$;
    $\mathcal{X}(i,v_i)=\mathcal{X}(i+1,v_{i+1})$;
    for every $(S,\sigma)\in\mathcal{X}(i+1,v_{i+1})$ **do**
      $S'=S\cup\{\eta_i\}$; $\sigma'(S')=$Serve(i,S,σ); $\mathcal{X}(i,v_i)=\mathcal{X}(i,v_i)\cup\{(S',\sigma')\}$;
    end for // end of computing $\mathcal{X}(i,v_i)$
    for every solution S in $\mathcal{X}(i,v_i)$ **do** // make $\mathcal{X}(i,v_i)$ non-dominating
      if S is not a solution of $T(i,v_i)$ **then** remove S from $\mathcal{X}(i,v_i)$;
      if S is dominated by some S' in $\mathcal{X}(i,v_i)$ **then** remove S from $\mathcal{X}(i,v_i)$;
    end for
    if $i-1$ is a merge point **then** // merge solutions (before $i-1$ is processed)
      let $i_1,\ldots,i_r$ be the parents of $i-1$ with $i_a<i_b$ for $a<b$;
      for $a=r$ to 2 **do** $\mathcal{X}(i_{a-1},v_{i-1})=$Merge$(\mathcal{X}(i_{a-1},v_{i_{a-1}}),\mathcal{X}(i_a,v_{i-1}))$;
    end if
  end if
end for
Procedure Serve(i,S,σ)
$\sigma'(j)=\sigma(j)$ for every $\eta_j\in S$;
if $i\notin\sigma(S)$ **then**
  $c=\min\{\lambda_i+1, |D_i\setminus\sigma(S)|\}$; $\sigma'(i)=N(i,c,S)$;
else
  $c=\min\{\lambda_i, |D_i\setminus\sigma(S)|\}$; $\sigma'(i)=N(i,c,S)\cup\{i\}$;
  let $\eta_k\in S$ s.t. $i\in\sigma(k)$; $\sigma'(k)=\sigma'(k)\setminus\{i\}$; $\sigma'(k)=\sigma'(k)\cup N(k,1,S')$;
end if
Procedure Merge$(\mathcal{X}(i_{a-1},v_{i_{a-1}}),\mathcal{X}(i_a,v_{i-1}))$
Include $S''=S\cup S'$ in $\mathcal{X}(i_{a-1},v_{i-1})$ for $S\in\mathcal{X}(i_{a-1},v_{i_{a-1}})$ and $S'\in\mathcal{X}(i_a,v_{i-1})$;
$\text{dist}(S'')=\text{dist}(S)+\text{dist}(S')$;
Set $|\sigma''(S'')|$ to $\min\{|T(i_{a-1},v_{i-1})|+|D_{i-1}|, |\sigma(S)|+|\sigma(S')|\}$;
Remove S'' from $\mathcal{X}(i_{a-1},v_{i-1})$ if S'' is dominated;
for every $S''\in\mathcal{X}(i_{a-1},v_{i-1})$ **do**
  $\sigma''(j)=\sigma'(j)$ for $\eta_j\in S'$; $\sigma''(j)=\sigma(j)\setminus\sigma'(S')$ for $\eta_j\in S$;
  for every $\eta_j\in S$ **do** $c_j=|\sigma(j)\cap\sigma'(S')|$ and $\sigma''(j)=\sigma''(j)\cup N(j,c_j,S'')$;
end for

Analysis of algorithm.

Next, we show the correctness of Algorithm 3 (Find Minimum Cost Solution) and its running time.

Lemma 3.12. *Let $T(i, v_i)$ be any maximal simple branch in T. For any solution S^* of $T(1, l)$, there is a solution S in $\mathcal{X}(i, v_i)$ such that S dominates $S^*(i, v_i)$.*

Proof. We prove the lemma by induction for $i \leq a \leq v_i$. Since v_i is a source in T, v_i can be served only by itself, so any solution S^* of $T(1, l)$ contains v_i, implying $S^*(v_i, v_i) = \{v_i\}$. From this and $S = \{v_i\} \in \mathcal{X}(v_i, v_i)$, the lemma holds for $a = v_i$. Assume that the lemma is true for $i < a \leq v_i$ and we prove it for $a-1$. By the induction hypothesis, there is a solution S in $\mathcal{X}(a, v_i)$ such that S dominates $S^*(a, v_i)$. Let $S' = S \cup \{a-1\}$ be the solution obtained by Algorithm 3. If $a - 1 \in S^*(a-1, v_i)$ then from the fact that S dominates $S^*(a, v_i)$,

$$\begin{aligned}|\sigma(S')| &= \min\{|D_{v_i}|, |\sigma(S)| + \lambda_{a-1} + 1\} \\ &\geq \min\{|D_{v_i}|, |\sigma^*(S^*(a, v_i))| + \lambda_{a-1} + 1\} \geq |\sigma^*(S^*(a-1, v_i))|\end{aligned}$$

and

$$\begin{aligned}\mathrm{dist}(S') &= \mathrm{dist}(S) + \mathrm{dist}(a-1) \\ &\leq \mathrm{dist}(S^*(a, v_i)) + \mathrm{dist}(a-1) = \mathrm{dist}(S^*(a-1, v_i)).\end{aligned}$$

Therefore, S' dominates $S^*(a-1, v_i)$.

Assume that $a - 1 \notin S^*(a-1, v_i)$. Because S^* is a solution of $T(1, l)$, $a-1$ is served by some trip in S^*. Further, $a-1$ can be served only by trips in $T(a-1, v_{a-1})$ and $v_{a-1} = v_i$. Therefore, $S^*(a, v_i)$ is a solution of $T(a-1, v_i)$. Since S dominates $S^*(a, v_i)$, S is a solution of $T(a-1, v_i)$. If S is in $\mathcal{X}(a-1, v_i)$ then the lemma is true. Otherwise, S is removed from $\mathcal{X}(a-1, v_i)$ because S is dominated by a solution $S' \in \mathcal{X}(a-1, v_i)$. This implies that S' dominates $S^*(a-1, v_i)$ and the lemma is proved. □

Lemma 3.13. *Let $i-1$ be a merge point in T such that there is no merge point in $A_{i-1} \setminus \{i-1\}$. For any solution S^* of $T(1, l)$, there is a solution S in $\mathcal{X}(i-1, v_{i-1})$ computed by Algorithm 3 such that S dominates $S^*(i-1, v_{i-1})$.*

Proof. Let $i_1, \ldots, i_r$ be the parents of $i-1$ such that $i_a < i_b$ if $a < b$. Then v_{i_a} is the unique source ancestor of i_a for each $1 \leq a \leq r$, $v_{i_r} = v_{i-1}$ and $i_1 = i$. We prove the following statement by induction: for every a with $1 \leq a \leq r$, there is a solution $S \in \mathcal{X}(i_a, v_{i-1})$ such that S dominates $S^*(i_a, v_{i-1})$. For $a = r$, from Lemma 3.12, the statement holds. Assume that the statement is true for $1 < a \leq r$ and we prove it for $a-1$. From Lemma 3.12, there is a solution S in $\mathcal{X}(i_{a-1}, v_{i_{a-1}})$ such that S dominates $S^*(i_{a-1}, v_{i_{a-1}})$. From the induction hypothesis, there is a solution $S' \in \mathcal{X}(i_a, v_{i-1})$ such that S' dominates $S^*(i_a, v_{i-1})$. Let

$S'' = S \cup S'$ as computed in Algorithm 3. Let $c = |T(i_{a-1}, v_{i-1})| + |D_{i-1}|$. Then

$$\begin{aligned} |\sigma''(S'')| &= \min\{c, |\sigma(S)| + |\sigma(S')|\} \\ &\geq \min\{c, |\sigma^*(S(i_{a-1}, v_{i_{a-1}}))| + |\sigma^*(S^*(i_a, v_{i-1}))|\} \\ &= |\sigma^*(S^*(i_{a-1}, v_{i-1}))| \end{aligned}$$

and

$$\begin{aligned} \mathrm{dist}(S'') &= \mathrm{dist}(S) + \mathrm{dist}(S') \\ &\leq \mathrm{dist}(S^*(i_{a-1}, v_{i_{a-1}})) + \mathrm{dist}(S^*(i_a, v_{i-1})) \\ &= \mathrm{dist}(S^*(i_{a-1}, v_{i-1})). \end{aligned}$$

That is, S'' dominates $S^*(i_{a-1}, v_{i-1})$. Therefore, there is a solution S in $\mathcal{X}(i, v_{i-1})$ such that S dominates $S^*(i, v_{i-1})$. By a similar argument for proving Lemma 3.12, there is a solution S in $\mathcal{X}(i-1, v_{i-1})$ such that S dominates $S^*(i-1, v_{i-1})$. □

Lemma 3.14. *For any solution (S^*, σ^*) of $T(1, l)$, there is a solution $(S, \sigma) \in \mathcal{X}(1, l)$ computed by Algorithm 3 such that S dominates S^*.*

Proof. If there is no merge point in T, then by Lemma 3.12, the lemma holds. If there is one merge point $i-1$ in T, then by Lemma 3.13, there is a solution $S \in \mathcal{X}(i-1, v_{i-1})$ such that S dominates $S^*(i-1, v_{i-1})$. Since $i-1$ is the only merge point of T, $v_{i-1} = l$, and D_{i-1} is the path consisting of all trips from $i-1$ to 1. By a similar argument for proving Lemma 3.12, there is a solution S in $\mathcal{X}(1, l)$ such that S dominates S^*.

Assume that $u_1, \ldots, u_s$, $1 < s$, are the merge points in T such that $u_a < u_b$ if $a < b$. For each u_a, $1 \leq a \leq s$, if the child of u_a is a merge point then let $w_a = u_a$, otherwise let w_a be the trip in D_{u_a} such that $T(w_a, v_{w_a})$ is a maximal branch and there is no merge point other than u_a in the path from u_a to w_a in T. We prove the following statement by induction: for $1 \leq a \leq s$, there is a solution S in $\mathcal{X}(w_a, v_{w_a})$ such that S dominates $S^*(w_a, v_{w_a})$.

For $a = s$, $A_{u_a} \setminus \{u_a\}$ does not contain any merge point. By Lemma 3.13, there is a solution $S \in \mathcal{X}(u_a, v_{u_a})$ such that S dominates $S^*(u_a, v_{u_a})$. Then by a similar argument for proving Lemma 3.12, there is a solution S in $\mathcal{X}(w_a, v_{w_a})$ such that S dominates $S^*(w_a, v_{w_a})$, implying the induction base. Assume that the statement holds for $1 < a \leq s$ and we prove it for $a-1$. If $A_{u_{a-1}} \setminus \{u_{a-1}\}$ does not have any merge point then by Lemma 3.13 and a similar argument for proving Lemma 3.12, the statement holds for $a-1$. Assume that $A_{u_{a-1}} \setminus \{u_{a-1}\}$ has some merge points. Notice that for every merge point $u_j \in A_{u_{a-1}} \setminus \{u_{a-1}\}$, $a \leq j$. Let $i_1, \ldots, i_r$ be the parents of u_{a-1}. By the induction hypothesis and a similar argument for proving Lemma 3.12, for every $1 \leq b \leq r$, there is a solution S in $\mathcal{X}(i_b, v_{i_b})$ such that S dominates $S^*(i_b, v_{i_b})$. By a similar argument for proving Lemma 3.13, there is a solution S in $\mathcal{X}(u_{a-1}, v_{u_{a-1}})$ such that S dominates $S^*(u_{a-1}, v_{u_{a-1}})$. Then by a similar argument

for proving Lemma 3.12, there is a solution S in $\mathcal{X}(w_{a-1}, v_{w_{a-1}})$ such that S dominates $S^*(w_{a-1}, v_{w_{a-1}})$. Therefore, the statement holds, implying the lemma. □

Theorem 3.23. *There is a dynamic programming algorithm that, given an instance $(N, \mathcal{A})$ of RSTwo satisfying Conditions C1-C5 and the transitive serve relation, computes a solution (S, σ) for $(N, \mathcal{A})$ in $O(M + l^3)$ time, where M is the size of the ridesharing instance which contains road network N and l trips.*

Proof. Let S^* be a solution for $T(1, l)$ with the minimum $\mathrm{dist}(S^*)$. By Lemma 3.14, Algorithm 3 finds a solution S with $\mathrm{dist}(S) \leq \mathrm{dist}(S^*)$ in $\mathcal{X}(1, l)$. From Lemma 3.10, Algorithm 3 computes a solution (S, σ) for $(N, \mathcal{A})$ with $\mathrm{dist}(S)$ minimized.

Algorithm 3 computes $\mathcal{X}(i, v_i)$ for every trip i. Since $\mathcal{X}(i+1, v_{i+1})$ is non-dominating, the solutions of $\mathcal{X}(i+1, v_{i+1})$ can be listed as $S_1, S_2, \ldots$ such that $|\sigma(S_a)| < |\sigma(S_b)|$ and $\mathrm{dist}(S_a) > \mathrm{dist}(S_b)$ for $a < b$. Hence, there are $O(l)$ solutions in $\mathcal{X}(i+1, v_{i+1})$ because $|\sigma(S_a)| \leq l$ for every S_a in $\mathcal{X}(i+1, v_{i+1})$. For every $S_a \in \mathcal{X}(i+1, v_{i+1})$, the solution $S'_a \in \mathcal{X}(i, v_i)$ can be computed in $O(l)$ time, so it takes $O(l^2)$ time to compute all solutions in $\mathcal{X}(i, v_i)$. It takes $O(l)$ time to make $\mathcal{X}(i, v_i)$ non-dominating. Therefore, it takes $O(l^3)$ time to process all of the l trips.

In Algorithm 3, Procedure Merge is called $O(l)$ times. We play a small trick to reduce the running time of the procedure: we compute $|\sigma''(S'')|$ before $\sigma''(j)$ is actually decided for each $j \in S''$. This allows us to get a non-dominating set of solutions. Then we decide $\sigma''(j)$ only for non-dominating solutions ($O(l)$ many) instead of all $S'' = S \cup S'$ ($O(l^2)$ many). In each call, it takes $O(l)$ time to compute $|T(i_{a-1}, v_{i-1})| + |D_{i-1}|$, and there are $O(l^2)$ solutions $S \cup S'$ for $S \in \mathcal{X}(i_{a-1}, v_{i_{a-1}})$ and $S' \in \mathcal{X}(i_a, v_{i-1})$ because each of $\mathcal{X}(i_{a-1}, v_{i_{a-1}})$ and $\mathcal{X}(i_a, v_{i-1})$ is non-dominating, and thus has $O(l)$ solutions. It takes $O(1)$ time to compute $\mathrm{dist}(S'')$ and $|\sigma''(S'')|$. Therefore, it takes $O(l^2)$ time to compute $O(l^2)$ solutions for $\mathcal{X}(i_{a-1}, v_{i-1})$ and $O(l^2)$ time to make $\mathcal{X}(i_{a-1}, v_{i-1})$ non-dominating. Then there are $O(l)$ non-dominating solutions in $\mathcal{X}(i_{a-1}, v_{i-1})$. For each $S'' \in \mathcal{X}(i_{a-1}, v_{i-1})$, it takes $O(l)$ time to compute $\sigma''(j)$ for every $j \in S''$. Therefore, each execution of Procedure Merge takes $O(l^2)$ time and total time for merge operations is $O(l^3)$ time. So the total time of Algorithm 1 is $O(l^3)$.

It takes $O(M)$ time to compute $\vec{G}$ (Lemma 3.11). The preprocessing for rearranging labels of trips takes $O(l)$ time. Therefore, the theorem holds. □

If we set $\mathrm{dist}(P_i) = 1$ for every $i \in \mathcal{A}$ (effective changing the definition of dominating relation to: *S dominates S'* if $|\sigma(S)| \geq |\sigma(S')|$), then by Theorem 3.23, the following result holds.

Theorem 3.24. *There is a dynamic programming algorithm that, given an instance $(N, \mathcal{A})$ of RSOne satisfying Conditions C1-C5 and the transitive serve relation, computes a solution (S, σ) for $(N, \mathcal{A})$ in $O(M + l^3)$ time, where M is the size of the ridesharing instance which contains road network N and l trips.*

3.3.4 Greedy algorithm for RSOne

In this section, we give a greedy algorithm for the ridesharing problem RSOne. A straightforward implementation of this algorithm runs in $O(M + l^2)$ time. With a more efficient implementation, the running time of the algorithm can be improved to $O(M + l \log l)$, which is much more efficient than Algorithm 3 with a running time $O(M + l^3)$.

Algorithm description.

Since the vertex set $V(T)$ is the trips of $\mathcal{A}$, we also call a vertex in T a trip. Our algorithm processes every trip in T starting from a source trip in T with the largest label. When a source trip y in T is processed, y is included in a partial solution (S, σ). That is, η_y is assigned as a driver and serves as many trips in D_y that are not served by S and are closest to η_y as possible, which is the set $N(y, c, S)$ of c such trips as defined in Subsection 3.3.3. In general, when any trip u is assigned as a driver, $\sigma(u) = N(u, c, S)$, where $c = \min\{\lambda_u + 1, |D_u \setminus \sigma(S)|\}$. A trip x is *marked* if x is assigned as a driver or a passenger by the algorithm. Each trip x which is not a source of T is processed only if all ancestors of x have been marked by the algorithm and $|\sigma(v)| = \lambda_v + 1$ for every $v \in S \cap A_x$. When a trip x is processed, a trip u with the largest capacity λ_u is selected from $A_x \setminus S$ and assigned as a driver in S.

At any execution point of the algorithm, whenever a trip has been assigned as a driver, it remains as a driver throughout the algorithm. On the other hand, a trip that has been assigned as a passenger can be changed to a driver when a new trip is processed. The pseudo code of the algorithm is given in Algorithm 4 (Greedy Assignment).

Algorithm 4 Greedy Assignment

Input: An inverse tree T of $\vec{G}$ with l trips.
Output: A solution (S, σ) for an instance $(N, \mathcal{A})$ of RSOne.
$S = \emptyset$; let $v_1, \ldots, v_r$ be the sources in T s.t. $v_i < v_j$ for $i < j$;
for $i = r$ to 1 **do**
    $x = v_i$; // initialization
    while $(x = v_i) \vee$ (all trips in $A_x \setminus \{x\}$ are marked) **do** // process x
        let u be a trip in $A_x \setminus S$ with the largest λ_u;
        if $u \neq x$ **then** // u has been assigned as a passenger
            let $\eta_k \in S$ s.t. $u \in \sigma(k)$;
            $\sigma(k) = (\sigma(k) \setminus \{u\}) \cup \{x\}$; mark x;
        end if
        $c = \min\{\lambda_u + 1, |D_u \setminus \sigma(S)|\}$; $S = S \cup \{\eta_u\}$;
        $\sigma(u) = N(u, c, S)$; mark all trips in $\sigma(u)$;
        if all trips in D_u are marked **then**
            break the **while** loop;
        else
            let x be the unmarked trip in D_u with the minimum dist(u, x) in T;
        end if
    end while
end for

Analysis of algorithm

Given a component T (inverse tree) of $\vec{G}$ with l trips and a partial solution (S, σ) for $T(1, l)$, we will use the following notations in the proof:

- $P(S) = \sigma(S) \setminus S$ is the set of passengers served by S;
- for $A \subseteq T(1, l)$, $S(A) = S \cap A$ is the set of drivers that are also trips of A; and
- for a trip i in $T(i, l)$, $\text{free}(i) = \lambda_i - |\sigma(i)| + 1$ is the number of additional trips i can serve.

For clarity, when a solution is denoted by (S', σ') or (S^*, σ^*), $\text{free}(i)$ is denoted as $\text{free}'(i)$ or $\text{free}^*(i)$ with respect to (S', σ') or (S^*, σ^*).

Lemma 3.15. *Given a simplified serve relation graph T of $\vec{G}$, Algorithm 4 finds a solution (S, σ) for $T(1, l)$ with the minimum $|S|$.*

Proof. Assume that the drivers in S are indexed such that $u_i \in S$ is the i^{th} driver included in S by Algorithm 4. Let (S^*, σ^*) be an optimal solution for $T(1, l)$. Let u_j be the driver in $S \setminus S^*$ with the smallest index and x_j be the trip processed by the algorithm for including u_j in S, implying $u_j \in A_{x_j}$. Let (S_F, σ_F) be the partial solution at the end of the iteration including u_{j-1} in S_F by the algorithm (note that $S_F = \{u_1, \ldots, u_{j-1}\}$). From the algorithm, x_j is either a source or a descendant of u_{j-1}. If x_j is a source, then $u_j = x_j$ must be included in both S and S^*, a contradiction. Thus, x_j is a descendant of u_{j-1} (see Figure 3.8). Then, $A_{x_j} \setminus \{x_j\} = \sigma_F(S_F(A_{x_j}))$ because all trips in $A_{x_j} \setminus \{x_j\}$ are marked. Further from the algorithm, $\sum_{u_i \in S_F(A_{x_j})}(\lambda_{u_i} + 1) = |A_{x_j} \setminus \{x_j\}|$. Notice that $S_F \subseteq S^*$. We use S_F^* for S_F when we refer the drivers of S_F as the drivers in S^*, i.e. $S_F^* = S^* \cap S_F = \{u_1, \ldots, u_{j-1}\}$.

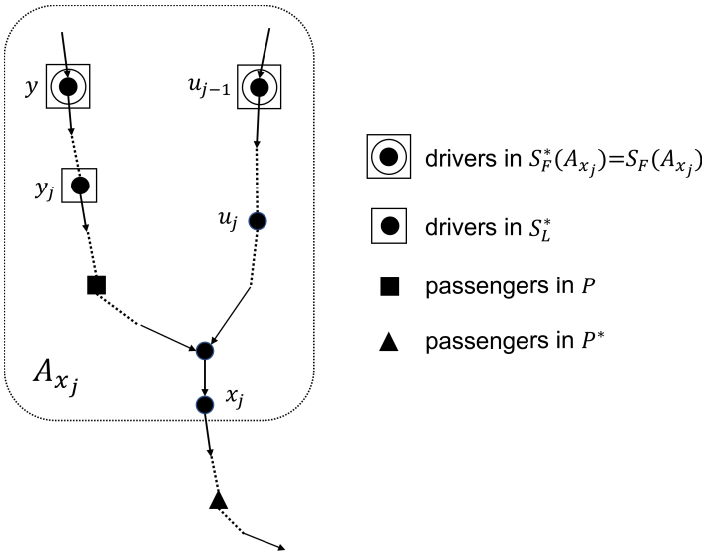


Figure 3.8: Modify (S^*, σ^*).

We modify (S^*, σ^*) to construct another optimal solution (S', σ') s.t. $\{u_1, \ldots, u_j\} \subseteq S'$. Let $S_L^* = S^*(A_{x_j}) \setminus S_F^*(A_{x_j})$. We replace a driver $v \in S_L^*$ by u_j to construct (S', σ'). Let v be a driver in S_L^*. Then the following property holds.

(A) Since $\lambda_{u_j} = \max_{u \in A_{x_j} \setminus S_F} \lambda_u$ and $v \in S^*(A_{x_j})$, $\lambda_v \leq \lambda_{u_j}$.

The construction is divided into two cases.

Case 1: S_L^* does not have any driver in $A_{x_j} \setminus \{x_j\}$. Then, all trips of $A_{x_j} \setminus \{x_j\}$ are served by $S_F^*(A_{x_j})$. From $S_F^* = S_F$ and Algorithm 4,

$$\sum_{u_i \in S_F^*(A_{x_j})} (\lambda_{u_i} + 1) = \sum_{u_i \in S_F(A_{x_j})} (\lambda_{u_i} + 1) = |A_{x_j} \setminus \{x_j\}|.$$

Therefore, x_j is not served by any driver in $S^*(A_{x_j} \setminus \{x_j\})$, implying $x_j \in S^*$. Thus, $S_L^* = \{x_j\}$. If $u_j = x_j$ then u_j is in both S and S^*, a contradiction. Thus, $u_j \neq x_j$, $u_j \in A_{x_j} \setminus \{x_j\}$, and there are drivers $y \in S_F(A_{x_j})$ and $y^* \in S_F^*(A_{x_j})$ such that $u_j \in \sigma_F(y)$ and $u_j \in \sigma^*(y^*)$. Let $S' = (S^* \setminus \{x_j\}) \cup \{u_j\}$ (replace x_j by u_j), $\sigma'(y^*) = (\sigma^*(y^*) \setminus \{u_j\}) \cup \{x_j\}$, $\sigma'(u_j) = \{u_j\} \cup (\sigma^*(x_j) \setminus \{x_j\})$ (this can be done due to Property (A)), and for each $v \in S' \setminus \{u_j, y^*\}$, $\sigma'(v) = \sigma^*(v)$. Then, (S', σ') is a solution for $T(1, l)$ with $|S'| = |S^*|$ and $\{u_1, \ldots, u_{j-1}, u_j\} \subseteq S'$.

Case 2: S_L^* has a driver v_j in $A_{x_j} \setminus \{x_j\}$ (see Figure 3.8). The construction has two rounds. In round 1, we modify σ^* so that S_L^* only serves the passengers in D_{x_j} and $S_F^*(A_{x_j})$ serves all passengers in $A_{x_j} \setminus (S_L^* \cup \{x_j\})$. In round 2, we replace some driver $v_j \in S_L^*$ by u_j to get (S', σ').

In round 1, let $P^* = P(S_F^*(A_{x_j})) \setminus P(S_F(A_{x_j}))$ be the set of trips which are passengers of $S_F^*(A_{x_j})$ but not passengers of $S_F(A_{x_j})$ (see Figure 3.8). Then $P^* \subseteq D_{x_j}$ because $P(S_F(A_{x_j})) = A_{x_j} \setminus (\{x_j\} \cup S_F^*(A_{x_j}))$. Let $P = P(S_F(A_{x_j})) \cap P(S_L^*)$ be the set of trips which are passengers of both $S_F(A_{x_j})$ and S_L^* (see Figure 3.8). We modify σ^* so that all passengers of P are served by $S_F^*(A_{x_j})$ and a subset of passengers of P^* are served by S_L^*. Notice that $S_L^* \subseteq A_{x_j}$, $P \subseteq A_{x_j} \setminus \{x_j\}$ and $|P^*| \leq |P| + |S_L^*|$. The modification of σ^* goes as follows. Let Q be a subset of P^* such that $|Q| = \min\{|P|, |P^*|\}$. For every $v \in S_L^*$, $\sigma^*(v) = \sigma^*(v) \setminus P$ and then assign at most free$^*(v)$ passengers of Q to $\sigma^*(v)$ (remove all passengers of P from $\sigma^*(S_L^*)$ and make all passengers of Q to be served by S_L^*; S_L^* only serves passengers in D_{x_j}). For every $v \in S_F^*(A_{x_j})$, $\sigma^*(v) = \sigma_F(v) \setminus S_L^*$ and then assign at most free$^*(v)$ passengers of $P^* \setminus Q$ to $\sigma^*(v)$ (remove passengers of Q from $\sigma_F^*(A_{x_j})$ and make all passengers of $P \cup P^* \setminus Q$ to be served by $S_F^*(A_{x_j})$; all passengers in $A_{x_j} \setminus (S_L^* \cup \{x_j\})$ are served by S_F^*). The modification can be done regardless of $|P| > |P^*|$ or $|P| < |P^*|$ because $S_F^*(A_{x_j}) = S_F(A_{x_j})$ can serve all trips of $A_{x_j} \setminus \{x_j\}$. As a result, S^* does not change, $S_F^*(A_{x_j})$ serves all trips in $A_{x_j} \setminus (S_L^* \cup \{x_j\})$ and at most $|S_L^*|$ trips in D_{x_j}, and every trip in $P(S_L^*)$ is in D_{x_j}.

In round 2, we replace some driver $v_j \in S_L^* \setminus \{x_j\}$ by u_j to construct (S', σ'). The following property is true for every $v_j \in S_L^* \setminus \{x_j\}$:

(B) There is a $y \in S_F(A_{x_j})$ such that $v_j \in \sigma_F(y)$ because $A_{x_j} \setminus \{x_j\} = \sigma(S_F(A_{x_j}))$, $v_j \in A_{x_j} \setminus \{x_j\}$ and $v_j \notin S_F(A_{x_j})$. Notice that y is also in $S_F^*(A_{x_j})$ and in round 1 for modifying σ^*, $\sigma^*(y) = \sigma_F(y) \setminus S_L^*$ and at most $\text{free}^*(y)$ passengers of $P^* \setminus Q$ ($P^* \subseteq D_{x_j}$) are assigned to $\sigma^*(y)$. Since $v_j \in S_L^*$, either (B1) $\text{free}^*(y) \geq 1$ or (B2) there is a $w \in D_{x_j}$ such that $w \in \sigma^*(y)$.

Let $S' = (S^* \setminus \{v_j\}) \cup \{u_j\}$ (replace v_j by u_j) and let $\sigma'(v) = \sigma^*(v) \setminus \{u_j\}$ for $v \in S' \setminus \{u_j\}$. Since $\sigma^*(v_j)$ only serves trips in D_{x_j} and $\lambda_{v_j} \leq \lambda_{u_j}$ (Property (A)), $\sigma'(u_j)$ can serve all passengers of $\sigma^*(v_j)$. Let $\sigma'(u_j) = \{u_j\} \cup (\sigma^*(v_j) \setminus \{v_j\})$. Then v_j is the only trip not served by S' right now. If Property (B1) holds, then let $\sigma'(y) = \sigma^*(y) \cup \{v_j\}$. If Property (B2) holds, let $\sigma'(y) = (\sigma^*(y) \setminus \{w\}) \cup \{v_j\}$. Then w is the only trip not served by S'. Since $u_j \notin S^*$, there is a $v \in S^*(A_{x_j})$ such that $u_j \in \sigma^*(v)$. If $v \in S_F^*(A_{x_j})$ then let $\sigma'(v) = (\sigma^*(v) \setminus \{u_j\}) \cup \{w\}$. Otherwise, $v \in S_L^*$. We select this v as v_j. Then $|\sigma'(u_j)| = |\sigma^*(v_j)| - 1$. From this and Property (A),

$$\text{free}'(u_j) = \lambda_{u_j} - |\sigma'(u_j)| + 1 = \lambda_{u_j} - (|\sigma^*(v_j)| - 1) + 1 \geq \lambda_{u_j} - \lambda_{v_j} + 1 \geq 1.$$

Let $\sigma'(u_j) = \sigma'(u_j) \cup \{w\}$. As a result, we obtain a solution (S', σ') with $|S'| = |S^*|$ and $\{u_1, \ldots, u_{j-1}, u_j\} \subseteq S'$.

By the modification of (S^*, σ^*), we get an optimal solution (S', σ') with $\{u_1, \ldots, u_j\} \subseteq S'$. Then the size of S_F is increased by at least one, and by repeating the above argument for each u_j one by one, we can get an optimal solution (S', σ') with $S' = S$. □

Theorem 3.25. *There is an algorithm that, given a ridesharing instance $(N, \mathcal{A})$ of RSOne satisfying Conditions C1-C5 and the transitive serve relation, computes a solution (S, σ) for $(N, \mathcal{A})$ in $O(M + l \log l)$ time, where M is the size of the ridesharing instance which contains road network N and l trips.*

Proof. By Lemma 3.15, Algorithm 4 computes a solution (S, σ) of T with minimum $|S|$, where T of $\vec{G}$ is constructed based on $(N, \mathcal{A})$. By Lemma 3.10, (S, σ) is an optimal solution for $(N, \mathcal{A})$.

By a straightforward approach, it takes $O(l)$ time to check if all vertices in $A_x \setminus \{x\}$ are marked for a non-source trip x in each iteration and $O(l^2)$ time in all iterations of the while loop. The time for the check can be improved to $O(1)$ for one iteration by the following observation: when a non-source trip x is selected after a source v_i (as defined in Algorithm 4) has been processed and v_{i-1} has not, if $x > v_{i-1}$ then all vertices in $A_x \setminus \{x\}$ are marked. Therefore, the time for checking the conditions in all iterations of the while loop is $O(l)$.

By a straightforward approach, it takes $O(l)$ time to find a trip $u \in A_x \setminus S$ with the largest λ_u in each iteration (Line 7) and $O(l^2)$ time in all iterations of the while loop. A more

efficient way to find the trip u is to use a max-heap, based on the capacity of passengers in $A_x \setminus S$. When a new source v_i is processed, create a max-heap H_i associated with v_i. Before any new source is processed, add any newly assigned passengers to H_i. When a passenger in H_i becomes a driver, it is removed from H_i. When a non-origin x is processed, we check if x is a merge point in T (defined in the end of Subsection 3.3.2) or a descendant of a merge point. If it is the case, merge all the heaps associated with sources that are ancestors of x into one heap. To keep track of the created heaps, we push each heap's associated source in a stack ST1 in the order the heaps are created. We check the two top elements of ST1: if they are ancestors of x, merge their associated heaps into one heap H. After the merge, both sources are associated to H. Then, remove the top element of ST1. Repeatedly checking the top two elements in ST1 until the second top element is not an ancestor of x. The heap associated with the source in the top element of ST1 is used to find the trip u with largest λ_u. The heap can be implemented using binomial heap [29]. Each of find-max, delete-max, insert and merge operations can be done in $O(\log |H|)$, where $|H| < l$ is the size of a heap H. Hence, using max-heaps, it takes $O(M + l \times \log l)$ time to find u with the largest λ_u in all iterations of the while loop.

By a straightforward approach, it takes $O(l)$ time to compute $c = \min\{\lambda_u+1, |D_u \setminus \sigma(S)|\}$ and $\sigma(u) = N(u, c, S)$ (Lines 12-13) in each iteration and $O(l^2)$ time in all iterations of the while loop. A more efficient approach to compute the above is as follows: For a trip u (as found at Line 7), let y_u be the unmarked trip in D_u closest to u. We try to check only unmarked trips of D_u one by one starting from y_u. Once an unmarked trip is checked, the trip is changed to marked and included in $\sigma(u)$. After u is processed, if D_u has an unmarked trip then y_u is updated and pushed into a stack ST2 to assist computing $y_{u'}$ for a trip u' selected for processing later. Let u be a trip in $A_x \setminus S$ with the largest λ_u in the algorithm. There are two cases:

1. u is a source. If the child of u is unmarked then take this child as y_u and perform Operation specified below (next paragraph). Assume that the child of u is marked. If ST2 is empty then include u in $\sigma(u)$ and the algorithm continues. Otherwise, remove the top element y of ST2, $y_u = y$ and perform Operation.

2. u is not a source ($u = x$ or $u \neq x$). If the child of x is unmarked then take this child as y_u and perform Operation. Assume that the child of x is marked. If ST2 is empty then include u in $\sigma(u)$ and the algorithm continues. Otherwise, remove the top element y of ST2, $y_u = y$ and perform Operation.

- **Operation:** Let u be a trip in $A_x \setminus S$ with the largest λ_u in the algorithm and y_u be the unmarked trip in D_u closest to u. We check the trips of D_u one by one from y_u. Let y be a trip checked. If y is not marked then mark y and include y in $\sigma(u)$. If y is the top element of stack ST2 then remove y from ST2. The check stops if (a) y is the sink of T, (b) $|\sigma(u)| = \lambda_u + 1$ or (c) y is marked.

– Case (a). The algorithm continues (select the next u for processing).
– Case (b). If there is an unmarked trip in D_u then update y_u to the unmarked trip in D_u closest to u. If y_u is not the top element of ST2 then we push y_u into stack ST2. The algorithm continues.
– Case (c). If stack ST2 is empty then the algorithm continues. Otherwise, let y be the top element of ST2. We remove y from ST2, update $y_u = y$ and goto Operation.

In the above processing, each unmarked trip is checked once and each marked trip is checked at most twice. Therefore, the total time for computing $c = \min\{\lambda_u + 1, |D_u \setminus \sigma(S)|\}$ and $\sigma(u) = N(u, c, S)$ of all trips is $O(l)$.

It takes $O(1)$ time for operations other than the above in each iteration and $O(l)$ time in all iterations of the while loop. Therefore, by straightforward implementations, the running time of Algorithm 2 is $O(l^2)$, and by more efficient implementations, the running time is improved to $O(l \log l)$. It takes $O(M)$ time to compute $\vec{G}$. Thus, the theorem holds. □

3.4 Ridesharing problem without the stop frequency condition

In this section, we first present two approximation algorithms for the ridesharing problem RSOne satisfying Conditions C1-C3 and C5; and such a variant is denoted as *RSOneStop*. These two approximation algorithms are modified from algorithms proposed by Kutiel and Rawitz [65] for the maximum carpool matching problem (MCMP). Then, we present our novel algorithm for the ridesharing problem RSOneStop, which is more efficient compared to the first two approximation algorithms. We assume all trips have the same destination of C1, but the algorithms can also apply to all trips have the same origin with small modifications. Let $\lambda = \max_{i \in \mathcal{A}} \lambda_i$ be the largest capacity of all trips in $\mathcal{A}$. All three algorithms have the same $\frac{\lambda+2}{2}$-approximation ratio.

3.4.1 Approximation algorithms based on MCMP

Recall that an instance of the maximum carpool matching problem (MCMP) consists of a digraph $\vec{G}(V, E)$, a capacity function $c : V \to \mathbb{N}$, and a weight function $w : E \to \mathbb{R}^+$, where the vertices of V represent the individuals and an edge $(u, v) \in E(\vec{G})$ implies v can serve u. We are only interested in the unweighted case, that is, $w(u, v) = 1$ for every $(u, v) \in E(\vec{G})$. Every $v \in V(\vec{G})$ can be assigned as a driver or passenger. The goal of MCMP is to find a set of drivers $S \subseteq V$ to serve all V such that the number of passengers is maximized.

A subset $M \subseteq E(\vec{G})$ of edges is a *feasible* solution to MCMP if the graph induced by M is a set $\mathcal{S}$ of vertex-disjoint stars in $\vec{G}$. Let S_v be a star in $\mathcal{S}$ rooted at center vertex v, and leaves of S_v is denoted by $P_v = V(S_v) \setminus \{v\}$. For each star $S_v \in \mathcal{S}$, $\text{outdeg}(v) = 0$, and

indeg$(u) = 0$ and outdeg$(u) = 1$ (directs to v) for every $u \in P_v$. The center vertex of each star S_v is assigned as a driver and the leaves are assigned as passengers. This is equivalent to the notation $\sigma(v) = \{v\} \cup P_v = V(S_v)$. The set of edges in $\mathcal{S}$ is called a *carpool matching* CM (CM is referred to as a matching for short). An edge in CM is called a *matched* edge. Notice that $|CM|$ equals to the number of passengers, and we want to maximize $|CM|$. Here, the term matching is different from the matching definition stated in Section 2.2.

For a carpool matching CM and a subset $V' \subseteq V$ of vertices, let $CM(V')$ be the set of edges in CM incident to V'. The *in-neighbors* of a vertex v is defined as $N^{in}(v) = \{u \mid (u, v) \in E(\vec{G})\}$, and the set of edges entering v (v is the head) is defined as *in-edges* $E^{in}(v) = \{(u, v) \mid (u, v) \in E(\vec{G})\}$. Table 3.2 lists the basic notation and definition for this section.

Notation	Definition
$\mathcal{S}$	A set of vertex-disjoint stars in $\vec{G}(V, E)$ (solution to MCMP)
S_v	A subgraph of $\vec{G}$ that is a star rooted at vertex v
P_v	$P_v = V(S_v) \setminus \{v\}$, the set of leaves of star S_v
$c(v)$	Capacity of vertex v (equivalent to λ_v in Table 2.1)
CM	The set of edges in $\mathcal{S}$, namely $E(S)$
$CM(V')$	The set of edges in CM incident to a set V' of vertices
$N^{in}(v)$	The set of *in-neighbors* of v, $N^{in}(v) = \{u \mid (u, v) \in E(\vec{G})\}$
$E^{in}(v)$	The set of edges entering v, *in-edges* $E^{in}(v) = \{(u, v) \mid (u, v) \in E(\vec{G})\}$
δP_v	The number of stops required for v to pick-up all of P_v

Table 3.2: Common notation and definition for MCMP used in this section.

There is a major difference between MCMP and RSOne in general. In MCMP, it is assumed that when an individual v is assigned as a driver, v can serve any combination of upto $c(v)$ passengers whose corresponding vertices are incident to v. However, this is not true for the ridesharing problem in general. For example, suppose there are three trips v_1, v_2, v_3 such that (v_2, v_1) and (v_3, v_1) are in $E(\vec{G})$ and $c(v_1) = 2$ ($\lambda_{v_1} = 2$), namely, v_1 can serve v_2 and v_3. A solution $\mathcal{S}$ to MCMP would assign v_1 as a driver to serve both v_2 and v_3. However, this solution may not be a valid solution to RSOne for the following reasons:

1. The detour z_{v_1} of v_1 is not large enough to serve both v_2 and v_3. Unless we can assume z_{v_1} is unlimited or $o_{v_2}, o_{v_3}, d_{v_2}$ and d_{v_3} are on a preferred path P_{v_1} of v_1. The latter requires the detour condition C2 to be satisfied.

2. Even if C2 is satisfied, v_1 could have two preferred paths, one that passes through o_{v_2} and d_{v_2} and the other passes through o_{v_3} and d_{v_3}. Unless there is only one preferred path (satisfying C3), $\mathcal{S}$ still cannot apply to RSOne.

3. Even if C2 and C3 are satisfied, the time constraints may be violated when both v_2 and v_3 are assigned to v_1, so we need to have C5 be satisfied as well.

Two approximation algorithms (*EdgeSwap* and *StarImprove*) are presented in [65]; both can achieve $\frac{1}{2}$-approximation ratio for MCMP, that is, the number of passengers found by the algorithms is at least half of that for the optimal solution. The EdgeSwap algorithm actually can be applied to RSOne satisfying Conditions C2-C3 and C5, but the algorithm is very inefficient as described in the next paragraph. Although we only show that the StarImprove algorithm can be applied to RSOneStop, we believe that:

Conjecture 3.1. *StarImprove can be applied to RSOne satisfying Conditions C2-C3 and C5 with approximation ratio* $\frac{\lambda+2}{2}$.

EdgeSwap. The EdgeSwap algorithm requires the input instance to have a bounded degree graph (or the largest capacity λ is bounded by a constant) to have a polynomial running time. The idea of EdgeSwap is to:

- swap a subset $CM' \subseteq CM$ of i edges with a subset $E' \subseteq (E \setminus CM)$ of $i+1$ edges such that $CM = (CM \setminus CM') \cup E'$ is feasible for $1 \leq i \leq k$, and k is a constant integer.

The running time of EdgeSwap is in the order of $O(|E|^{2k+1})$. A small modification of EdgeSwap is required so that it can apply to RSOneStop. Let $\mathcal{S}$ be the set of stars induced by CM. For each star $S_v \in \mathcal{S}$, let $\sigma(v) = V(S_v)$. During the feasibility check of CM, make sure that $\sigma(v)$ does not require more than δ_v to pick-up and more than δ_v to drop-off all of P_v. Then, EdgeSwap can be applied to RSOneStop to achieve $\frac{\lambda+2}{2}$-approximation ratio in $O(l^{2\lambda})$ time, which may not be practical even if λ is a small constant.

StarImprove. Let $(\vec{G}(V,E), c, w)$ be an instance of MCMP. Let $\mathcal{S}$ be the current set of stars found by StarImprove and CM be the set of matched edges. The idea of the StarImprove algorithm is to iteratively check in a *for-loop* for every vertex $v \in V(\vec{G})$:

- check if there exists a star S_v with $E(S_v) \cap CM = \emptyset$ such that the resulting set of stars $CM = (CM \setminus CM(V(S_v))) \cup S_v$ gives a larger cardinality matching.

Such a star S_v is called an *improvement* and $|P_v| \leq c(v)$. The algorithms stops when no improvement can be found.

Given an instance $(N, \mathcal{A})$ of RSOneStop, the StarImprove algorithm cannot apply to $(N, \mathcal{A})$ directly. For example, suppose v can serve u_1 and u_2 with $\lambda_v = 2$ and $\delta_v = 1$. The StarImprove assigns v as a driver to serve both u_1 and u_2. However, if u_1 and u_2 have different origins ($o_v \neq o_{u_1} \neq o_{u_2}$), this assignment is not valid for $(N, \mathcal{A})$. Hence, we need to modify StarImprove for computing a star.

For a vertex v and matching CM, let $N^{in}_{\text{-}CM}(v) = \{u \mid (u,v) \in E^{in}(v)$ and u is not incident to any edge of CM$\}$. For a star S_v, let δP_v be the number of stops required for v to pick-up P_v. Suppose the in-neighbors $N^{in}_{\text{-}CM}(v)$ are partitioned into $g_1(v), \ldots, g_m(v)$ groups such that trips with same origins are grouped together. When stop frequency constraint is

not satisfied, finding a star S_v with maximum $|P_v|$ is similar to solving a fractional knapsack instance using a greedy approach as shown in Algorithm 5 (Compute Star). The idea is, in each iteration, to select the largest group of in-neighbors $N^{in}_{\neg CM}(v)$ until the capacity $c(v)$ is reached.

Algorithm 5 Compute Star

The in-neighbors $N^{in}_{\neg CM}(v)$ are already partitioned into $g_1(v), \ldots, g_m(v)$.
$P_v = \emptyset$; $c = c(v)$; $\delta P_v = 0$;
if $\exists$ a group $g_j(v)$ s.t. $o_u = o_v$ for any $u \in g_j(v)$ **then**
    // there is at most one such group $g_j(v)$
    Let $g'_j(v) \subseteq g_j(v)$ be a maximum subset of $g_j(v)$ such that $|g'_j(v)| \leq c$.
    $P_v = P_v \cup g'_j(v)$; $c = c - |g'_j(v)|$;
end if
while $c > 0$ and $\delta P_v < \delta_v$ **do**
    Select $g_i(v) = \text{argmax}_{1 \leq i \leq m} |g_i(v) \setminus P_v|$;
    if $|g_i(v) \setminus P_v| = 0$ **then break** the **while** loop;
    Let $g'_i(v) \subseteq g_i(v)$ be a maximum subset of $g_i(v)$ such that $|g'_i(v)| \leq c$.
    $P_v = P_v \cup g'_i(v)$; $c = c - |g'_i(v)|$; $\delta P_v = \delta P_v + 1$;
end while
return the star S_v induced by $\{v\} \cup P_v$;

Lemma 3.16. *Let v be the trip being processed and S_v be the star found by Algorithm 5 with respect to matching CM. Then $|P_v| \geq |P'_v|$ for any star S'_v such that $E(S'_v) \cap CM = \emptyset$.*

Proof. Assume for contradiction that $|P'_v| > |P_v|$ for some star S'_v s.t. $E(S'_v) \cap CM = \emptyset$. Since $|P'_v| > |P_v|$, $c(v) > |P_v|$. For any trip $u \in N^{in}_{\neg CM}(v)$, let $g_{i_u}(v)$ be the group s.t. $u \in g_{i_u}(v)$. Let $u' \in P'_v \setminus P_v$. Note that $o_{u'} \neq o_v$; otherwise, u' would have been included in P_v by the greedy algorithm, and hence, $\delta_v > 0$. From $c(v) > |P_v|$ and $\delta_v > 0$, the greedy algorithm must have executed the while-loop and checked all the groups in decreasing order of their size $|g_i(v) \setminus P_v|$, and $\delta P_v = \delta_v$ at the end of the while-loop. Because $c(v) > |P_v|$, $|P_v \cap g_{i_w}(v)| = |g_{i_w}(v)| \geq |P'_v \cap g_{i_w}(v)|$ for any $w \in P'_v \cap P_v$. Since groups are checked in decreasing order of their size, $|P_v \cap g_i(v)| \geq |P'_v \cap g_{i_u}(v)|$ for every group $g_i(v)$ and every $u \in P'_v \setminus P_v$. Recall that $\delta P_v = \delta_v$. Hence, $|P_v| \geq |P'_v|$, which is a contradiction. □

Definition 3.1. A star S_v rooted at v is an *improvement* with respect to matching CM if $|P_v| \leq c(v), \delta P_v \leq \delta_v$ and $|E(S_v)| - \sum_{(u,v) \in E(S_v)} |CM(u)| > |CM(v)|$.

Definition 3.1 is equivalent to the original definition in [65], except the former is for the unweighted case and stop constraint. When an improvement star S_v is found, the current matching CM is increased by exactly $|E(S_v)| - \sum_{(u,v) \in E(S_v)} |CM(u)|$ edges (an augmentation $CM = (CM \setminus \cup_{(u,v) \in E(S_v)} CM(u)) \cup E(S_v)$ is performed). For a vertex v and a subset $S \subseteq E^{in}(v)$, let $N^{in}_S(v) = \{u \mid (u,v) \in S\}$.

Lemma 3.17. *Let CM be the current matching and v be a vertex with no improvement with respect to CM. Let S_v be a star with $E(S_v) \subseteq E^{in}(v)$ such that $|E(S_v)| \leq c(v)$ and $\delta P_v \leq \delta_v$.*

Then, $|E(S_v)| \leq |CM(v)| + |CM(N^{in}_{S_v}(v))|$. *Further, if any star* S_v *found by Algorithm 5 with respect to* CM *is not an improvement, then no other* S'_v *is an improvement.*

Proof. When no improvement exists for a vertex v, we get $|E(S_v)| - |CM(N^{in}_{S_v}(v))| = |E(S_v)| - \sum_{(u,v)\in E(S_v)} |CM(u)| \leq |CM(v)|$ by Definition 3.1.

To maximize $|E(S_v)|$, we need to maximize $|E(S_v)| - \sum_{(u,v)\in E(S_v)} |CM(u)|$, which can be done by selecting only in-neighbors of v that are not incident to any matched edges. This is because for any $(u, v) \in E(S_v)$ s.t. u is incident to a matched edge, $|CM(u)| \geq 1$. In other words, including such a vertex u cannot increase $|E(S_v)| - \sum_{(u,v)\in E(S_v)} |CM(u)|$. Algorithm 5 considers only in-neighbors $N^{in}_{\neg M}(v) = \{i \mid i \in N^{in} \setminus V(M)\}$. By Lemma 3.16, $|P_v|$ is maximized among all stars rooted at v w.r.t. CM. Hence, the lemma holds. □

Lemma 3.17 is equivalent to Lemma 5 of [65], except the former is for the unweighted case and stop constraint. By Lemma 3.17 and the same argument of Lemma 6 in [65], we have the following lemma.

Lemma 3.18. *The modified StarImprove algorithm computes a solution for an instance* $(N, \mathcal{A})$ *of RSOneStop with at least* $(l - |S^*|)/2$ *trips assigned as passengers, where* (S^*, σ^*) *is an optimal solution for* $(N, \mathcal{A})$.

Theorem 3.26. *Let* $(N, \mathcal{A})$ *be a ridesharing instance of RSOneStop. Let* (S^*, σ^*) *be an optimal solution for* $(N, \mathcal{A})$, $l = |\mathcal{A}|$ *and* $\lambda = \max_{i\in\mathcal{A}} \lambda_i$. *Then,*

- *The EdgeSwap algorithm computes a solution* (S, σ) *for* $(N, \mathcal{A})$ *such that* $|S^*| \leq |S| \leq \frac{\lambda+2}{2}|S^*|$ *with running time* $O(M + l^{2\lambda})$.
- *The modified StarImprove algorithm computes a solution* (S, σ) *for* $(N, \mathcal{A})$ *such that* $|S^*| \leq |S| \leq \frac{\lambda+2}{2}|S^*|$ *with running time* $O(M + \lambda \cdot l^3)$, *where* M *is the size of the ridesharing instance which contains road network* N *and* l *trips.*

Proof. First, we need to construct a digraph $\vec{G}$ to represent the serve relation of the trips in $\mathcal{A}$ using Algorithm 1, which takes $O(M)$ time. Then reverse the direction of all arcs in $\vec{G}$, and this gives an instance can be solved by the EdgeSwap and modified StarImprove algorithms. Then, the first bullet point of the lemma is due to the EdgeSwap paragraph stated previously. The rest of the proof is for the second bullet point.

By Lemma 3.18, the modified StarImprove algorithm finds a solution for $(N, \mathcal{A})$ with at least $(l - |S^*|)/2$ passengers, and hence, at most $|S| \leq l - (l - |S^*|)/2 = (l + |S^*|)/2$ trips are assigned as drivers. Since there are $l - |S^*|$ passengers in the optimal solution, $|S^*| \geq (l - |S^*|)/\lambda$, implying $l \leq (\lambda + 1)|S^*|$. Therefore,

$$|S| \leq (l + |S^*|)/2 \leq ((\lambda + 1)|S^*| + |S^*|)/2 = (\lambda + 2)|S^*|/2.$$

The original StarImprove algorithm has a for-loop to check each vertex v to see if an improvement can be found, that is, it takes $O(l)$ time to check all in-neighbors of v to see if a

star S_v that can increase $|CM|$ exists, where CM is the current matching. In total, the for-loop takes $O(l^2)$ time since there are $O(l)$ vertices to check. As for the modified StarImprove, it takes $O(\lambda \cdot l)$ time to computing S_v if it exists (the running time of Algorithm 5). Hence, the for-loop becomes $O(\lambda \cdot l^2)$ for the modified StarImprove. After an improvement is made each time, StarImprove scans every vertex again to check for another improvement until no improvement can be found, and this takes $O(l)$ time due to at most $O(l)$ improvements can be made for the unweighted case. Thus, in total, the modified StarImprove has a running time of $O(M + \lambda \cdot l^3)$. □

3.4.2 A novel algorithm for RSOneStop

In this section, we present our approximation algorithm for RSOneStop; our algorithm has a better running time than the approximation algorithms based on MCMP in the previous section. For our proposed algorithm, we assume the serve relation is transitive, that is, trip i can serve trip j and j can serve trip k imply i can serve k. As mentioned in Subsection 3.3.1, transitive serve relation typically occurs when each trip uses the shortest path from the origin to destination, computed by the centralized ridesharing system (CRS), as the preferred path. This is usually the case in practice. Since the preferred path condition C3 is satisfied for $\mathcal{A}$, trip i can serve trip j implies that P_j is a subpath of P_i, so the serve relation is transitive (assuming unique shortest path in N).

Preprocessing.

Given an instance $(N, \mathcal{A})$ of RSOneStop, we first construct a simplified serve relation meta graph $\Gamma(V, E)$ to express the serve relation, where $V(\Gamma)$ represents the origins of all trips in $(N, \mathcal{A})$. Each node μ of $V(\Gamma)$ contains all trips with the same origin. There is an edge (μ, ν) in $E(\Gamma)$ if a trip in μ can serve a trip in ν. Since C1-C3 and C5 are satisfied, if one trip in μ can serve a trip in ν, any trip in μ can serve any trip in ν. We say node μ can serve node ν. An edge (μ, ν) in Γ is called a *short cut* if after removing (μ, ν) from Γ, there is a path from μ to ν in Γ. We simplify Γ by removing all short cuts from Γ. The construction of $\Gamma(V, E)$ can be done using an algorithm similar to Algorithm 1. The only major differences are to create a node μ_i for each group G_i (on Line 4 of Algorithm 1) and do not create a chain (Line 17 - Line 20) since a node μ_i contains all trips of G_i.

Similarly, as stated in Subsection 3.3.2, we assume w.l.o.g. that $\vec{G}(V, E)$ contains a single connected component T, which is an inverse tree. We label the nodes of $V(\vec{G})$ as $V(\vec{G}) = \{\mu_p, \mu_{p-1}, ..., \mu_1\}$, where $p = |V(\vec{G})|$, in such a way that for every edge (μ_b, μ_a) of Γ, $b > a$, and we say μ_b has a larger label than μ_a. Again, the labeling can done by Algorithm 2 that starts at integer label p instead of l. Hence, the running time for constructing and labelling Γ is $O(M)$, where M is the size of the ridesharing instance which contains road network N and l trips.

Each node in Γ without an incoming edge is called a *source*, and μ_1 is the unique sink. For a node μ in $V(\Gamma)$, the set of trips contained in mode μ is denoted by $\mathcal{A}(\mu)$. For a set U of nodes in $V(\Gamma)$, $\mathcal{A}(U) = \bigcup_{\mu \in U} \mathcal{A}(\mu)$. Similarly, given a set S of drivers, we denote the set of drivers in the nodes of U by $S(U)$ and the set of drivers in a node μ by $S(\mu)$. For a trip $i \in \mathcal{A}$, the node that contains i is denoted by node(i), that is, if $i \in \mathcal{A}(\mu)$ then node$(i) = \mu$. Table 3.3 contains the basic notation and definition for this section.

Notation	**Definition**
$\Gamma(V, E)$ and p	A digraph expressing the serve relation and $p = \|V(\Gamma)\|$
μ is an ancestor of ν	If $\exists$ a nonempty path from μ to ν in Γ (ν is a descendant of μ)
A_μ and A^*_μ	Set of ancestors of μ and $A^*_\mu = A_\mu \cup \{\mu\}$, respectively
D_μ and D^*_μ	Set of descendants of μ and $D^*_\mu = D_\mu \cup \{\mu\}$, respectively
$\mathcal{A}(\mu)$ and $\mathcal{A}(U)$	Set of trips in a node μ and in a set U of nodes, respectively
$S(\mu)$ and $S(U)$	Set of drivers in a node μ and in nodes U, respectively
node(i)	The node that contains trip i (if $i \in \mathcal{A}(\mu)$ then node$(i) = \mu$)
free(i)	The remaining seats (capacity) of i w.r.t. solution (S, σ)
stop(i)	The number of stops i has to made to pick-up all trips assigned to i

Table 3.3: Basic notation and definition used in this section.

Algorithm Description.

We divide all trips of $\mathcal{A}$ into two sets W and X as follows:

$W = \{i \in \mathcal{A} \mid \lambda_i = 0\} \cup \{i \in \mathcal{A}(\mu) \mid \delta_i = 0 \text{ and } |\mathcal{A}(\mu) = 1| \text{ for every node } \mu \in V(\Gamma)\}$ and $X = \mathcal{A} \setminus W$.

For a node μ in Γ, let $X(\mu) = X \cap \mathcal{A}(\mu)$ and $W(\mu) = W \cap \mathcal{A}(\mu)$. Our algorithm tries to minimize the number of drivers that only serve itself. There are three phases in the algorithm. In Phase-I, it serves all trips of W and tries to minimize the number of trips in W that are assigned as drivers since each trip of W can serve only itself. Let Z be the set of unserved trips after Phase-I such that for every $i \in Z$, $\delta_i = 0$. In Phase-II, it serves all trips of Z and tries to minimize the number of trips in Z to be assigned as drivers, each only serves itself. In Phase III, it serves all remaining trips. Let (S, σ) be the current partial solution and $\eta_i \in S$. Denoted by free$(i) = \lambda_i - |\sigma(i)| + 1$ is the remaining seats (capacity) of i w.r.t. solution (S, σ). Denoted by stop(i) is the number of stops i has to made in order to pick-up all trips in $\sigma(i)$ w.r.t. (S, σ). For the initial solution $(S, \sigma) = (\emptyset, \emptyset)$, free$(i) = \lambda_i$ and stop$(i) = 0$ for all $i \in \mathcal{A}$. For a driver i and node μ, we define $\mathcal{A}(i, \mu, S)$ as the set of $\min\{\text{free}(i), |\mathcal{A}(\mu) \setminus \sigma(S)|\}$ trips in $\mathcal{A}(\mu) \setminus \sigma(S)$ and $W(i, \mu, S)$ as the set of $\min\{\text{free}(i), |W(\mu) \setminus \sigma(S)|\}$ trips in $W(\mu) \setminus \sigma(S)$, and similarly for $Z(i, \mu, S)$. The three phases of the approximation algorithm are described in following, and the pseudo code is given in Algorithm 6 (Approximating RSOneStop).

(Phase-I) In this phase, the algorithm assigns a set of drivers to serve all trips of W, and it ends once all trips of W are served. Let $\Gamma(W) = \{\mu \in V(\Gamma) \mid W(\mu) \setminus \sigma(S) \neq \emptyset\}$, and in each iteration, a node of $\Gamma(W)$ is processed. In each iteration, the node $\mu = \text{argmax}_{\mu \in \Gamma(W)} |W(\mu) \setminus \sigma(S)|$ is selected and a subset of trips in $W(\mu) \setminus \sigma(S)$ is served by a driver as follows:

- Let $\hat{X}_1 = \{i \in S(A_\mu) \mid \text{free}(i) > 0 \wedge \text{stop}(i) < \delta_i\}$ and $\bar{X} = \{i \in X \cap \mathcal{A}(A^*_\mu) \setminus \sigma(S) \mid \text{stop}(i) < \delta_i \vee i \in \mathcal{A}(\mu)\}$. The algorithm finds and assigns a trip x as a driver to serve $W(x, \mu, S)$ such that $x = \text{argmin}_{x \in \hat{X}_1 \cup \bar{X} \,:\, \lambda_x \geq |W(\mu) \setminus \sigma(S)|} \delta_x - \text{stop}(x)$.
  - If such a trip x does not exist, it means that $\lambda_x < |W(\mu) \setminus \sigma(S)|$ for every $x \in \hat{X}_1 \cup \bar{X}$ assuming $\hat{X}_1 \cup \bar{X} \neq \emptyset$. Then, $x = \text{argmax}_{x \in \hat{X}_1 \cup \bar{X}} \text{free}(x)$ is assigned as a driver to serve $W(x, \mu, S)$. If there is more than one x with same $\text{free}(x)$, the trip with smallest $\delta_x - \text{stop}(x)$ is selected.
- When $\hat{X}_1 \cup \bar{X} = \emptyset$, assign every $w \in W(\mu) \setminus \sigma(S)$ as a driver to serve itself.

(Phase-II) In the second phase, all trips of $Z = \{i \in \mathcal{A} \setminus \sigma(S) \mid \delta_i = 0\}$ will be served. Let $\Gamma(Z) = \{\mu \in V(\Gamma) \mid Z(\mu) = (Z \cap \mathcal{A}(\mu)) \neq \emptyset\}$. Each node μ of $\Gamma(Z)$ is processed in the decreasing order of their node labels.

- If $|Z(\mu)| \geq 2$, trip $x = \text{argmax}_{x \in Z(\mu)} \lambda_x$ is assigned as a driver and serves a subset of trips in $Z(x, \mu, S)$ with smallest capacity among trips in $Z(\mu) \setminus \sigma(S)$.
- This repeats until $|Z(\mu)| \leq 1$. Then next node in $\Gamma(Z)$ is processed.

After all nodes of $\Gamma(Z)$ are processed, each node μ of $\Gamma(Z)$ such that $|Z(\mu)| > 0$ is processed again; note that every μ contains exactly one $z \in Z(\mu)$ now, that is, $|Z(\mu)| = 1$.

- A driver $x \in \hat{X}_2 = \{i \in S(A^*_\mu) \mid \text{free}(i) > 0 \wedge (\text{stop}(i) < \delta_i \vee i \in \mathcal{A}(\mu))\}$ with largest $\text{free}(x)$ is selected to serve $z = Z(\mu)$ if $\hat{X}_2 \neq \emptyset$.
- If $\hat{X}_2 = \emptyset$, a trip $x \in \bar{X} = \{i \in X \cap \mathcal{A}(A^*_\mu) \setminus \sigma(S) \mid \text{stop}(i) < \delta_i \vee i \in \mathcal{A}(\mu)\}$ with largest δ_x is selected to serve $z = Z(\mu)$.

(Phase-III) To serve all remaining trips, the algorithm processes each node of Γ in decreasing order of node labels from μ_p to μ_1. Let μ_j be the node being processed by the algorithm. Suppose there are trips in $\mathcal{A}(\mu_j)$ that have not be served, that is, $\mathcal{A}(\mu_j) \nsubseteq \sigma(S)$.

- A driver $x \in \hat{X}_2 = \{i \in S(A^*_{\mu_j}) \mid \text{free}(i) > 0 \wedge (\text{stop}(i) < \delta_i \vee i \in \mathcal{A}(\mu_j))\}$ with largest $\text{free}(x)$ is selected if $\hat{X}_2 \neq \emptyset$.
- If $\hat{X}_2 = \emptyset$, a trip $x = \text{argmax}_{x \in X(\mu_j) \setminus \sigma(S)} \lambda_x$ is assigned as a driver. If the largest λ_x is not unique, the trip with the smallest δ_x is selected.
- In either case, x is assigned to serve $\mathcal{A}(x, \mu_j, S)$. This repeats until all of $\mathcal{A}(\mu_j)$ are served. Then, next node μ_{j-1} is processed.

Algorithm 6 Approximating RSOneStop

Input: The simplified serve relation meta graph Γ for an instance $(N, \mathcal{A})$ of RSOneStop.
Output: A solution (S, σ) for the instance $(N, \mathcal{A})$ with $\frac{\lambda+2}{2}$-approximation ratio.
$(S, \sigma) = (\emptyset, \emptyset)$; let $\Gamma(W) = \{\mu \in V(\Gamma) \mid W(\mu) \setminus \sigma(S) \neq \emptyset\}$.
while $\Gamma(W) \neq \emptyset$ **do** // Beginning of Phase-I
    Compute $\mu = \text{argmax}_{\mu \in \Gamma(W)} |W(\mu) \setminus \sigma(S)|$. Let $\hat{X}_1 = \{i \in S(A_\mu) \mid \text{free}(i) > 0 \wedge \text{stop}(i) < \delta_i\}$
    and $\bar{X} = \{i \in X \cap \mathcal{A}(A^*_\mu) \setminus \sigma(S) \mid \text{stop}(i) < \delta_i \vee i \in \mathcal{A}(\mu)\}$.
    if $\hat{X}_1 \cup \bar{X} \neq \emptyset$ **then**
        Compute $x = \text{argmin}_{x \in \hat{X}_1 \cup \bar{X} \,:\, \lambda_x \geq |W(\mu) \setminus \sigma(S)|} \delta_x - \text{stop}(x)$.
        if $x = \emptyset$ **then** $x = \text{argmax}_{x \in \hat{X}_1 \cup \bar{X}} \text{free}(x)$ with the smallest $\delta_x - \text{stop}(x)$.
        if $x \notin S$ **then** $S = S \cup \{\eta_x\}$; $\sigma(x) = \{x\}$;
        $\sigma(x) = \sigma(x) \cup W(x, \mu, S)$; update free$(x)$ and stop(x);
    else
        for each $w \in W(\mu) \setminus \sigma(S)$, $S = S \cup \{\eta_w\}$, $\sigma(w) = \{w\}$; update free(w);
    end if
end while // End of Phase-I. Below is Phase-II
Let $Z = \{i \in \mathcal{A} \setminus \sigma(S) \mid \delta_i = 0\}$ and $\Gamma(Z)$ be the set of nodes containing Z.
for each node $\mu \in \Gamma(Z)$ in decreasing order of the node labels **do**
    while $|Z(\mu)| \geq 2$ **do**
        Compute $x = \text{argmax}_{x \in Z(\mu)} \lambda_x$. $S = S \cup \{\eta_x\}$; $\sigma(x) = \{x\}$;
        $\sigma(x) = \sigma(x) \cup Z(x, \mu, S)$ where $Z(x, \mu, S)$ consists of trips with smallest capacity in
        $Z(\mu) \setminus \sigma(S)$; update free$(x)$ and stop(x); update Z.
    end while
end for
for each node $\mu \in \Gamma(Z)$ in decreasing order of node labels **do** // implying $|Z(\mu)| = 1$
    Let $\hat{X}_2 = \{i \in S(A^*_\mu) \mid \text{free}(i) > 0 \wedge (\text{stop}(i) < \delta_i \vee i \in \mathcal{A}(\mu))\}$.
    if $\hat{X}_2 \neq \emptyset$ **then**
        Compute $x = \text{argmax}_{x \in \hat{X}_2} \text{free}(x)$.
    else
        Let $\bar{X} = \{i \in X \cap \mathcal{A}(A^*_\mu) \setminus \sigma(S) \mid \text{stop}(i) < \delta_i \vee i \in \mathcal{A}(\mu)\}$. Compute $x = \text{argmax}_{x \in \bar{X}} \delta_x$.
    end if
    if $x \notin S$ **then** $S = S \cup \{\eta_x\}$; $\sigma(x) = \{x\}$;
    $\sigma(x) = \sigma(x) \cup Z(x, \mu, S)$; update free$(x)$ and stop(x);
end for // End of Phase-II. Below is Phase-III
for each node μ from μ_p to μ_1 **do**
    while $\mathcal{A}(\mu) \not\subseteq \sigma(S)$ **do**
        Let $\hat{X}_2 = \{i \in S(A^*_\mu) \mid \text{free}(i) > 0 \wedge (\text{stop}(i) < \delta_i \vee i \in \mathcal{A}(\mu))\}$.
        if $\hat{X}_2 \neq \emptyset$ **then** Compute $x = \text{argmax}_{x \in \hat{X}_2} \text{free}(x)$.
        else Compute $x = \text{argmax}_{x \in X(\mu) \setminus \sigma(S)} \lambda_x$ (with smallest δ_x as a tiebreaker)
        if $x \notin S$ **then** $S = S \cup \{\eta_x\}$; $\sigma(x) = \{x\}$;
        $\sigma(x) = \sigma(x) \cup \mathcal{A}(x, \mu, S)$; update free$(x)$ and stop(x);
    end while
end for

Analysis of the approximation algorithm

A driver in a solution is called a *solo* driver if it serves only itself. Algorithm 6 tries to minimize the number of solo drivers. Recall that W is the set of trips, each of which can serve only itself. The algorithm, in Phase-I, computes a partial solution to serve all trips of W and tries to assign as few trips of W to be drivers as possible. In Phase-II, the set Z of

unserved trips after Phase-I (every $i \in Z$ has $\delta_i = 0$) is served. The rationale to serve such set of trips is that many trips of Z can become solo drivers if all trips of $\mathcal{A}(\text{node}(i)) \setminus \{i\}$ for $i \in Z$ are served before i is processed or considered. This can cause Z to have the same characteristic as W, so we treat Z separately. Let K be the number of solo drivers in a solution computed by Algorithm 6 and K^* be the number of solo drivers in any optimal solution. Then there are at most $(|\mathcal{A}| - K)/2 + K$ drivers in the solution computed by Algorithm 6 and at least $(|\mathcal{A}| - K^*)/(K+1) + K^*$ drivers in the optimal solution. A central line of the analysis is to show that K is close to K^* which guarantees the approximation ratio of Algorithm 6.

We now introduce some notation used in our analysis. Denoted by (S, σ) is the complete solution computed by Algorithm 6. Denoted by $(S_{\text{I}}, \sigma_{\text{I}})$ is the partial solution computed at the end of Phase-I, so all trips of W are served by drivers in S_{I}. For every driver $i \in S_{\text{I}}$, $(\sigma_{\text{I}}(i) \setminus \{i\}) \cap (\mathcal{A} \setminus W) = \emptyset$. Let $S_{\text{I}}(X) = S_{\text{I}} \cap X$ and $S_{\text{I}}(W) = S_{\text{I}} \cap W = S_{\text{I}} \setminus S_{\text{I}}(X)$. Note that each driver $i \in S_{\text{I}}(X)$ must serve at least one trip from W and $\sigma_{\text{I}}(S_{\text{I}}(X)) \setminus S_{\text{I}}(X) \subseteq W$ if $S_{\text{I}}(X) \neq \emptyset$. Let $W = \{W_1, \ldots, W_e\}$ such that each W_j $(1 \leq j \leq e)$ is the set of trips served by a driver (or drivers when $\hat{X}_1 \cup \bar{X} = \emptyset$) in S_{I} for iteration j, where e is the last iteration of Phase-I. For each W_j, W_j is a subset of $W(\mu_{a_j})$ for some node μ_{a_j} (indexed at a_j), and let (S_j, σ_j) be the partial solution just after serving W_j, $1 \leq j \leq e$. For a driver $i \in S_j$, $\text{free}_j(i) = \lambda_i - |\sigma_j(i)| + 1$ is the remaining available seats (capacity) of i w.r.t. (S_j, σ_j), and $\text{stop}_j(i)$ is the number of stops i has to made in order to pick-up all trips in $\sigma_j(i)$ w.r.t. (S_j, σ_j).

Property 3.1. *For every trip i that is assigned as a driver, i remains a driver until the algorithm terminates and* $\text{free}(i)$ *is non-increasing throughout the algorithm.*

Recall that each set W_j of trips either are served by one driver or $W_j \subseteq S_{\text{I}}(W)$ are drivers themselves. For clarity, we denote each set $W_j \subseteq S_{\text{I}}(W)$ by $\tilde{W}_j$. When trips of $\tilde{W}_j$ are assigned as drivers to serve themselves, all other trips $W(\mu_{a_j}) \setminus \tilde{W}_j$ must have been served by drivers in S_{j-1} such that no driver in S_{j-1} or trip in $X \setminus \sigma_{j-1}(S_{j-1})$ can serve $\tilde{W}_j$. In other words, $\tilde{W}_j = W(\mu_{a_j}) \setminus \sigma_{j-1}(S_{j-1})$ and $\bar{X}_1 \cup \hat{X} = \emptyset$ w.r.t. (S_{j-1}, σ_{j-1}), so the algorithm has the following property.

Property 3.2. *For every pair $\tilde{W}_i$ and $\tilde{W}_j$ $(i \neq j)$, $\mu_{a_i} \neq \mu_{a_j}$.*

Suppose (S^*, σ^*) is an optimal solution for $(N, \mathcal{A})$ with $|S^*|$ minimized. We first show, in Lemma 3.19, that the number of trips in $S_{\text{I}}(W)$ served by S^* is at most that of the passengers served by $\sigma_{\text{I}}(S_{\text{I}}(X))$ ($S_{\text{I}}(X)$ not included). The proof idea is as follows. Let $U \subseteq S^*$ be the set of drivers such that for every $u \in U$, $\sigma^*(u) \cap S_{\text{I}}(W) \neq \emptyset$ and $U \cap W = \emptyset$. We prove that U are also drivers in S_{I} (specifically, $U \subseteq S_{\text{I}}(X)$) and $\sigma_{\text{I}}(u)$ serves at least $|\sigma^*(u) \cap S_{\text{I}}(W)|$ passengers for each $u \in U$.

Lemma 3.19. *Let (S^*, σ^*) be an optimal solution for $(N, \mathcal{A})$ and $S^*(W) = W \cap S^*$. Let $U \subseteq S^*$ be the set of drivers that serve all trips of $W \setminus S^*(W)$. Then $|\sigma_{\mathrm{I}}(S_{\mathrm{I}}(X)) \setminus S_{\mathrm{I}}(X)| \geq |\sigma^*(U) \cap S_{\mathrm{I}}(W)|$.*

Proof. Let U_j be the set of drivers in S^* that serve $(W_1 \cup \ldots \cup W_j) \setminus S^*(W)$ for $1 \leq j \leq e$. Note that $W = W_1 \cup \ldots \cup W_e$ and $U_e = U$. Let $\tilde{W}_{a_1}, \ldots, \tilde{W}_{a_d}$ be the sets computed by the algorithm s.t. $\tilde{W}_{a_b} \subseteq S_{\mathrm{I}}(W), 1 \leq b \leq d$, and for $1 \leq b < c \leq d$, $\tilde{W}_{a_b}$ is computed before $\tilde{W}_{a_c}$. For each $\tilde{W}_{a_b}$, the drivers of U_{a_b} that serve $\tilde{W}_{a_b} \setminus S^*(W)$ can be partitioned into two sets: (1) $U'_{a_b} = \{u \in U_{a_b} \mid \sigma^*(u) \cap \tilde{W}_{a_b} \neq \emptyset \text{ and } u \in \mathcal{A}(\mu_{a_b})\}$ and (2) $U''_{a_b} = \{u \in U_{a_b} \mid \sigma^*(u) \cap \tilde{W}_{a_b} \neq \emptyset \text{ and } u \in \mathcal{A}(A_{\mu_{a_b}})\}$. We consider them separately.

(1) Due to $W(\mu_{a_b}) \neq \emptyset$ ($\mu_{a_b} \in \Gamma(W)$), the algorithm must have already assigned every $u \in U'_{a_b}$ as a driver in $S_{a_{b-1}}(X)$ when node μ_{a_b} is selected to be processed since such a trip u must be included in $\bar{X}$ w.r.t. the partial solution just before μ_{a_b} is processed. Further, it must be that $\mathrm{free}_{a_{b-1}}(u) = 0$. Otherwise, $\sigma_{a_{b-1}}(u)$ would have served trips from $\tilde{W}_{a_b}$, a contradiction to the algorithm. From $\mathrm{free}_{a_{b-1}}(u) = 0$, $|\sigma_{a_b}(u) \cap W| \geq |\sigma^*(u) \cap W|$ for every $u \in U'_{a_b}$, that is, $|\bigcup_{u \in U'_{a_b}} \sigma_{a_b}(u) \cap W| \geq |\bigcup_{u \in U'_{a_b}} \sigma^*(u) \cap W|$.

(2) Every $u \in U''_{a_b}$ must also be a driver in $S_{a_{b-1}}(X)$ with $\mathrm{free}_{a_b}(u) < \lambda_u$. Otherwise, u would have been assigned (from unassigned) as a driver in S_{a_b} to serve trips from $\tilde{W}_{a_b}$. We further divide U''_{a_b} into two subsets: $U''_{a_b}(0) = \{u \in U''_{a_b} \mid \mathrm{free}_{a_b}(u) = 0\}$ and $U''_{a_b}(1) = \{u \in U''_{a_b} \mid \mathrm{free}_{a_b}(u) \geq 1\}$. We consider $U''_{a_b}(0)$ in case (2.1) and $U''_{a_b}(1)$ in case (2.2).

(2.1) For every $u \in U''_{a_b}(0)$, $|\sigma_{a_b}(u) \cap W| \geq |\sigma^*(u) \cap W|$ since $\mathrm{free}_{a_b}(u) = 0$. This implies that $|\bigcup_{u \in U''_{a_b}(0)} \sigma_{a_b}(u) \cap W| \geq |\bigcup_{u \in U''_{a_b}(0)} \sigma^*(u) \cap W|$. (2.2) Consider any driver $u \in U''_{a_b}(1)$. Let W_j be a non-empty set of passengers served by $\sigma_{a_b}(u)$ where $j < a_b$. In other words, W_j is computed before $\tilde{W}_{a_b}$. Recall that $W_j \subseteq W(\mu_{a_j})$, W_j are the only passengers in $W(\mu_{a_j})$ served by u, and (S_{j-1}, σ_{j-1}) is the partial solution just before trips of W_j are served. From $\mathrm{free}_{j-1}(u) > \mathrm{free}_{a_b}(u) > 0$, $W_j = W(\mu_{a_j}) \setminus \sigma_{j-1}(S_{j-1})$ must be served by $\sigma_j(u)$, implying $|W_j| < \mathrm{free}_{j-1}(u)$. Since W_j is computed before $\tilde{W}_{a_b}$, $|\tilde{W}_{a_b}| \leq |W(\mu_{a_b}) \setminus \sigma_{j-1}(S_{j-1})| \leq |W(\mu_{a_j}) \setminus \sigma_{j-1}(S_{j-1})| < \mathrm{free}_{j-1}(u)$, meaning $|W_j| \geq |\tilde{W}_{a_b}|$ for every set W_j of passengers served by $\sigma_{a_b}(u)$. From the proofs of Cases (1) and (2), we have the following property.

Property 3.3. *Every $u \in U'_{a_b} \cup U''_{a_b}$ is also a driver in $S_I(X)$, that is, $U \subseteq S_I(X)$.*

Consider any pair $u_b \in U''_{a_b}(1)$ and $u_c \in U''_{a_c}(1)$ with $u_b \neq u_c$ for any $1 \leq b < c \leq d$. Since $u_b \neq u_c$, the analysis of Case (2.2) can be applied to u_b and u_c independently, that is, $|W_{j_b}| \geq |\tilde{W}_{a_b}|$ for every set W_{j_b} of passengers served by $\sigma_{j_b}(u_b)$, and $|W_{j_c}| \geq |\tilde{W}_{a_c}|$ for every set W_{j_c} served by $\sigma_{j_c}(u_c)$. Now, consider the case $u_b = u_c$. Assume that $U''_{a_b}(1) \cap U''_{a_c}(1) \neq \emptyset$ for some $1 \leq b < c \leq d$. Consider any driver $u \in U''_{a_b}(1) \cap U''_{a_c}(1)$. By definition, u serves trips from both $\tilde{W}_{a_b}$ and $\tilde{W}_{a_c}$. Since $\mathrm{free}_{a_c}(u) > 0$, $\mathrm{stop}_{a_c}(u) = \delta_u$. It must be that $\mathrm{free}_{a_b}(u) \geq \mathrm{free}_{a_c}(u) > 0$ and $\mathrm{stop}_{a_b}(u) = \delta_u$ (otherwise, $\sigma_{a_b}(u)$ would have served trips from $\tilde{W}_{a_b}$). From this and $\mu_{a_b} \neq \mu_{a_c}$ (by Property 3.2), $\delta_u \geq 2$ and $\sigma_{a_c}(u)$ serves at least two sets W_{j_b} and W_{j_c} of passengers before $\tilde{W}_{a_b}$ is computed. By the conclusion of previous

paragraph (Case 2.2), $|W_{j_b}| \geq |\tilde{W}_{a_b}|$ and $|W_{j_c}| \geq |\tilde{W}_{a_c}|$. This can be generalized to all sets $\tilde{W}_{a_1}, \ldots, \tilde{W}_{a_d} \subseteq S_{\mathrm{I}}(W)$ s.t. trips of $\tilde{W}_{a_b} \setminus S^*(W)$ are served by U_{a_b} for $1 \leq b \leq d$. We get $|\bigcup_{u \in U'_{a_b} \cup U''_{a_b}, 1 \leq b \leq d} \sigma_{a_b}(u) \cap W| \geq |\bigcup_{u \in U'_{a_b} \cup U''_{a_b}, 1 \leq b \leq d} \sigma^*(u) \cap \tilde{W}_{a_b}|$. By definition, $\bigcup_{u \in U'_{a_b} \cup U''_{a_b}, 1 \leq b \leq d} \sigma_{a_b}(u) \cap W = \sigma_{\mathrm{I}}(U) \setminus U$ and $\bigcup_{u \in U'_{a_b} \cup U''_{a_b}, 1 \leq b \leq d} \sigma^*(u) \cap \tilde{W}_{a_b} = \sigma^*(U) \cap S_{\mathrm{I}}(W)$. Since $U \subseteq S_{\mathrm{I}}(X)$ (Property 3.3), $|\sigma_{\mathrm{I}}(S_{\mathrm{I}}(X)) \setminus S_{\mathrm{I}}(X)| \geq |\sigma_{\mathrm{I}}(U) \setminus U| \geq |\sigma^*(U) \cap S_{\mathrm{I}}(W)|$. □

Lemma 3.20. *Let (S^*, σ^*) be any optimal solution for $(N, \mathcal{A})$. Let $F_{\mathrm{I}} \subseteq S^*$ be the set of drivers such that $\sigma_{\mathrm{I}}(S_{\mathrm{I}}) \subseteq \sigma^*(F_{\mathrm{I}})$. Then, $|F_{\mathrm{I}}| \geq \frac{2|S_{\mathrm{I}} \cup F_{\mathrm{I}}|}{\lambda+2}$.*

Proof. Three sets U, B_1, B_2 of drivers in S^* are considered, each of which serves a portion of trips of $\sigma_{\mathrm{I}}(S_{\mathrm{I}})$ w.r.t. (S^*, σ^*), and altogether $\sigma^*(U \cup B_1 \cup B_2) \cup S^*(W) \supseteq \sigma_{\mathrm{I}}(S_{\mathrm{I}})$, where $S^*(W) = S^* \cap W$. Let $U \subseteq S^*$ be the set of drivers s.t. $(S_{\mathrm{I}}(W) \setminus S^*(W)) \subseteq \sigma^*(U)$. By Property 3.3 and Lemma 3.19, all of U must be drivers in $S_{\mathrm{I}}(X)$ and $|\sigma_{\mathrm{I}}(U) \setminus U| \geq |\sigma^*(U) \cap S_{\mathrm{I}}(W)|$. In this proof, the drivers in S_{I} are partitioned into three sets: $S_{\mathrm{I}}(W), U$, and $S_X = S_{\mathrm{I}} \setminus (S_{\mathrm{I}}(W) \cup U)$.

It requires another set $B_1 \subseteq S^*$ of drivers to serve all trips of $(\sigma_{\mathrm{I}}(U) \setminus U) \subseteq W$ for (S^*, σ^*) because $\sigma_{\mathrm{I}}(U) \cap S_{\mathrm{I}}(W) = \emptyset$ and $|\sigma_{\mathrm{I}}(U) \setminus U| \geq |\sigma^*(U) \cap S_{\mathrm{I}}(W)|$. From $|\sigma_{\mathrm{I}}(U) \setminus U| \geq |\sigma^*(U) \cap S_{\mathrm{I}}(W)|$, $\sigma_{\mathrm{I}}(U) \cap S_{\mathrm{I}}(W) = \emptyset$ and that $\sigma^*(U) \cap S_{\mathrm{I}}(W) = S_{\mathrm{I}}(W) \setminus S^*(W)$, we have $|(S_{\mathrm{I}}(W) \setminus S^*(W)) \cup (\sigma_{\mathrm{I}}(U) \setminus U)| \geq 2|S_{\mathrm{I}}(W) \setminus S^*(W)|$. Therefore, $|U \cup B_1| \geq 2|S_{\mathrm{I}}(W) \setminus S^*(W)|/\lambda$ is the minimum number of drivers required in S^* to serve all of $(S_{\mathrm{I}}(W) \setminus S^*(W)) \cup (\sigma_{\mathrm{I}}(U) \setminus U)$. In the worst case, the algorithm can assign each trip $v \in B_1$ to be a driver in $S \setminus S_{\mathrm{I}}$ s.t. v serves only itself.

Consider the remaining set of drivers $S_X = S_{\mathrm{I}} \setminus (S_{\mathrm{I}}(W) \cup U)$. For each driver $x \in S_X$, $\sigma_{\mathrm{I}}(x)$ must serve at least one trip from W, meaning $|\sigma_{\mathrm{I}}(x)| \geq 2$ and $|\sigma_{\mathrm{I}}(S_X)| \geq 2|S_X|$. Let $B_2 \subseteq S^*$ s.t. $\sigma_{\mathrm{I}}(S_X) \subseteq \sigma^*(B_2)$. We now consider the size of B_2. Note that $B_2 \cap S_X$ may or may not be empty. In the worst case, each trip $v \in B_2 \setminus S_X$ can be assigned as a driver in $S \setminus S_{\mathrm{I}}$ s.t. v serves itself only. Hence, the ratio between the number of drivers in S that serve $\sigma_{\mathrm{I}}(S_X) \cup B_2$ and B_2 is $(|S_X| + |B_2 \setminus S_X|)/|B_2|$. This function is monotone increasing in $|B_2 \setminus S_X|$. Thus, $B_2 \cap S_X = \emptyset$ gives the worst case. From this and $|\sigma^*(v) \cap \sigma_{\mathrm{I}}(S_X)| \leq \lambda$ for each driver in $v \in B_2$, $|B_2| \geq 2|S_X|/\lambda$. Since $\sigma_{\mathrm{I}}(S_X) \cap \sigma_{\mathrm{I}}(U) = \emptyset$ and $\sigma_{\mathrm{I}}(S_X) \cap \sigma_{\mathrm{I}}(W) = \emptyset$,

$$|(S_{\mathrm{I}}(W) \setminus S^*(W)) \cup (\sigma_{\mathrm{I}}(U) \setminus U) \cup \sigma_{\mathrm{I}}(S_X)| \geq 2|S_{\mathrm{I}}(W) \setminus S^*(W)| + 2|S_X|.$$

Thus, $|U \cup B_1 \cup B_2| \geq 2(|S_{\mathrm{I}}(W) \setminus S^*(W)| + |S_X|)/\lambda$.

Let $F_{\mathrm{I}} = U \cup B_1 \cup B_2 \cup S^*(W)$, which is the set of drivers in S^* required to serve all of $\sigma_{\mathrm{I}}(S_{\mathrm{I}})$ for (S^*, σ^*). Then $|F_{\mathrm{I}}| = |U \cup B_1 \cup B_2| + |S^*(W)| \geq 2(|S_{\mathrm{I}}(W) \setminus S^*(W)| + |S_X|)/\lambda + |S^*(W)|$. Recall that $S_{\mathrm{I}} = S_{\mathrm{I}}(W) \cup U \cup S_X$. The ratio between the number of drivers in S to serve $\sigma_{\mathrm{I}}(S_{\mathrm{I}}) \cup B_1 \cup B_2$ and F_{I} is

$$\frac{|S_{\mathrm{I}} \cup B_1 \cup B_2|}{|F_{\mathrm{I}}|} \leq \frac{|S_{\mathrm{I}} \cup F_{\mathrm{I}}|}{|F_{\mathrm{I}}|} \leq \frac{|S_{\mathrm{I}}(W) \setminus S^*(W)| + |S_X| + |U \cup B_1 \cup B_2 \cup S^*(W)|}{|U \cup B_1 \cup B_2 \cup S^*(W)|} \tag{3.1}$$

$$
\begin{aligned}
&\leq \frac{|S_{\mathrm{I}}(W) \setminus S^*(W)| + |S_X|}{2(|S_{\mathrm{I}}(W) \setminus S^*(W)| + |S_X|)/\lambda + |S^*(W)|} + 1 \\
&\leq \frac{|S_{\mathrm{I}}(W) \setminus S^*(W)| + |S_X|}{2(|S_{\mathrm{I}}(W) \setminus S^*(W)| + |S_X|)/\lambda} + 1 \\
&= \frac{\lambda}{2} + 1 = \frac{\lambda + 2}{2}.
\end{aligned}
$$

Hence, it requires at least $|F_{\mathrm{I}}| \geq 2|S_{\mathrm{I}} \cup F_{\mathrm{I}}|/(\lambda+2)$ drivers in S^* to serve all trips of $\sigma_{\mathrm{I}}(S_{\mathrm{I}})$. □

Notice that Equation (3.1) holds when each driver in $u \in F_{\mathrm{I}}$ serves at most λ trips of $\sigma_{\mathrm{I}}(S_{\mathrm{I}})$, that is, $|\sigma^*(u)| \leq \lambda+1$. Next, we consider the minimum number of drivers in S^* that is required to serve all trips of $\sigma_{\mathrm{II}}(S_{\mathrm{II}})$ for (S^*, σ^*), where $(S_{\mathrm{II}}, \sigma_{\mathrm{II}})$ is the partial solution computed at the end of Phase-II. Recall that $Z = \{i \in \mathcal{A} \setminus \sigma_{\mathrm{I}}(S_{\mathrm{I}}) \mid \delta_i = 0\}$ and all of Z are served in $\sigma_{\mathrm{II}}(S_{\mathrm{II}})$.

Lemma 3.21. *Let (S^*, σ^*) be any optimal solution for $(N, \mathcal{A})$. Let $F_{\mathrm{II}} \subseteq S^*$ be the set of drivers such that $\sigma_{\mathrm{I}}(S_{\mathrm{I}}) \subseteq \sigma^*(F_{\mathrm{II}})$. Then, $|F_{\mathrm{II}}| \geq \frac{2|S_{\mathrm{II}} \cup F_{\mathrm{II}}|}{\lambda+2}$.*

Proof. We consider $F_{\mathrm{II}} = F_{\mathrm{I}} \cup C' \cup V' \cup C''$, each of C', V' and C'' is a set of drivers in S^* that serves a portion of trips of $\sigma_{\mathrm{II}}(S_{\mathrm{II}} \setminus S_{\mathrm{I}})$ w.r.t. (S^*, σ^*). Let $S' = \{x \in S_{\mathrm{II}} \setminus S_{\mathrm{I}} \mid |\sigma_{\mathrm{II}}(x)| = 1\}$ be the set of solo drivers in $S_{\mathrm{II}} \setminus S_{\mathrm{I}}$. Since $S' \subseteq X$, $\lambda_x > 0$ for every $x \in S'$. Each $x \in S'$ belongs to a distinct node of Γ since otherwise, one of them can serve the other. This implies that $S' \subseteq Z$. Let $C' = C'_0 \cup C'_1$ be the set of drivers in S^* s.t. $S' \subseteq \sigma^*(C')$, where $C'_0 = \{v \in S^* \mid \sigma^*(v) \cap S' \neq \emptyset \text{ and } \delta_v = 0\}$ and $C'_1 = \{v \in S^* \mid \sigma^*(v) \cap S' \neq \emptyset \text{ and } \delta_v \geq 1\}$. By definition and $S' \subseteq Z$, $C' \subseteq X$ and $C'_1 \cap Z = \emptyset$. Let $S' = S'_0 \cup S'_1$, where S'_0 is served by C'_0 and S'_1 is served by C'_1. Then $|C'_0| = |S'_0|$ because each $x \in S'_0$ belongs to a distinct node and $\delta_v = 0$ for all $v \in C'_0$.

Consider any driver $z \in S'_1$. Let (S_z, σ_z) be the partial solution just before z is assigned as a driver by the algorithm. All trips in $C'_1 \cap \mathcal{A}(A^*_{\mathrm{node}(z)})$ must have been assigned as drivers in S_z. Otherwise, any $v \in C'_1 \cap \mathcal{A}(A^*_{\mathrm{node}(z)})$ would have been assigned as a driver in S_{II} to serve z when node(z) is processed. Hence, $C'_1 \subseteq S_{\mathrm{II}}$, and for every driver $v \in C'_1 \cap \mathcal{A}(A^*_{\mathrm{node}(z)})$, $\mathrm{free}_z(v) = 0$ or $\mathrm{free}_z(v) > 0$ with $\mathrm{stop}_z(v) = \delta_v$. From these and each $z \in S'_1$ belongs to a distinct node, $|\bigcup_{v \in C'_1} \sigma_z(v) \setminus \{v\}| \geq |\bigcup_{v \in C'_1} \sigma^*(v) \cap S'_1| = |S'_1|$, and $|C'_1| \geq |S'_1|/\lambda$ to serve all of $S'_1 \subseteq Z$ since $S'_1 \cap C'_1 = \emptyset$. Recall that for every driver $v \in C'_1$, each passenger served by $\sigma_{\mathrm{II}}(v)$ is either in W or Z. For any $v \in C'_1$ s.t. $\sigma_{\mathrm{II}}(v) \cap W \neq \emptyset$, $v \in S_{\mathrm{I}}$ and v is included in the calculation of Equation (3.1). For any such v (regardless if $\sigma_{\mathrm{II}}(v) \cap Z \neq \emptyset$), the ratio $|S_{\mathrm{II}} \setminus S_{\mathrm{I}}|/|F_{\mathrm{II}}|$ decreases because $v \in S_{\mathrm{I}}$ and $v \in F_{\mathrm{II}}$. To get the approximation ratio for the worst case, we assume that all $C'_1 \subseteq (S_{\mathrm{II}} \setminus S_{\mathrm{I}})$, that is, $\sigma_{\mathrm{II}}(v) \cap W = \emptyset$ and $\sigma_{\mathrm{II}}(v) \cap Z \neq \emptyset$ for all $v \in C'_1$. Let $V' \subseteq S^*$ s.t. $(\bigcup_{v \in C'_1} \sigma_{\mathrm{II}}(v) \cap Z) \subseteq \sigma^*(V')$. By the algorithm (Phase-II part 2 specifically), each passenger $z \in \sigma_{\mathrm{II}}(v) \cap Z$ belongs to a distinct node, implying $|\bigcup_{v \in C'_1} \sigma_{\mathrm{II}}(v) \cap Z| \geq |S'_1|$. From these, each driver in $V' \subseteq S^*$ can serve at most λ trips of $\bigcup_{v \in C'_1} \sigma_{\mathrm{II}}(v) \cap Z$, and hence, $|V'| \geq |S'_1|/\lambda$. Since $S'_1 \cap (\bigcup_{v \in C'_1} \sigma_{\mathrm{II}}(v) \cap Z) = \emptyset$,

$|V' \cup C'_1| \geq 2|S'_1|/\lambda$. In the worst case, the algorithm can assign all of C' and V' to be drivers in S. From $|S'_0| = |C'_0|$, the ratio between $|S' \cup C' \cup V'|$ and $|C' \cup V'|$ is

$$\frac{|S' \cup C' \cup V'|}{|C' \cup V'|} \leq \frac{|S'| + |C' \cup V'|}{|C' \cup V'|} = \frac{|S'_0| + |S'_1|}{|C'_0| + |C'_1 \cup V'|} + 1 = \frac{|C'_0| + |S'_1|}{|C'_0| + |C'_1 \cup V'|} + 1.$$

Since $|S'_1| \geq |C'_1|$, $(|C'_0| + |S'_1|)/(|C'_0| + |C'_1 \cup V'|)$ is monotone decreasing in $|C'_0|$. Therefore,

$$\frac{|S' \cup C' \cup V'|}{|C' \cup V'|} \leq \frac{|C'_0| + |S'_1|}{|C'_0| + |C'_1 \cup V'|} + 1 \leq \frac{|S'_1|}{|C'_1 \cup V'|} + 1 \leq \frac{|S'_1|}{2|S'_1|/\lambda} + 1 = \frac{\lambda + 2}{2}. \tag{3.2}$$

Consider the remaining drivers in $S'' = S_{\mathrm{II}} \setminus (S_{\mathrm{I}} \cup S' \cup C')$. Since each driver $x \in S''$ serves at least one passenger, $|\sigma_{\mathrm{II}}(S'')| \geq 2|S''|$. Let $C'' = C''_0 \cup C''_1$ be the set of drivers in S^* s.t. $\sigma_{\mathrm{II}}(S'') \subseteq \sigma^*(C'')$, where $C''_0 = \{v \in S^* \mid \sigma^*(v) \cap \sigma_{\mathrm{II}}(S'') \neq \emptyset \text{ and } v \in Z\}$ and $C''_1 = \{v \in S^* \mid \sigma_{\mathrm{II}}(S'') \neq \emptyset \text{ and } v \in X \setminus Z\}$. Note that $C''_0 \subseteq \sigma_{\mathrm{II}}(S_{\mathrm{II}})$ by definition. From the algorithm (Phase-II), $\sigma_{\mathrm{II}}(S'') \subseteq Z$ and $\sigma_{\mathrm{II}}(S'') \cap \sigma_{\mathrm{II}}(S') = \emptyset$. In the worst case, each trip $v \in C''$ can be assigned as a driver in S s.t. v serves itself only. From this, $C''_0 \subseteq \sigma_{\mathrm{II}}(S_{\mathrm{II}})$ and that every $v \in C''$ can serve at most λ trips of $\sigma_{\mathrm{II}}(S'')$, the ratio between the number of drivers in S that serve $\sigma_{\mathrm{II}}(S'') \cup C''$ and C'' is

$$\frac{|S'' \cup C''|}{|C''|} \leq \frac{|S''| + |C''|}{|C''|} \leq \frac{|S''|}{2|S''|/\lambda} + 1 \leq \frac{\lambda}{2} + 1 = \frac{\lambda + 2}{2}. \tag{3.3}$$

Next, we combine Equations (3.2) and (3.3) with Equation (3.1). Let $F_{\mathrm{II}} = F_{\mathrm{I}} \cup C' \cup V' \cup C''$, which is the set of drivers in S^* required s.t. $\sigma_{\mathrm{II}}(S_{\mathrm{II}})$ in $\sigma^*(F_{\mathrm{II}})$. Note that $F_{\mathrm{I}} \subseteq S^*$ is the minimum set of drivers that serve all of $\sigma_{\mathrm{I}}(S_{\mathrm{I}}) \subseteq W$ and $(C' \cup V') \subseteq S^*$ is the minimum set of drivers that serve all of $S' \cup (\bigcup_{v \in S'} \sigma_{\mathrm{II}}(v) \cap Z)$, and $C'' \subseteq S^*$ is the minimum set of drivers that serve all of $\sigma_{\mathrm{II}}(S'') \subseteq Z$ s.t. $\sigma_{\mathrm{II}}(S'') \cap \sigma_{\mathrm{II}}(S') = \emptyset$. The minimum number of drivers in each set of F_{I}, $C' \cup V'$ and C'' is calculated based on each driver $u \in F_{\mathrm{II}}$ serving λ trips of $\sigma_{\mathrm{II}}(S_{\mathrm{II}})$, as stated in Equations (3.1), (3.2) and (3.3). Hence,

$$\begin{aligned} |F_{\mathrm{II}}| &\geq 2|S_{\mathrm{I}} \cup F_{\mathrm{I}}|/(\lambda + 2) + 2|S' \cup C' \cup V'|/(\lambda + 2) + 2|S'' \cup C''|/(\lambda + 2) \\ &= 2(|S_{\mathrm{I}} \cup F_{\mathrm{I}}| + |S' \cup C' \cup V'| + |S'' \cup C''|)/(\lambda + 2). \end{aligned}$$

The ratio between $S_{\mathrm{II}} \cup F_{\mathrm{II}}$ and F_{II} is at most

$$\begin{aligned} \frac{|S_{\mathrm{II}} \cup F_{\mathrm{II}}|}{|F_{\mathrm{II}}|} &\leq \frac{|S_{\mathrm{I}} \cup S' \cup S'' \cup F_{\mathrm{I}} \cup C' \cup V' \cup C''|}{|F_{\mathrm{I}} \cup C' \cup V' \cup C''|} \\ &\leq \frac{|S_{\mathrm{I}} \cup F_{\mathrm{I}}| + |S' \cup C' \cup V'| + |S'' \cup C''|}{2(|S_{\mathrm{I}} \cup F_{\mathrm{I}}| + |S' \cup C' \cup V'| + |S'' \cup C''|)/(\lambda + 2)} \\ &= \frac{\lambda + 2}{2}. \end{aligned} \tag{3.4}$$

Therefore, it requires at least $|F_{\mathrm{II}}| \geq 2|S_{\mathrm{II}} \cup F_{\mathrm{II}}|/(\lambda+2) \geq 2|S_{\mathrm{II}}|/(\lambda+2)$ drivers in S^* to serve all of $\sigma_{\mathrm{II}}(S_{\mathrm{II}})$. □

Again, Equation (3.4) holds if each driver $u \in F_{\mathrm{II}}$ serves at most λ trips of $\sigma_{\mathrm{II}}(S_{\mathrm{II}})$, that is, $|\sigma^*(u)| \leq \lambda + 1$. Recall that B_1 and B_2 are subsets of F_{I} defined in the proof of Lemma 3.20, and C', V' and C'' are subsets of F_{II} defined in the proof of Lemma 3.21. Each trip v in $B_1 \cup B_2 \cup C' \cup V' \cup C''$ can be a driver in S that serves itself only. This can happen if before $v \in B_1 \cup B_2 \cup C' \cup V' \cup C''$ is processed by the algorithm, $\mathcal{A}(D^*_{\mathrm{node}(v)}) \setminus \{v\} \subseteq \sigma_v(S_v)$ for $\delta_v > 0$ or $\mathcal{A}(\mathrm{node}(v)) \setminus \{v\} \subseteq \sigma_v(S_v)$ for $\delta_v = 0$, where (S_v, σ_v) is the partial solution just before v is processed. In other words, all trips that can be served by v are already served w.r.t. (S_v, σ_v).

Remark 3.1. Let B_1 and B_2 be the set of trips defined in the proof of Lemma 3.20. Let S' and S'' be the sets of drivers defined in the proof of Lemma 3.21. Trips of $B_1 \cup B_2$ can be assigned as drivers in Phase-II or Phase-III. Suppose $v \in B_1 \cup B_2$ is assigned as a driver in Phase-II. If $\sigma(v)$ serves only itself, v is included in S'. If $\sigma(v)$ serves more than one trip, v is included in S''. For either case, Equation (3.4) holds.

From Remark 3.1, let $B'_1 \cup B'_2 \subseteq B_1 \cup B_2$ be the trips assigned as drivers in Phase-III. Let $\bar{S} = S \setminus (S_{\mathrm{II}} \cup B'_1 \cup B'_2 \cup C' \cup V' \cup C'')$ be the set of drivers found during Phase-III of the algorithm.

Lemma 3.22. *For (S^*, σ^*), it requires at least $\frac{2|S|}{\lambda+2}$ drivers in S^* to serve all of $\sigma(S)$.*

Proof. Any trip x in $\mathcal{A} \setminus \sigma_{\mathrm{II}}(S_{\mathrm{II}})$ has $\lambda_x > 0$ and $\delta_x > 0$ since all of W and Z are served in $\sigma_{\mathrm{II}}(S_{\mathrm{II}})$. Consider the moment a trip $x \in \bar{S}$ is assigned as a driver. Let (S_x, σ_x) be the partial solution just before x is processed by the algorithm. Since $\lambda_x > 0$ and $\delta_x > 0$, x will serve at least one passenger ($|\sigma(x)| \geq 2$) if there exists an un-assigned trip in $\mathcal{A}(D^*_{\mathrm{node}(x)}) \setminus \{x\}$, that is, $\mathcal{A}(D^*_{\mathrm{node}(x)}) \setminus \{x\} \not\subseteq \sigma_x(S_x)$. Let $X(1) = \{x \in \bar{S} \mid |\sigma(x)| = 1\}$. For every pair $x, x' \in X(1)$, $x \notin \mathcal{A}(D^*_{\mathrm{node}(x')}) \cup \mathcal{A}(A^*_{\mathrm{node}(x')})$. Otherwise, one of them can serve the other. For every $x \in X(1)$, any driver $x' \in \bar{S}(A^*_{\mathrm{node}(x)}) \setminus \{x\}$ must serve at least two trips, where $\bar{S}(A^*_{\mathrm{node}(x)}) = \bar{S} \cap \mathcal{A}(A^*_{\mathrm{node}(x)})$. For any $x \in X(1)$, let Y_x be the set of drivers in S^* that serve all of $\bigcup_{x' \in \bar{S}(A^*_{\mathrm{node}(x)})} \sigma(x')$ in (S^*, σ^*). For a driver $y \in Y_x$, $\sigma^*(y) \setminus \{y\}$ can contain at most λ trips of $\bigcup_{x' \in \bar{S}(A^*_{\mathrm{node}(x)})} \sigma(x')$. If $y \in \bigcup_{x' \in \bar{S}(A^*_{\mathrm{node}(x)})} \sigma(x')$, y can serve at most $\lambda + 1$ trips of $\bigcup_{x' \in \bar{S}(A^*_{\mathrm{node}(x)})} \sigma(x')$. Hence, $|Y_x| \geq (2|\bar{S}(A_{\mathrm{node}(x)})| + 1)/(\lambda + 1)$. For any pair $x, x' \in X(1)$ with $x \neq x'$, since drivers in Y_x cannot serve any trip of $\bigcup_{x'' \in \bar{S}(A^*_{\mathrm{node}(x')})} \sigma(x'')$, it must be that $Y_x \cap Y_{x'} = \emptyset$. Let $Y = \bigcup_{x \in X(1)} Y_x$, $\bar{S}_{X(1)} = \bigcup_{x \in X(1)} \bar{S}(A^*_{\mathrm{node}(x)})$ and $\sigma(\bar{S}_{X(1)}) =$

$\bigcup_{x\in\bar{S}_{X(1)}}\sigma(x)$. Note that $|Y|\geq|X(1)|$. Then,

$$\begin{aligned}|Y| = \sum_{x\in X(1)}|Y_x| &\geq \sum_{x\in X(1)}(2|\bar{S}(A_{\mathrm{node}(x)})|+1)/(\lambda+1)\\ &= (\sum_{x\in X(1)}2|\bar{S}(A_{\mathrm{node}(x)})|+|X(1)|)/(\lambda+1).\end{aligned}$$

The above can be rewritten as

$$\frac{(\lambda+1)\cdot|Y|-|X(1)|}{2}\geq\sum_{x\in X(1)}|\bar{S}(A_{\mathrm{node}(x)})| = \sum_{x\in X(1)}|\bar{S}(A^*_{\mathrm{node}(x)})|-X(1),$$

and hence,

$$\sum_{x\in X(1)}|\bar{S}(A^*_{\mathrm{node}(x)})|\leq\frac{(\lambda+1)\cdot|Y|+|X(1)|}{2}.$$

Consider the remaining drivers in $\bar{S}_{\overline{X(1)}}=\bar{S}\setminus\bar{S}_{X(1)}$. Each driver $x\in\bar{S}_{\overline{X(1)}}$ serves at least two trips, implying $|\sigma(\bar{S}_{\overline{X(1)}})|\geq 2|\bar{S}_{\overline{X(1)}}|$. Let Y' be the set of drivers in S^* that serve all of $\bigcup_{x\in\bar{S}_{\overline{X(1)}}}\sigma(x)$ in (S^*,σ^*). Any driver $x\in\bar{S}_{\overline{X(1)}}$ is not in $\sigma(\bar{S}_{X(1)})$ by the definition of $\bar{S}_{\overline{X(1)}}$, and x is not in $\mathcal{A}(D^*_{\mathrm{node}(x')})$ for any $x'\in X(1)$ since otherwise, x' would have served x. From these, for every $y'\in Y'$, $y'\notin\mathcal{A}(D^*_{\mathrm{node}(x)})\cup\mathcal{A}(A^*_{\mathrm{node}(x)})$ for all $x\in\bar{S}_{X(1)}$, which implies that $|Y\cup Y'|=|Y|+|Y'|$. Similar to Y, each driver in Y' can serve at most $\lambda+1$ trips of $\bigcup_{x\in\bar{S}_{\overline{X(1)}}}\sigma(x)$. Hence, $|Y'|\geq 2|\bar{S}_{\overline{X(1)}}|/(\lambda+1)$, implying $((\lambda+1)\cdot|Y'|)/2\geq|\bar{S}_{\overline{X(1)}}|$. Each $y\in Y\cup Y'$ must be in either $\sigma(S\setminus S_{\mathrm{II}})$ or $\sigma(S_{\mathrm{II}})$ since all trips must be served at the end by the algorithm. In other words, if $y\in S\setminus\bar{S}$, y has been considered in Equation (3.4). This means that we only need to consider $\bar{S}$, and the ratio between $|\bar{S}|$ and $|Y\cup Y'|$ is

$$\begin{aligned}\frac{|\bar{S}|}{|Y\cup Y'|}=\frac{|\bar{S}_{X(1)}|+|\bar{S}_{\overline{X(1)}}|}{|Y|+|Y'|} &\leq \frac{((\lambda+1)\cdot|Y|+|X(1)|)/2+((\lambda+1)\cdot|Y'|)/2}{|Y|+|Y'|}\\ &= \frac{(\lambda+1)\cdot(|Y|+|Y'|)+|X(1)|}{2(|Y|+|Y'|)}\\ &= \frac{\lambda+1}{2}+\frac{|X(1)|}{2(|Y|+|Y'|)}\\ &\leq \frac{\lambda+1}{2}+\frac{1}{2}=\frac{\lambda+2}{2}.\end{aligned}\tag{3.5}$$

Finally, we calculate the ratio between S and S^*, where $F_{\mathrm{II}}\cup Y\cup Y'\subseteq S^*$. Recall that $S=\bar{S}\cup S_{\mathrm{II}}\cup F_{\mathrm{II}}$ and all trips served by each driver $x\in\bar{S}$ are in X. From this and the same reason stated in the proof of Lemma 3.22 for Equation (3.4), the minimum number of drivers in each set of F_{II}, Y and Y' is calculated based on using all capacity λ of every

driver $u \in F_{\mathrm{II}} \cup Y \cup Y'$. Hence, with Equations (3.4) and (3.5),

$$\begin{aligned} |F_{\mathrm{II}} \cup Y \cup Y'| &\geq 2|S_{\mathrm{II}} \cup F_{\mathrm{II}}|/(\lambda+2) + 2|\bar{S}|/(\lambda+2) \\ &= 2(|S_{\mathrm{II}} \cup F_{\mathrm{II}}| + |\bar{S}|)/(\lambda+2) \end{aligned}$$

The ratio between S and S^* is

$$\begin{aligned} \frac{|S|}{|S^*|} \leq \frac{|S|}{|F_{\mathrm{II}} \cup Y \cup Y'|} &\leq \frac{|\bar{S}| + |S_{\mathrm{II}} \cup F_{\mathrm{II}}|}{|F_{\mathrm{II}} \cup Y \cup Y'|} \\ &\leq \frac{|\bar{S}| + |S_{\mathrm{II}} \cup F_{\mathrm{II}}|}{2(|S_{\mathrm{II}} \cup F_{\mathrm{II}}| + |\bar{S}|)/(\lambda+2)} \\ &= \frac{\lambda+2}{2} \end{aligned} \tag{3.6}$$

Therefore, it requires at least $\frac{2|S|}{\lambda+2}$ drivers in S^* to serve all of $\sigma(S)$. □

Next, we show that Algorithm 6 always computes a valid solution to any instance of RSOneStop. Let (S', σ') be the partial solution computed by Algorithm 6 for a given time point, i.e., current partial solution.

Lemma 3.23. *Let (S, σ) be a complete solution found by Algorithm 6. Then for each pair $i, j \in S$, $\sigma(i) \cap \sigma(j) = \emptyset$ and $\sigma(S) = \mathcal{A}$, implying (S, σ) is indeed a valid solution for an instance $(N, \mathcal{A})$.*

Proof. Phase-I of the algorithm ends until all trips of W are served, that is, $\Gamma(W) = \emptyset$ at the end of Phase-I. During each iteration of Phase-I, a node $\mu \in \Gamma(W)$ containing trips of W is chosen w.r.t. current solution (S', σ'). A trip x is selected from $\hat{X}_1 \cup \bar{X}$, where $\hat{X}_1 = \{i \in S'(A_\mu) \mid \mathrm{free}'(i) > 0 \text{ and } \mathrm{stop}'(i) < \delta_i\}$ and $\bar{X} = \{i \in X \cap \mathcal{A}(A^*_\mu) \setminus \sigma'(S') \mid \mathrm{stop}'(i) < \delta_i \text{ or } i \in \mathcal{A}(\mu)\}$. By the definition of $\hat{X}_1$ and $\bar{X}$, x is either a driver or an unassigned trip that can still serve other trips in $\mathcal{A}(D^*_\mu)$. From this, x is a valid assignment for serving $W(x, \mu, S')$. If $\hat{X}_1 \cup \bar{X} = \emptyset$, each trip of $W(\mu) \setminus \sigma'(S')$ is assigned as a driver to serve itself.

Phase-II of the algorithm ends until all trips of Z are served, where $Z = \{i \in \mathcal{A} \setminus \sigma'(S') \mid \delta_i = 0\}$. Since all of W are served before Phase-II starts, $\lambda_i \geq 1$ for every $i \in \mathcal{A} \setminus \sigma'(S')$, that is, $Z \subseteq X$. From this, every $x \in Z \setminus \sigma'(S')$ that is assigned as a driver to serve other trips in $Z(x, \mathrm{node}(x), S')$ is valid, as described in the first part (first for-loop) of Phase-II. The second part of Phase-II is similar to Phase-I. A node $\mu \in \Gamma(Z)$ is chosen, where $\Gamma(Z)$ is the set of nodes containing the rest of Z w.r.t. (S', σ'). Either a driver $x \in \hat{X}_2$ or an unassigned trip $x \in \bar{X}$ is selected to serve $Z(x, \mu, S')$, where $\hat{X}_2 = \{i \in S(A^*_\mu) \mid \mathrm{free}(i) > 0 \text{ and } (\mathrm{stop}(i) < \delta_i \text{ or } i \in \mathcal{A}(\mu))\}$ and $\bar{X}$ is the same as defined above. The assignment of x is valid as mentioned above.

In Phase-III of the algorithm, the rest of $X \setminus Z$ are served. The algorithm processes each node from μ_p to μ_1. All trips in $\mathcal{A}(\mu_j)$ must be served before μ_{j-1} is processed. In each

iteration, either a driver $x \in \hat{X}_2$ (as defined above) or an unassigned trip $x \in X(\mu) \setminus \sigma'(S')$ is selected to serve $\mathcal{A}(x, \mu, S')$. The assignment of x is valid as mentioned above. Therefore, Algorithm 6 produces a valid solution. □

Theorem 3.27. *Let $(N, \mathcal{A})$ be an instance of RSOneStop satisfying the transitive serve relation, (S^*, σ^*) be an optimal solution for $(N, \mathcal{A})$ and $\lambda = \max_{i \in \mathcal{A}} \lambda_i$. Algorithm 6 computes a solution (S, σ) for $(N, \mathcal{A})$ such that $|S^*| \leq |S| \leq \frac{\lambda+2}{2}|S^*|$ with running time $O(M + l^2)$, where M is the size of the ridesharing instance which contains a road network and l trips.*

Proof. By Lemma 3.22, Lemma 3.23 and Lemma 3.10, Algorithm 6 computes a solution (S, σ) for $(N, \mathcal{A})$ with $\frac{\lambda+2}{2}$-approximation ratio. It takes $O(M)$ time to construct the simplified serve relation meta graph $\Gamma(V, E)$. The labeling of nodes in Γ takes $O(l)$ time. Sorting the trips in a node μ according to their capacity takes $O(\lambda \cdot |\mathcal{A}(\mu)|)$ time for each node μ, so in total $O(\lambda \cdot l)$ to sort all trips in $\mathcal{A}$. The total time for the preprocessing is $O(M + \lambda \cdot l)$; we assume $\lambda < l$. For Phase-I, there are at most $O(l)$ iterations (in the while-loop). In each iteration, it takes $O(l)$ time to pick the required node μ from $\Gamma(W)$ and $O(l)$ time to select a trip x from $\hat{X}_1 \cup \bar{X}$. To serve all of $W(x, \mu, S')$ or $W(\mu)$, $|W(\mu)| \leq O(l)$ is required. Hence, Phase-I runs in time $O(l^2)$. For Phase-II, we can first scan the tree Γ following the node labels in decreasing order, which takes $O(l)$ time. Whenever a node μ with $|Z(\mu)| \geq 2$ is encountered, a trip $x \in Z(\mu) \setminus \sigma'(S')$ is selected to serve $Z(x, \mu, S')$ repeated until $|Z(\mu)| \leq 1$. This takes $O(l)$ time since the trips in $\mathcal{A}(\mu)$ are sorted according to their capacity. Hence, it takes $O(l^2)$ time for the first for-loop in Phase-II. The second for-loop in Phase-II is similar to Phase-I, which requires $O(l)$ time for each iteration. Thus, it requires $O(l^2)$ time for Phase-II. For Phase-III, in each iteration when processing a node μ, it takes $O(l)$ time to select a trip x from $\hat{X}_2$ or $X(\mu) \setminus \sigma'(S')$. Then in total, it requires $O(l + \lambda)$ time to serve $\mathcal{A}(x, \mu, S')$. Collectively, Phase-III may require $O(l)$ iterations to process trips of all nodes in $V(\Gamma)$. Thus, it requires $O(l^2)$ time for Phase-III. Therefore, the running time of Algorithm 6 is $O(M + l^2)$. □

3.5 Summary

In this chapter, we show that restricted variants of the ridesharing problem are NP-hard. Specifically, if any one of Conditions C1-C5 is not satisfied, both RSOne (RSOne*) and RSTwo (RSTwo*) are NP-hard, even if all four other conditions are satisfied. Further, when one of Conditions C2-C5 is not satisfied, both RSOne and RSTwo are NP-hard to approximate within a constant factor. Then, we give two polynomial-time exact algorithms for RSOne and RSTwo when all of C1-C5 are satisfied: One is a dynamic programming algorithm that can solve RSOne and RSTwo, and the other algorithm is more efficient than the dynamic programming algorithm that can solve only RSOne. Both of these two algorithms utilize the serve relation graph, introduced in Section 2.3. An description of

how to implement the serve relation graph in linear time (in the input size) is given in Subsection 3.3.2. Three polynomial-time approximation algorithms are presented for the ridesharing problem RSOneStop (an instance of RSOne that satisfies Conditions C1-C3 and C5). Two of these three approximation algorithms are modified from the approximation algorithms for the MCMP [65]. The third approximation algorithm is our novel algorithm that is much more efficient than the two aforementioned approximation algorithms. The speed-up is realized by further utilizing the serve relation graph.

Chapter 4

Ridesharing with Profit Constraint Problem

In this chapter, we give exact and approximation algorithms for the RPC problem on fixed trips. Ridesharing is a promising way to improve transportation efficiency, but the current use of mobility-on-demand (MoD) has actually created a negative effect on the traffic, such as worsen congestion. One of the reasons is that actual ridesharing still has not been adopted in practice despite its potential. An important factor within a ridesharing system is the profit/pricing scheme. Since MoD platform operators and drivers are driven by profit, the intention is to have more rides fulfilled. This results in most ridesharing rides contain only one passenger. Demand-and-pricing of a ridesharing system is important for its adaptability of actual ridesharing in practice [111]. The ridesharing optimization problems we study in this thesis provide a new framework to consider maximizing both the number of passengers served and drivers' profit target. This may balance both sides: profit and transportation efficiency. In particular, the main optimization goal aims at improving the number of passengers served and occupancy rate of vehicles while meeting an overall drivers' profit target.

The studying of this problem is motivated by the fact that profits-as-incentives may promote ridesharing in practice for both MoD operators and drivers. The potential of ridesharing has been recognized in the academia, but the potential of ridesharing in profit-maximizing platforms/MoDs is not well understood. We propose exact and approximation algorithms for the optimization problems studied. We now formally introduce the problem *Ridesharing with Profit* (RP).

In the centralized ridesharing system (CRS), we are given two sets of trips: a set $D = \{\eta_1, \ldots, \eta_k\}$ of k drivers (each operates a vehicle), and a set $R = \{r_1, \ldots, r_l\}$ of l trip requests from passengers. Each driver $\eta_i \in D$ and each passenger $r_i \in R$ have the same parameters related to a road network N as stated in Section 2.3, Table 2.1. In addition to those parameters, a maximum trip time (duration) γ_i is given by each driver $\eta_i \in D$ and each passenger $r_i \in R$. A maximum trip time γ_i means that total duration from o_i to

d_i using any route should not exceed γ_i. In this chapter, we denote a serve relation $\sigma(i)$ more explicitly by (η_i, R_i) ($\sigma(i) = \{\eta_i\} \cup R_i$) between a driver η_i and a group of passengers $R_i \subseteq R$; and call (η_i, R_i) a *match*. A *match* (η_i, R_i) is *feasible* if there exists a *feasible path* $\text{FP}(\eta_i, R_i)$ in N by which all passengers are delivered satisfying all constraints of a feasible serve relation stated in Section 2.3; in other words, the feasible path $\text{FP}(\eta_i, R_i)$ is path $P(i, J)$ stated in the ridesharing route constraint. In addition, the travel time constraint is extended to include the maximum trip time, specifically:

- Travel time constraint: each trip $j \in \{\eta_i\} \cup R_i$ departs from o_j no earlier than α_j, arrives at d_j no later than β_j, and the total travel duration of j is at most γ_j.

An *assignment* Π is a set of feasible matches such that for every two matches (η_i, R_i) and (η_j, R_j) in Π, $\eta_i \neq \eta_j$ and $R_i \cap R_j = \emptyset$. Each feasible match (η_i, R_i) is associated with a revenue $rev(\eta_i, R_i)$ computed by the CRS; most or all of $rev(\eta_i, R_i)$ are given to the driver η_i for serving all of R_i. The revenue $rev(\eta_i, R_i)$ is assumed to be computed based on $\text{FP}(\eta_i, R_i)$ and should be at most what the passengers of R_i pay to the CRS. Each match (η_i, R_i) is associated with a travel cost $tc(\eta_i, R_i)$ for η_i to traverse $\text{FP}(\eta_i, R_i)$. The profit of a feasible match (η_i, R_i) is $w(\eta_i, R_i) = rev(\eta_i, R_i) - tc(\eta_i, R_i)$, which can be negative, and we assume it is expressed in integers (e.g., cents, smallest payable amount). For an assignment Π, let $w(\Pi) = \sum_{(\eta_i, R_i) \in \Pi} w(\eta_i, R_i)$ be the profit of Π and $R(\Pi) = \cup_{(\eta_i, R_i) \in \Pi} R_i$ be the set of passengers in Π.

The RP problem is to find an assignment Π such that $w(\Pi)$ is maximized. In this thesis, we focus on a more complex optimization problem, called the *Ridesharing with Profit Constraint* (RPC) problem. In application, the CRS may want to serve as many passengers as possible while maintaining a *profit target* c. With this in mind, we introduce the RPC problem: which is to find an assignment Π such that $|R(\Pi)|$ is maximized and $w(\Pi) \geq c$ for some integer c.

4.1 Model

Our approach uses a model similar to the one proposed in [6, 95], which is similar to [36, 42, 99, 102]. All feasible matches between all drivers and passengers are computed first; and then based on some optimization goal/objective, an assignment consisting of a set of disjoint feasible matches is computed. Our model for the RPC problem allows a flexible pricing for the MoD system operators; and different pricing schemes (e.g., [67, 72, 113]) can be incorporated into our model. Further, Our approach allows for a more general set of drivers, including personal/ad-hoc drivers and designated drivers (e.g., taxi drivers).

The revenue $rev(\eta_i, R_i)$ given by the CRS to driver η_i is heavily impacted by $\text{FP}(\eta_i, R_i)$. We assume that every driver uses the feasible path computed by the CRS. Specifically, for an assignment $\Pi = \{(\eta_i, R_i) \mid \eta_i \in D, R_i \subseteq R\}$, each driver η_i in the assignment

follows a *shortest feasible path* $\text{SFP}(\eta_i, R_i)$ to serve passengers in R_i. Here, shortest means either shortest distance or shortest time; and it is predefined by the CRS. For example, let $R_i = \{r_a, r_q\}$ be the set of passengers in match (η_i, R_i). There are six different visiting orders of R_i in which the passengers of R_i can be picked-up and dropped-off by η_i, which correspond to six paths in road network N (each starts at o_i and ends at d_i). Below are the six visiting orders of R_i for driver η_i:

$$\{(o_i, o_a, o_q, d_a, d_q, d_i), (o_i, o_a, d_a, o_q, d_q, d_i), (o_i, o_a, o_q, d_q, d_a, d_i),$$
$$(o_i, o_q, o_a, d_a, d_q, d_i), (o_i, o_q, d_q, o_a, d_a, d_i), (o_i, o_q, o_a, d_q, d_a, d_i)\}.$$

Path $\text{SFP}(\eta_i, R_i)$ is the path in N corresponds to one of the six visiting orders that is feasible and has either the shortest distance or shortest time. Then, the RP problem can be formulated as follows.

$$\max_{\Pi} \quad \sum_{(\eta_i, R_i) \in \Pi} w(\eta_i, R_i) \tag{i}$$

$$\text{subject to} \quad \eta_i \neq \eta_j \wedge R_i \cap R_j = \emptyset, \qquad \forall (\eta_i, R_i) \neq (\eta_j, R_j) \in \Pi \tag{ii}$$

The objective function (i) is to maximize the overall profit obtained from served trips. Constraint (ii) ensures that each passenger request is assigned to only one driver and each driver serves at most one feasible match (a unique group of passengers). Note that not all requests of R are required to be served in an assignment Π. Again, our main focus is the RPC problem, and it can be formulated as follows.

$$\max_{\Pi} \quad \sum_{(\eta_i, R_i) \in \Pi} |R_i| \tag{iii}$$

$$\text{subject to} \quad \eta_i \neq \eta_j \wedge R_i \cap R_j = \emptyset, \qquad \forall (\eta_i, R_i) \neq (\eta_j, R_j) \in \Pi \tag{iv}$$

$$\sum_{(\eta_i, R_i) \in \Pi} w(\eta_i, R_i) \geq c \tag{v}$$

The objective function (iii) is to maximize the total number of passengers served. Constraint (iv) is the same as constraint (ii). Constraint (v) ensures the system profit meets a given target. An assignment Π containing any feasible match (η_i, R_i) with negative profit $(w(\eta_i, R_i) < 0)$ means that the driver η_i loses money.

As can be seen in the formulation (i)-(ii) (and formulation (iii)-(v)), we only need a few sets of constraints. The more traditional formulations for the ridesharing problems, VRP and DARP usually require more than 10 sets of constraints. For example, the mathematical formulation in [97] has 13 sets of constraints, not including the range constraints on the variables. A simpler formulation is very useful in computational studies for the problem.

We construct an integer-weighted hypergraph $H(V, E, w)$ to represent the formulation (iii)-(v) as follows. Initially, $V(H) = D \cup R$. For each $\eta_i \in D$ and for every subset R_i of R with $1 \leq |R_i| \leq \lambda_i$, create a hyperedge $e = \{\eta_i\} \cup R_i$ in $E(H)$ if (η_i, R_i) is a feasible

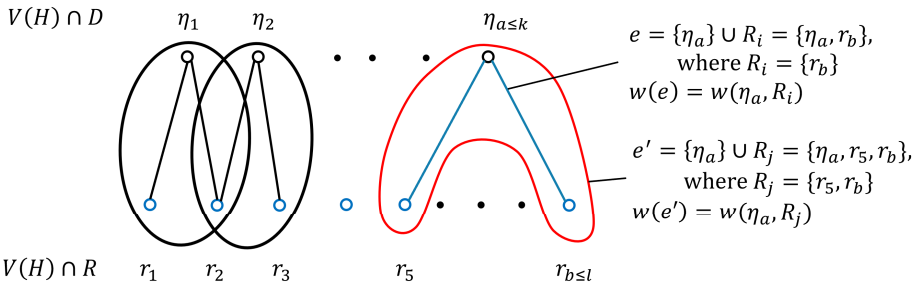


Figure 4.1: A bipartite hypergraph $H(V, E, w)$ representing all feasible matches of an instance $(N, \mathcal{A})$, where $|D(H)| = a$ and $|R(H)| = b$.

match. Each edge $e = \{\eta_i\} \cup R_i \in E(H)$ has weight $w(e) = w(\eta_i, R_i)$, the profit of η_i. Remove all isolated vertices from H. An example of $H(V, E, w)$ is shown in Figure 4.1. Let H^- be the subgraph of H such that H^- contains all edges of H with negative weight and $H^+ = H \setminus H^-$. There are at most $\sum_{1 \le a \le \lambda_i} \binom{l}{a}$ edges incident to each η_i in H. Let $\lambda = \max_{\eta_i \in D} \lambda_i$. If λ is a small constant, the size of H is polynomially bounded.

We use the concept of matching/set packing in our approach, where the definitions of matching have been stated in Section 2.2. In particular, the RPC problem is to find a matching M in $H(V, E, w)$ such that $\sum_{\{\eta_i\} \cup R_i \in M} |R_i|$ is maximized and $w(M) > c$. Let M_1 and M_2 be two matchings in an edge-weighted (hyper)graph $G(V, E)$. Denoted by $F(V, E) = M_1 \Delta M_2$ is the resulting graph of the symmetric difference of M_1 and M_2. Let $\mathcal{F}$ be the set of connected components in F. It is well known that when G is a graph, each component of $\mathcal{F}$ is either a path or an even cycle where the edges are alternating between M_1 and M_2. For any subset $\mathcal{F}_1 \subseteq \mathcal{F}$, let $E(\mathcal{F}_1) = \cup_{C \in \mathcal{F}_1} E(C)$ and $w(\mathcal{F}_1) = \sum_{C \in \mathcal{F}_1} w(E(C))$.

Let c^* be the weight of a maximum weight matching in the above constructed hypergraph H. It is easy to see that finding a maximum weight matching M^* in H^+ solves the formulation (i)-(ii) and vice versa. Since formulation (i)-(ii) is a weighted $(\lambda+1)$-set packing formulation, finding M^* and c^* (RP problem) is NP-hard in general for $\lambda \ge 2$ [40, 62].

Theorem 4.1. *The RPC problem is NP-hard for an arbitrary target c and $\lambda \ge 2$.*

Proof. Given the formulation (i)-(ii) for an instance of the RP problem, construct an instance of the RPC problem containing the same set of feasible matches $\cup_{\eta_i \in D, R_i \subseteq R}(\eta_i, R_i)$. For a driver $\eta_i \in D$, let ω_i^* be the profit of the match containing η_i with the largest profit. Set the profit target $c = \sum_{\eta_i \in D} \omega_i^*$ for the constructed RPC problem instance. Note that a driver η_i cannot appear in more than one match in any solution of the RP and RPC problems. Hence, the RPC problem has a feasible solution if and only if the objective function value of formulation (i)-(ii) is equal to c. Since solving the formulation (i)-(ii) is NP-hard for $\lambda \ge 2$, the RPC problem is NP-hard for $\lambda \ge 2$. □

Further, Hazan et al. [51] showed that the $(\lambda + 1)$-set packing problem cannot be approximated to within $\Omega(\frac{\ln(\lambda+1)}{\lambda+1})$ in general for $\lambda \ge 2$. There exists a polynomial-time $\frac{2}{\lambda+2}$-

approximation algorithm for approximating the maximum profit of Π [16]. However, similar algorithms [16, 24] cannot be directly applied to the RPC problem since these algorithms only approximate the maximum profit $w(\Pi)$ and do not consider the size of each subset (match) in Π and the different elements in the subset/match. Algorithms for the maximum set packing problem (e.g., [38, 103]) cannot apply to the RPC problem either since such algorithms do not consider general integer weight.

Due to the NP-hardness of the RPC problem (Theorem 4.1) and the inapproximability of the weighted set packing problem, we study two variants of the RPC problem: RPC1 and RPC+. The RPC1 problem variant assumes that for a given instance of the RPC problem, $\lambda_i = 1$ for every driver $\eta_i \in D$ ($\lambda = 1$). To solve the RPC1 variant, we use an approach in solving the maximum matching problem on bipartite graphs. For the RPC+ problem variant, we include one more constraint (called the *non-negative profit constraint*) to the formulation (iii)-(v) of the RPC problem:

$$w(\eta_i, R_i) \geq 0, \forall (\eta_i, R_i) \in \Pi. \tag{vi}$$

To solve the RPC+ variant, we use a local search approach similar to the ones in [16, 24].

4.2 Related work

As seen in Section 4.1, the RP problem can be formulated as a maximum weight set packing problem, but the RPC problem cannot. To the best of our knowledge, the RPC optimization problem has not been studied before, whether in the form of set packing problem or matching in hypergraphs. Having said that, there are studies on ridesharing with profit. Hsieh [56] proposed a model to match drivers and passengers such that the ratio between the cost savings and the original costs is maximized. A non-linear integer programming is given and only (meta)heuristics are discussed. An interesting study [112] suggests that the pickup locations (origins) of passengers that are matched to a driver should not be too far away from the driver. The authors of [112] further suggested that when the destination of a passenger is in a high demand zone (or zone with low driver supply) should have a higher priority to be matched. Such an approach may balance the supply and demand in a non-balanced zone, increasing the overall matching rate.

A class of the vehicle routing problem (VRP) focuses on profit, called Vehicle Routing Problems with Profits (VRPPs). The variant of VRPPs that is related to RPC is called the Team Orienteering Problem (TOP) [8, 49]. The TOP can be summarized as follows (using ridesharing terms). A set of nodes is given, each represents a customer/passenger and has a score/profit. A set K of vehicles is given. When a node is visited by a vehicle, the profit is collected, and each node can be visited at most once. The goal of TOP is to find at most $|K|$ vehicle routes that maximize the total collected profit, while satisfying a maximum duration constraint for each route [8]. One major difference between RPC and TOP is that

the profit of a passenger is not fixed for RPC but fixed for TOP. For RPC, the profit of driver η_i for serving a group R_i of passengers is path $\text{FP}(\eta_i, R_i)$ dependent, which can be different for different drivers. Another major difference is that the vehicles in TOP do not have a capacity constraint. In the literature of TOP, many studies focus on exact algorithms (IP-solver dependent), which can only solve small instances (less than 100 nodes when time windows and travel time are considered) in a reasonable time [49]. Many heuristics are also proposed for TOP. Most experiments for these heuristics are also for small common benchmark instances. Some heuristics are tested on medium sized instances (less than 500 nodes); and majority of them can finish in a reasonable time. The instances used in TOP usually have a small number of vehicles (less than 5), which is not a representative of the ridesharing problem. In our experiment for RPC (Section 4.5), we have over 500 drivers and 2000 passengers.

In many previous studies, ridesharing with profit is studied as taxi ridesharing [61, 78, 90, 97]. In [61], two optimization objectives are considered, and one of them is to maximize system profit from selectively accepting passengers. The optimization problem is solved by using hybrid-simulated annealing, as mentioned in Section 3.1. The system model proposed in [78] handles dynamic ridesharing requests by incorporating monetary constraints in their proposed taxi-dispatch system. The monetary constraints ensure that drivers make more money and passengers pay less, compared to as if they are in non-shared taxi rides. Their taxi-dispatch system focuses on fulfilling the ride requests quickly (satisfying all constraints), and no global optimization goal is considered. The ridesharing problem studied in [90] is the TGR system model mentioned in Section 3.1. Passengers are grouped together if their itineraries are similar and the grouping can reduce the travel distance and not increasing fare paid for each passenger. The optimization goal is to maximize the total travel distance saved of all passengers. An exact and two heuristics (one is greedy) are proposed in [90]. The DARP/ridesharing problem studied by Santos and Xavier [97] also allows taxi drivers and personal drivers. Their optimization goal is to maximize the number of served passengers and minimize the total cost at the same time (a multi-criteria objective function). A mathematical model is presented and solved by a heuristic, as mentioned in Section 3.1.

In general, a passenger r_i in a shared-ride does not want to share the ride with another passenger r_j if r_i needs to pay more or have a much longer ride by doing so. This can lead to some limitations for ridesharing by taxi, compared to ridesharing by personal vehicles. (1) A taxi driver expects to earn more than a personal vehicle driver for serving a passenger, which is highly affected by total travel distance and duration. As a result, it would be expected from the taxi passengers that they would have a shorter travel distance/time, compared to rides shared by personal vehicles. This can result in less shared rides as a whole. (2) When a passenger requests for a taxi, the wait time expected by the passenger is usually lower, compared to that for a personal vehicle. The taxi ridesharing model needs to have a tighter maximum passenger wait time, which can lead to a lower successful match rate.

There are also studies that focus on the design and effectiveness of pricing schemes related to monetary profit and cost for drivers and passengers, such as the studies in [10, 17, 67, 72, 88, 113, 114]. These studies propose different pricing scheme that should charge the passengers under different scenarios. The main purpose of these studies focus on how to improve passenger ridership based on pricing (or verify that demand can be met under the available supply); and some studies discuss how to improve profit for ridesharing platforms or drivers without damaging the demand. Dynamic pricing (also called surge pricing) is considered in many studies. Dynamic pricing means that the prices for rides change based on real-time demand and supply conditions, which can be affected by both time and location. There is a line of recent research that focuses on platform (MoD) equilibrium analysis based on pricing factors (e.g., [17, 21, 57, 67, 114, 115]). There are two types of equilibrium: market equilibrium and network equilibrium. Market equilibrium concerns about the balance of supply and demand. Network equilibrium concerns about the matchability of drivers and passengers, that is, at the equilibrium state, no one can further reduce their effective travelling cost by changing decisions, such as using a different route. Most studies focus on market equilibrium, namely, proposing an optimal pricing strategy to achieve the balance of supply and demand. Dynamic pricing is a common tool to achieve market equilibrium. Instead of choosing a theoretical pricing scheme from one of the studies in the literature, we closely estimated the pricing scheme used by a current large ridesharing operator in practice, Uber. Then, we use this estimated pricing scheme for our experiments in Section 4.5.

4.3 RPC1 variant - capacity of one

In this section, we consider a simplified variant, denoted as RPC1, where $\lambda = 1$. In this case, the weighted hypergraph $H(V, E, w)$, constructed in Section 4.1, becomes a weighted bipartite graph. A solution to the RPC problem for $\lambda = 1$ (with $c \leq c^*$) is a matching M in H with $w(M) \geq c$ and $|M|$ maximized. We first give a polynomial-time exact algorithm (referred to as **ExactNF**) framework that uses network flow to find an optimal solution. Then, we describe two implementations of ExactNF that are suitable for practice. Finally, we give a simple $\frac{1}{2}$-approximation algorithm (referred to as **Greedy**).

4.3.1 Exact algorithm

The framework description of ExactNF is given below.

1. Construct a flow network $FN(V, E)$ from H, where $V(FN) = \{s, t\} \cup V(H)$, s is the source, and t is the sink. For each $\eta_i \in V(H)$, create an edge (s, η_i) in $E(FN)$ with cost 0 and capacity 1. For each $\{\eta_i, r_j\} \in E(H)$, create an edge (η_i, r_j) in $E(FN)$ with cost $-w(\eta_i, r_j)$ and capacity 1. For each $r_j \in V(H)$, create an edge (r_j, t) in $E(FN)$ with cost 0 and capacity 1. Note that the maximum amount of flow that can be sent

from s to t in FN is at most $n_{min} = \min\{|V(H) \cap D|, |V(H) \cap R|\}$. An example of the flow network $FN(V, E)$ is given in Figure 4.2.

2. For $1 \leq y \leq n_{min}$, find a minimum cost flow f_y of value y (sent from s to t) or conclude that there is no flow of value y in FN.

3. For an edge $e \in E(FN)$, let $f_y(e)$ be the flow value passing through e in f_y. Let $c(f_y) = \sum_{e \in E(FN)|f_y(e)>0} w(e)$ be the cost of flow f_y. If there is a flow f_y computed in Step 2 with $c(f_y) \leq -c$, then $y = \text{argmax}_y - c(f_y) \geq c$, and output the edges $\cup_{e \in E(FN)|f_y(e)>0 \wedge e \in E(H)}$ with positive flow value in f_y as solution M; otherwise, conclude there is no matching in H with profit at least c.

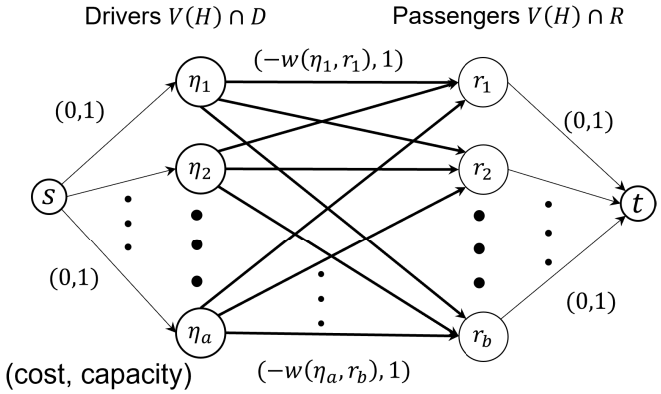


Figure 4.2: The flow network $FN(V, E)$ in the ExactNF algorithm, constructed from H, where $a = |V(H) \cap D|$ and $b = |V(H) \cap R|$.

Theorem 4.2. *Algorithm ExactNF finds a matching M with $w(M) \geq c$ and $|M|$ maximized or concludes that there is no matching M with $w(M) \geq c$ in H in polynomial time.*

Proof. The edges $\cup_{e \in E(FN)|f_y(e)>0 \wedge e \in E(H)}$ of a flow f_y form a matching M in H of cardinality y. Since the cost $c(f_y)$ is minimum among all flows of value y and the profit $w(M)$ is the negation of $c(f_y)$, $w(M)$ is maximum of all matchings in H of cardinality y. If there is a matching M in H with $w(M) \geq c$, then $1 \leq |M| \leq n_{min}$ (since $\lambda = 1$) and Algorithm ExactNF finds the matching M of the largest cardinality with $w(M) \geq c$. An upper bound on the running time of ExactNF is $O(n_{min} \cdot t(FN))$, where $t(FN)$ is the time to compute a min-cost flow f_y and is polynomial in the size of FN [3]. □

Next, we describe two practical implementations of ExactNF (referred to as Algorithm **ExactNF1** and Algorithm **ExactNF2**). The real computational time of $t(FN)$ depends on the algorithm used to compute f_y. Each of the implementations has its own approach to compute f_y. Algorithm **ExactNF1** uses a Linear Programming approach and is described in the following.

1. Let N' be the network $FN(V, E)$ constructed above without the edge costs. Construct a maximum flow LP formulation for N' and solve the LP. Let y^* be the maximum flow value (optimal solution of the LP).

2. For $y = y^*$ to 1, find a minimum cost flow of value y in FN (sent from s to t) by solving another LP formulation for a min-cost flow in FN. If $c(f_y) \leq -c$, stop and output $\cup_{e \in E(FN) | f_y(e) > 0 \wedge \{u,v\} \in E(H)}$. Otherwise, continue until $c(f_y) \leq -c$ or conclude there is no such flow in FN, implying H does not have a feasible solution.

In the worst case, such a simple implementation may need to solve the min-cost flow at most n_{min} times. However, most (majority) of the edges in H have positive weight (negative weight in FN) in practical scenarios, the minimum cost flow f_y that satisfies $c(f_y) \leq -c$ usually is or nearly is a maximum flow, that is, $y = y^*$ or y is close to y^*. Further, if c is smaller than the largest profit obtainable for a noticeable amount (say 20%), y is very close to y^*. These suggest that the number of min-cost flows to be computed is a small constant in practice. We implemented this approach using CPLEX, and it is fast in most cases as shown in the experiment Subsection 4.5.4. From Theorem 4.2, we have the following corollary.

Corollary 4.1. *Algorithm ExactNF1 finds a matching M with $w(M) \geq c$ and $|M|$ maximized or concludes that there is no matching M with $w(M) \geq c$ in H in $O(n_{min} \cdot t(FN))$ time, where $t(FN)$ is the time to find a min-cost flow f_y by an LP solver.*

The second approach for computing f_y is by graph algorithms described in [3]: Let f_y be the min-cost flow of flow value y in FN and N_{f_y} be the residual network of FN w.r.t. f_y, where $N_{f_0} = FN$. We compute f_y in $N_{f_{y-1}}$ w.r.t. f_{y-1} for $y = 1, 2, \ldots, n_{min}$. As shown in [3], the min-cost flow f_y can be computed by the successive shortest path algorithm. The detailed implementation of Algorithm **ExactNF2** is described below.

1. If $FN(V, E)$ has negative weighted edges, first change the negative edge weights into non-negative weights as in Johnson's shortest path algorithm [29, 60]: Use Bellman-Ford algorithm to compute the shortest distance $\text{dist}(s, u)$ for every $u \in V(FN)$. Assign $h(u) = \text{dist}(s, u)$ and compute a new cost $\hat{w}(u, v) = w(u, v) + h(u) - h(v)$ for every $(u, v) \in E(FN)$, then $\hat{w}(u, v) \geq 0$. Label the network with the new weights as $\hat{N}(V, E)$ (a min-cost flow f_y of value y in $\hat{N}$ is a min-cost flow of value y in FN).

2. The rest is similar to the successive shortest path algorithm, except the stopping conditions are different. For completeness and implementation detail, we give the details of the successive shortest path algorithm, including how our stopping conditions are applied. Further, definitions used in the correctness proof of Algorithm ExactNF2 are also introduced.

   First, initialize the *node potential* $\pi(u) = 0$ for each $u \in V(\hat{N})$ and *reduced cost* $w_\pi(u, v) = \hat{w}(u, v) - \pi(u) + \pi(v)$ (which is $\hat{w}(u, v)$ initially) for each $(u, v) \in E(\hat{N})$.

Let f_0 be an empty initial flow on $\hat{N}$. Let $\hat{N}_{f_{y-1}}(\pi)$ be the residual network w.r.t. flow f_{y-1} and reduced costs w_π, where $\hat{N}_{f_0}(\pi) = \hat{N}$.

For $y = 1, \ldots, n_{min}$, find a minimum cost flow f_y of value y from f_{y-1} as follows. Compute single-source shortest paths from s, $SSSP(s)$, to every other vertex in $u \in V(\hat{N}_{f_{y-1}}(\pi))$ to get $\text{dist}_{y-1}(s,u)$. Update node potential $\pi(u) = \pi(u) - \text{dist}_{y-1}(s,u)$ for every $u \in V(\hat{N})$. Update reduced cost for every edge of $\hat{N}_{f_{y-1}}(\pi)$: $w_\pi(u,v) = \hat{w}(u,v) - \pi(u) + \pi(v)$ if $(u,v) \in E(\hat{N}) \cap E(\hat{N}_{f_{y-1}}(\pi))$; otherwise, $w_\pi(u,v)$ is unchanged. Let P_{y-1} be the shortest $s-t$ path found by $SSSP(s)$. Then, augment flow along P_{y-1} to get a flow f_y, and construct the residual network $\hat{N}_{f_y}(\pi)$ w.r.t. f_y and w_π. Note that each edge (u,v) in P_{y-1} is in $\hat{N}_{f_{y-1}}(\pi) \setminus \hat{N}_{f_y}(\pi)$; and its reduced cost $w_\pi(u,v)$ is updated prior to augmenting flow along P_{y-1}. After the augmentation, each such edge (u,v) in P_{y-1} has reduced cost $w_\pi(v,u) = -w_\pi(u,v)$ in $\hat{N}_{f_y}(\pi)$. Due to the properties of the successive shortest path algorithm, $w_\pi(u,v) \geq 0$ for every $(u,v) \in E(\hat{N}_{f_y}(\pi))$ (reduced cost optimality conditions [3]).

3. Let f_y be the flow found by Step 2 in each iteration. For an edge $(u,v) \in E(\hat{N})$, let $f_y(u,v)$ be the flow value passing through (u,v) w.r.t. f_y. Let $c(f_y)$ be the cost of flow f_y in FN, namely,

$$c(f_y) = \sum_{(u,v) \in E(\hat{N}) | f_y(u,v) > 0} \hat{w}(u,v) + h(v) - h(u) = \sum_{(u,v) \in E(FN) | f_y(u,v) > 0} w(u,v).$$

Stop Step 2 if:

- either the value $y \leq n_{min}$ of flow f_y cannot be increased (an $s-t$ path cannot be found in $\hat{N}_{f_y}(\pi)$), or $c(f_{y+1}) > c(f_y)$ such that $c(f_{y+1}) > -c$. For either case, if $c(f_y) \leq -c$ then output $\cup_{e \in E(\hat{N}) | f_y(e) > 0 \wedge e \in E(H)}$ as a solution. Otherwise, conclude there is no matching in H with profit at least c.

Johnson's shortest path algorithm has the following properties that also apply to ExactNF2.

Property 4.1. *[29, 60].*

a) *For every edge $(u,v) \in E(\hat{N})$, $\hat{w}(u,v) \geq 0$.*

b) *A flow f_y of flow value y is a min-cost flow in FN if and only if f_y is a min-cost flow of flow value y in $\hat{N}$.*

Proof. Since there is no cycle in $FN(V,E)$, there is no negative cycle in $FN(V,E)$. There are only out-going edges from source s with zero weight. It follows from the analysis in [29] that part (a) holds; and all edges of $E(FN)$ can be changed to non-negative weighted edges $E(\hat{N})$ by using the Bellman-Ford algorithm.

Part (b), which is an extension of a property in [29] on the length of a path in FN and $\hat{N}$. Let $Q' = (s = v_0, v_1, \ldots, v_{z-1}, v_z = t)$ be an $s-t$ path in $\hat{N}$. As shown in [29], the weight of Q' in $\hat{N}$ is

$$\begin{aligned}\hat{w}(Q') = \sum_{1\le j\le z} \hat{w}(v_{j-1}, v_j) &= \sum_{1\le j\le z} w(v_{j-1}, v_j) + h(v_{j-1}) - h(v_j) \\ &= h(s) - h(t) + \sum_{1\le j\le z} w(v_{j-1}, v_j) \\ &= h(s) - h(t) + w(Q'),\end{aligned} \tag{4.1}$$

where $w(Q')$ is the weight of Q' in FN. For any flow f_y in FN or $\hat{N}$, the edges with positive flow of f_y form a set $\mathcal{Q} = \{Q_1, \ldots, Q_y\}$ of edge-disjoint $s-t$ paths since each edge in FN and $\hat{N}$ has unit capacity. From Eq (4.1), the weight of $\mathcal{Q}$ in $\hat{N}$ is

$$\hat{w}(\mathcal{Q}) = \sum_{1\le i\le y} \hat{w}(Q_i) = y(h(s) - h(t)) + \sum_{1\le i\le y} w(Q_i). \tag{4.2}$$

Since $h(s)$ and $h(t)$ are independent of any $s-t$ path, if f_y is a min-cost flow ($\sum_{1\le i\le y} w(Q)$ is minimum) in FN, then f_y is a min-cost flow in $\hat{N}$ and vice versa. □

The successive shortest path algorithm has the following properties that also apply to ExactNF2 since the initial network $\hat{N}$ has non-negative edge weights and $SSSP(s)$ on $\hat{N}$ can be computed correctly by Property 4.1.

Property 4.2. *[3]*

a) *(Reduced cost optimality conditions) For $y \ge 0$ and $\hat{N}_{f_y}(\pi)$, a feasible flow f_y is an optimal solution to the min-cost flow problem if and only if some set of node potentials π satisfy that $w_\pi(u, v) \ge 0$ for every edge $(u, v) \in \hat{N}_{f_y}(\pi)$.*

b) *Sending flow along an $s-t$ shortest path p_y in $\hat{N}_{f_y}(\pi)$ w.r.t. reduced costs w_π still maintains the reduced cost optimality conditions in each iteration.*

Next, we give a general definition of the weight of a path in $\hat{N}_{f_i}(\pi)$ for FN and $\hat{N}$. Let $\hat{N}_{f_i}(\pi)$ be a residual network w.r.t. flow f_i ($0 \le i \le n_{min}$) and reduced cost w_π, where $\hat{N}_{f_0}(\pi) = \hat{N}$. For any path P in $\hat{N}_{f_i}(\pi)$, define the weight of P in FN as

$$w(P) = \sum_{(u,v)\in P\cap E(FN)} w(u, v) - \sum_{(u,v)\in P\setminus E(FN)} w(v, u)$$

and the weight of P in $\hat{N}$ as

$$\hat{w}(P) = \sum_{(u,v)\in P\cap E(\hat{N})} \hat{w}(u, v) - \sum_{(u,v)\in P\setminus E(\hat{N})} \hat{w}(v, u).$$

Note that if $P \setminus E(FN) = \emptyset$, the weight $w(P)$ is the regular definition of $w(P)$ (same for $\hat{w}(P)$ as $E(FN) = E(\hat{N})$).

Corollary 4.2. *Let P be an $s-t$ path in the residual network $\hat{N}_{f_i}(\pi)$ w.r.t. flow f_i and reduced cost w_π. The weight of P in $\hat{N}$ is $\hat{w}(P) = h(s) - h(t) + w(P)$.*

Proof. If $P \setminus E(FN) = \emptyset$, then from Eq (4.1), corollary holds. Suppose $P \setminus E(FN) \neq \emptyset$. Then, the weight of P in $\hat{N}$ is

$$\begin{aligned}
\hat{w}(P) &= \sum_{(u,v)\in P\cap E(\hat{N})} \hat{w}(u,v) - \sum_{(u,v)\in P\setminus E(\hat{N})} \hat{w}(v,u) \\
&= [\sum_{(u,v)\in P\cap E(FN)} w(u,v) + h(u) - h(v)] - [\sum_{(u,v)\in P\setminus E(FN)} w(v,u) + h(v) - h(u)] \\
&= [\sum_{(u,v)\in P\cap E(FN)} w(u,v) + h(u) - h(v)] + [\sum_{(u,v)\in P\setminus E(FN)} h(u) - h(v)] - \sum_{(u,v)\in P\setminus E(FN)} w(v,u) \\
&= h(s) - h(t) + \sum_{(u,v)\in P\cap E(FN)} w(u,v) - \sum_{(u,v)\in P\setminus E(FN)} w(v,u) \\
&= h(s) - h(t) + w(P).
\end{aligned}$$

Hence, we have the corollary. □

Lemma 4.1. *Let f_y and f_{y+1} be two flows found in Step 2 of ExactNF2. If $c(f_y) < c(f_{y+1})$, then $c(f_{y+1}) \leq c(f_z)$ for every flow f_z found after f_{y+1}, $z > y+1$.*

Proof. Assume for contradiction that there is a flow f_z found after f_{y+1} such that $c(f_z) < c(f_{y+1})$ for the smallest $z > y+1$. Let P_y be the path found in $\hat{N}_{f_y}(\pi)$ to get f_{y+1} from f_y (by augmenting flow along P_y), and similarly, let $P_{z-1} = (s, v_1, v_2, \ldots, v_q, t)$ be the path found in $\hat{N}_{f_{z-1}}(\pi)$. Then, $c(f_{y+1}) - c(f_y) = w(P_y) > 0$ and $c(f_z) - c(f_{z-1}) = w(P_{z-1}) < 0$. By Corollary 4.2, $\hat{w}(P_{z-1}) < \hat{w}(P_y)$. If P_{z-1} exists in $\hat{N}_{f_y}(\pi)$, by Property 4.2, the algorithm would have augmented flow along P_{z-1} instead of P_y since it gives a flow f'_{y+1} of value $y+1$ in $\hat{N}$ with a cost lower than that of f_{y+1}. Suppose P_{z-1} does not exist in $\hat{N}_{f_y}(\pi)$. Let $g \geq y+1$ be the smallest iteration such that P_{z-1} exists in $\hat{N}_{f_g}(\pi)$ ($g \leq z-1$). Let $P_{g-1} = (s, u_1, u_2, \ldots, u_x, t)$ be the path found in $\hat{N}_{f_{g-1}}(\pi)$. Some edges of P_{z-1} must have inverse directions in P_{g-1}. Consider any maximal subpath $(v_i, v_{i+1}, \ldots, v_j)$ of P_{z-1} such that the reverse path $(v_j, \ldots, v_{i+1}, v_i)$ is in P_{g-1}. Label P_{z-1} as three subpaths: $P_{z-1}(1) = (s, v_1, \ldots, v_i)$, $P_{z-1}(2) = (v_i, \ldots, v_j)$ and $P_{z-1}(3) = (v_j, \ldots, v_q, t)$. Label P_{g-1} as three subpaths: $P_{g-1}(1) = (s, u_1, \ldots, u_a, v_j)$, $P_{g-1}(2) = (v_j, \ldots, v_i)$ and $P_{g-1}(3) = (v_i, u_b, \ldots, u_x, t)$. Then, $P'_{g-1} = P_{z-1}(1) \cup P_{g-1}(3)$ and $P''_{g-1} = P_{g-1}(1) \cup P_{z-1}(3)$ are two $s-t$ paths in $\hat{N}_{f_{g-1}}(\pi)$ (see Figure 4.3 for an example). Recall that $z > y+1$ is the smallest such that $c(f_z) < c(f_{y+1})$, which means $w(P_{g-1}) \geq 0$ and $w(P_{z-1}) < 0$. Next, we show

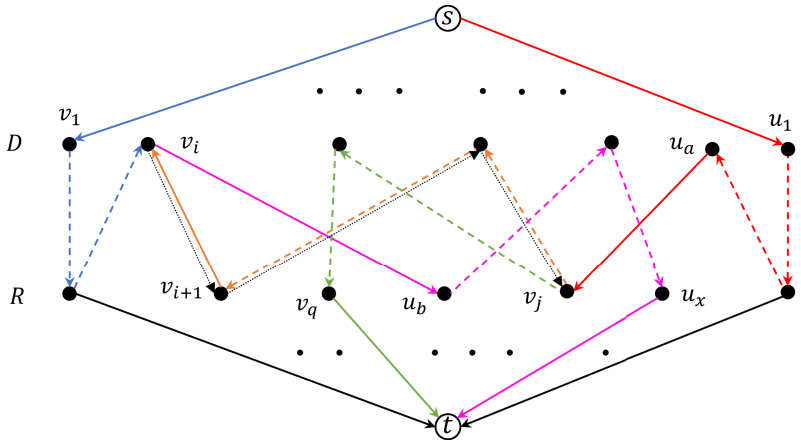


Figure 4.3: Residual network $\hat{N}_{f_{g-1}}(\pi)$. Path $P_{g-1} = (s, u_1, \ldots, u_x, t)$ found in $\hat{N}_{f_{g-1}}(\pi)$ by the algorithm is labeled as three subpaths $P_{g-1}(1)$ (red lines), $P_{g-1}(2)$ (orange lines) and $P_{g-1}(3)$ (pink lines). The path $P_{z-1} = (s, v_1, \ldots, v_q, t)$ exists in $\hat{N}_{f_g}(\pi)$ and is labeled as three subpaths $P_{z-1}(1)$ (blue lines), $P_{z-1}(2)$ (black dotted lines, these edges are in $\hat{N}_{f_g}(\pi)$ and not in $\hat{N}_{f_{g-1}}(\pi)$) and $P_{z-1}(3)$ (green lines).

$w(P'_{g-1}) < w(P_{g-1})$ or $w(P''_{g-1}) < w(P_{g-1})$. If $w(P_{z-1}(3)) < w(P_{g-1}(2)) + w(P_{g-1}(3))$, then

$$\begin{aligned} w(P''_{g-1}) &= w(P_{g-1}(1) + w(P_{z-1}(3)) \\ &< w(P_{g-1}(1)) + w(P_{g-1}(2)) + w(P_{g-1}(3)) = w(P_{g-1}). \end{aligned}$$

Suppose $w(P_{z-1}(3)) \geq w(P_{g-1}(2)) + w(P_{g-1}(3))$. Note that $P_{z-1}(2)$ is the reversed path of $P_{g-1}(2)$, and $w(P_{z-1}(2)) = -w(P_{g-1}(2))$ since for each (u, v) in $P_{z-1}(2)$, $w(u, v) = -w(v, u)$, where (v, u) is in $P_{g-1}(2)$. Because $w(P_{z-1}) < 0$, $w(P_{z-1}(1)) + w(P_{z-1}(3)) < -w(P_{z-1}(2))$, so

$$w(P_{g-1}(2)) > w(P_{z-1}(1)) + w(P_{z-1}(3)).$$

From this,

$$\begin{aligned} w(P_{z-1}(3)) &\geq w(P_{g-1}(2)) + w(P_{g-1}(3)) \\ &> w(P_{z-1}(1)) + w(P_{z-1}(3)) + w(P_{g-1}(3)), \end{aligned}$$

implying $w(P'_{g-1}) = w(P_{z-1}(1)) + w(P_{g-1}(3)) < 0$. Thus, $w(P'_{g-1}) < w(P_{g-1})$. From the above and Corollary 4.2, at least one of $\hat{w}(P'_{g-1}) < \hat{w}(P_{g-1})$ and $\hat{w}(P''_{g-1}) < \hat{w}(P_{g-1})$ is true. This implies that we can obtain a min-cost flow f'_g of value g in $\hat{N}$ by augmenting flow along P'_{g-1} (or P''_{g-1}) instead of P_{g-1} such that f'_g has a cost lower than that of f_g. This is a contradiction to Property 4.2 that f_g is a min-cost flow of value g. Therefore, a flow f_z with $c(f_z) < c(f_{y+1})$ cannot exist for $z > y + 1$. □

Theorem 4.3. *Algorithm ExactNF2 finds a matching M with $w(M) \geq c$ and $|M|$ maximized or concludes that there is no matching M with $w(M) \geq c$ in H in time $O(nm +$*

$n_{min} \cdot t(FN))$, *where* $t(FN)$ *is the time for computing* $SSSP(s)$ *in a residual network of* $\hat{N}$, $n = |V(FN)|$ *and* $m = |E(FN)|$.

Proof. From Property 4.2, the flow f_y found by Step 3 at each iteration is a minimum cost flow of value y in $\hat{N}$; and by Property 4.1 (b), f_y is a min-cost flow of value y in FN. If Algorithm ExactNF2 terminates due to f_y is maximum, then by Theorem 4.2, either f_y is an optimal solution, or there is no solution for H.

Suppose Algorithm ExactNF2 terminates due to $c(f_{y+1}) > c(f_y)$ and $c(f_{y+1}) > -c$. By Lemma 4.1, any flow f_z found after f_{y+1} has $c(f_z) \geq c(f_{y+1}) > -c$. If $c(f_y) \leq -c$, then f_y implies an an optimal solution; otherwise, there is no solution for H.

Johnson's shortest path algorithm runs in $O(nm)$, due to the Bellman-Ford algorithm having a running time of $O(nm)$. There are at most n_{min} iterations after re-weighting the edges using Johnson's algorithm. There are at most n_{min} iterations. In each iteration, it takes $t(FN) \geq O(m + n \log n)$ time to compute $SSSP(s)$ and $O(m + n)$ time to update node potentials, reduced costs and residual network. Therefore, Algorithm ExactNF2 runs in time $O(nm + n_{min} \cdot t(FN))$. □

In the worst case, ExactNF2 has n_{min} iterations (after re-weighting the edges). The stopping condition ($c(f_{y+1}) > c(f_y)$ and $c(f_{y+1}) > -c$) can reduce the number of iterations in practice. Dijkstra's algorithm can be used to compute $SSSP(s)$ in each iteration since the edge cost in $\hat{N}_{f_y}(\pi)$ is non-negative. As described in [3], one can compute an $s - t$ path instead of computing $SSSP(s)$ to improve the computational time. After an $s - t$ path is found using Dijkstra's algorithm (with early termination), the update to each node potential $\pi(u)$ is based on whether vertex $\pi(u)$ has been permanently (u is visited and explored) or temporarily labeled by Dijkstra's algorithm. We implement this version in our experiment. Note, however, that Algorithm ExactNF2 still runs in time $O(nm+n_{min} \cdot t(FN))$ asymptotically. Due to the structure of the graph FN, the length of any $s - t$ path in FN is exactly 3. We can terminate the Bellman-Ford algorithm after 3 iterations (instead of $n - 1$ iterations), resulting in time $O(m)$ for Johnson's shortest path algorithm. From this and Theorem 4.3, we have the following corollary.

Corollary 4.3. *The running time of Algorithm ExactNF2 can be reduced to* $O(m + n_{min} \cdot t(FN))$.

4.3.2 Approximation algorithm

Next, we present a simple $\frac{1}{2}$-approximation algorithm (referred to as **Greedy**) for an arbitrary profit target c. This algorithm is useful, in terms of actual computational time, only if the instance H contains many negative edges.

1. Compute a maximum weight matching M' in H.

2. Let $M = M'$. For each iteration, select an edge e'' in $H^- = H \setminus H^+$ such that

$$e'' = \text{argmax}_{e \in E(H^-) \setminus M \mid e \cap e' = \emptyset \ \forall e' \in M} w(e).$$

If $w(M) + w(e'') \geq c$, then add e'' to M. Repeat this until such an edge e'' does not exist (every edge of H intersects with an edge of M) or $w(M) + w(e'') < c$.

Algorithm Greedy has a running time of $O(t(H) + m \log m)$, where $t(H)$ is the time to find a maximum weight matching in H and $m = |E(H)|$. Next, we show Algorithm Greedy has a $\frac{1}{2}$-approximation ratio. Let M' be the initial maximum weight matching computed in the first step of Greedy. Note that $M' \subseteq E(H^+)$. Let M^* be a matching in H with $w(M^*) \geq c$ and $|M^*|$ maximized.

Property 4.3. *Every edge $e \in E(H^+) \setminus M'$ is incident to at least one edge of M', implying any edge e of H not incident to M' must have weight $w(e) < 0$.*

Let $M_1 = M \setminus M'$ be the set of edges added to M' during the second step of Greedy. Let $M_1^* = \{e \in M^* \mid e \text{ is not incident to any edge of } M'\}$. From Property 4.3, every $e \in M_1 \cup M_1^*$ has weight $w(e) < 0$, namely, $(M_1 \cup M_1^*) \subseteq E(H^-)$. From Property 4.3 and each edge of M' is incident to at most two edges of $M^* \setminus M_1^*$, $|M'| \geq |M^* \setminus M_1^*|/2$ since M' is a maximal matching in H^+. Since $w(M')$ is maximum among all matchings in H, $w(M') \geq w(M^* \setminus M_1^*)$.

Theorem 4.4. *Let M be the matching found by the Greedy algorithm and M^* be a matching in H with $w(M^*) \geq c$ and $|M^*|$ maximized. Then, $\frac{|M|}{|M^*|} \geq \frac{1}{2}$, implying Greedy is $\frac{1}{2}$-approximate to RPC1.*

Proof. First, divide $F(V, E) = M_1 \Delta M_1^*$ into two collections of components: $\mathcal{F}_1^* = \{C \in \mathcal{F} \mid E(C) \subseteq M_1^*\}$ and $\mathcal{F}_0 = \mathcal{F} \setminus \mathcal{F}^*$. Notice that $|E(\mathcal{F}_1^*)| = |\mathcal{F}_1^*|$ and $(E(F) \cap M_1) \subseteq E(\mathcal{F}_0)$. We prove that $E(F) \cap M_1$ can be divided into two subsets E_0 and E_1 such that $|E_0| \geq |E(\mathcal{F}_0) \cap M_1^*|/2$ and $|E_1| \geq |E(\mathcal{F}_1^*)|$. Initially, $E_0 = \emptyset$ and $E_1 = \emptyset$. For each component $C \in \mathcal{F}_0$, let $e_1, e_2, \ldots, e_b$ be the set of edges in $E(C) \cap M_1$ selected by Greedy, where e_i is the i^{th} edge added to $E(C) \cap M_1$ and $b = |E(C) \cap M_1|$. We divide $M_1 \cap E(\mathcal{F}_0)$ into E_0 and E_1 as follows: for each $C \in \mathcal{F}_0$ and $i = 1, \ldots, b$, if there is an edge in $E(C)$ incident to e_i then add e_i to E_0 and remove every edge in $E(C)$ incident to e_i from $E(C)$ (each removed edge is in $E(C) \cap M_1^*$); otherwise, add e_i to E_1. Since each edge $e_i \in E_0$ is incident to at most two edges of M_1^*, $|E_0| \geq |E(\mathcal{F}_0) \cap M_1^*|/2$. By the Greedy algorithm, $w(E_0) \geq w(E(\mathcal{F}_0) \cap M_1^*)$. From this and $w(M') \geq w(M^* \setminus M_1^*)$, if there is any edge e^* in $E(\mathcal{F}_1^*)$ then there is a distinct edge e in E_1 with $w(e) \geq w(e^*)$, implying $|E_1| \geq |E(\mathcal{F}_1^*)|$. Therefore,

$$\frac{|M_1|}{|M_1^*|} = \frac{|E_0| + |E_1| + |M_1 \cap M^*|}{|E(\mathcal{F}_0) \cap M_1^*| + |E(\mathcal{F}_1^*)| + |M_1 \cap M^*|} \geq \frac{|E(\mathcal{F}_0) \cap M_1^*|/2 + |E(\mathcal{F}_1^*)| + |M_1 \cap M^*|}{|E(\mathcal{F}_0) \cap M_1^*| + |E(\mathcal{F}_1^*)| + |M_1 \cap M^*|} \geq \frac{1}{2}.$$

From this and $|M'| \geq |M^* \setminus M_1^*|/2$,

$$\frac{|M|}{|M^*|} = \frac{|M'| + |M_1|}{|M^* \setminus M_1^*| + |M_1^*|} \geq \frac{|M^* \setminus M_1^*|/2 + |M_1^*|/2}{|M^* \setminus M_1^*| + |M_1^*|} \geq \frac{1}{2}. \qquad \square$$

4.4 RPC+ variant

Due to the non-negative profit constraint (vi), only edges in H^+ can be selected to solve the RPC+ problem (formulation (iii)-(vi)) Inherently, the profit target must be non-negative for the RPC+ problem. In this case, a matching M with $w(M) \geq c$ and $|M|$ maximized may not be an optimal solution to the RPC+ problem if $\lambda \geq 2$. For instance, a matching $M_1 = \{e_1, e_2, e_3\}$ with three edges may contain only three passenger vertices of $V(H) \cap R$, whereas a matching $M_2 = \{e_4\}$ with one edge can contain four passenger vertices (assuming $\lambda \geq 4$). We need to find a matching M in H such that the number of passenger vertices $V(H) \cap R$ contained in M is maximized and $w(M) \geq c$.

4.4.1 The LS2 Algorithm

We propose a local search algorithm for $\lambda \geq 2$, called **LS2**. For an edge $e = \{\eta_i\} \cup R_i \in E(H)$, let $D(e) = \{\eta_i\}$ (the driver of edge e) and $R(e) = R_i$ (the passengers of e). For a subset $E' \subseteq E(H)$, let $D(E') = \cup_{e \in E'} D(e)$ and $R(E') = \cup_{e \in E'} R(e)$. For an edge $e \in E(H)$, let $N(e)$ be the set of edges incident to e, called the *neighborhood* of e, and $N^+(e) = N(e) \cap E(H^+)$. By constraint (vi), we only need to consider the subgraph H^+. As mentioned in Section 4.1, it is NP-hard to find a maximum weight matching in hypergraph H and the largest weight c^*. We use a heuristic to compute a weight $\tilde{c}$ to approximate c^* and set $c \leq \tilde{c}$ as a profit target. There are two steps in Algorithm LS2. In the first step, LS2 uses the *simple greedy* in [16, 24] to find an initial weighted set packing (hypergraph matching) to get $\tilde{c}$. In the second step of LS2, a local search is used to improve the solution computed in the first step. The first step produces a solution with a $\frac{1}{2\lambda}$-approximation ratio, and the second step gives a solution with a $\frac{2}{3\lambda}$-approximation ratio when a specific condition on the profit target is met. Algorithm LS2 is given in the following, starting with $M' = \emptyset$.

1. In each iteration, select an edge $e'' \in E(H^+)$ that does not intersect with any edge of M' and has maximum weight. That is, find an edge e'' in $E(H^+)$ such that

   $$e'' = \text{argmax}_{e \in E(H^+) \setminus M' \mid e \cap e' = \emptyset \ \forall e' \in M'} w(e),$$

   and add e'' to M'. Repeat this until every edge of $E(H^+) \setminus M'$ intersects with M'. Determine c by setting $c \leq \tilde{c} = w(M')$.

2. Let $M = M'$ be the matching obtained after Step 1. Let $A = \{e \in M \mid |R(e)| = 1\}$ and assume $A = \{a_1, \ldots, a_q\}$ with $w(a_i) \leq w(a_j)$ for $1 \leq i < j \leq q$. An *improvement* δ_e of an edge $e \in M$ is a subset of edges in $N^+(e)$ such that

- $|\delta_e| \leq 2$, all edges of $(M \cup \delta_e) \setminus \{e\}$ are pairwise vertex-disjoint, $|R(\delta_e)| > |R(e)|$ and $w(M) + w(\delta_e) - w(e) \geq c$.

An improvement δ_e is *maximum* if $|R(\delta_e) \setminus R(M)|$ is maximum among all improvements of e.

(a) If $\lambda = 2$, execute the following for-loop for each $a_i \in A$.

- For $i = 1$ to q do, if there is an improvement δ_{a_i} of a_i such that $|R(\delta_{a_i})| = 4$, then perform an *augmentation* as $M = (M \cup \delta_{a_i}) \setminus \{a_i\}$.

(b) Else if $\lambda \geq 3$, execute the following for-loop for each $a_i \in A$.

- For $i = 1$ to q do, if there is an improvement of a_i, then find a maximum improvement δ_{a_i} and perform an *augmentation* as $M = (M \cup \delta_{a_i}) \setminus \{a_i\}$.

Output M.

4.4.2 Analysis of LS2

Let M' be the solution (a matching) found after Step 1 of the LS2 algorithm. Let $0 \leq c \leq w(M')$ and M^* be a matching in H^+ such that $|R(M^*)|$ is maximized and $w(M^*) \geq c$, representing an optimal solution to the RPC+ problem. We first show that M' is already $\frac{1}{2\lambda}$-approximate for any $\lambda \geq 1$.

Property 4.4. *Every edge $e \in E(H^+) \setminus M'$ is incident to at least one edge of M'.*

For a matching M'' found during the execution of algorithm LS2 and an edge $e \in M''$, let $M^*(e) = \{e^* \in M^* \mid e^* \text{ is incident to } e\}$. For two incident edges $e \in M'$ and $e^* \in M^*$, we say they are incident/intersected *by trip* if $R(e) \cap R(e^*) \neq \emptyset$, *by driver* if $D(e) = D(e^*)$, or *by both* if $R(e) \cap R(e^*) \neq \emptyset$ and $D(e) = D(e^*)$.

Theorem 4.5. *Let M' be the matching found by Step 1 of the LS2 algorithm, $0 \leq c \leq w(M')$ and M^* be a matching in H^+ such that $|R(M^*)|$ is maximized and $w(M^*) \geq c$. Then, $\frac{|R(M')|}{|R(M^*)|} \geq \frac{1}{2\lambda}$ for $\lambda \geq 1$.*

Proof. For every $e \in M'$, e is incident to at most one edge of M^* by driver and $|R(e)|$ edges of M^* by trip. From this,

$$\frac{|R(e)|}{|R(M^*(e))|} \geq \frac{|R(e)|}{(|R(e)|+1)\lambda} \geq \frac{1}{2\lambda}.$$

From Property 4.4, every edge $e^* \in M^*$ must be incident to some edge $e \in M'$ (or $e^* \in M'$). Therefore, $\frac{|R(M')|}{|R(M^*)|} \geq \frac{1}{2\lambda}$. □

As can be seen in the analysis of Theorem 4.5, the approximation ratio is dominated by $\frac{|R(e)|}{|R(M^*(e))|} \geq \frac{|R(e)|}{(|R(e)|+1)\lambda}$ for every $e \in M'$. We can generalize it as the following corollary.

Corollary 4.4. *If $|R(e)| \geq b$ for every edge $e \in M$ and a constant $b \geq 1$, then $\frac{|R(M)|}{|R(M^*)|} \geq \frac{b}{(b+1)\lambda}$.*

If the number of edges in M' containing only one passenger is small, then a better approximation ratio can be obtained, which is the main purpose of Step 2 of LS2. Recall that $A = \{e \in M' \mid |R(e)| = 1\}$. When an edge e of A is replaced by an improvement δ_e, $|R(\delta_e)| - |R(e)| \geq 1$, increasing $|R(M')|$ by at least one. However, an improvement δ_e can decrease the total weight $w(M')$. If the profit target c is smaller than $w(M')$ by some fraction of $w(A)$ (as stated in Assumption 4.1), then we can get a $\frac{2}{3\lambda}$-approximation, and we prove this in the remainder of this section.

Assumption 4.1. The profit target c is within $0 \leq c \leq w(M' \setminus A) + 2w(A)/(\lambda + 1)$.

Let $F(V, E) = M'\Delta M^*$ be the resulting graph of the symmetric difference of M' and M^* for the rest of the analysis. Let $\mathcal{F}$ be the set of connected components in $F(V, E)$. Let $\mathcal{C}(3) = \{C \in \mathcal{F} \mid |E(C)| = 3$ and $|R(M' \cap E(C))| = 1$ and $|R(e^*)| > 1$ for each $e^* \in M^* \cap E(C)\}$. Let $Q = \cup_{C \in \mathcal{C}(3)} M' \cap E(C)$ and $Q^* = \cup_{C \in \mathcal{C}(3)} M^* \cap E(C)$. By the definition of $\mathcal{C}(3)$, every $C \in \mathcal{C}(3)$ contains exactly one edge of M' (due to $|R(M' \cap E(C))| = 1$) and two edges e_1^* and e_2^* of M^*. Then, there exists an improvement δ_e with $\delta_e = \{e_1^*\}$, $\delta_e = \{e_2^*\}$, or $\delta_e = \{e_1^*, e_2^*\}$ for every $e \in Q$. Such an improvement δ_e is independent of each $C \in \mathcal{C}(3)$. For an edge $e \in Q$ and an improvement δ_e with $|\delta_e| = 2$ w.r.t. M', $|R(\delta_e)| \geq |R(e_1^* \cup e_2^*)| - |R(e)| \geq 3$ because $|R(e_1^*)| \geq 2$ and $|R(e_2^*)| \geq 2$ by definition. Similarly for an improvement δ_e with $|\delta_e| = 1$, $|R(\delta_e)| - |R(e)| \geq 1$. Since each edge of Q is incident to two edges of Q^* and any edge e^* of Q^* is only incident to one edge of Q, $|Q^*| = 2|Q|$. Hence, we have the following property.

Property 4.5. *Let $\mathcal{C}(3)$, Q and Q^* be defined as above.*

(1) There exists an improvement δ_e w.r.t. M' for every $e \in Q$ such that edges of δ_e are not incident to edges of $\delta_{e'}$ for every pair $e, e' \in Q$.

(2) $|R(Q)| \geq \frac{|R(Q^*)|}{2\lambda}$.

Lemma 4.2. $|R(M' \setminus Q)| \geq \frac{2}{3\lambda}|R(M^* \setminus Q^*)|$.

Proof. Let $\mathcal{F}_1 = \{C \in \mathcal{F} \setminus \mathcal{C}(3) \mid |R(M' \cap E(C))| = 1\}$ and $\mathcal{F}_2 = \mathcal{F} \setminus (\mathcal{F}_1 \cup \mathcal{C}(3))$. Let $M_1' = \cup_{C \in \mathcal{F}_1} M' \cap E(C)$, $M_1^* = \cup_{C \in \mathcal{F}_1} M^* \cap E(C)$, $M_2' = \cup_{C \in \mathcal{F}_2} M' \cap E(C)$ and $M_2^* = \cup_{C \in \mathcal{F}_2} M^* \cap E(C)$. We first consider (1) $\mathcal{F}_1$, and then (2) $\mathcal{F}_2$.

(1) For each $C \in \mathcal{F}_1$, C has exactly one edge e of M' and at most two edges of M^*. If C contains two edges e_1^* and e_2^* of M^*, then one of e_1^* and e_2^* contains only one passenger by the definition of $\mathcal{C}(3)$. This implies that

$$\frac{|R(e)|}{|R(e_1^*) \cup R(e_2^*)|} \geq \frac{1}{\lambda + 1} \geq \frac{2}{3\lambda} \quad \text{for } \lambda \geq 2.$$

If C contains only one edge e^* of M^*, $|R(e)|/|R(e^*)| \geq 1/\lambda$. From these, $|R(M'_1)| \geq \frac{2}{3\lambda}|R(M^*_1)|$.

(2) Let $\mathcal{F}'_2 = \{C \in \mathcal{F}_2 \mid |M' \cap E(C)| = 1\}$. For each $C \in \mathcal{F}'_2$ and an edge $e \in M' \cap E(C)$, $|R(e)| \geq 2$ by the definition of $\mathcal{F}_1$. Then,

$$\frac{|R(e)|}{|R(M^*(e))|} \geq \frac{|R(e)|}{(|R(e)|+1)\lambda} \geq \frac{2}{3\lambda}. \tag{4.3}$$

Let $\mathcal{F}''_2 = \{C \in \mathcal{F}_2 \mid |M' \cap E(C)| \geq 2\}$. For any $C \in \mathcal{F}''_2$ and every two edges e_1 and e_2 in $M' \cap E(C)$ that are connected by an edge of $M^* \cap E(C)$, there can be at most $|R(e_1) \cup R(e_2)| + 1$ edges of M^* incident to e_1 and e_2. From this,

$$\frac{|R(e_1 \cup e_2)|}{|R(M^*(e_1) \cup M^*(e_2))|} \geq \frac{|R(e_1) \cup R(e_2)|}{(|R(e_1) \cup R(e_2)|+1)\lambda} \geq \frac{2}{3\lambda}. \tag{4.4}$$

Eq (4.3) and Eq (4.4) imply that $\frac{|R(M'_2)|}{|R(M^*_2)|} \geq \frac{2}{3\lambda}$. Since $M' \setminus Q = M'_1 \cup M'_2 \cup (M' \cap M^*)$ and $M^* \setminus Q^* = M^*_1 \cup M^*_2 \cup (M' \cap M^*)$, $|R(M' \setminus Q)| \geq \frac{2}{3\lambda}|R(M^* \setminus Q^*)|$ from the above. □

Let $B = \{a_1, a_2, \ldots, a_m\}$ be any subset of A such that $w(a_1) \leq w(a_i) \leq w(a_j) \leq w(a_m)$ for $1 \leq i < j \leq m$, $m \geq 2$ and $\lambda \geq 2$. For two integers $1 \leq x \leq y \leq m$, let $B(x, y) = \{a_x, \ldots, a_y\}$. Since $B \subseteq A$, we know that

$$w(A) - w(B) \geq z \cdot (w(A) - w(B)) = z \cdot w(A) - z \cdot w(B)$$

for any constant $0 \leq z \leq 1$. Hence, we have the following property.

Property 4.6. $\frac{2}{\lambda+1}w(B) - w(B) \geq \frac{2}{\lambda+1}w(A) - w(A)$ *for* $\lambda \geq 1$.

Lemma 4.3. *Let* $B' = B(1, \lfloor \frac{(\lambda-1)m}{\lambda+1} \rfloor)$ *for* $\lambda \geq 2$. *For matching* $M'' = (M' \setminus B') \cup E_{B'}$, *where* $E_{B'} \subseteq (E(H^+) \setminus B')$ *is a (an empty) set of edges incident to* B' *s.t.* M'' *remains as a matching,* $w(M'') \geq c$.

Proof. Each edge $e \in B$ that is replaced by the algorithm can reduce $w(M')$ by at most $w(e)$ since the improvement δ_e of e has weight $w(\delta_e) \geq 0$. Recall that the elements of B are sorted in the increasing order of their weights. For $B' = B(1, \lfloor \frac{(\lambda-1)m}{\lambda+1} \rfloor)$, B' contains at most $\frac{\lambda-1}{\lambda+1}$ smallest elements in $B \subseteq A$. Hence,

$$w(B') \leq \frac{\lambda-1}{\lambda+1}w(B) = (1 - \frac{2}{\lambda+1})w(B).$$

Let $M'' = (M' \setminus B') \cup E_{B'}$. From $w(E_{B'}) \geq 0$, Property 4.6 and Assumption 4.1,

$$\begin{aligned} w(M'') \geq w(M') - w(B') &\geq w(M') - (1 - \frac{2}{\lambda+1})w(B) \\ &= w(M' \setminus B) + \frac{2}{\lambda+1}w(B) \end{aligned}$$

$$\geq w(M' \setminus A) + \frac{2}{\lambda+1} w(A) \geq c. \qquad \square$$

During Step 2 of Algorithm LS2, an edge $e^* \in Q^*$ remains *unblocked* if e^* is not incident to any edge of improvement δ_e for any $e \in A$; and, after the augmentation of any improvement δ_e for some $e \in A$, an unblocked edge $e^* \in Q^*$ becomes *blocked* if e^* is incident to any edge of δ_e, that is, $D(e^*) \cap D(\delta_e) \neq \emptyset$ or $R(e^*) \cap R(\delta_e) \neq \emptyset$.

Lemma 4.4. *Let M be the final matching found by the LS2 algorithm. Let M^* be a matching in H^+ such that $|R(M^*)|$ is maximized and $w(M^*) \geq c$, representing an optimal solution to the RPC+ problem. For $\lambda = 2$, $\frac{|R(M)|}{|R(M^*)|} \geq \frac{2}{3\lambda}$.*

Proof. We show that at least $|Q|/3$ additional edges are added to M', that is, $|R(M)| - |R(M')| \geq |Q|/3$. For $\lambda = 2$, removing $|B(1, \lfloor \frac{(\lambda-1)m}{\lambda+1} \rfloor)| \geq |B(1, \lfloor \frac{m}{3} \rfloor)|$ edges of A from M' results in a matching M'' with $w(M'') \geq c$ by Lemma 4.3. Letting $m = |A| - \lfloor 2|A|/3 \rfloor \geq \lceil |A|/3 \rceil \geq \lceil |Q|/3 \rceil$ implies that $\lfloor \lceil |A|/3 \rceil /3 \rfloor \geq \lceil |Q|/9 \rceil$ improvements (in augmentations) can be performed on M' such that the resulting matching M'' has weight $w(M'') \geq c$. By the definition of $\mathcal{C}(3)$, each improvement δ_e of $e \in Q$ contains the two edges $e_1^*, e_2^* \in Q^*$ incident to e. Let $\delta_Q = \{\delta_e \mid e \in Q\}$ be the set of such improvements. An improvement δ_e for $e \in A \setminus Q$ adds 3 additional edges to M' and can be incident to at most 4 improvements of δ_Q since e is not incident to any edge of Q^*. An improvement δ_e for $e \in Q$ adds at least 3 additional edges to M' and can be incident to at most 5 improvements of δ_Q (four different improvements plus the one already incident to e). This implies that, in the worst case, after $\lceil |Q|/9 \rceil$ improvements with $|Q|/3$ additional edges, there are still some improvements of δ_Q not incident to these $\lceil |Q|/9 \rceil$ improvements. Hence, at least $|Q|/3$ additional edges are added to M'.

Recall from Lemma 4.2 and Property 4.5 (2) that $|R(M' \setminus Q)| \geq \frac{2}{3\lambda} |R(M^* \setminus Q^*)|$ and $|R(Q)| \geq |R(Q^*)|/2\lambda$. We have

$$\begin{aligned} |R(M)| \geq |R(M' \setminus Q)| + |R(Q)| + \frac{|Q|}{3} &= |R(M' \setminus Q)| + \frac{4|R(Q)|}{3} \\ &\geq \frac{2}{3\lambda} |R(M^* \setminus Q^*)| + \frac{2}{3\lambda} |R(Q^*)| \end{aligned} \tag{4.5}$$

From Eq (4.5),

$$\frac{|R(M)|}{|R(M^*)|} \geq \frac{\frac{2}{3\lambda}|R(M^* \setminus Q^*)| + \frac{2}{3\lambda}|R(Q^*)|}{|R(M^* \setminus Q^*)| + |R(Q^*)|} = \frac{2}{3\lambda}. \tag{4.6}$$

Therefore, the lemma holds. $\square$

Lemma 4.5 is proved in a similar way as the proof of Lemma 4.4.

Lemma 4.5. *Let M be the final matching found by the LS2 algorithm. Let M^* be a matching in H^+ such that $|R(M^*)|$ is maximized and $w(M^*) \geq c$, representing an optimal solution to the RPC+ problem. For $\lambda \geq 3$, $\frac{|R(M)|}{|R(M^*)|} \geq \frac{2}{3\lambda}$.*

Proof. We show that at least $|Q|/3$ additional edges are added to M', that is, $|R(M)| - |R(M')| \geq |Q|/3$. For $\lambda \geq 3$, removing $|B(1, \lfloor \frac{(\lambda-1)m}{\lambda+1} \rfloor)| \geq |B(1, \lfloor \frac{m}{2} \rfloor)|$ edges of A from M' results in a matching M'' with $w(M'') \geq c$ by Lemma 4.3. Letting $m = |A| - \lfloor |A|/3 \rfloor \geq \lceil 2|A|/3 \rceil \geq \lceil 2|Q|/3 \rceil$ implies that $\lfloor \lceil 2|Q|/3 \rceil /2 \rfloor \geq \lceil |Q|/3 \rceil$ improvements (in augmentations) can be performed on M' such that the resulting matching M'' has weight $w(M'') \geq c$. By Property 4.5 (1), there are $2|Q|$ improvements w.r.t. M', each of which is an edge in Q^*. An improvement δ_e for $e \in A \setminus Q$ adds $|R(\delta_e)| - 1 \geq 1$ additional edges to M' and can be incident to at most $|R(\delta_e)|$ unblocked edges of Q^* since e is not incident to any edge of Q^*; and these unblocked edges become blocked and are no longer improvements after augmenting δ_e. Similarly, an improvement δ_e for $e \in Q$ with $|\delta_e| = 2$ adds at least $|R(\delta_e)| - 1 \geq 3$, due to maximum improvement, additional edges to M' and can be incident to at most $|R(\delta_e)| + 2$ unblocked edges of Q^*. It is possible that after some augmentations, an improvement δ_e for $e \in Q$ contains only one edge due to blocked edges of Q^*; and in this case, each such improvement δ_e adds at least $|R(\delta_e)| - 1 \geq 1$ additional edges to M' and can be incident to at most $|R(\delta_e)| + 1$ unblocked edges of Q^*. These imply that, in the worst case, even after $\lceil |Q|/3 \rceil$ improvements with $|Q|/3$ additional edges, not all edges of Q^* are incident to some edges of these $\lceil |Q|/3 \rceil$ improvements, that is, there are edges of Q^* remain unblocked. Hence, at least $|Q|/3$ additional edges are added to M'. Therefore, by Eq (4.5) and Eq (4.6), the lemma holds. □

From Theorem 4.5, Lemma 4.4 and Lemma 4.5, we have Theorem 4.6.

Theorem 4.6. *Let M' be the matching found by Step 1 of the LS2 algorithm and M be the final matching found by the LS2 algorithm. Let $A = \{e \in M' \mid |R(e)| = 1\}$. Let $0 \leq c \leq w(M')$ and M^* be a matching in H^+ such that $|R(M^*)|$ is maximized and $w(M^*) \geq c$. $\frac{|R(M')|}{|R(M^*)|} \geq \frac{1}{2\lambda}$ for $\lambda \geq 1$, and if $c \leq w(M' \setminus A) + \frac{2w(A)}{\lambda+1}$ for $\lambda \geq 2$, then $\frac{|R(M)|}{|R(M^*)|} \geq \frac{2}{3\lambda}$.*

4.5 Experiment

We conduct an extensive empirical study to evaluate our model and algorithms for RPC1 and RPC+. Our goal is to have a simulation as realistic as possible. To the best of our knowledge, there is no practical test dataset publicly available for the RPC problem at the moment. To clear this hurdle, we create a simulation dataset by incorporating a real-world ridesharing dataset from Chicago City with the driver profit model of Uber. In Subsection 4.5.1, we introduce the simulation setup and describe the ridesharing dataset of Chicago City. We describe how to apply the profit/revenue model of Uber to assign a profit/revenue

value to each feasible match in Subsection 4.5.2. In Subsection 4.5.3, we describe the test instances in detail. Experimental results are reported in Subsection 4.5.4.

4.5.1 Simulation and dataset overview

The simulated centralized system receives a batch of driver offer trips D and passenger request trips R in a fixed time interval, where the origin o_i and destination d_i are within Chicago City for every driver $\eta_i \in D$ and passenger $r_i \in R$. The roadmap data of Chicago city is retrieved from OpenStreetMap (BBBike.org)[1]. We used the GraphHopper[2] library to construct the logical graph data structure of the roadmap, which contains 290048 vertices and 414124 edges.

The ridesharing dataset (denoted by *TNP*) we use is publicly available on Chicago Data Portal (CDP), maintained by Chicago Transit Authority (CTA)[3]. The ridesharing dataset TNP contains completed trips records, reported by Transportation Network Providers (which are rideshare companies) to CTA. The TNP dataset range is chosen from May 1st, 2022 to May 31st, 2022. The Chicago city is divided into 77 official community areas (**area** for brevity and labeled as A1 to A77). Each record in the TNP dataset describes a passenger trip served by a driver who provides the ride service. A trip record in the TNP dataset contains:

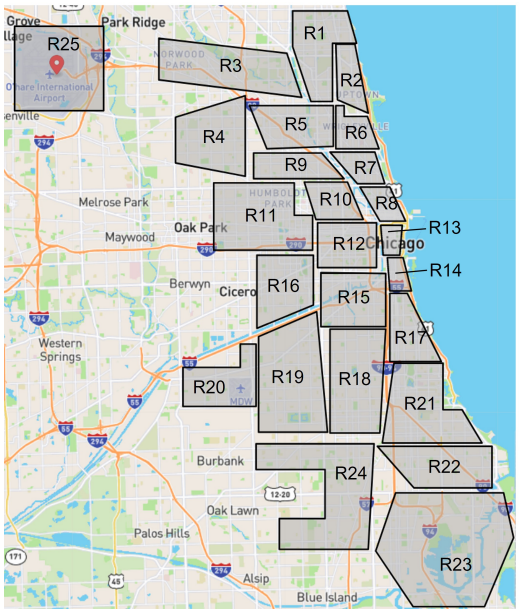
- A pick-up time, a drop-off time, a pick-up area (Census Tract) and a drop-off area (Census Tract) for the passenger.
- Duration and distance travelled of the trip.
- A fare and an optional tip paid by the passenger, where the fare does not include the tip.

Note that the exact pick-up and drop-off locations are not provided from the dataset and times are rounded to the nearest 15 minutes. We removed any trip record that is missing any of the essential information, "short" trips (less than 1.5 miles or 6 minutes) and "long" trips (greater than 35 miles or 70 minutes) from the dataset. Our experiment focuses on weekdays only. This results in 2453435 trip records from the TNP dataset. We group one or more adjacent areas together to create 25 ***regions*** to represent the Chicago City. Areas of a region are grouped together based on the total number of trips with pick-up and drop-off areas in that region. That is, areas with smaller number of trips are grouped together as shown in Figure 4.4. Since the trip records in TNP are aggregated in every 15 minutes, we partition a day from 6:00 to 23:59 into 72 time intervals (each has 15 minutes). In each time interval, drivers and passengers are generated for each of the 25 regions.

[1]Planet OSM. `https://planet.osm.org`. BBBike. `https://download.bbbike.org/osm/`

[2]GraphHopper 6.0. `https://www.graphhopper.com`

[3]CDP. `https://data.cityofchicago.org`. CTA. `https://www.transitchicago.com`



Region	Areas & Number of trips in and out of these areas
R1	[A1, A2, A4] : 149680
R2	[A3,A77] : 186192
R3	[A9, A10, A11, A12, A13, A14] : 94569
R4	[A15, A17, A18, A19] : 102257
R5	[A5, A16, A21] : 172496
R6	[A6] : 319515
R7	[A7] : 250254
R8	[A8] : 687226
R9	[A20, A22] : 183768
R10	[A24] : 270568
R11	[A23, A25, A26, A27] : 163813
R12	[A28] : 413587
R13	[A32] : 414864
R14	[A33] : 135633

Region	Areas & Number of trips in and out of these areas
R15	[A31, A34, A59, A60] : 134796
R16	[A29, A30] : 75457
R17	[A35, A36, A38, A39] : 103504
R18	[A37, A61, A67, A68] : 86558
R19	[A57, A58, A62, A63, A65, A66] : 99091
R20	[A56, A64] : 97296
R21	[A40, A41, A42, A43, A69] : 241527
R22	[A44, A45, A46, A47, A48] : 83033
R23	[A49, A50, A51, A52, A53, A54, A55] : 78509
R24	[A70, A71, A72, A73, A74, A75] : 96965
R25	[A76] : 265712

Figure 4.4: The 77 community areas are grouped into 25 regions.

4.5.2 Profit for feasible matches

First, we describe how we estimate the revenue $rev(\eta_i, R_i)$ of a feasible match (η_i, R_i) for a driver η_i using the TNP dataset. Since the trip records in TNP are reported by rideshare companies in the US, we use the price scheme from Uber, based on its upfront cost estimator[4] (Lyft has a similar scheme). Recall that $\text{SFP}(\eta_i, R_i)$ is the shortest feasible path η_i needs to traverse to serve all of R_i. Let $\zeta = |R_i|$ and $\text{SFP}(\eta_i, R_i) = (l_0, l_1, \ldots, l_{2\zeta+1})$ such that for $1 \leq a \leq 2\zeta$, l_a is a location (a vertex in the road network graph) representing either an origin o_j or a destination d_j of passenger r_j in R_i, where $l_0 = o_i$ and $l_{2\zeta+1} = d_i$. For $1 \leq a < b \leq 2\zeta$, let $p(a, b)$ be the subpath of $\text{SFP}(\eta_i, R_i)$ from l_a to l_b. For a passenger $r_j \in R_i$, let $p(j_1, j_q)$ be the subpath such that $l_{j_1} = o_j$ and $l_{j_q} = d_j$. In other words, $p(j_1, j_q)$ is the path passenger r_j needs to traverse. Denoted by $t(l_a, l_b)$ is the estimated travel time (duration) for traversing $p(a, b)$.

We define a cost function $g(\eta_i, r_j)$ representing how much r_j needs to pay. The cost of a trip for a passenger from Uber's cost estimator includes: a base fare f_1, a per-minute cost multiplier f_2, a per-mile cost multiplier f_3 and a booking/service fee f_4. Then the cost for passenger $r_j \in R_i$ is

$$g(\eta_i, r_j) = \gamma(r_j) \cdot [f_1 + f_2 \cdot t(j_1, j_q) + f_3 \cdot dist(p(j_1, j_q))] + f_4,$$

[4]Uber cost estimator. `https://www.uber.com/global/en/price-estimate`.

where $\gamma(r_j) \geq 1$ is a surge pricing factor. Surge pricing factor $\gamma(r_j)$ fluctuates based on passenger-demand and driver-supply, which depends on when r_j is picked-up, o_j and d_j. Let $f(\eta_i, r_j) = g(\eta_i, r_j) - f_4$. Uber takes all of the booking fee f_4 and takes a portion of $f(\eta_i, r_j)$, which is known as the "take-rate" $\theta(r_j, R_i)$, and it is usually $0.2 \leq \theta(r_j, R_i) \leq 0.25$. In addition, r_j has the option to include a tip $\epsilon(r_j) > 0$ for driver η_i. The estimated revenue $rev(\eta_i, R_i)$ for driver η_i and $|R_i| = \{r_j\}$ is $(1 - \theta(r_j, R_i)) \cdot \sum_{r_j \in R_i} f(\eta_i, r_j) + \epsilon(r_j)$.

If $|R_i| > 1$, the match (η_i, R_i) may become shared trips. The price scheme is similar, except a discounted rate $\omega(r_j, R_i)$ is applied to $f(\eta_i, r_j)$, where $0 \leq \omega(r_j, R_i) \leq 1$ and $\omega(r_j, R_i) = 1$ means no discount. Let $dp(r_j, R_i)$ be the number of different passengers in $R_i \setminus \{r_j\}$ encountered by r_j while traversing $p(j_1, j_q)$. We set $\omega(r_j, R_i) = \max\{1.0 - 0.2dp(r_j, R_i), 0.2\}$ (a linear relation). In addition, the cost for shared distance travelled (per-minute and per-mile) is split among the passengers in the car at the time. For $1 \leq a \leq 2\zeta - 1$, let $np(l_a, l_{a+1})$ be the number of passengers in the car while travelling from l_a to l_{a+1}. The take-rate $\theta(r_j, R_i)$ for shared trips (e.g., UberPool) can be set lower to adjust for the earnings of the drivers: take-rate $\theta(r_j, R_i)$ for a passenger $r_j \in R_i$ is selected uniformly at random from $[\max\{0.05, 0.2\omega(r_j, R_i)\}, \max\{0.1, 0.25\omega(r_j, R_i)\}]$. The final estimated revenue $rev(\eta_i, R_i)$ is

$$\sum_{r_j \in R_i} (1 - \theta(r_j, R_i)) \cdot \omega(r_j, R_i) \cdot \gamma(r_j) \cdot (f_1 + \sum_{j_1 \leq a \leq j_{q-1}} \frac{f_2 \cdot t(l_a, l_{a+1}) + f_3 \cdot dist(l_a, l_{a+1})}{np(l_a, l_{a+1})}) + \epsilon(r_j).$$

Next, we describe how we estimate the fee components, surge pricing factor, travel time, tip amount, and profit $w(\eta_i, R_i)$ in detail. From Uber's cost estimator, we can determine that f_1, f_2 and f_3 are fixed regardless of the distance of the trip. The booking fee increases as the estimated distance of the trip increases. Table 4.1 shows the cost for each fee component

base fare f_1	per-minute f_2	per-mile f_3	booking fee f_4
1.8	0.27	0.8	$\min\{\max\{1, 1 + 0.25(\text{miles} - 2)\}, 10\}$

Table 4.1: The cost (in USD) for each fee component of a trip.

for a single-passenger trip used in our experiment. The choice of the fees is validated by examining the TNP dataset as follows. Let $Z(h, x, y)$ be the set of trips with pick-up times during hour h, origins (pick-up areas) in region x, and destinations (drop-off areas) in y. For each set $Z(h, x, y)$ of trips, we calculate the average fare $avg(Z(h, x, y))$ paid by the passengers of these trips, namely, sum the fare of the trips in $Z(h, x, y)$ and divide it by $|Z(h, x, y)|$. Then, we examine the periods of time containing many trips during which surge pricing is unlikely to occur (before noon, as demonstrated by [113]). We compare the estimated average fare $f'(Z(h, x, y))$, calculated using the chosen fee component costs in Table 4.1, against the the average fare $avg(Z(h, x, y))$ for $h \in \{10, 11\}$. From the stats in Table 4.2, the cost for each fee component shown in Table 4.1 is reasonable. Note that fares are rounded to the nearest \$2.50, so some inaccuracy may occur due to this. The ratio $\gamma(h, x, y) = avg(Z(h, x, y))/f'(Z(h, x, y))$ is also the estimated average surge pricing factor

Ratio range	Percentage of $\cup_{x,y} Z(h,x,y)$ fall in the range $h=10$	$h=11$
$\frac{avg(Z(h,x,y))}{f'(Z(h,x,y))} > 1.25$	8.64%	13.48%
$1.25 \leq \frac{avg(Z(h,x,y))}{f'(Z(h,x,y))} \leq 0.75$	84.48%	82.02%
$0.9 \leq \frac{avg(Z(h,x,y))}{f'(Z(h,x,y))} \leq 1.1$	40.32%	38.52%
$\frac{avg(Z(h,x,y))}{f'(Z(h,x,y))} < 0.75$	6.88%	4.49%

Table 4.2: Percentage of $\cup_{x,y} Z(h,x,y)$ that fall in different ranges of $\frac{avg(Z(h,x,y))}{f'(Z(h,x,y))}$.

(same surge pricing factor $\gamma(r_j)$ mentioned above) for a passenger r_j with an origin in region x and a destination in region y that is picked-up during hour h. Promotion discounts are given out regularly by ridesharing companies, which can cause the fare to be lower than normal. We simulate this type of discount by applying a surge pricing factor $\gamma(h,x,y) < 1$ to any passenger trip r_j in $Z(h,x,y)$, causing $f(\eta_i, r_j)$ to be lower.

We use a similar process to estimate the average vehicle (driving) speed from one region to another region. For each set $Z(h,x,y)$ of trips, we calculate the average speed $spd(h,x,y)$ by these trips, namely, the sum of the distance of trips in $Z(h,x,y)$ is divided by the sum of the duration of the trips in $Z(h,x,y)$. In this way, the average speed $spd(h,x,y)$ gives variable vehicle speeds according to the time h of day from region x to region y. Since the TNP does not contain exact coordinates for pick-up and drop-off locations and travel time is required for profit/revenue calculations, we estimate the travel time during hour h from a location l_a in region x to location l_b in region y as $t(l_a, l_b) = dist(l_a, l_b)/spd(h,x,y)$. Note that for both estimated surge pricing factor $\gamma(h,x,y)$ and $spd(h,x,y)$, $x = y$ is also considered.

From the TNP dataset, we calculate the average amount of tips and the percentage of trips which a tip is given as follows. The distance of a trip is rounded to the nearest mile; and we denote the set of trips with distance d as Z_d. Let Z_d^+ be the set of trips

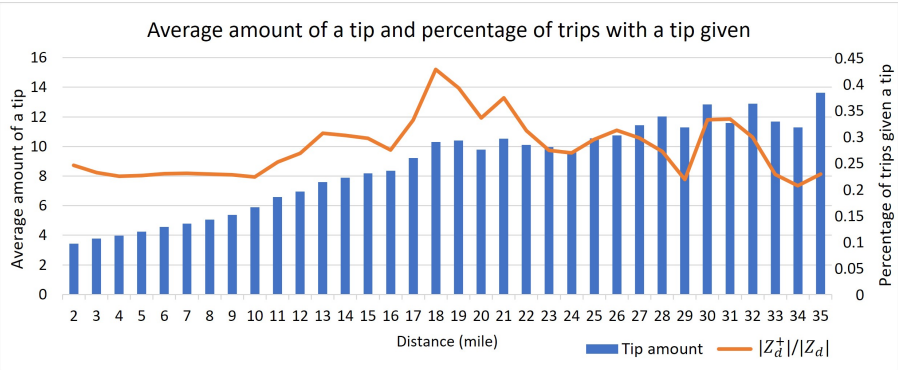


Figure 4.5: Average amount of a tip $\epsilon(r_j, d)$ and ratio $\frac{|Z_d^+|}{|Z_d|}$ for each rounded distance d.

with tips given and $\epsilon(r_j, d)$ be the average amount of a tip given by a passenger r_j for trip distance d. Figure 4.5 shows $\epsilon(r_j, d)$ and $p(d) = |Z_d^+|/|Z_d|$ for each distance d round to the nearest mile. We use this result to determine a tip given by a passenger. Given a match (η_i, R_i), SFP(η_i, R_i) and a passenger $r_j \in R_i$, let d_{r_j} be the nearest mile of the subpath (o_j, d_j) in SFP(η_i, R_i). Recall that SFP(η_i, R_i) is the computed feasible shortest path to serve all passengers of R_i (defined in Section 4.1). Then $\epsilon(r_j, d_{r_j})$ is the average amount of tip from r_j (blue bar in Figure 4.5) if r_j pays a tip, and $p(d_{r_j})$ is the probability r_j pays a tip (orange line in Figure 4.5). The average tip amount of a passenger trip $r_j \in R_i$ is $\epsilon(r_j) = p(d_{r_j}) \cdot \epsilon(r_j, d_{r_j})$.

The cost $tc(\eta_i, R_i)$ of a match (η_i, R_i) for a driver η_i is the travel cost for traversing SFP(η_i, R_i). Then, $tc(\eta_i, R_i) = dist(\text{SFP}(\eta_i, R_i)) \cdot cost(type)$, where $cost(type)$ is an estimate average cost per mile for a vehicle type under normal traffic conditions. We use the estimate costs from [11, 20] (which are based on the average gas prices for a 12-month period ending May 2022 in the US): the cost per mile for small Sedan, medium Sedan and medium SUV are \$0.1251, \$0.1437 and \$0.1889 (USD), respectively.

$\omega(r_j, R_i)$	discounted rate for the fee components
$\theta(r_j, R_i)$	take-rate for a passenger trip $r_j \in R_i$ and is selected uniformly at random from $[\max\{0.05, 0.2\omega(r_j, R_i)\}, \max\{0.1, 0.25\omega(r_j, R_i)\}]$
$\gamma(h, x, y)$	surge pricing factor for a passenger picked-up during hour h, from region x to region y
$spd(h, x, y)$	average vehicle speed during hour h from region x to region y
$\epsilon(r_j)$	the average amount of a tip given by a passenger r_j
$tc(\eta_i, R_i)$	the travel cost for traversing SFP(η_i, R_i) based on a vehicle-type mileage-cost $cost(type)$
$t(l_a, l_b)$	estimated travel time (duration) of from location l_a to location l_b in SFP(η_i, R_i)

Table 4.3: Notation used in estimating revenue $rev(\eta_i, R_i)$ and profit $w(\eta_i, R_i)$.

Table 4.3 summarizes the notation for all the estimations. Putting everything together, the estimated revenue $rev(\eta_i, R_i)$ is

$$\sum_{r_j \in R_i} (1 - \theta(r_j, R_i)) \cdot \omega(r_j, R_i) \cdot \gamma(r_j) \cdot (1.8 + \sum_{j_1 \le a \le j_{q-1}} \frac{0.27 \cdot t(l_a, l_{a+1}) + 0.8 \cdot dist(l_a, l_{a+1})}{np(l_a, l_{a+1})}) + \epsilon(r_j),$$

and the estimated profit $w(\eta_i, R_i)$ after all of R_i are served is

$$w(\eta_i, R_i) = rev(\eta_i, R_i) - tc(\eta_i, R_i) = rev(\eta_i, R_i) - dist(\text{SFP}(\eta_i, R_i)) \cdot cost(type).$$

4.5.3 Driver and passenger trips generation

For each 15-minute time interval h_t, $1 \le t \le 4$, in hour h, we first generate a set of passengers and then a set of drivers. Passengers are generated according to the average number of trips occurred per hour, calculated using the TNP dataset. Let $Z(h_t, x, y)$ be the set of trip records in the TNP dataset with pick-up time in interval h_t, origins (pick-

up areas) in region x, and destinations (drop-off areas) in region y. Let $R(h_t, x, y)$ be the set of passengers generated for interval h_t with origins in x and destinations in y. Then, $|R(h_t, x, y)| = \lceil |Z(h_t, x, y)|/days(h_t, x, y) \rceil$, where $days(h_t, x, y)$ is the number of weekdays in TNP that contain at least a trip record of $Z(h_t, x, y)$. After $R(h_t, x, y)$ is generated, a set $D(h_t, x)$ of drivers with origins in region x is generated. The destination region y of a driver $\eta_i \in D(h_t, x)$ is decided as follows. Let $sum(h_t, x) = \sum_y |R(h_t, x, y)|$ for each region x. Then, η_i has a destination in region y with probability $|R(h_t, x, y)|/sum(h_t, x)$. For any driver $\eta_i \in D(h_t, x)$ or passenger $r_i \in R(h_t, x, y)$, the actual origin o_i and destination d_i are two random locations in regions x and y, respectively. The only exceptions are regions R25 and R20 (where the O'Hare International and Midway airports are located, respectively); all drivers' and passengers' origins and destinations in R25 are the O'Hare airport, and there is 50% chance that origins and destinations in R20 are the Midway airport. The ridesharing instance in interval h_t consists of $D = \cup_x D(h_t, x)$ and $R = \cup_{x,y} R(h_t, x, y)$.

For variant **RPC1**, we set $0.9 \leq \lceil \frac{|D(h_t,x)|}{|R(h_t,x,y)|} \rceil \leq 1.1$, depending on the hour of the day. For variant **RPC+**, $|D(h_t, x)| = \lceil \frac{|R(h_t,x,y)|}{4} \rceil$ for each h_t during peak hours (7:00-9:59) and (16:00-19:59); and $\lceil \frac{|R(h_t,x,y)|}{3} \rceil \leq |D(h_t, x)| \leq \lceil \frac{|R(h_t,x,y)|}{2} \rceil$ during non-peak hours. Figure 4.6 shows the number of drivers and passenger generated for each interval. These numbers are

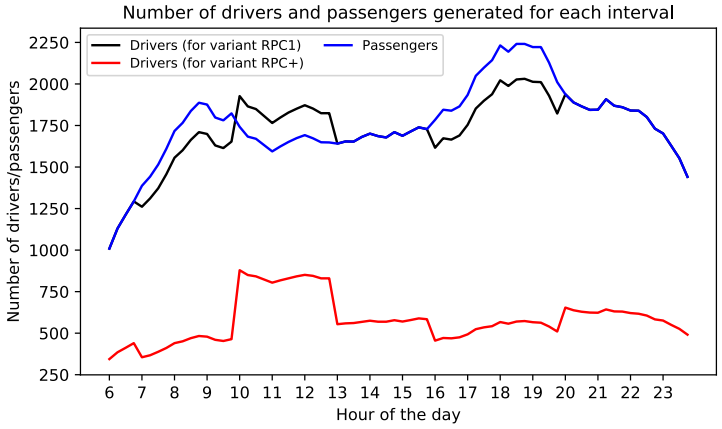


Figure 4.6: The number of drivers and passenger generated for each interval.

selected with the consideration of surge pricing factor and capacities of drivers' vehicles. Since the trip records in the TNP dataset are for served (completed) trips, the number of drivers should not be too low. We consider only Sedan vehicle types with capacity for variant RPC1. For variant RPC+, we consider three vehicle types: small and medium Sedans with capacity [1,3] and SUVs with capacity [1,5] inclusive. During peak hours, roughly 95% and 5% of vehicles have capacities randomly selected from Sedan and SUV, respectively. During non-peak hours, roughly 90% and 10% of vehicles have capacities randomly selected from Sedan and SUV, respectively. Other parameters for drivers and passengers are listed in Table 4.4.

$\mathrm{SP}(o, d)$	shortest path from location o to location d
Earliest departure time α_i	immediate to end of a time interval h_t for any driver η_i or passenger r_i in $D(h_t, x) \cup R(h_t, x, y)$
Driver detour limit z_i for $\eta_i \in D(h_t, x)$	at most $\max\{\frac{a \cdot dist(\mathrm{SP}(o_i, d_i))}{spd(h,x,y)}, 45\}$, $a \in [1.2, 1.4]$
Latest arrival time for $\eta_i \in D(h_t, x)$	$\alpha_i + a(\frac{dist(\mathrm{SP}(o_i, d_i))}{spd(h,x,y)} + z_i)$, $a \in [1.0, 1.25]$
Max travel duration for $\eta_i \in D(h_t, x)$	$\frac{dist(\mathrm{SP}(o_i, d_i))}{spd(h,x,y)} + z_i$
Latest arrival time for $r_j \in R(h_t, x, y)$	$\alpha_j + \frac{a \cdot dist(\mathrm{SP}(o_j, d_j))}{spd(h,x,y)}$, $a \in [2.0, 3.0]$
Max travel duration for $r_j \in R(h_t, x, y)$	$\frac{a \cdot dist(\mathrm{SP}(o_j, d_j))}{spd(h,x,y)}$, $a \in [1.5, 2.0]$

Table 4.4: Parameters for drivers and passengers.

Feasible matches are computed from D and R in each interval h_t. For a driver η_i, a feasible match (η_i, R_i) with $|R_i| = 1$ is called a *base match* because for any feasible match (η_i, R_i') with $|R_i'| > 1$, the match (η_i, R_i) with $R_i \subset R_i'$ must exist [101, 102] (restated as Observation 5.1). We compute base matches first and then feasible matches with $|R_i| > 1$ as in [6, 99] (the details of computing the feasible matches are described in Chapter 5, Subsection 5.2.2). Shortest paths in our simulation are computed in *real-time*. To speedup the computation of feasible matches for practical reasons, we first apply a check to see if a driver η_i and a passenger r_j is a *candidate pair*. That is, test η_i and r_j should be considered in a base match by estimating the travel distance without computing any shortest path. We also limit the number of base matches and total feasible matches a driver η_i can have.

1. Let $\overline{dist}(o, d)$ be the straight-line distance from location o to location d. Let $ed(\eta_i, r_j) = \tau \cdot (\overline{dist}(o_i, o_j) + \overline{dist}(o_j, d_j) + \overline{dist}(d_j, d_i))$ be the estimating travel distance, for some $\tau > 0$. If the maximum travel distance for η_i is at least $ed(\eta_i, r_j)$, then the match $(\eta_i, R_i = \{r_j\})$ is a candidate. Otherwise, $(\eta_i, \{r_j\})$ is unlikely a feasible match (so $(\eta_i, \{r_j\})$ is not checked).
2. Any driver η_i can have at most 50 base matches and at most 500 feasible matches in total; and each passenger can belong to at most 20 base matches (so 20 different drivers).

4.5.4 Computational results

All algorithms were implemented in Java, and the experiments were conducted on an Intel Core i7-6700 processor with 2133 MHz of 12 GBs RAM available to JVM. The ILP formulations in the algorithms are solved by CPLEX v12.10.1. We label the algorithm CPLEX uses to solve ILP formulations (iii)-(v) and (iii)-(vi) for RPC1 and RPC+ by **Exact**. Note that by default, CPLEX uses multithreading, and we leave it as it is. Recall that for RPC1, the implementation use ILPs is labeled as **ExactNF1**, the implementation based on graph algorithms is labeled as **ExactNF2** and the greedy $\frac{1}{2}$-approximation algorithm is labeled as **Greedy**. A passenger $r_j \in R$ is called *served* if $r_j \in R_i$ such that (η_i, R_i) is a feasible match belongs to a solution computed by one of the algorithms.

RPC1 results.

The base case instances use the profit calculation described in Section 4.5.2. The estimated distance factor used is $\tau = 0.6$ for the computation heuristic described in 4.5.3. The overall results are shown in Table 4.5 for profit targets $c_1 = w(M')$, $c_2 = 0.8 \cdot w(M')$ and $c_3 = 0.6 \cdot w(M')$, where M' is a maximum weight matching in H for each interval. Due to the

Algorithm	Total number of passengers served in all intervals		
	(c'_1=\$1587436)	(c'_2=\$1269949)	(c'_3=\$952462)
Greedy	109770	109775	109775
ExactNF1	109771	110035	110035
ExactNF2	109771	110035	110035
Exact	109771	110035	110035
	Total profit of served matches in all intervals		
Greedy	\$1587436	\$1587432	\$1587432
ExactNF1	\$1587436	\$1586707	\$1586707
ExactNF2	\$1587436	\$1586707	\$1586707
Exact	\$1587436	\$1465676	\$1457338
	Avg running time (sec) per interval		
Greedy	5.573	5.670	5.765
ExactNF1	6.248	6.223	6.379
ExactNF2	4.765	4.484	4.565
Exact	7.030	6.591	6.298

Avg running time to compute the matches per interval	238.713 seconds
Total number of drivers and passengers generated resp.	124340 and 126625

Table 4.5: Performances of algorithms for RPC1 on base case instances. For $1 \leq a \leq 3$, $c'_a = \sum_{h=1}^{18} \sum_{h_t=1}^{4} c_a$ (in dollar).

profit calculation, there are only 7.28 negative-profit matches per interval on average, which is about 0.019% of the average number of matches per interval (37939.99). This is by design for drivers to make money in practice, which results in the excellent performance of Greedy, as optimal solutions have very few negative-profit matches. The Greedy solutions serve about 99.76% of passengers served by optimal solutions computed by the exact algorithms. In some intervals, Greedy solutions have higher profits due to negative matches are assigned in the optimal solutions. From Table 4.5, ExactNF2 runs faster than other algorithms, and ExactNF1/2 always produce optimal solutions with the highest profits of all optimal solutions.

The base case instances may not truly reflect the whole picture when retail gas price increases and traffic congestion occurs. The gas price (regular) in Chicago was increased by nearly 35% from March 2022 to June 2022 and nearly 80% from Dec. 2021 to June 2022[5]. According to [69], fuel consumption can increase 30% under heavily congestion. Note that the cost increase from these two together are multiplicative. Although the cost due to

[5]Energy Information Administration. `https://www.eia.gov`

congestion is compensated by a higher revenue (longer travel time), not all cost caused by congestion is recovered. We considered five different settings for travel cost increases due to extra fuel cost (gas price + congestion):

- 0-20% increase (from 0% for non-peak hours to 20% for peak hours), 20-40%, 40-60%, 60-80% and 80-100%.

In addition to gas price, other major operating costs for drivers include maintenance, depreciation, insurance and tax, especially for drivers that provide frequent ridesharing service. From the findings in [11, 20], we add the following maintenance + depreciation (based on 20k miles/year) operating costs, labeled as OP, to $tc(\eta_i, R_i)$ for each match (η_i, R_i):

- \$0.0887 + \$0.1851 per mile for Small Sedan and \$0.1064 + \$0.2505 per mile for Medium Sedan.

For all other costs, an extra 20%/40% cost increase is applied to $tc(\eta_i, R_i)$ for each match (η_i, R_i). Altogether, we tested the following six cost settings (fuel and operating costs) added to the base case:

- *S1* (20-40% cost increase + operating costs OP), *S2* (40-60% + OP), *S3* (60-80% + OP), *S4* (80-100% + OP), *S5* (100-120% + OP), *S6* (120-140% + OP).

The results of these six settings are depicted in Table 4.6 for profit target $c = 0.8 \cdot w(M')$ for each interval. From Tables 4.5 and 4.6, the performance of Greedy is from about 99.76%

		Greedy	ExactNF1	ExactNF2	Exact
#S1		107233	110035	110035	110035
#S2		106953	110035	110035	110035
#S3		106658	110035	110035	110035
#S4		106387	110035	110035	110035
#S5		106081	110035	110035	110035
#S6		105795	110035	110035	110035
\$S1	c'=\$931569	\$1162693	\$1150650	\$1150650	\$1006764
\$S2	c'=\$907483	\$1132014	\$1118309	\$1118309	\$977763
\$S3	c'=\$883652	\$1101575	\$1086034	\$1086034	\$949875
\$S4	c'=\$860079	\$1071137	\$1053821	\$1053821	\$922326
\$S5	c'=\$836773	\$1041053	\$1021658	\$1021658	\$897687
\$S6	c'=\$813739	\$1010972	\$989553	\$989553	\$871236
ΘS1 (second)		3.316	3.976	4.066	4.577
ΘS2 (second)		3.246	3.921	4.658	4.564
ΘS3 (second)		3.308	3.909	3.986	4.672
ΘS4 (second)		3.301	3.962	4.526	4.696
ΘS5 (second)		3.351	3.919	3.904	4.496
ΘS6 (second)		3.343	3.989	4.600	4.720

Table 4.6: (#) Total number of passengers served and (\$) total profit of served matches in all intervals, and (Θ) average running time per interval for RPC1 using different cost settings. $c' = \sum_{h=1}^{18} \sum_{h_t=1}^{4} c$ (in dollar).

(base case) to 96.1% (the worst case S6) of the exact algorithms in the total number of passengers served. In the tested instances, Greedy has the fastest average running time in this scenario. Exact is still the slowest, and ExactNF1 is slightly faster than ExactNF2 on average. Table 4.7 shows results related to negative profits for selected cost settings: S2, S4, and S6. Solutions produced by Greedy have higher total profits than that of Ex-

		Greedy	ExactNF1	ExactNF2	Exact
Avg % of negative-profit matches served per interval out of all served matches	(S2)	0.8219%	1.2332%	1.2314%	3.5371%
	(S4)	1.2361%	1.8867%	1.8867%	4.6540%
	(S6)	1.8073%	2.8391%	2.8373%	5.9127%
Avg number of negative-profit matches per interval	(S2)	3651.903 (9.625% of total matches)			
	(S4)	5013.125 (13.213% of total matches)			
	(S6)	6517.472 (17.178% of total matches)			

Table 4.7: Results relate to profit for S2, S4, and S6.

actNF1/ExactNF2 and Exact because Greedy solutions have lower number of matches with negative profit, compared to the ExactNF1/ExactNF2/Exact solutions, as stated in Table 4.7. Of course, ExactNF always produces optimal solutions. As a result, ExactNF1 and ExactNF2 always outperform Exact. As the number of matches with negative profit increases (S1→S6), Greedy serves less passengers, and the exact algorithms stay the same. These may suggest that when there are more matches with negative profit, ExactNF1 is a better choice as it runs slightly faster than ExactNF2 on average. On the other hand, if profit is also important, using Greedy is acceptable since it produces the highest total profits and its performance is about 96.1% of the exact algorithms in the total number of passengers served.

The mean occupancy rate (in each interval) for exact algorithms stays the same for all six settings S1 to S6. The mean occupancy rate is calculated as, in each interval, (the number of served passengers + the total number of drivers) divided by the total number of drivers. The mean occupancy rates for each time interval in S2 and S6 are shown in Figure 4.7. The mean

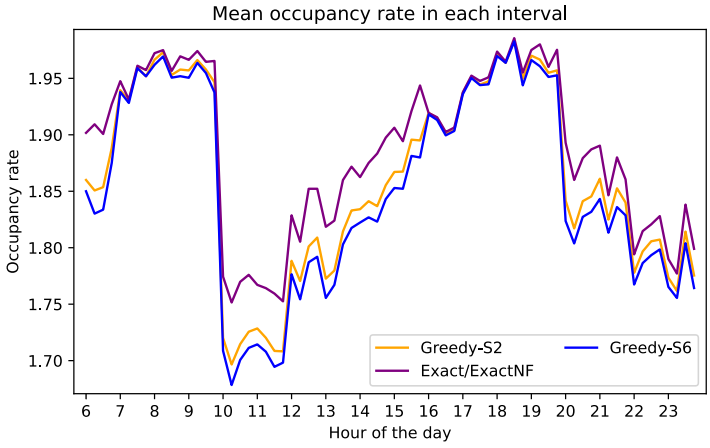


Figure 4.7: The mean occupancy rate in each interval for S2 and S6.

occupancy rates follow a similar distribution as the number of passengers generated. The average occupancy rate for exact algorithms (calculated as the sum of the mean occupancy rate for each interval and divided it by 72) is 1.8868. When there are many drivers and passengers generated (balanced supply and demand), many of them are matched together. During morning and afternoon peak/rush hours, the mean occupancy rates are close to 2, meaning most of the generated drivers are assigned a passenger. It may be beneficial to decrease the the parameter τ for candidate pair checks when supply/demand are lower in some time intervals, so more driver-passenger pairs are considered. Finally, from Table 4.5, the total maximum number of passengers served is at 110035 for c_2 and c_3 (same in Table 4.6 for $c = 0.8 \cdot w(M')$). This may indicate that $c_2 = 0.8 \cdot w(M')$ is a relative stable choice for profit target in general.

RPC+ results.

The base case instances use the profit calculation described in Section 4.5.2. The estimated distance factor used is $\tau = 0.8$ for the computation heuristic described in 4.5.3. We assume all passengers are willing to participant in ridesharing. Recall that the $\frac{2}{3\lambda}$-approximation algorithm that solves RPC+ is labeled as **LS2** and the first step of LS2 is labeled as **SimpleGreedy**. For this variant, the profit target c is upper bounded by the weight $w(M')$ of the matching M' found by SimpleGreedy. Recall that $A = \{e \in M' \mid |R(e)| = 1\}$, as defined in the description of algorithm LS2, and Assumption 4.1 requires $c \leq w(M' \setminus A) + 2w(A)/(\lambda+1)$. We set a lower bound $LB = \min\{w(M' \setminus A) + 2w(A)/(\lambda + 1), 0.6w(M')\}$. We tested three profit targets $c_1 = w(M')$, $c_2 = 0.5 \cdot (w(M') - LB) + LB$ and $c_3 = LB$. The overall results are shown in Table 4.8. The performances of SimpleGreedy and LS2 are about 89.25% and

Algorithm	Total number of passengers served in all intervals		
	(c'_1=\$845817)	(c'_2=\$676653)	(c'_3=\$507490)
SimpleGreedy	63554	63554	63554
LS2	64099	64118	64118
Exact	71197	71208	71208
	Total profit of served matches in all intervals		
SimpleGreedy	\$845817	\$845817	\$845817
LS2	\$848677	\$848271	\$848271
Exact	\$846967	\$702130	\$681472
	Avg running time (sec) per interval		
SimpleGreedy	0.0445	0.0386	0.0397
LS2	0.0761	0.0708	0.0695
Exact	47.880	33.279	35.664

Avg number of feasible matches per interval	103612.431
Avg running time to compute the matches per interval	389.176 seconds
Total number of drivers and passengers generated resp.	40573 and 126625

Table 4.8: Performances of algorithms for RPC+ on base case instances. For $1 \leq a \leq 3$, $c'_a = \sum_{h=1}^{18} \sum_{h_t=1}^{4} c_a$ (in dollar).

90.04% of the exact algorithm (Exact), in the total number of passengers served. The running time of LS2 is only 0.0325 second longer than that of SimpleGreedy. On the other hand, the running time of Exact is 470-630 times longer than that of LS2, depending on the profit target. For an even larger instance, Exact may not be suitable for real-time computation. The overall running time is practical enough as shown in Table 4.8.

In terms of occupancy rates, Exact has the best mean occupancy rate in each interval as expected, and its mean occupancy rate is also more stable compared to the other two algorithms (see an example for c_1 in Figure 4.8). In most intervals, the mean occupancy rate of Exact is 2.7 or higher, and the mean occupancy rates of LS2 and SimpleGreedy close to 2.6. The average occupancy rate for each algorithm is depicted in Table 4.9. This aligns with previous studies that there is potential in ridesharing. Our experiment further shows that there is potential in profit-maximizing MoD platforms by utilizing ridesharing as profit targets are achieved by our algorithms. It is beneficial to use different algorithms for different intervals (depending on the supply and demand) if time limit for computing a solution is important.

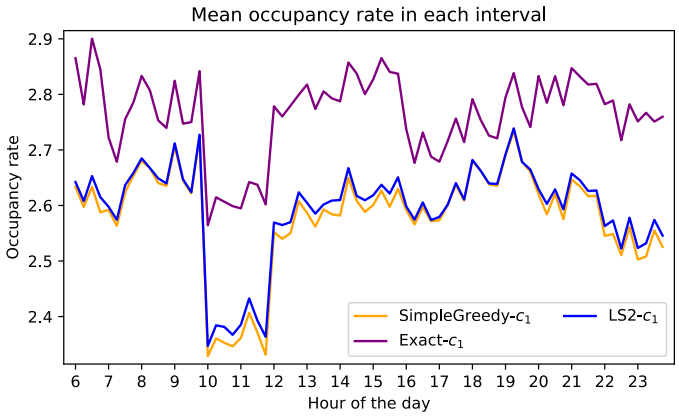


Figure 4.8: The mean occupancy rate in each interval for RPC+ and c_1.

Table 4.9: The average occupancy rate.

		SimpleGreedy	LS2	Exact
Average occupancy rate	c_1	2.5808	2.5933	2.7628
	c_2	2.5808	2.5938	2.7631
	c_3	2.5808	2.5938	2.7631

Discussion.

Based on the RPC1 and RPC+ results, we can see that our algorithms are effective for achieving the optimization goal in practical scenarios. The exact algorithms are efficient to find optimal solutions. The approximation algorithms Greedy and LS2 can achieve 96.1%

and 90.04% of the optimal solutions to RPC1 and RPC+, respectively, in the number of passengers served. The average occupancy rates for RPC1 and RPC+ improve the reported occupancy rate in the US (which is 1.5 in 2017) for all algorithms tested. Our experiment results suggest that there is potential in profit-maximizing/profit-incentive MoD platforms by utilizing ridesharing.

We also computed optimal solutions to the RP problem (formulation (i)-(ii)) using some RPC1 and RPC+ test instances. The results are shown in Table 4.10. From this, optimal

	RPC1 base	RPC1: S2	RPC1: S4	RPC1: S6	RPC+ base
*	#109770 $1587436	#106074 $1134354	#105072 $1075099	#103883 $1017174	#65913 $893879
◇	#109771 $1586707	#110035 $1118309	#110035 $1053821	#110035 $989553	#71197 $846893

Table 4.10: Total number # of passengers served and total profit $ of served matches in all intervals. (*) Optimal solutions to RP. (◇) Optimal solutions to RPC1 (for ExactNF and c_2) and RPC+ (for Exact and c_1).

solutions to RPC1 and RPC+ serve 5.92% (S6) and 8.02% (RPC+) more passengers than the respective RP optimal solutions. The profits of the optimal solutions for RPC1 and RPC+ are reasonably close to that of the RP optimal solutions. RPC can be an alternative to RP in practice since more passengers are served with a controllable profit target, which can be adjusted for each interval. MoDs can choose to compensate the drivers that serve negative-profit matches (e.g., lower the take-rate).

4.6 Summary

In this chapter, we study the RPC problem: Find a set Π of pairwise disjoint feasible matches such that $|R(\Pi)|$ is maximized and $w(\Pi) \geq c$ for some integer c. The RPC problem provides a new framework to incorporate a flexible pricing scheme to maximize the number of passengers served while meeting a profit target; and the studying of the RPC problem gives some insight into the potential of ridesharing in profit-maximizing platforms. The RPC problem is a more complex variant of the well-studied weighted set packing problem (an ILP formulation to the problem is given); and inherently, RPC is NP-hard. Two variants of RPC are studied (labeled as RPC1 and RPC+). In RPC1, the vehicle capacity of each driver is one, that is, each driver can serve at most one passenger in any solution to RPC1. We present a polynomial-time exact algorithm framework (labeled as ExactNF) and a $\frac{1}{2}$-approximation algorithm for RPC1. ExactNF is a network flow based algorithm, and two practical implementations of ExactNF are presented: one uses LP that may need to compute min-cost flows anew repeatedly, and the other uses graph algorithm that computes a min-cost flow from the min-cost flow computed in the previous iteration. In RPC+, drivers have arbitrary vehicle capacity, but only feasible matches with positive profit are considered in

any solution to RPC+. We present a fast $\frac{2}{3\lambda}$-approximation algorithm for RPC+. Lastly, we create a dataset to evaluate our model and algorithms, based on a real-life ridesharing dataset in Chicago City. The dataset provides the features and a base to simulate the real world RPC problems.

Chapter 5

Multimodal Transportation with Ridesharing Problem

Caused by poor city planning or underdevelopment, a major drawback of public transit is the inconvenience and inflexibility of first mile/last mile (**FM/LM**) transportation, compared to personal vehicles. The spareness of transit networks usually is the main cause of the inconvenience in public transit. Such transit networks have infrequent transit schedule and choiceless for customers to reach their destinations. All of the above can cause longer waiting time and more transfers for customers, and/or overcrowded transit. As a result, many people to choose to use personal vehicles for work commute. In this chapter, we investigate the potential effectiveness of integrating public transit with ridesharing to reduce travel time for commuters and increase occupancy rate in such sparse transit networks (with the focus on work commute). For example, people who drive their vehicles to work can pick-up transit users, who use public transit regularly, and drop-off them at some transit stops close to the users' destinations. In this way, passengers are presented with a cheaper alternative than ridesharing for the entire trip, and it is more convenient than using public transit only. The transit system also gets a higher ridership, which aligns with the recommendation of [34, 116] for a more sustainable transportation system. Our research focuses on an integrated transportation system (**ITS**) that is capable of matching drivers and passengers satisfying their trips' requirements while achieving an optimization goal. When a passenger (a transit user) is assigned a driver, we call this *ridesharing route*, and it is compared with the fastest *public transit route* for this passenger, which uses only public transit. If the ridesharing route is faster than the public transit route, the ridesharing route becomes an *acceptable ridesharing route* (acceptable route for short); and the acceptable route is provided to both the passenger and driver for acceptance confirmation. To increase the number of passenger participants, the optimization goal of the ITS is to maximize the number of passengers, each is assigned an acceptable route. We call this the multimodal transportation with ridesharing (**MTR**) problem.

We now formally introduce the MTR problem. In an instance of the MTR problem, we have the ITS, and for every fixed time interval, the system receives a set $\mathcal{A} = D \cup R$ of participant trips with $D \cap R = \emptyset$, where $D = \{\eta_1, \ldots, \eta_k\}$ is the set of driver trips and $R = \{r_1, \ldots, r_n\}$ is the set of passengers trips. Each trip consists of an individual (driver or passenger) and is represented by an integer i for convenience, as in the ridesharing problem. Each driver of D provides ridesharing service, and each passenger of R is a transit users and wants to receive ridesharing service to reduce his/her travel time.

A connected public transit network with a fixed timetable T is given by the ITS. The timetable T contains each transit vehicle's departure and arrival times for each transit stop/station (a transit vehicle includes bus, metro train, rail and so on). We assume that given an earliest departure time from any source location o to a destination location d in the public transit network, the fastest travel time from o to d (including transfer time) can be computed from T quickly. Given an earliest departure time dt from a passenger $r_i \in R$, a *public transit route* $\hat{\pi}_i(dt)$ for r_i is a travel plan using only public transportation, whereas a *ridesharing route* $\pi_i(dt)$ for r_i is a travel plan using a combination of public transportation and ridesharing to reach from r_i's origin to r_i's destination satisfying r_i's requirements.

Each trip $i \in \mathcal{A}$ has the same parameters related to a road network N as in the ridesharing problems as well as the RPC problem (Table 4.4). In addition to those parameters, a passenger r_i also contains an acceptance threshold θ_i for a ridesharing route $\pi_i(\alpha_i)$, that is, $\pi_i(\alpha_i)$ is given to passenger r_i if $t(\pi_i(\alpha_i)) \leq \theta_i \cdot t(\hat{\pi}_i(\alpha_i))$ for every public transit route $\hat{\pi}_i(\alpha_i)$ and $0 < \theta_i \leq 1$, where $t(\cdot)$ is the travel time of a route. Such a route $\pi_i(\alpha_i)$ is called an *acceptable ridesharing route* (*acceptable route* for brevity). For example, suppose the fastest public transit route $\hat{\pi}_i(\alpha_i)$ takes 100 minutes for r_i and $\theta_i = 0.9$. An acceptable route $\pi_i(\alpha_i)$ implies that $t(\pi_i(\alpha_i)) \leq \theta_i \cdot t(\hat{\pi}_i(\alpha_i)) = 90$ minutes.

We consider two match types for practical reasons (although our system can extend to different match types):

- **Type 1 (rideshare-transit)**: a driver may make multiple stops to pick-up different passengers, but makes only one stop to drop-off all passengers. In this case, the *pick-up locations* are the passengers' origin locations, and the *drop-off location* is a public station.
- **Type 2 (transit-rideshare)**: a driver makes only one stop to pick-up passengers and may make multiple stops to drop-off all passengers. In this case, the *pick-up location* is a public station and the *drop-off locations* are the passengers' destination locations. Passengers have to go to the pick-up location themselves.

Drivers and passengers specify which match type to participate in; they are allowed to choose both in hope to increase the chance being selected, but the system will assign them only one of the match types such that the optimization goal of the MTR problem is achieved, which is to assign acceptable routes to as many passengers as possible. Formally, the optimization

Notation	Definition
o_i	Origin (start location) of i (a vertex in N)
d_i	Destination of i (a vertex in N)
λ_i	Number of seats (capacity) of η_i available for passengers
z_i	Maximum detour time η_i willing to spend for offering ridesharing services
p_i	An optional preferred path of η_i from o_i to d_i in N
δ_i	Maximum number of stops η_i willing to make to pick-up passengers for match Type 1 or drop-off passengers for match Type 2
α_i	Earliest departure time of i
β_i	Latest arrival time of i
γ_i	Maximum trip time of i
θ_i	Acceptance threshold ($0 \leq \theta_i < 1$) for a ridesharing route $\pi_i(\alpha_i)$ for r_i
$\pi_i(\alpha_i)$	Route for r_i using a combination of public transit and ridesharing
$\hat{\pi}_i(\alpha_i)$	Route for r_i using only public transit
$d(\pi_i(\alpha_i))$	The driver who provides the ridesharing route $\pi_i(\alpha_i)$
$t(p_i)$	Shortest travel time for traversing path p_i by private vehicle
$t(\pi_i(\alpha_i))$ & $t(\hat{\pi}_i(\alpha_i))$	Shortest travel time for traversing route $\pi_i(\alpha_i)$ and $\hat{\pi}_i(\alpha_i)$ resp.
$t(u,v)$ & $\hat{t}(u,v)$	Shortest travel time from location u to v by vehicle and transit resp.

Table 5.1: Parameters for a trip announcement i.

goal of the MTR problem is to maximize the number of passengers, each is assigned an acceptable route. We denote an instance of this maximization MTR problem by $(N, \mathcal{A}, T)$.

For clarity, we summarize the parameters of trips in $\mathcal{A}$ in Table 5.1 and restate the constraints/requirements of the parameters from the definitions in Section 2.3 (which has some minor differences) in the following. For a driver $\eta_i \in D$ and a set $J \subseteq R$ of passengers, the set $\sigma(i) = \{\eta_i\} \cup J$ is called a *feasible match* if driver i can serve this group J of passengers together, using a route in N from o_i to d_i, while all requirements (constraints) specified by the parameters of the trips in $\sigma(i)$ are satisfied collectively, as listed below:

1. *Ridesharing route constraint*: for $J = \{r_{j_1}, \ldots, r_{j_p}\}$, there is a path $\text{FP}(\eta_i, J) = (o_i, o_{j_1}, \ldots, o_{j_p}, s, d_i)$ in N, where s is the drop-off location for Type 1 match; or there is a path $\text{FP}(\eta_i, J) = (o_i, s, d_{j_1}, \ldots, d_{j_p}, d_i)$ in N, where s is the pick-up location for Type 2 match. Note that if p_i is given and detour limit $z_i = 0$, path $\text{FP}(\eta_i, J) = p_i$ for either match type (assuming driver η_i specifies a station s). Otherwise, the centralized system computes the path $\text{FP}(\eta_i, J)$.
2. *Capacity constraint*: limits the number of passengers a driver can serve, $1 \leq |J| \leq \lambda_i$ with the assumption $\lambda_i \geq 1$.
3. *Acceptable constraint*: each passenger $r_j \in J$ is given an acceptable route $\pi_j(\alpha_j)$ such that $t(\pi_j(\alpha_j)) \leq \theta_j \cdot t(\hat{\pi}_j(\alpha_j))$ for $0 < \theta_j \leq 1$, where the ridesharing part of $\pi_j(\alpha_j)$ is a subpath of $\text{FP}(\eta_i, J)$ and $\hat{\pi}_j(\alpha_j)$ is the fastest public transit route for r_j given α_j.
4. *Travel time constraint*: each trip $j \in \sigma(i)$ departs from o_j no earlier than α_j, arrives at d_j no later than β_j, and the total travel duration of j is at most γ_j. The exact

application of these time constraints is described in Subsection 5.2.2. (Algorithm 7 and Algorithm 8).

5. *Stop frequency constraint*: the number of unique locations visited by driver η_i to pick-up (for Type 1) or drop-off (for Type 2) all passengers of $\sigma(i)$ is at most δ_i.

5.1 Related work

As pointed out by Ma et al. [80], some basic form of collaboration between MoD services and public transit already exists (for first and last mile transportation). For example, Thao et al. [106] mentioned in their study that a basic integration of ridesharing and public transport in rural Switzerland is already in place; and there have been pilot projects (e.g., Taxito, Ebuxi/mybuxi, Kollibri, sowiduu) promoting the integration of public transit and MoD in Switzerland. There is an increasing interest for collaboration between private companies and public sector entities [91]. These studies/reports suggest that multimodal transportation with ridesharing is practical, which further motives us.

There is a rich literature on standalone ridesharing/carpooling, from theoretical to computational studies (e.g., [1, 6, 48, 112]). Since ridesharing related work has been reviewed in Section 3.1, we refer readers to [2, 39, 81, 85, 104, 111] for more in-depth literature reviews on ridesharing.

A few studies on the integration of public transit with dynamic ridesharing have also been reported. Aissat and Varone [4] proposed an approach which, given a public transit route for a passenger, substitutes each part of the route with ridesharing if ridesharing is better than the original part. Their algorithm finds the best route for each passenger in first-come first-serve (FCFS) basis, where an optimization goal of the system is not considered, and the algorithm is computational intensive. Huang et al. [58] presented a more robust approach, compared to [4], by combining two networks N, N' (representing the public transit and ridesharing network, respectively) into one single routable graph G. The graph G uses the *time-expanded model* to maintain the information about all public-vehicle schedule, passengers' and drivers' origins, destinations and time constraints. The two networks N, N' are connected by creating edges between them whenever a passenger can be picked-up/dropped-off from/at a public stop within time constraints. For any passenger travel query, a ridesharing route is found on G, if available, by a shortest path algorithm. Their approach is also FCFS basis. Masoud et al. [82] used a similar idea of time-expanded network. Their algorithm and experiment only consider a limited number of transfer points, which may constrain their algorithm in large-scale transit systems. Due to the nature of FCFS basis in the above mentioned papers, no exact and approximation algorithms are considered to achieve an overall optimization goal.

Kumar and Khani [64] studied the FM/LM problem for transit in which individuals have no or limited transit service due to limited transit coverage and connectivity. A schedule-

based transit network graph N is used in [64], which connects the road network and the public transit service network to capture time constraints and related time sensitivity movements for passengers. Then, acceptable waiting time/walking time for feasible transfers can be computed from N. A set of feasible matches is determined by using N (each feasible match consists of a driver and a passenger that satisfies all the constraints). Then, they formulate an ILP to find an optimal match from the computed feasible matches. Their approach only consider at most one passenger per driver. Molenbruch et al. [84] studied a similar FM/LM problem where public transit is not available to passengers, so the passengers have to rely on demand-responsive services to connect to major transit stations. The difference is that [84] focuses more on the dial-a-ride problem (DARP). The authors use a variant of the metaheuristic Large Neighborhood Search to compute a list of candidate passengers, that can be served by each DAR vehicle. Then, candidates are verified based on their (users' and DAR vehicles') time constraints. Objective is to design minimum-distance DAR routes, satisfying all user requests.

Luo et al. [76] proposed a different multimodal transportation system for FM/LM problem. Their transportation system integrates micromobility services with MoD, where micromobility services include bikes and electric scooters sharing services for shorter trips. In the system, travellers use micromobility services for the FM/LM connections to hubs for ridesharing supported by MoD. As pointed out by the authors, the placement of hubs is crucial since the hub locations are fixed in their model, but it is difficult to find an optimum placement for the hubs (NP-hard). Their model also incorporates the idea of re-positioning micromobility vehicles to balanced demand/supply. The optimization goals include finding optimum placements of hubs and micromobility vehicles for given demands and supplies (repositioning the micromobility vehicles as well). Salazar et al. [94] introduced autonomous vehicle into the integration of public transit and MoD while considering the energy consumption MoD autonomous vehicles. A relevant optimization goal considered in [94] is to minimize passengers' travel time together with the operational costs of the autonomous fleet and public transportation.

Several transportation systems integrating public transit and ridesharing have been proposed to address the FM/LM problem. Ma et al. [80] presented a transportation system with policy design that integrates public transit and ridesharing using a fleet of dedicated vehicles to provide the ridesharing service. When a request from a passenger that enters the system, a *best option* (fastest travel option) is computed approximately and provided to the passenger, and the system is FCFS basis. Their model/system also includes the relocation of the ridesharing vehicles if they are idle. Narayan et al. [87] gave a similar system (without vehicle re-location). Their conditions for best routes are different from [80] and the routes are computed differently. The models in [80, 87] are different from our model in which personal drivers are the main focus.

Ma [79] and Stiglic et al. [102] proposed models to integrate public transit and ridesharing as graph matching problems to achieve certain optimization goals. In the models of [79, 102], a set of drivers, a set of passengers and a public transit network are given. The models assign passengers to drivers to use ridesharing for replacing FM/LM transit (exclusive ridesharing can be supported, as described in [102]). A group of passengers can be assigned to a driver if all constraints of the passengers in the group and the driver are satisfied. Each of passenger groups and drivers is represented as a node in a shareability graph (RV graph [95] and RTV graph [6]); and there is an edge between a passenger group node and a driver node if the group of passengers can be assigned to the driver. The passenger and driver assignment problem is modelled as graph matching problem in the graph (then formulated as an ILP problem and solved by an ILP solver).

The optimization goal in [79] is to minimize the cost related to waiting time and travel time. Two optimization goals are considered in [102]: one is to maximize the number of passengers assigned to drivers, and the other is to minimize the total distance increase for all drivers. These models are closely related to ours, but there are differences and limitations. The goal in [79] is different from ours and does not guarantee ridesharing routes better than the public transit routes. One of the goals in [102] aligns with ours, but there are restrictions: a passenger group can have at most two passengers, each passenger must use the transit stop closest to the passenger's destination, and more importantly, the ridesharing routes are not guaranteed to be better than public transit routes. We extend the work in [102] to eliminate the limitations described above. That is, a ridesharing match allows more than two passengers, passengers can be picked-up/dropped-off at any feasible transit stop, and ridesharing routes assigned to passengers is quicker than the fastest public transit routes.

It is worth to mention that the ILP formulation (described in Section 5.2) for the MTR problem is a special case of the Separable Assignment Problem (SAP), which is a generalization of the Generalized Assignment Problem (GAP). Given a β-approximation algorithm for the single-bin subproblem in SAP, Fleischer et al. [37] presented two approximation algorithms for SAP: an LP-rounding based $((1-\frac{1}{e})\beta)$-approximation algorithm and a local-search $(\frac{\beta}{\beta+1}-\epsilon)$-approximation algorithm, $\epsilon > 0$. If interested, a problem related to SAP is the Multiple Knapsack Problem with Assignment Restrictions (e.g., [30]).

5.2 Exact algorithm approach

Our exact algorithm approach for the MTR problem is presented in this section, which is similar to the matching approach described in [6, 95] for ridesharing and in [79, 102] for integration of public transit and ridesharing. Our approach computes a hypergraph representing all feasible matches in an instance $(N, \mathcal{A}, T)$ of the MTR problem, which is similar to [77, 95]. The hypergraph allows a better intuition of the approximation algorithms described in Section 5.3. We give a detailed description of our hypergraph approach as some

of the definitions/notations are used in our approximation algorithms. More importantly, we show that the MTR problem is in fact NP-hard using our ILP formulation from the hypergraph representing.

5.2.1 Integer program formulation

The exact algorithm approach is described in the following. Given an instance $(N, \mathcal{A}, T)$ of the MTR problem, we first compute all feasible matches for each driver $\eta_i \in D$. Then, we create a bipartite (hyper)graph $H(V, E)$, where $V(H) = D \cup R$. For each driver $\eta_i \in D$ and a non-empty subset $J \subseteq R$, if $\{\eta_i\} \cup J$ is a feasible match, create a hyperedge $e = \{\eta_i\} \cup J$ in $E(H)$. Any driver $\eta_i \in D$ or passenger $r_j \in R$ does not belong to any feasible match is removed from $V(H)$, that is, H contains no isolated vertex (such passengers must use public transit). For each edge $e = \{\eta_i\} \cup J$ in $E(H)$, assign a weight $w(e) = |J|$ (representing the number of passengers in e). Let $D(H) = D \cap V(H)$ and $R(H) = R \cap V(H)$. For an edge $e = \{\eta_i\} \cup J$ in $E(H)$, let $D(e) = \eta_i$ (the driver of e) and $R(e) = J$ (the passengers of e). For a subset $E' \subseteq E(H)$, let $D(E') = \cup_{e \in E'} D(e)$ and $R(E') = \cup_{e \in E'} R(e)$. For a vertex $j \in V(H)$, define $E_j = \{e \in E(H) \mid j \in R(e)\}$ to be the set of edges in $E(H)$ incident to j. An example of the hypergraph $H(V, E)$ is given in Figure 5.1. To solve the MTR problem,

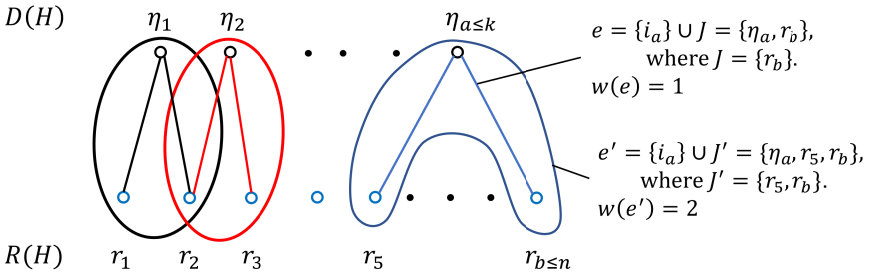


Figure 5.1: A bipartite hypergraph $H(V, E)$ representing all feasible matches of an instance $(N, \mathcal{A}, T)$, where $|D(H)| = a$ and $|R(H)| = b$.

we give an integer linear programming (ILP) formulation:

$$\text{maximize} \quad \sum_{e \in E(H)} w(e) \cdot x_e \tag{5.1}$$

$$\text{subject to} \quad \sum_{e \in E_j} x_e \leq 1, \qquad \forall\, j \in \mathcal{A} \tag{5.2}$$

$$x_e \in \{0, 1\}, \qquad \forall\, e \in E(H) \tag{5.3}$$

The binary variable x_e indicates whether the edge $e = (\eta_i, J)$ is in the solution ($x_e = 1$) or not ($x_e = 0$). If $x_e = 1$, it means that all passengers of J are assigned to η_i and can be delivered by η_i satisfying all constraints. Inequality (5.2) in the ILP formulation guarantees that each driver serves at most one feasible set of passengers and each passenger is served

by at most one driver. Note that the ILP (5.1)-(5.3) is similar to a set packing formulation. An advantage of this ILP formulation is that the number of constraints is substantially decreased, compared to traditional ridesharing formulation.

Observation 5.1. *A match $\sigma(i)$ for any driver $\eta_i \in D$ is feasible if and only if for every subset $J \subseteq (\sigma(i) \setminus \{i\})$, $\{\eta_i\} \cup J$ is a feasible match [101].*

From Observation 5.1, it is not difficult to see that Proposition 5.1 holds.

Proposition 5.1. *Let $D' \subseteq D$ and $P(D')$ be a maximal set of passengers served by D'. There always exists a set of feasible matches for D' such that $\sigma(i) \cap \sigma(i') = \emptyset$ for every $\eta_i, \eta_{i'} \in D'$ and $\bigcup_{\eta_i \in D'} \sigma(i) \setminus \{\eta_i\} = P(D')$.*

Theorem 5.1. *Given a hypergraph $H(V, E)$ for an instance of the multimodal transportation with ridesharing (MTR) problem, an optimal solution to the ILP (5.1)-(5.3) is an optimal solution to the MTR problem and vice versa.*

Proof. From inequality (5.2) in the ILP, the solution found by the ILP is always feasible to the MTR problem. By Proposition 5.1 and objective function (5.1), an optimal solution to the ILP (5.1)-(5.3) is an optimal solution to the MTR problem. Obviously, an optimal solution to the MTR problem is an optimal solution to the ILP (5.1)-(5.3). □

5.2.2 Computing feasible matches

Recall that λ_i is the capacity of a driver η_i (the maximum number of passengers i can serve at once). The maximum number of feasible matches for η_i is $\sum_{p=1}^{\lambda_i} \binom{|R|}{p}$. Assuming the capacity λ_i is a very small constant (which is reasonable in practice), the above summation is polynomial in R, that is, $O((|R|+1)^{\lambda_i})$ (partial sums of binomial coefficients). Let $\lambda = \max_{i \in D} \lambda_i$ be the maximum capacity among all vehicles/drivers. Then, in the worst case, $|E(H)| = O(|D| \cdot (|R|+1)^{\lambda})$.

In the next two subsections, we describe how to compute all feasible matches between drivers and passengers in $\mathcal{A} = D \cup R$, given an instance $(N, \mathcal{A}, T)$ of the MTR problem. Let $\sigma(i) = \{\eta_i\} \cup J$ be a feasible match for driver $\eta_i \in D$ and subset $J \subseteq R$. Let $\mathrm{FP}(\eta_i, J)$ be the actual path in N (*route* for short) that satisfies the feasibility of the match $\sigma(i)$, namely, $\mathrm{FP}(\eta_i, J)$ is the route driver η_i uses to deliver all passengers of J that starts at o_i and ends at d_i. During the computation of $\sigma(i) = \{\eta_i\} \cup J$, the route $\mathrm{FP}(\eta_i, J)$ of shortest travel time is computed, assuming a shortest path from one location to another can be computed from roadmap network N. Because the number of locations η_i needs to visit for a feasible match $\sigma(i)$ is limited, enumerating all possible locations to compute $\mathrm{FP}(\eta_i, J)$ is still quick (detailed description is given in Algorithm 8). The general procedure to compute all feasible matches is similar to [102] with some minor differences to further extend and overcome the limitations of [102], as mentioned in the related work (Section 5.1). Further,

some definitions are required to show that the route r_i computed for driver i indeed has the shortest travel time. Hence, we give a full description for computing all feasible matches.

Computing all feasible matches between D and R is done in two phases. In phase one, for each driver η_i, we find all feasible matches $\sigma(i) = \{\eta_i, r_j\}$ consisting of only one passenger r_j. Such a feasible match is called the *base match*. In phase two, for each driver i, we compute all feasible matches $\sigma(i) = \{\eta_i, r_{j_1}, \ldots, r_{j_p}\}$ with p passengers, based on the previously computed feasible matches $\sigma(i)$ with $p-1$ passengers, for $p = 2$ upto the number of passengers η_i can serve. Recall that, from Table 5.1, each trip $i \in \mathcal{A}$ is specified by the parameters $(o_i, d_i, \lambda_i, z_i, p_i, \delta_i, \alpha_i, \beta_i, \gamma_i, \theta_i)$. The maximum trip time γ_i of a driver $\eta_i \in D$ can be calculated as $\gamma_i = \min\{\gamma_i, t(p_i) + z_i\}$ if p_i is given, where $t(p_i)$ is the shortest travel time on path p_i; otherwise $\gamma_i = \min\{\gamma_i, t(o_i, d_i) + z_i\}$, where $t(o_i, d_i)$ is the shortest travel time of a path from o_i to d_i. For a passenger r_j, γ_j is more flexible; it is default to be $\gamma_j = t(\hat{\pi}_j(\alpha_j))$ in our experiment, where $\hat{\pi}_j(\alpha_j)$ is the fastest public transit route. We make two simplifications in our algorithms:

- Given an origin o_j and a destination d_j of a passenger r_j with earliest departure time α_j at o_j, we use a simplified transit system in our experiments to calculate the fastest public transit route $\hat{\pi}_j(\alpha_j)$ from o_j to d_j.
- We use a simplified model for the transit travel time, transit waiting time and ridesharing service time (time it takes to pick-up and drop-off passengers, walking time between locations and stations). Given the fastest travel time $t(u, v)$ by car from location u to location v, we multiply a small constant $\epsilon > 1$ with $t(u, v)$ to simulate the transit time and ridesharing service time. In this model, the transit time and ridesharing service time are considered together, as a whole.

Phase one (Algorithm 7).

We now describe how to compute a feasible match between a driver and a passenger for Type 1. The computation for Type 2 is similar and we omit it. For every trip $i \in D \cup R$, we first compute the set $S_{do}(i)$ of feasible drop-off locations for trip i. Each element in $S_{do}(i)$ is a *station-time* tuple $(s, \alpha_i(s))$ of i, where $\alpha_i(s)$ is the earliest possible time i can reach station s. The station-time tuples are computed by the following preprocessing procedure.

- We find all feasible station-time tuples for each passenger $r_j \in R$. A station s is *time feasible* for r_j if r_j can reach d_j from s within time window $[\alpha_j, \beta_j]$ and $t(o_j, s) + \hat{t}(s, d_j) \leq \min\{\gamma_j, \theta_j \cdot \hat{t}(o_j, d_j)\}$.
  - The earliest possible time to reach station s for r_j can be computed as $\alpha_j(s) = \alpha_j + t(o_j, s)$ without pick-up time and drop-off time. Since we do not consider waiting time and ridesharing service time separately, $\alpha_j(s)$ also denotes the earliest time for j to depart from station s.

– Let $\hat{t}(s, d_j)$ be the travel time of a fastest public route. Station s is *time feasible* if $\alpha_j(s) + \hat{t}(s, d_j) \leq \beta_j$ and $t(o_j, s) + \hat{t}(s, d_j) \leq \min\{\gamma_j, \theta_j \cdot \hat{t}(o_j, d_j)\}$.

- The feasible station-time tuples for each driver $\eta_i \in D$ is computed by a similar calculation.

  – Without considering pick-up time and drop-off time separately, the earliest arrival time of η_i to reach s is $\alpha_i(s) = \alpha_i + t(o_i, s)$. Station s is *time feasible* if $\alpha_i(s) + t(s, d_i) \leq \beta_i$ and $t(o_i, s) + t(s, d_i) \leq \gamma_i$.

After the preprocessing, Algorithm 7 finds all base matches. For each pair (η_i, r_j) in $D \times R$, let $\tau_i(o_j) = \max\{\alpha_i, \alpha_j - t(o_i, o_j)\}$ be the latest departure time of driver η_i from o_i such that η_i can still pick-up r_j at the earliest; this minimizes the time (duration) needed for driver η_i to wait for passenger r_j. Hence, the total travel time of η_i is minimized when i uses a path $\mathrm{FP}(\eta_i, J)$ with shortest travel time and departure time $\eta_i(o_j)$. The process of checking if the match $\sigma(i) = \{\eta_i, r_j\}$ is feasible for all pairs of (η_i, r_j) can be performed as in Algorithm 7.

Algorithm 7 (Phase one) Compute base matches

```
1: for each pair (η_i, r_j) in D × R do
2:     for each station s in S_do(i) ∩ S_do(j) do
3:         t_1 = t(o_i, o_j) + t(o_j, s); t_2 = t(o_j, s);    // travel duration for η_i and r_j to reach s resp.
4:         t = τ_i(o_j) + t_1;                 // earliest departure time from station s for everyone.
5:         if (t + t(s, d_i) ≤ β_i ∧ t_1 + t(s, d_i) ≤ γ_i) and (t + t̂(s, d_j) ≤ β_j ∧ t_2 + t̂(s, d_j) ≤ min{γ_j, θ_j ·
           t̂(o_j, d_j)}) then
6:             create an edge (η_i, J = {r_j}) in E(H) to represent σ(i) = {η_i, r_j}.
7:             break inner for-loop;    // can be allowed to run to completion for a better route
8:         end if
9:     end for
10: end for
```

Phase two (Algorithm 8).

We extend Algorithm 7 to create matches with more than one passenger. Let $H(V, E)$ be the graph after computing all feasible base matches (instance computed by Algorithm 7). We start with computing, for each driver η_i, feasible matches consisting of two passengers, then three passengers, and so on until $\min\{\delta_i, \lambda_i\}$. Let $\Omega(i)$ be the set of feasible matches found so far for driver η_i and $\Omega(i, p-1) = \{\sigma(i) \in \Omega(i) \mid |\sigma(i) \setminus \{\eta_i\}| = p - 1\}$ be the set of matches with $p-1$ passengers, and we try to extend $\Omega(i, p-1)$ to $\Omega(i, p)$ for $2 \leq p \leq \min\{\delta_i, \lambda_i\}$. Let $\sigma(i) = \{\eta_i\} \cup J$ be a match in $\Omega(i, p)$, where $J = \{r_{j_1}, \ldots, r_{j_p}\}$. Let $\mathrm{FP}(\eta_i, J) = (l_0, l_1, \ldots, l_p, s, d_i)$ denotes an ordered potential route for driver η_i to pick-up all p passengers of J and drop-off them at station s, where l_0 is the origin of η_i and l_y is the pick-up location (origin of passenger r_{j_y}), $1 \leq y \leq p$. We extend the notion of $\tau_i(o_j)$, defined above in Phase one, to every pick-up location of $\mathrm{FP}(\eta_i, J)$. That is, $\tau_i(l_p)$ is the

latest time of η_i to depart from o_i to pick-up each of the passengers $r_{j_1}, \ldots, r_{j_p}$ such that the waiting time of η_i is minimized, and hence, travel time of η_i is minimized. We simply call $\tau_i(l_p)$ the latest departure of η_i to pick-up $\sigma(i)$. All possible combinations of $\mathrm{FP}(\eta_i, J)$ are enumerated to find a feasible path $\mathrm{FP}(\eta_i, J)$; the process of finding $\mathrm{FP}(\eta_i, J)$ is described in the following.

- First, we fix a combination of $\mathrm{FP}(\eta_i, J)$ such that $|\sigma(i)| \leq \lambda_i + 1$ and $\mathrm{FP}(\eta_i, J)$ satisfies the stop constraint. The visiting order of the pick-up origin locations is known when we fix a route for $\mathrm{FP}(\eta_i, J)$.
- Then, the algorithm determines the actual drop-off station s, by checking each time feasible station, in $\mathrm{FP}(\eta_i, J) = (l_0, l_1, \ldots, l_p, s, d_i)$. Let r_{j_y} be the passenger corresponds to pick-up location l_y for $1 \leq y \leq p$ and $l_0 = o_i$. For each station s in $\bigcap_{0 \leq y \leq p} S_{do}(r_{j_y})$, the algorithm checks if $\mathrm{FP}(\eta_i, J) = (l_0, l_1, \ldots, l_p, s, d_i)$ admits a time feasible path for all trips in $\sigma(i)$ as follows.
  - The total travel time (duration) for η_i from l_0 to s is $t_i = t(l_0, l_1) + \cdots + t(l_{p-1}, l_p) + t(l_p, s)$. The total travel time (duration) for r_{j_y} from l_y to s is $t_{j_y} = t(l_y, l_{y+1}) + \cdots + t(l_{p-1}, l_p) + t(l_p, s)$, $1 \leq y \leq p$.
  - Since the order for η_i to pick up r_{j_y} $(1 \leq y \leq p)$ is fixed, $\tau_i(l_p)$ can be calculated as $\tau_i(l_p) = \max\{\alpha_i, \alpha_{j_1} - t(l_0, l_1), \alpha_{j_2} - t(l_0, l_1) - t(l_1, l_2), \ldots, \alpha_{j_p} - t(l_0, l_1) - \cdots - t(l_{p-1}, l_p)\}$. The earliest arrival time at s for all trips in $\sigma(i)$ is $t = \tau_i(l_p) + t_i$.
  - If $t + t(s, d_i) \leq \beta_i$, $t_i + t(s, d_i) \leq \gamma_i$, and for $1 \leq y \leq p$, $t + \hat{t}(s, d_{j_y}) \leq \beta_{j_y}$ and $t_{j_y} + \hat{t}(s, d_{j_y}) \leq \theta_{j_y} \cdot \hat{t}(o_{j_y}, d_{j_y})$, then $\mathrm{FP}(\eta_i, J)$ is feasible.
- If $\mathrm{FP}(\eta_i, J)$ is feasible, add to H the match (η_i, J). Otherwise, check next combination of $\mathrm{FP}(\eta_i, J)$ until a feasible path is found or all combinations are exhausted.

The pseudo code for the above process is given in Algorithm 8. We show that the latest departure $\tau_i(l_p)$ used in Algorithm 8 indeed minimizes the total travel time of η_i to reach l_p.

Theorem 5.2. *Given a feasible path* $\mathrm{FP}(\eta_i, J) = (l_0, \ldots, l_p, s, d_i)$ *for driver* η_i *to serve* p *passengers in a match* $\sigma(i)$*. The latest departure time* $\tau_i(l_p)$ *calculated above minimizes the total travel time of* η_i *to reach* l_p*.*

Proof. Prove by induction. For the base case $\tau_i(l_1) = \max\{\alpha_i, \alpha_{j_1} - t(l_0, l_1)\}$, and by choosing departure time $\tau_i(l_1)$, driver η_i does not need to wait for passenger r_{j_1} at α_{j_1}. Hence, using a shortest (time) path from l_0 to l_1 with departure time $\tau_i(l_1)$ minimizes the travel time of η_i to pick-up r_{j_1}. Assume the lemma holds for $1 \leq y - 1 < p$, that is, $\tau_i(l_{y-1})$ minimizes the total travel time of η_i to reach l_{y-1}. We prove for y. From the calculation of $\tau_i(l_{y-1})$, $\tau_i(l_y) = \max\{\eta_i(l_{y-1}), \alpha_{j_y} - t(l_0, l_1) - t(l_1, l_2) - \cdots - t(l_{y-1}, l_y)\}$. By the induction hypothesis, $\tau_i(l_y)$ minimizes the total travel time of η_i when using a shortest path $(l_0, \ldots, l_y)$. □

Algorithm 8 (Phase two) compute all feasible matches

for $i = 1$ to $|D|$ **do**
  $p = 2$;
  while ($p \leq \min\{\delta_i, \lambda_i\}$ and $\Omega(i, p-1) \neq \emptyset$) **do**
    for each match $\sigma(i)$ in $\Omega(i, p-1)$ **do**
      for each $r_j \in R$ s.t. $r_j \notin \sigma(i)$ **do**
        if $\sigma(i) \cup \{r_j\}$ does not satisfy Observation 5.1, **then continue**;
(skip r_j)
        if $((\sigma(i) \setminus \{q\}) \cup \{r_j\}) \in \Omega(i, p-1)$ for all $q \in \sigma(i) \setminus \{\eta_i\}$ **then**
          if ($\sigma(i) \cup \{r_j\}$ has not been checked) and (feasibleInsert($\sigma(i), r_j$)) **then**
            create an edge $\{\eta_i\} \cup J$ in $E(H)$, where $J = \sigma(i) \setminus \{\eta_i\}$;
            add $\sigma(i) \cup \{\eta_j\}$ to $\Omega(i, p)$;
          end if
        end if
      end for
    end for
    $p = p + 1$;
  end while
end for
Procedure feasibleInsert($\sigma(i), r_j$) // find a feasible path for η_i to serve $\sigma(i) \cup \{r_j\}$ if exists
Let $\mathrm{FP}(\eta_i, J) = (l_0, l_1, \ldots, l_p, s, d_i)$ denotes a potential path for driver η_i to serve all passengers in $J = \{r_{j_1}, \ldots, r_{j_p}\}$;
for each station s in $\bigcap_{0 \leq y \leq p} S_{do}(j_y)$ **do**
  for each combination of $\mathrm{FP}(\eta_i, J)$ that satisfies the stop constraint **do**
    $t_i = t(l_0, l_1) + \cdots + t(l_{p-1}, l_p) + t(l_p, s)$; $t_{j_y} = t(l_y, l_{y+1}) + \cdots + t(l_{p-1}, l_p) + t(l_p, s)$;
    $t = \tau_i(l_p) + t_i$; // the earliest arrival time at s for all trips in $\sigma(i)$
    if ($t + t(s, d_i) \leq \beta_i \wedge t_i + t(s, d_i) \leq \gamma_i$) and (for $1 \leq y \leq p$, $t + \hat{t}(s, d_{j_y}) \leq \beta_{j_y} \wedge t_{j_y} + \hat{t}(s, d_{j_y}) \leq \min\{\gamma_j, \theta_j \cdot \hat{t}(o_{j_y}, d_{j_y})\}$) **then**
      return True;
    end if
  end for
end for
return False;

The running time of Algorithm 8 heavily depends on the number of subsets of passengers to be checked for feasibility. One way to speed up Algorithm 8 is to use dynamic programming (or memoization) to avoid redundant checks on a same subset. For each feasible match $\sigma(i) = \{\eta_i\} \cup J$ of $p-1$ passengers for a driver $\eta_i \in D$, we store every feasible path $\mathrm{FP}(\eta_i, J) = (l_0, l_1, \ldots, l_{p-1}, s, d_i)$ and extend from $\mathrm{FP}(\eta_i, J)$ to insert a new trip to minimize the number of ordered potential paths we need to test. We can further make sure that no path $\mathrm{FP}(\eta_i, J)$ is tested twice during execution. First, the set R of passengers is given a fixed ordering (based on the integer labels). For a feasible path $\mathrm{FP}(\eta_i, J)$ of a driver η_i, the check of inserting a new passenger r_j into $\mathrm{FP}(\eta_i, J)$ is performed only if r_j has a label larger than every passenger in $\mathrm{FP}(\eta_i, J)$ according to the fixed ordering. Furthermore, A heuristic approach to speed up Algorithm 8 is given at the end of Subsection 5.4.2.

5.3 Approximation Algorithms

We show that the MTR problem is NP-hard and give approximation algorithms for the problem. When every edge in $H(V, E)$ consists of only two vertices (one driver and one passenger), the ILP (5.1)-(5.3) formulation is equivalent to the maximum weight matching problem, which can be solved in polynomial time. However, if the edges contain more than two vertices, they become hyperedges. In this case, the ILP (5.1)-(5.3) becomes a formulation of the maximum weighted set packing problem (MWSP), which is NP-hard in general [40, 62]. In fact, the ILP (5.1)-(5.3) formulation gives a special case of MWSP (due to the structure of $H(V, E)$). We first show that this special case is also NP-hard, and by Theorem 5.1, the MTR problem is NP-hard.

5.3.1 NP-hardness

It was mentioned in [95] that their minimization problem related to shareability hypernetwork is NP-complete, which is similar to the MTR problem formulation. However, an actual reduction proof was not described. We prove the MTR problem is NP-hard by a reduction from a special case of the maximum 3-dimensional matching problem (3DM). An instance of 3DM consists of three disjoint finite sets A, B and C, and a collection $\mathcal{F} \subseteq A \times B \times C$. That is, $\mathcal{F}$ is a collection of triplets (a, b, c), where $a \in A, b \in B$ and $c \in C$. A 3-dimensional matching is a subset $\mathcal{M} \subseteq \mathcal{F}$ such that all sets in $\mathcal{M}$ are pairwise disjoint. The decision problem of 3DM is that given $(A, B, C, \mathcal{F})$ and an integer q, decide whether there exists a matching $\mathcal{M} \subseteq \mathcal{F}$ with $|\mathcal{M}| \geq q$. We consider a special case of 3DM: $|A| = |B| = |C| = q$; it is still NP complete [40, 62]. Given an instance $(A, B, C, \mathcal{F})$ of 3DM with $|A| = |B| = |C| = q$, we construct an instance $H(V, E)$ (bipartite hypergraph) of the MTR problem as follows:

- Create a set of drivers $D(H) = A$ with capacity $\lambda_i = 2$ for every driver $\eta_i \in D(H)$ and a set of passengers $R(H) = B \cup C$.
- For each $f \in \mathcal{F}$, create a hyperedge $e(f)$ in $E(H)$ containing elements $\{a, b, c\}$, where a represents a driver and b, c represent two different passengers. Further, create edges $e'(f) = \{a, b\}$ and $e''(f) = \{a, c\}$ so that Observation 5.1 is satisfied.

Theorem 5.3. *The MTR problem is NP-hard.*

Proof. By Theorem 5.1, we only need to prove the ILP (5.1)-(5.3) is NP-hard, which is done by showing that an instance $(A, B, C, \mathcal{F})$ of the maximum 3-dimensional matching problem has a solution $\mathcal{M}$ of cardinality q if and only if the objective function value of ILP (5.1)-(5.3) is $2q$.

Assume that $(A, B, C, \mathcal{F})$ has a solution $\mathcal{M} = \{f_1, f_2, \ldots, f_q\}$. For each f_i $(1 \leq i \leq q)$, set the corresponding binary variable $x_{e(f_i)} = 1$ in ILP (5.1)-(5.3). Since $f_i \cap f_j = \emptyset$ for $1 \leq i \neq j \leq q$, constraint (5.2) of the ILP is satisfied. Further, each edge $e(f_i)$ corresponding

to $f_i \in \mathcal{M}$ has weight $w(e(f_i)) = 2$, implying the objective function value of ILP (5.1)-(5.3) is $2q$.

Assume that the objective function value of ILP (5.1)-(5.3) is $2q$. Let $X = \{e(f) \in E(H) \mid x_{e(f)} = 1\}$, where $x_{e(f)}$'s are the binary variables of ILP (5.1)-(5.3). For every edge $e(f) \in X$, add the corresponding set $f \in \mathcal{F}$ to $\mathcal{M}$. From constraint (5.2) of the ILP, X is pairwise disjoint and $|X| \leq |D(H)|$. Hence, $\mathcal{M}$ is a valid solution for $(A, B, C, \mathcal{F})$ with $|\mathcal{M}| = |X|$. Since every $e(f) \in X$ contains at most two different passengers and $|X| \leq |D(H)| = q$, $|X| = q$ for the objective function value to be $2q$. Thus, $|\mathcal{M}| = q$.

The size of $H(V, E)$ is polynomial in q. It takes a polynomial time to convert a solution of $H(V, E)$ to a solution of the 3DM instance $(A, B, C, \mathcal{F})$ and vice versa. □

5.3.2 Proposed approximation algorithms

Since solving the ILP (5.1)-(5.3) formulation exactly is NP-hard, it may require exponential time in a worst case, which is not acceptable in practice. One way to solve this is to have a time limit on any solver (or exact algorithm). When the time limit is reached, output the current solution or the best solution found so far. However, this does not guarantee the quality of the solution. Hence, it is important to use an approximation algorithm as a fallback plan.

The approximation ratio of a ρ-approximation algorithm for a maximization problem is defined as $\frac{w(\mathcal{M})}{w(\text{OPT})} \geq \rho$ for $\rho < 1$, where $w(\mathcal{M})$ and $w(\text{OPT})$ are the values of approximation and optimal solutions, respectively. In this section, we give a $(1 - \frac{1}{e})$-approximation algorithm and a $\frac{1}{2}$-approximation algorithm for the MTR problem. Our $(1 - \frac{1}{e})$-approximation algorithm (refer to as ***LPR***) is a simplified version of the LP-rounding based algorithm obtained by Fleischer et al. [37]. Our $\frac{1}{2}$-approximation algorithm (refer to as ***ImpGreedy***) is a simplified version of the simple greedy [16, 24] discussed in Subsection 5.3.3. By computing a solution directly from $H(V, E)$ without solving the independent set/weighted set packing problem, the running time and memory usage of ImpGreedy are significantly improved over the simple greedy.

The LPR algorithm.

The ILP (5.1)-(5.3) formulation is a special case of the Separable Assignment Problem (SAP): Given a set U of bins, a set I of items, a value f_{ij} for assigning item j to bin i, and a collection $\mathcal{I}_i$ of subsets of I for each bin i, SAP asks to find an assignment of items to bins such that each bin i can be assigned at most one set of $\mathcal{I}_i$, each item can be assigned to at most one bin and the total value f_{ij} of the assigned item is maximized. When only one bin i is considered, the problem is called the single-bin subproblem of SAP. It can be seen that the ILP (5.1)-(5.3) formulation of a hypergraph $H(V, E)$ is a special case of SAP, where the bins are drivers, items are passengers and the edges of H are $\cup_{i \in U} \mathcal{I}_i$ with unit value f_{ij} for all drivers i and passengers j.

Given a β-approximation algorithm for the single-bin subproblem of SAP, Fleischer et al. [37] obtained a local-search $(\frac{\beta}{\beta+1} - \epsilon)$-approximation algorithm ($\epsilon > 0$) and an LP-rounding based $((1 - \frac{1}{e})\beta)$-approximation algorithm for SAP. Both of these algorithms approximate the ILP (5.1)-(5.3). The local-search $(\frac{\beta}{\beta+1} - \epsilon)$-approximation algorithm presented by Fleischer et al. [37] is not efficient if one wants to have an approximation ratio as close to 1/2 as possible, assuming $\beta \approx 1$. This is because the number of iterations of the local-search algorithm is inverse-related to ϵ. The authors of [37] gave an LP for SAP, but it can have exponential number of variables due to $|\mathcal{I}_i|$ can be exponentially large in general. By the assumption that the maximum capacity λ of all vehicles is a small constant, $|\mathcal{I}_i|$ is polynomially bounded in our case. From this and unit value, the single-bin subproblem of the MTR problem can be solved efficiently ($\beta = 1$). This gives a $(1 - \frac{1}{e})$-approximation algorithm for the MTR problem. More importantly, the LP of ILP (5.1)-(5.3) can be solved directly because $|E(H)|$ ($|\mathcal{I}_i|$) is polynomially bounded. For completeness, we describe the LPR algorithm using our notation as follows.

1. Obtain a linear programming LP of ILP (5.1)-(5.3) by relaxing the 0-1 variables x_e to nonnegative real variables; and solve the LP.
2. Independently for each driver $\eta_i \in D(H)$, assign η_i a match $\sigma(i) = \{\eta_i\} \cup J$ corresponding to the edge $e = \{\eta_i\} \cup J$ with probability x_e (based on all edges containing η_i, namely, for all $e \in E_i$ such that $x_e > 0$). Let M be the resulting intermediate solution, which contains a set of feasible matches.
3. For any passenger $r_j \in R(H)$, let $M_j = \{\sigma(i) \in M \mid r_j \in \sigma(i)\}$ be the set of matches in M containing r_j.

   If $|M_j| \geq 2$, then remove r_j from every match of M_j except one match (any one match) of M_j. Finally, remove from M every match $\sigma(i) = \{\eta_i\}$.

The matches in M are pairwise disjoint. From Step 2, no two matches of M contain a same driver. From Step 3, no two matches of M contain a same passenger. In Step 3, after the removal of a set of passengers J from a match $\sigma(i) \in M$, $\sigma(i) \setminus J$ is still a feasible match if $|\sigma(i) \setminus J| \geq 2$ by Observation 5.1. Therefore, M is a feasible solution to an instance $(N, \mathcal{A}, T)$ of the MTR problem.

Theorem 5.4. *Let* OPT *be the objective function value of the ILP (5.1)-(5.3) formulation, which is the maximum number of passengers can be served. Then the expected value Q of the rounded solution M of Algorithm LPR is at least* $(1 - \frac{1}{e})$OPT.

Proof. Let OPT* be the objective function value of the LP relaxation. Then OPT$^* \geq$ OPT. From Theorem 2.1 in [37], $Q \geq (1 - (1 - \frac{1}{m})^m)\text{OPT}^* \geq (1 - \frac{1}{e})\text{OPT}$, where $m = |M|$. Since M is a feasible solution as explained above, the theorem holds. □

The ImpGreedy algorithm.

The ImpGreedy algorithm is similar to the $\frac{1}{\lambda+1}$-approximation algorithm obtained by Santi et al. [95] assuming the maximum capacity λ is at least two. However, a detailed analysis for the approximation ratio of their greedy algorithm is not presented in [95]. Hence, in this section, we describe our ImpGreedy algorithm along with a complete proof for its constant $\frac{1}{2}$-approximation ratio. For the hypergraph $H(V, E)$ constructed for an instance $(N, \mathcal{A}, T)$ of the MTR problem, denoted by $\Sigma \subseteq E(H)$ is the current partial solution computed by ImpGreedy (recall that each edge of $E(H)$ represents a feasible match). Let $R(\Sigma) = \bigcup_{e\in\Sigma} R(e)$, called the *covered passengers*. Initially, $\Sigma = \emptyset$. In each iteration, we add an edge with the most number of uncovered passengers to Σ, that is, select an edge e such that $|R(e)|$ is maximum, and then add e to Σ. Remove $E_e = \cup_{j\in e} E_j$ from $E(H)$, where E_j is the set of edges in $E(H)$ incident to j. Repeat until $|R(\Sigma)| = |R(H)|$ or $|\Sigma| = |D(H)|$. The pseudo code of ImpGreedy is shown in Algorithm 9.

Algorithm 9 ImpGreedy.

1: **Input:** The hypergraph $H(V, E)$ for problem instance $(N, \mathcal{A}, T)$.
2: **Output:** A solution Σ for $(N, \mathcal{A}, T)$ with $\frac{1}{2}$-approximation ratio.
3: $\Sigma = \emptyset$; $P(\Sigma) = \emptyset$;
4: **while** ($|P(\Sigma)| < |R(H)|$ and $|\Sigma| < |D(H)|$) **do**
5: compute $e = \text{argmax}_{e\in E(H)}|R(e)|$; $\Sigma = \Sigma \cup \{e\}$; update $P(\Sigma)$; remove E_e from $E(H)$;
6: **end while**

Next, we prove the correctness of Algorithm ImpGreedy. In ImpGreedy, when an edge e is added to Σ, E_e is removed from $E(H)$, so Property 5.1 holds for Σ. Further, the edges in Σ are pairwise vertex-disjoint, implying Σ is a feasible solution.

Property 5.1. *For every $\eta_i \in D(H)$, at most one edge e from E_i can be selected in any solution.*

Let $\Sigma = \{x_1, x_2, \ldots, x_a\}$ be a solution found by Algorithm ImpGreedy, where x_i is the i^{th} edge added to Σ. Throughout the analysis, we use OPT to denote an optimal solution, that is, OPT is a set of edges that are pairwise vertex-disjoint and $R(\text{OPT}) \geq R(\Sigma)$. Further, $\Sigma_i = \bigcup_{1\leq b\leq i} x_b$ for $1 \leq i \leq a$, $\Sigma_0 = \emptyset$ with $R(\Sigma_0) = \emptyset$, and $\Sigma_a = \Sigma$. Since each edge e of $E(H)$ represents a feasible match, we overload any edge $x_i \in \Sigma$ to denote a match as well. For each $x_i \in \Sigma$, by Property 5.1, there is at most one $y \in \text{OPT}$ with $D(y) = D(x_i)$. We order OPT and introduce dummy edges to OPT such that $D(y_i) = D(x_i)$ for $1 \leq i \leq a$. Formally, for $1 \leq i \leq a$, define

$$\text{OPT}(i) = \{y_1, .., y_i \mid 1 \leq b \leq i, D(y_b) = D(x_b) \text{ if } y_b \in \text{OPT}, \text{ otherwise } y_b \text{ a dummy edge}\}.$$

A dummy edge $y_b \in \text{OPT}(i)$ is defined as $D(y_b) = D(x_b)$ with $R(y_b) = \emptyset$. Notice that for any $y \in \text{OPT} \setminus \text{OPT}(a)$, $D(y) \neq D(x)$ for every $x \in \Sigma$.

Lemma 5.1. *Let* OPT *be an optimal solution and* $\Sigma = \{x_1, \ldots, x_a\}$ *be a solution found by ImpGreedy. For any* $1 \leq i \leq a$, $|R(y_i) \setminus R(\Sigma_{i-1})| \leq |R(x_i)|$.

Proof. If $|R(y_i)| \leq |R(x_i)|$, then the lemma holds. Suppose $|R(y_i)| > |R(x_i)|$. Since the algorithm selects x_i instead of y_i, it must mean that $R(y_i) \cap R(\Sigma_{i-1}) \neq \emptyset$, and y_i has been removed from $E(H)$ while searching for x_i. By Observation 5.1, there is an edge $e_z \in E(H)$ such that $D(e_z) = D(y_i)$ and $R(e_z) = R(y_i) \setminus R(\Sigma_{i-1})$; and $|R(e_z)| \leq |R(x_i)|$ from the algorithm. Hence, $|R(y_i) \setminus R(\Sigma_{i-1})| \leq |R(x_i)|$. □

Lemma 5.2. *Let* $\mathrm{OPT}' = \mathrm{OPT} \setminus \mathrm{OPT}(a)$. *Then,* $R(\mathrm{OPT}') \subseteq R(\Sigma)$.

Proof. Assume for contradiction that there exists an edge $y \in \mathrm{OPT}'$ s.t. $R(y) \setminus R(\Sigma) \neq \emptyset$. By Observation 5.1, there is an edge $e_z \in E(H)$ such that $D(e_z) = D(y)$ and $R(e_z) = R(y) \setminus R(\Sigma)$, and $e_z \notin \Sigma$. Since e_z is not incident to any vertex of $D(\Sigma) \cup R(\Sigma)$, the algorithm should have added e_z to Σ, a contradiction. □

Theorem 5.5. *Given a hypergraph instance* $H(V, E)$, *Algorithm ImpGreedy computes a solution* Σ *for* H *such that* $\frac{|R(\Sigma)|}{|R(\mathrm{OPT})|} \geq \frac{1}{2}$, *where* OPT *is an optimal solution, with running time* $O(|D(H)| \cdot |E(H)|)$ *and* $|E(H)| = O(|D| \cdot (|R| + 1)^{\lambda})$.

Proof. Let $\Sigma = \{x_1, \ldots, x_a\}$, $\mathrm{OPT}(a)$ as defined above, and $\mathrm{OPT}' = \mathrm{OPT} \setminus \mathrm{OPT}(a)$. From Lemma 5.1, we get

$$|\bigcup_{i=1}^{a} R(y_i) \setminus R(\Sigma_{i-1})| \leq |\bigcup_{i=1}^{a} R(x_i)| = |R(\Sigma)|.$$

From this and Lemma 5.2, we obtain

$$\begin{aligned} |R(\mathrm{OPT})| &= |R(\mathrm{OPT}(a)) \cup R(\mathrm{OPT}')| \\ &\leq |\bigcup_{i=1}^{a} (R(y_i) \setminus R(\Sigma_{i-1})) \cup R(\Sigma) \cup R(\mathrm{OPT}')| \\ &= |(\bigcup_{i=1}^{a} R(y_i) \setminus R(\Sigma_{i-1}))| + |R(\Sigma)| \\ &\leq |R(\Sigma)| + |R(\Sigma)| = 2|R(\Sigma)|. \end{aligned}$$

In each iteration of the while-loop, it takes $O(|E(H)|)$ to find an edge x with maximum $|R(x)|$, and there are at most $|D(H)|$ iterations. Hence, Algorithm ImpGreedy runs in $O(|D(H)| \cdot |E(H)|)$ time. □

5.3.3 Approximation algorithms for maximum weighted set packing

Now, we explain the algorithms for the maximum weighted set packing problem, which can also solve the MTR problem. Given a universe $\mathcal{U}$ and a family $\mathcal{S}$ of subsets of $\mathcal{U}$ such

that every subset $S \in \mathcal{S}$ has at most k elements, the maximum weighted k-set packing problem (MWSP) asks to find a packing $\mathcal{C}$ with the largest total weight. We can see that the ILP (5.1)-(5.3) formulation of a hypergraph $H(V, E)$ is a special case of the maximum weighted k-set packing problem, where the trips of $D(V) \cup R(V)$ is the universe $\mathcal{U}$ and $E(H)$ is the family $\mathcal{S}$ of subsets, and every $e \in E(H)$ is a set in $\mathcal{S}$ representing at most $k = \lambda + 1$ trips. Hence, solving MWSP also solves the MTR problem. Hazan et al. [51] showed that the k-set packing problem cannot be approximated to within $O(\frac{k}{\ln k})$ in general unless P = NP. Chandra and Halldórsson [24] presented a $\frac{3}{2(k+1)}$-approximation and a $\frac{5}{2(2k+1)}$-approximation algorithms (referred to as ***BestImp*** and ***AnyImp***, respectively), and Berman [16] presented a $(\frac{2}{k+1})$-approximation algorithm (referred to as ***SquareImp***) for the weighted k-set packing problem. Here, $k \geq 3$.

The three algorithms in [16, 24] (AnyImp, BestImp and SquareImp) solve the weighted k-set packing problem by first transferring it into a weighted independent set problem, which consists of a vertex weighted graph $G(V, E)$ and asks to find a maximum weighted independent set in $G(V, E)$. We briefly describe the common local search approach used in these three approximation algorithms. A *claw* C in G is defined as an induced connected subgraph that consists of an independent set T_C of vertices (called talons) and a center vertex C_z that is connected to all the talons (C is an induced star with center C_z). For any vertex $v \in V(G)$, let $N(v)$ denotes the set of vertices in G adjacent to v, called the *neighborhood* of v. For a set U of vertices, $N(U) = \cup_{v \in U} N(v)$. The *local search* of AnyImp, BestImp and SquareImp uses the same central idea, summarized as follows:

1. The approximation algorithms start with an initial solution (independent set) I in G found by a *simple greedy* (referred to as ***Greedy***) as follows: select a vertex $u \in V(G)$ with largest weight and add to I. Eliminate u and all u's neighbors from being selected. Repeatedly select the largest weight vertex until all vertices are eliminated from G.
2. While there exists claw C in G w.r.t. I such that independent set T_C improves the weight of I (different for each algorithm), augment I as $I = (I \setminus N(T_C)) \cup T_C$; such an independent set T_C is called an *improvement*.

To apply these algorithms to the MTR problem, we need to convert the bipartite hypergraph $H(V, E)$ to a weighted independent set instance $G(V, E)$, which is straightforward. Each hyperedge $e \in E(H)$ is represented by a vertex $v_e \in V(G)$. The weight $w(v_e) = w(e)$ for each $e \in E(H)$ and $v_e \in V(G)$. There is an edge between $v_e, v_{e'} \in V(G)$ if $e \cap e' \neq \emptyset$ where $e, e' \in E(H)$. We observed the following property.

Property 5.2. *When the size of each set in the set packing problem is at most k ($|w(e)| = k - 1, e \in E(H)$), the graph $G(V, E)$ has the property that it is $(k + 1)$-claw free, that is, $G(V, E)$ does not contain an independent set of size $k+1$ in the neighborhood of any vertex.*

Applying this property, we only need to search a claw C consists of at most k talons, which upper bounds the running time for finding a claw within $O(n^k)$, where $n = |V(G)|$.

When k is very small, it is practical enough to approximate the ILP (5.1)-(5.3) of a hypergraph $H(V, E)$ computed by Algorithm 8. It has been mentioned in [95] that the approximation algorithms in [24] can be applied to their ridesharing problem. However, only the simple greedy (*Greedy*) was implemented in [95]. Notice that ImpGreedy (Algorithm 9) is a simplified version of the Greedy algorithm, and Greedy is used to get an initial solution in algorithms AnyImp, BestImp and SquareImp. From Theorem 5.5, we have Corollary 5.1.

Corollary 5.1. *Each of Greedy, AnyImp, BestImp and SquareImp algorithms computes a solution to $H(V, E)$ with $\frac{1}{2}$-approximation ratio.*

Since ImpGreedy finds a solution directly on $H(V, E)$ without converting it to an independent set problem $G(V, E)$ and solving it, ImpGreedy is more time and space efficient than the algorithms for MWSP. In the rest of this chapter, Algorithm 9 is referred to as ImpGreedy.

5.4 Experiment

We create a simulation environment, which consists of a integrated transportation system (ITS) that integrates public transit and ridesharing. The ITS receives continuous batches of discrete driver and passenger trips. We implement the approximation algorithms ImpGreedy, LPR, Greedy, AnyImp and BestImp, and an exact algorithm that solves ILP formulation (5.1)-(5.3) to evaluate the benefits of having such an integrated transportation system. The results of SquareImp are not discussed because its performance is the same as AnyImp when using the smallest improvement factor ($\alpha > 1$ in [24]); this is due to the implementation of the independent set instance $G(V, E)$ having a fixed search/enumeration order of the vertices and edges, and each vertex in $V(G)$ has an integer weight.

We use a simplified transit network of Chicago to simulate the public transit and ridesharing. The data instances generated in our experiments focus more on trips that commute to and from work (to and from the downtown area of Chicago). To the best of our knowledge, a mass transportation system in large cities integrating public transit and ridesharing has not been implemented in real-life. There is not any large dataset containing customers that use both public transit and ridesharing transportation modes. Hence, we use two related datasets to generate representative instances for our experiments. One dataset contains transit ridership data, and the other dataset contains ridesharing trips data. The transit ridership dataset allows us to determine the busiest transit routes, and we use this information to create passenger demand in these busiest regions. We describe this more in Subsection 5.4.1. We assume passengers of longer transit trips would like to reduce their travel duration by using the integrated ridesharing service. The ridesharing dataset reveals whether there are enough personal drivers willing to provide ridesharing services. We understand that these drivers may not be the ones who drive their vehicles to work, but at least

it shows that there are currently enough drivers to support the proposed transportation system. We describe how to generate the drivers and passengers in Subsection 5.4.2.

5.4.1 Description and characteristics of the datasets

We built a simplified transit network of Chicago to simulate practical scenarios of public transit and ridesharing. The roadmap data of Chicago is retrieved from OpenStreetMap[1]. We used the GraphHopper[2] library to construct the logical graph data structure of the roadmap, which contains 177037 vertices and 263881 edges. The Chicago city is divided into 77 official community areas, each of which is assigned an area code. We examined two different datasets in Chicago to reveal some basic traffic pattern (the datasets are provided by the Chicago Data Portal (CDP) and Chicago Transit Authority (CTA)[3], maintained by the City of Chicago). The first dataset contains bus and rail ridership, which shows the monthly averages and monthly totals for all CTA bus routes and train station entries. We denote this dataset as *PTR, public transit ridership*. The PTR dataset range is chosen from June 1st, 2019 to June 30th, 2019. The second dataset contains rideshare trips reported by Transportation Network Providers (sometimes called rideshare companies) to the City of Chicago. We denote this dataset as *TNP*. The TNP dataset range is chosen from June 3rd, 2019 to June 30th, 2019, total of 4 weeks of data. Table 5.2 and Table 5.3 show some basic stats of both datasets.

Total Bus Ridership	20,300,416
Total Rail Ridership	19,282,992
12 busiest bus routes	3, 4, 8, 9, 22, 49, 53, 66, 77, 79, 82, 151
The busiest bus routes selected	4, 9, 49, 53, 77, 79, 82

Table 5.2: Basic stats of the PTR dataset.

# of original records	8,820,037
# of records considered	7,427,716
# of shared trips	1,015,329
# of non-shared trips	6,412,387
The most visited community areas selected	1, 4, 5, 7, 22, 23, 25, 32, 41, 64, 76

Table 5.3: Basic stats of the TNP dataset.

In the PTR dataset, the total ridership for each bus route is recorded; there are 127 bus routes in the dataset. We examined the 12 busiest bus routes based on the total ridership. 7 out of the 12 routes are selected (excluding bus routes that are too close to train stations) as listed in Table 5.2 to support the selection of the community areas. We also selected all the major trains/metro lines within the Chicago area except the Brown Line and Purple Line since they are too close to the Red and Blue lines. Note that the PTR dataset also provides the total rail ridership. However, it only provides the number of people entering

[1]Planet OSM. `https://planet.osm.org`

[2]GraphHopper 1.0. `https://www.graphhopper.com`

[3]CDP. `https://data.cityofchicago.org`. CTA. `https://www.transitchicago.com`

every station in each day; it does not provide the number of people exiting a station nor the time related to the entries.

Each record in the TNP dataset describes a passenger trip served by a driver who provides the rideshare service; a trip record consists of a pick-up and a drop-off time and a pick-up and a drop-off community area of the trip, and exact locations are not provided. We removed records where the pick-up or drop-off community area is hidden for privacy reason or not within Chicago, which results in 7.4 million ridesharing trips. We calculated

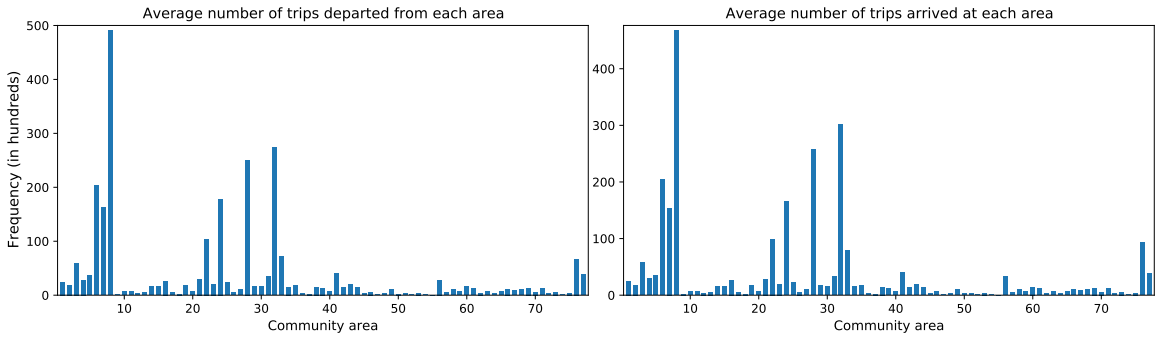


Figure 5.2: The average number of trips per day departed from and arrived at each area.

the average number of trips per day departed from and arrived at each area. The results are plotted in Figure 5.2; the community areas that have the highest numbers of departure trips are almost the same as that of the arrival trips.

We selected 11 of the 20 most visited areas as listed in Table 5.3 (area 32 is Chicago downtown, areas 64 and 76 are airports) to build the transit network for our simulation. From the selected bus routes, trains and community areas (22 areas in total), we created a simplified public transit network connecting the community areas, depicted in Figure 5.3. Three of the 22 community areas are the *designated locations* which include the downtown region in Chicago and the two airports. We label the rest of the 19 community areas as *urban community areas*. Each rectangle on the figure represents an *urban community* within one urban community area or across two urban community areas, labeled in the rectangle. The blue dashed rectangles/urban communities are chosen due to the busiest bus routes from the PTR dataset. The rectangles/urban communities labeled with red area codes are chosen due to the most visited community areas from the TNP dataset. The dashed lines are the trains, which resemble the major train services in Chicago. The solid lines are the selected bus routes connecting the urban communities to their closest train stations. We assume that there is a major bus route travels within each urban community or some minor bus route (not labeled in Figure 5.3) that travels to the nearest train station from each urban community. From the datasets, many people travel to/from the designated locations (downtown region and the two airports).

The travel time between two locations by car (each location consists of the latitude and longitude coordinates) uses the fastest/shortest route computed by the GraphHopper

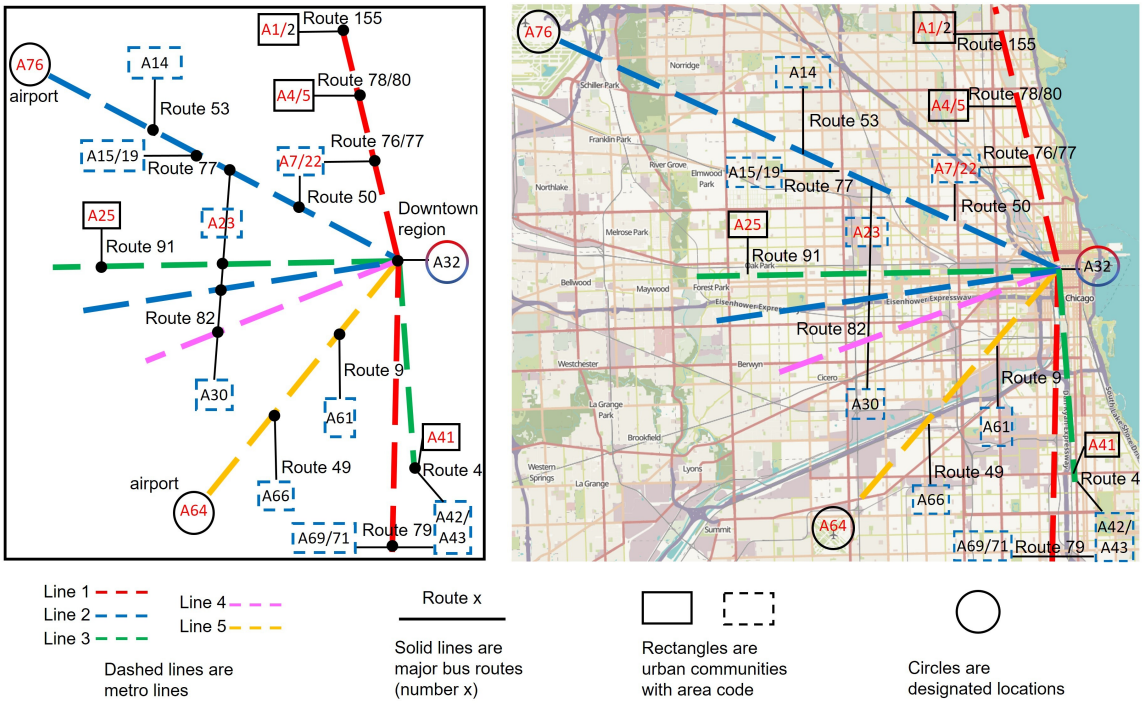
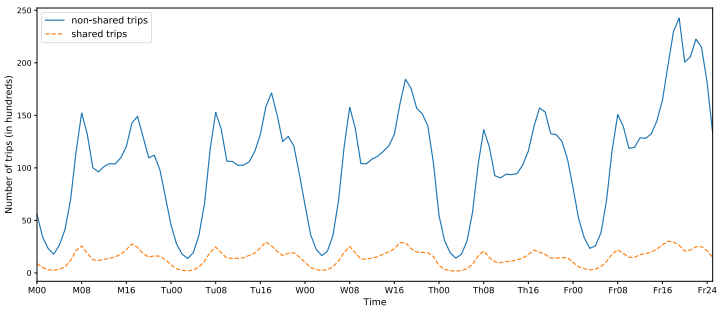


Figure 5.3: Simplified public transit network of Chicago with 19 urban community areas and 3 designated locations (minor bus routes are not shown). Figure on the right has the Chicago City map overlay for scale.

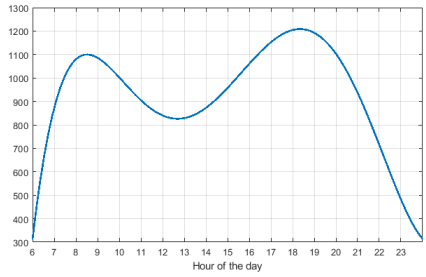
library. The shortest paths are **computed in real-time**, unlike many previous simulations where the shortest paths are pre-computed and stored. As mentioned in Subsection 5.2.2, transit travel and waiting time (transit time for short) and service time are considered in a simplified model; we multiply a small constant $\epsilon > 1$ to the fastest route to mimic transit time and service time. For instance, consider two consecutive metro stations s_1 and s_2. The travel time $t(s_1, s_2)$ is computed by the fastest route travelled by personal cars, and the travel time by train between from s_1 to s_2 is $\hat{t}(s_1, s_2) = 1.15 \cdot t(s_1, s_2)$. The constant ϵ for bus service is 2. Passenger trips originated from all locations (except airports) must take a bus to reach a metro station when ridesharing service is not involved.

5.4.2 Generating instances

In our simulation, we partition a day from 6:00 to 23:59 into 72 time intervals (each has 15 minutes), and we only focus on weekdays. To observe the common ridesharing traffic pattern, we calculated the average number of served passenger trips per hour for each day of the week using the TNP dataset. The dashed (orange) line and solid (blue) line of the plot in Figure (5.4a) represent shared trips and non-shared trips, respectively. A set of trips are called *shared trips* if this set of trips are matched for the same vehicle consecutively such that their trips may potentially overlap, namely, one or more passengers are in the same vehicle. The number of shared trips shown in Figure 5.4a suggests that drivers and



(a) Average numbers of shared and non-shared trips in TNP dataset.



(b) Total number of driver and passenger trips generated for each time interval.

Figure 5.4: Plots for the number of trips for every hour from data and generated.

passengers are willing to share the same vehicle. For all other trips, we call them *non-shared trips*. From the plot, the peak hours are between 7:00AM to 10:00AM and 5:00PM to 8:00PM on weekdays for both non-shared and shared trips. The number of trips generated for each interval roughly follows the function plotted in Figure (5.4b), which is a scaled down and smoothed version of the TNP dataset for weekdays. For the base instance, the ratio between the number of drivers and passengers generated is roughly 1:3 (1 driver and 3 passengers) for each interval. Such a ratio is chosen because it should reflect the system's potential as capacity of 3 is common for most vehicles. For each time interval, we first generate a set R of passengers and then a set D of drivers. We do not generate a trip where its origin and destination are close. For example, any trip with an origin in Area25 and destination in Area15 is not generated.

Generation of passenger trips.

We assume that the numbers of passengers entering and exiting a station are roughly the same each day. Next we assume that the numbers of passengers in PTR over the time intervals each day follow a similar distribution of the TNP trips over the time intervals. Each day is divided into 6 different consecutive time periods (each consists of multiple time intervals): *morning rush, morning normal, noon, afternoon normal, afternoon rush*, and *evening* time periods. Each time period determines the probability and distribution of origins and destinations. Based on the PTR dataset and Rail Capacity Study by CTA [25], many users are going into downtown in the morning and leaving downtown in the afternoon.

For each passenger trip r_i generated, we first randomly decide a pickup area where origin o_i is located within, then decide a dropoff area where destination d_i is located within. A *pickup area* or a *dropoff area* is one of the 22 community areas we selected to build our geological map for the simulation. For each community area, a set of points spanning the area is defined (each point is represented by a latitude-longitude pair). To generate a passenger trip r_i during **morning rush** time period, the pickup area for r_i is selected uniformly at random from the list of 22 community areas. The origin o_i is a point selected uniformly

at random from the set of points in the selected pickup area. Then, we use the standard normal distribution to determine the *dropoff area*, namely, the 22 selected community areas are transformed to follow the standard normal distribution. Specifically, downtown area is within two SDs (standard deviations), airports are more than two and at most three SDs, and the other urban community areas are more than three SDs away from the mean. Then, the dropoff area is sampled/selected randomly from this distribution. The destination d_i is a point selected uniformly at random from the set of points in the selected dropoff area.

The above is repeated until a_t passengers are generated, where $a_t + a_t/3$ (passengers + drivers so that it is roughly 1:3 driver-passenger ratio) is the total number of trips for time interval t shown in Figure (5.4b). For any pickup area c, let c_t be the number of generated passengers originated from c for time interval t, that is, $\sum_c c_t = a_t$. Other time periods follow the same procedure, and all urban communities and designated locations can be selected as pickup and dropoff areas.

1. **Morning normal** (10:00AM to 12:00AM). For selecting pickup areas, the 22 community areas are transformed to follow the standard normal distribution: urban community areas are within two SDs, downtown is more than two and at most three SDs, and airports are more than three SDs away from the mean; and destination areas are selected using uniform distribution.
2. **Noon** (12:00PM to 2:00PM). Pickup/dropoff areas are selected uniformly at random from the list of 22 community areas.
3. **Afternoon normal** (2:00PM to 5:00PM). For selecting pickup areas, downtown and airport are within two SDs and urban community areas are more than two SDs away from the mean. For selecting dropoff areas, urban community areas are within two SDs, and downtown and airports are more than two SDs away from the mean.
4. **Afternoon rush** (5:00PM to 8:00PM). For selecting pickup areas, downtown is within two SDs, airports are more than two SDs and at most three SDs, and urban community areas are more than three SDs away from the mean. For selecting dropoff areas, urban community areas are within two SDs, airports are more than two SDs and at most three SDs, and downtown is more than three SDs away from the mean.
5. **Evening** (8:00PM to 11:59PM). For both pickup and dropoff areas, urban community areas are within two SDs, downtown is more than two and at most three SDs, and airports are more than three SDs away from the mean.

Generation of driver trips.

We examined the TNP dataset to determine whether, in practice, there are enough drivers who can provide ridesharing service to passengers that follow match Types 1 and 2 traffic pattern. First, we removed any trip from TNP if it is too short (less than 15 minutes or

origin and destination are adjacent areas). We calculated the average number of trips per hour originated from every pre-defined area in the transit network (Figure 5.3), and then plotted the destinations of such trips in a grid heatmap. In other words, each cell (c, r) in the heatmap represents the the average number of trips per hour originated from area c to destination area r in the transit network (Figure 5.3). An example is depicted in Figure 5.5. From the heatmaps, many trips are going into the downtown area (A32) in the morning; and

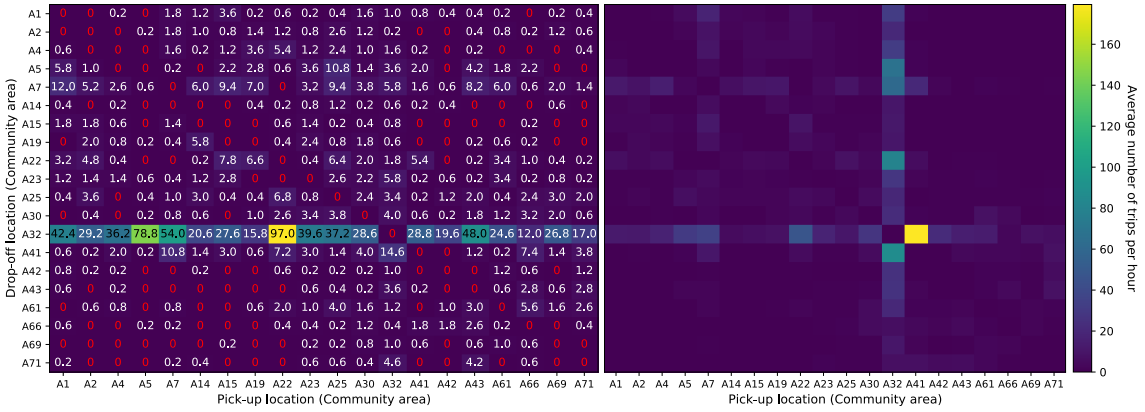


Figure 5.5: Traffic heatmaps for the average number of trips originated from one area (x-axis) during hour 7:00 (left) and hour 17:00 (right) to every other destination area (y-axis).

as time progresses, more and more trips leave downtown. This traffic pattern confirms that there are enough drivers to serve the passengers in our simulation. The number of shared trips shown in Figure 5.4a also suggests that many passengers are willing to share a same vehicle. We slightly reduce the difference between the values of each cell in the heatmaps and use the idea of marginal probability to generate driver trips. Let $d(c, r, h)$ be the value at the cell (c, r) for origin area c, destination r and hour h. Let $P(c, h)$ be sum of the average number of trips originated from area c for hour h (the column for area c in the heatmap corresponds to hour h), that is, $P(c, h) = \sum_r d(c, r, h)$ is the sum of the values of the whole column c for hour h. Given a time interval t, for each area c, we generate $c_t/3$ drivers (c_t is defined in Generation of passenger trips) such that each driver i has origin $o_i = c$ and destination $d_i = r$ with probability $d(c, r, h)/P(c, h)$, where t is contained in hour h. The probability of selecting an airport as destination is fixed at 5%.

Deciding other parameters for each trip.

After the origin and destination of a passenger or driver trip have been determined, we decide other parameters of the trip. The capacity λ_i of drivers' vehicles is selected from three ranges: the *low range* [1,2,3], *mid range* [3,4,5], and *high range* [4,5,6]. During morning/afternoon peak hours, roughly 95% and 5% of vehicles have capacities randomly selected from the low range and mid range, respectively. It is realistic to assume vehicle capacity is lower for morning and afternoon peak-hour commute. While during off-peak hours, roughly 80%, 10%

and 10% of vehicles have capacities randomly selected from low range, mid range and high range, respectively. The number δ_i of stops equals to λ_i if $\lambda_i \leq 3$, else it is chosen uniformly at random from $[\lambda_i - 2, \lambda_i]$ inclusive. The detour limit z_i of each driver is within 5 to 20 minutes because traffic is not considered, and transit time and service time are considered in a simplified model. Earliest departure time α_i of a driver or passenger i is from immediate to two time intervals. Latest arrival time β_i of a driver η_i is at most $1.5 \cdot (t(o_i, d_i) + z_i) + \alpha_i$. Latest arrival time β_j of a passenger r_j is $\alpha_j + t(\hat{\pi}_j(\alpha_j))$, where $\hat{\pi}_j(\alpha_j)$ is the fastest public transit route for r_j. The acceptance threshold θ_j of every passenger r_j is 0.8 for the base instance. The general information of the base instance is summarized in Table 5.4. Note that the earliest departure time all trips generated in the last four time intervals is immediate for computational-result purpose.

Major trip patterns	from urban communities to downtown and vice versa for peak and off-peak hours, respectively; trips specify one match type for peak hours and can be in either type for off-peak hours
# of intervals simulated	start from 6:00 AM to 11:59 PM; each interval is 15 minutes
# of trips per interval	varies from [350, 1150] roughly, see Figure 5.4
Driver-passenger ratio	1:3 approximately
Capacity λ_i of vehicles	low range: [1,3], mid range: [3,5] and high range: [4,6] inclusive
Number of stops limit	$\delta_i = \lambda_i$ if $\lambda_i \leq 3$, or $\delta_i \in [\lambda_i - 2, \lambda_i]$ if $\lambda_i \geq 4$
Earliest departure time α_i	immediate to 2 intervals after a trip i (driver or passenger) is generated
Driver detour limit z_i	5 minutes to $\min\{2 \cdot t(o_i, d_i)$ (driver's fastest route), 20 minutes$\}$
Latest arrival time of driver η_i	$\beta_i \leq 1.5 \cdot (t(o_i, d_i) + z_i) + \alpha_i$
Latest arrival time of passenger r_j	$\beta_j = \alpha_j + t(\hat{\pi}_j(\alpha_j))$, where $\hat{\pi}_j(\alpha_j)$ is the fastest public transit route for j with earliest departure time α_j from o_j
Travel duration of driver η_i	$\gamma_i = t(o_i, d_i) + z_i$
Travel duration of passenger r_j	$\gamma_j = t(\hat{\pi}_j(\alpha_j))$, where $\hat{\pi}_j(\alpha_j)$ is the fastest public transit route for j
Acceptance threshold	80% for all passengers (0.8 times the fastest public transit route)
Train and bus travel time	average at 1.15 and 2 times the fastest route by car, respectively

Table 5.4: General information of the base instance.

Reduction configuration procedure.

When the number of trips increases, the running time for Algorithm 8 and the time needed to construct the k-set packing instance (and independent set instance) also increase. This is due to the increased number of feasible matches for each driver $\eta_i \in D$. In a practical setup, we may restrict the number of feasible matches a driver can have. Recall that each match produced by Algorithm 7 is a *base match*, consisting of one driver and one passenger. To make the simulation feasible and practical, we heuristically limit the numbers of base

matches for each driver and each passenger and the number of total feasible matches for each driver. We use $(x\%, y, z)$, called *reduction configuration* (*Config* for short), to denote that for each driver η_i, the number of base matches of η_i is reduced to x percentage and at most y total feasible matches are computed for η_i; and for each passenger r_j, at most z base matches containing r_j are used.

After Algorithm 7 is completed. A reduction procedure may be evoked with respect to a Config. Let $H(V, E)$ be the graph after computing all feasible base matches (instance computed by Algorithm 7 and before Algorithm 8 is executed). For a trip $i \in \mathcal{A}$, let E_i be the set of base matches of i. The reduction procedure works as follows.

- First of all, the set of drivers is sorted in descending order of the number of base matches each driver has.
- Each driver η_i is then processed one by one.
    1. If driver η_i has at least 10 base matches, then E_i is sorted, based on the number of base matches each passenger included in E_i has, in descending order. Otherwise, skip η_i and process the next driver.
    2. For each base match $e = (\eta_i, r_j)$ in the sorted E_i, if passenger r_j belongs to z or more other matches, remove e from E_i.
    3. After above step 2, if E_i has not been reduced to $x\%$, sort the remaining matches in descending order of the travel time from o_i to o_j for remaining matches $e = (\eta_i, r_j)$. Remove the first x' matches from E_i until $x\%$ is reached.

The original sorting of the drivers allows us to first remove matches from drivers that have more matches than others. The sorting of the base matches of driver η_i in step 1 allows us to first remove matches containing passengers that also belong to other matches. Passengers farther away from a driver η_i may have lower chance to be served together by η_i; this is the reason for the sorting in step 3.

5.4.3 Computational results

We use the same transit network and same set of generated trip data for all algorithms. All algorithms were implemented in Java, and the experiments were conducted on Intel Core i7-2600 processor with 1333 MHz of 8 GB RAM available to JVM. To solve the ILP formulation (5.1)-(5.3) and the formulation in LPR, we use CPLEX v12.10.0; and we label the algorithm CPLEX uses to solve these ILP formulations by **Exact**. Since the optimization goal is to assign acceptable ridesharing routes to as many passengers as possible, the performance measure is focused on the number of passengers serviceable by acceptable ridesharing routes, followed by the total time saved for the passengers as a whole. We record both of these numbers for each of the algorithms: ImpGreedy, LPR, Exact, Greedy, AnyImp and BestImp.

A passenger $r_j \in R$ is called *served* if $r_j \in \sigma(i)$ for some driver $\eta_i \in D$ such that $\sigma(i)$ belongs to a solution computed by one of the algorithms.

Results on a base instance

The base case instance uses the parameter setting described in Subsection 5.4.2 and Config (30%, 600, 20). The overall experiment results are shown in Table 5.5. Although the solutions

	ImpGreedy	LPR	Exact	Greedy	AnyImp	BestImp
Total number of passengers served	26597	22583	27940	26597	27345	27360
Avg number of passengers served per interval	369.4	313.7	388.1	369.4	379.8	380.0
Total time saved of all passengers riders	309369.1	260427.3	324718.4	309369.1	318729.6	318983.9
Avg time saved of served passengers per interval	4296.8	3617.0	4510.0	4296.8	4426.8	4430.3
Avg time saved per served passenger	11.63	11.53	11.62	11.63	11.65	11.66
Avg time saved per passenger	6.68	5.75	7.17	6.68	7.03	7.04
Avg public transit duration per passenger		30.54 minutes				
Total number of passengers and public transit duration		45314 and 1384100.97 minutes				

Table 5.5: Base case solution comparison between all algorithms. Every time unit is measured in minute.

computed by AnyImp and BestImp are slightly better than that of ImpGreedy, it takes much longer for AnyImp and BestImp to run to completion, as shown in a later experiment (Figure 5.8). The average number of passengers served per interval is calculated as the total number of passengers served divided by 72 (the number of intervals). The average time saved per served passenger is calculated as the total time saved divided by the total number of served passengers. The results of ImpGreedy and Greedy are aligned since they are essentially the same algorithm: 58.69% of total number of all passengers are assigned acceptable routes and 22.35% of total time are saved for those passengers. The results of AnyImp and BestImp are similar because of the density of the independent set graph $G(V, E)$ due to Observation 5.1. For AnyImp and BestImp, roughly 60.38% of total number of all passengers are assigned acceptable routes and 23.05% of total time are saved. For LPR, 49.8% of total number of all passengers are assigned acceptable routes and 18.82% of total time are saved. We show that LPR is worse than ImpGreedy in terms of performance and running time in a later experiment. For Exact, 61.66% of total number of all passengers are assigned acceptable routes and 23.46% of total time are saved.

The average public transit duration per passenger is calculated as the total public transit duration divided by total number of all passengers, which is 30.54 minutes. The average time saved per passenger is calculated as the total time saved divided by the total number of all passengers (served and unserved). From Algorithm Exact, a passenger is able to reduce their travel duration from 30.54 minutes to 23.37 minutes on average (save 7.17 minutes) with the integration of public transit and ridesharing.

If we consider only the served passengers (26597 for ImpGreedy and 27940 for Exact), the average origin public transit duration per served passenger is 30.29 (30.30) minutes for ImpGreedy (Exact respectively) In this case, the average public transit + ridesharing

duration per served passenger is 18.65 (18.67) minutes, 11.63 (11.62) minutes saved, for ImpGreedy (Exact respectively). Although this is only a side-effect of the optimization goal of MTR, it reduces passengers' travel duration significantly. The results of these algorithms are not too far apart. However, it takes too long for AnyImp and BestImp to run to completion. A 10-second limit is set for both algorithms in each iteration for finding an independent set improvement. With this time limit, AnyImp and BestImp run to completion within 10 minutes for almost all intervals. The optimal solution computed by Exact serves only about 5% more total passengers than that of the solution computed by ImpGreedy; and it is most likely constrained by the number of feasible matches each driver has, which is also limited by the base match reduction Config (30%, 600, 20). We explore more about this in Subsection 5.4.3.

We also examined results from the drivers' perspective; we recorded both the mean occupancy rate and vacancy rate of drivers. The results are depicted in Table 5.6 and Figure 5.6. The mean occupancy rate is calculated as, in each interval, (the number of served passengers + the total number of drivers) divided by the total number of drivers. The mean vacancy rate describes the number of empty vehicles, so it is calculated as, in each interval, the number of drivers who are not assigned any passenger divided by the total number of drivers. The average occupancy rate per interval is the sum of mean occupancy rate in each interval divided by the number of intervals (72); and similarly for average vacancy rate per interval. The occupancy rate results show that in many intervals, 1.7-1.8 passengers are served by each driver on average (except Algorithm LPR). The vacancy rate results show that in many intervals, only 4-7.5% and 2-5% of drivers are not assigned any passenger for ImpGreedy and BestImp/Exact, respectively, during all hours except afternoon peak hours. This is most likely due to the origins of many trips are from the same

	ImpGreedy	LPR	Exact	BestImp
Average occupancy rate per interval	2.703	2.417	2.789	2.753
Average vacancy rate per interval	0.0693	0.193	0.0289	0.0436

Table 5.6: The average occupancy rate and vacancy rate per interval.

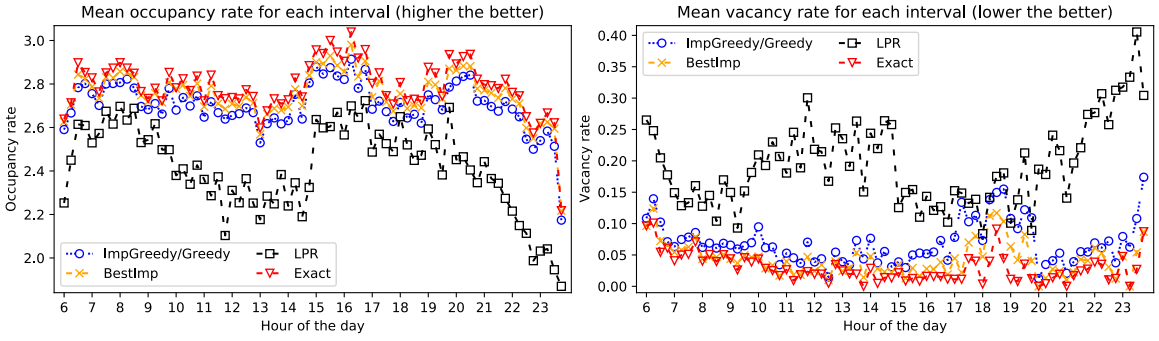


Figure 5.6: The average occupancy rate and vacancy rate of drivers for each interval.

area (downtown); and if the destinations of drivers and passengers do not have the same general direction originated from downtown, the drivers may not be able to serve many passengers. On the other hand, when their destinations are aligned, drivers are likely to serve more passengers. The occupancy rate is much lower in the last interval because the number of passengers is low, causing the number of served passengers low.

Results on different reduction configurations

Another major component of the experiment is to measure the performance of the algorithms using different reduction configurations. We tested 12 different Configs:

- *Small1* (20%,300,10), *Small2* (20%,600,10), *Small3* (20%,300,20), *Small4-10* (20%,600,20).
- *Medium1* (30%,300,10), *Medium2* (30%,600,10), *Medium3* (30%,300,20), *Medium4-10* (30%,600,20).
- *Large1* (40%,300,10), *Large2* (40%,600,10), *Large3-10* (40%,300,20), and *Large4-10* (40%,600,20).

Any Config with label "-10" at the end means there is a 10-second limit for AnyImp and BestImp to find an independent set improvement (Configs without any label have a 20-second limit). Note that all 12 Configs have the same sets of driver/passenger trips and base matches but have different feasible matches generated at the end (after Algorithm 8). The performance and running time results of all 12 Configs are depicted in Figures 5.7 and 5.8, respectively. Since the performance results of ImpGreedy and Greedy are the same, we skip Greedy.

The results are divided into peak and off-peak hours for each Config, averaging all intervals of peak hours and off-peak hours. As expected, larger Configs give better performance (more passengers are served by drivers). The increase in performance of Exact, compared to ImpGreedy, remains at about 5% for each different Config. This shows that ImpGreedy is practical in terms of performance. For all algorithms, the increase in performance from Small1 to Small3 is much larger than that from Small1 to Small2 (same for Medium and Large), implying any parameter in a Config should not be too small. The increase in performance from Large1 to Large4 is higher than that from Medium1 to Medium4 (similarly

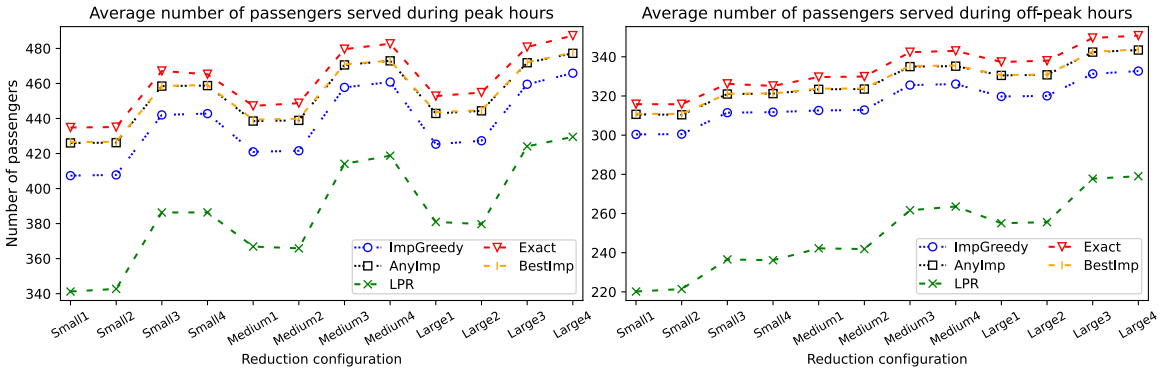


Figure 5.7: Average performance of peak and off-peak hours for different configurations.

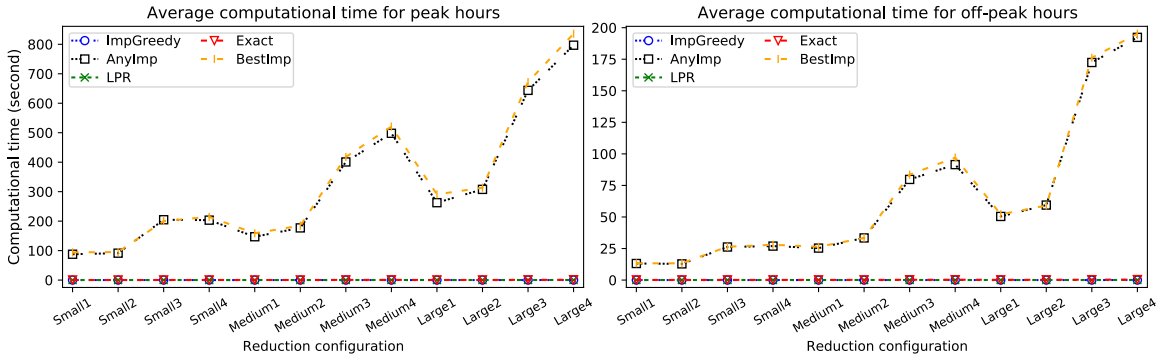


Figure 5.8: Average running time of peak and off-peak hours for different configurations.

for Small). Therefore, a balanced configuration is more important than a configuration emphasizes only one or two parameters. The average running times of ImpGreedy, LPR and Exact are under a second for all Configs. On the other hand, for AnyImp and BestImp during peak hours, they require 600-800 seconds and 400-500 seconds for Large3/Large4 and for Medium3/Medium4 Configs, respectively. By reducing more matches, we are able to improve the running time of AnyImp and BestImp significantly by sacrificing performance slightly. However, it may still be not practical to use AnyImp and BestImp for peak hours.

We specifically compared the performance of ImpGreedy and LPR since these two are the more practical approximation algorithms. The performance and running time results of ImpGreedy and LPR using Medium4 and Large4 configs are depicted in Figure 5.9. For

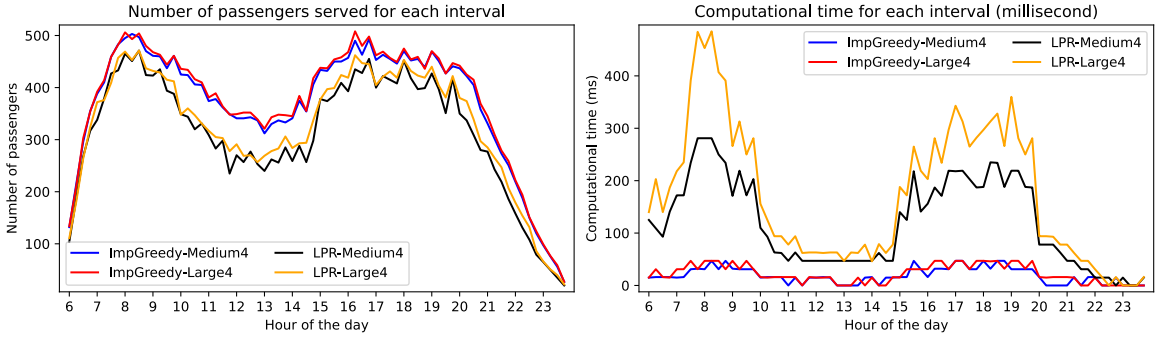


Figure 5.9: Performance of ImpGreedy and LPR using Medium4 and Large4 Configs.

both Configs, ImpGreedy is better than LPR in both performance and running time for each interval. The difference in performance is most likely due to the removal of passengers (step 3 of LPR). After a passenger r_j is removed from a match $\sigma(i)$, a match $\sigma'(i)$ with $j \notin \sigma'(i)$ and $|\sigma'(i)| > |\sigma(i) \setminus \{j\}|$ for driver η_i is not searched (even if such a match exists).

We further tested ImpGreedy, Exact and Greedy with the following Configs: *Huge1* (100%,600,10), *Huge2* (100%,2500,20) and *Huge3* (100%,10000,30) (these Configs have the same sets of driver/passenger trips and base match sets as those in the previous 12 Configs).

The focus of these Configs is to see if these algorithms can handle large number of feasible matches. The results are shown in Table 5.7.

ImpGreedy	Huge1	Huge2	Huge3
Avg number of riders served for peak/off-peak hours	405.8 / 329.2	458.7 / 347.5	482.5 / 354.2
Avg time saved of riders for peak/off-peak hours (min)	3462.7 / 4500.1	3987.6 / 4756.9	4237.4 / 4836.9
Avg time saved of riders per interval (min)	4154.3	4500.5	4637.0
Avg running time for peak/off-peak hours (sec)	0.0690 / 0.0254	0.327 / 0.0824	0.806 / 0.170
Exact	Huge1	Huge2	Huge3
Avg number of riders served for peak/off-peak hours	442.4 / 355.5	488.9 / 371.0	507.7 / 375.6
Avg time saved of riders for peak/off-peak hours (min)	3701.6 / 4911.7	4118.8 / 5128.3	4322.9 / 5216.2
Avg time saved of riders per interval (min)	4508.3	4791.8	4918.4
Avg running time for peak/off-peak hours (sec)	1.246 / 0.818	6.689 / 2.621	26.315 / 5.757
Greedy	Huge1	Huge2	Huge3
Avg running time for peak/off-peak hours (sec)	10.499 / 2.371	N/A	N/A
Avg instance size $G(V, E)$ of morning peak ($\|E(G)\|$)	0.014 billion	0.25 billion	2.4 billion
Avg time creating $G(V, E)$ of morning peak (sec)	10.43	200.41	1391.48

Table 5.7: The results of ImpGreedy and Greedy using Huge Configs.

Because ImpGreedy does not create the independent set instance, it runs quicker and uses less memory space than those of Greedy. Greedy cannot run to completion for Huge2 and Huge3 Configs because in many intervals, the whole graph $G(V, E)$ of the independent set instance is too large to hold in memory (8.00 GB for JVM). The average numbers of edges in $G(V, E)$ for morning peak hours are 0.014, 0.25 and 2.4 billion for Huge1, Huge2 and Huge3, respectively. There are techniques in graph processing to solve this problem. For example, one can use the out-of-core technique, which is to load the needed portion of a graph $G(V, E)$ for processing and unload the processed portion if necessary (e.g., [70, 109, 110]). However, this increases the total running time of the algorithms as the number of I/Os increases. Another way is to use distributed architecture to process large graphs [18, 74]. This approach may not create a burden in the running time, but it complicates the implementation and maintenance of the system. More importantly, the time it takes to create $G(V, E)$ can excess practicality in the first place regardless of what technique is used. The time, displayed in the last row of Table 5.7, is only the duration for finding all overlapping feasible matches to see if edges of $G(V, E)$ should be created (no actual independent set instance was created for Huge2 and Huge3).

Hence, using Greedy for large instances may not be practical, whereas both ImpGreedy and Exact can handle large instances and can run to completion quickly. For Huge3, there are 393738 feasible matches on average per interval during peak hours. ImpGreedy and Exact are able to compute a solution from these many feasible matches in about a second and 26 seconds, respectively. This shows that both algorithms are scalable when a reasonable Config is used or number of feasible matches is only reasonably large. Note that the average numbers of served passengers (405.8) and running time (0.06904 second) per interval during peak-hours of ImpGreedy with Huge1 is worse than that (452, 0.02475 second) produced by ImpGreedy with Medium3. Similarly, the average numbers of served passengers (442.4) and

running time (1.246 second) per interval during peak-hours of Exact with Huge1 is worse than that (475.0, 0.6230 second) produced by Exact with Medium3. These support the observation that a balanced configuration is more important than a configuration emphasizes only one or two parameters.

Lastly, we looked at the total (CPU) running times of the algorithms including the time for computing feasible matches (Algorithms 7 and 8). Table 5.8 shows the average running time of a time interval during peak hours for Algorithm 7 (Alg7), Algorithm 8 (Alg8), and the total time from Algorithm 7 to the finish of each tested algorithm. The running time

	Alg7 + Alg8	Total (CPU) time of Alg7 + Alg8 + each algorithm					
		ImpGreedy	LPR	Exact	Greedy	AnyImp	BestImp
Small3	500.58 + 34.63	535.2	535.3	535.6	535.9	739.9	735.5
Small4	500.58 + 35.41	536.0	536.1	536.4	536.9	739.3	750.4
Medium3	500.58 + 62.97	563.6	563.7	564.2	566.3	964.4	981.3
Medium4	500.58 + 55.32	555.9	556.1	556.7	558.8	1053.9	1076.7
Large4	500.58 + 83.26	583.9	584.1	584.9	590.3	1380.8	1419.7
Huge2	500.58 + 310.78	811.7	814.5	818.0	N/A	N/A	N/A
Huge3	500.58 + 368.93	870.3	878.1	895.8	N/A	N/A	N/A

Table 5.8: Average computational time (in seconds) of an interval during peak hours for all algorithms.

of Alg7 solely depends on computing the shortest paths between the trips and stations. Alg7 runs to completion in about 500 seconds on average per interval during peak hours (7AM-10AM and 5PM-8PM). As for Algorithm 8, when many trips' origins/destinations are concentrated in one area, the running time increases significantly, especially for drivers with high capacity. Running time of Alg8 can be reduced significantly by Configs with aggressive reductions. ImpGreedy and Exact are capable of handling large instances tested. Exact provides better solutions than any of the approximation algorithms. ImpGreedy gives solutions with quality close to other algorithms with running time less than a second for instances tested. ImpGreedy may be more practical for instances larger than those tested.

In conclusion, both ImpGreedy and Exact are much faster and uses less memory space, thus can handle large instances, compared to the other approximation algorithms. From the experiment results in Figure 5.7 and Table 5.8, it is beneficial to dynamically select different reduction configurations for each interval depending on the number of trips and the number of feasible matches. When the size of an instance is large and a solution must be computed within some time-limit, ImpGreedy may have a slight advantage over the Exact algorithm. Recall that the MTR problem (the ILP formulation (5.1)-(5.3)) is NP-hard by Theorem 5.3 (Theorem 5.1). From this, if the size of an instance or the number of feasible matches is larger, the running time of Exact for computing an optimal solution is not known and can be time consuming. As indicated by the results of Huge3, the running time of Exact is 26 times higher than that of ImpGreedy. A fallback plan would be to run ImpGreedy after Exact. If after a pre-defined time limit is reached and Exact still cannot compute an optimal

solution, the solution computed by ImpGreedy can be used. As shown by the experiments, the performance of ImpGreedy is still competitive.

Effects from different acceptance thresholds.

We consider three different acceptance thresholds for passengers: 0.9, 0.7 and 0.6 (in addition to 0.8, specified in Table 5.4, that is already tested in the base instance). As a reminder, an acceptance threshold (AT for short) 0.9 means that the acceptable ridesharing route given to every passenger r_j has travel time (duration) at most 0.9 times r_j's public transit duration $t(\hat{\pi}_j(\alpha_j))$. All other parameters in the base instance remain the same. To see the effect of different acceptance thresholds, two Configs are used: Large4-(40%, 600, 20) and Huge3-(100%, 10000, 30). Only ImpGreedy and Exact were tested.

The overall results are shown in Table 5.9 for Config Large4 and Table 5.10 for Config Huge3. The results for Large4 and Huge3 are consistent for both ImpGreedy and Exact. As somewhat expected, the total number of passengers served decreases for both Configs as the acceptance threshold decreases, since shorter travel duration is required according to passengers' requests. The data in Table 5.9 and Table 5.10 show that the total time saved of all served passengers increases when the acceptance threshold decreases, which implies that the number of passengers served is inversely related to the total time saved when the acceptance threshold changes.

Let us focus on the results using Huge3. From AT 0.9 to 0.8, total number of passengers served decreases by 3.635% (3.061%), whereas total time saved of all served passengers increases by 13.055% (15.789%) for ImpGreedy (Exact respectively). From this, decreasing the acceptance threshold from 0.9 to 0.8 only reduces the number of served passengers slightly but significantly reduces the travel time for each served passenger. For a smaller AT,

ImpGreedy	AT:0.9	AT:0.8	AT:0.7	AT:0.6
Total number of passengers served	27712	27008	26099	23456
Avg number of passengers served per interval	384.9	375.1	362.5	325.8
Total time saved of all served passengers	275329.5	315851.8	344727.3	354368.0
Avg time saved of served passengers per interval	3824.0	4386.8	4787.9	4921.8
Avg time saved per served passenger	9.94	11.69	13.21	15.11
Avg time saved per passenger	6.07	6.97	7.61	7.82
Exact	AT:0.9	AT:0.8	AT:0.7	AT:0.6
Total number of passengers served	29176	28430	27561	25184
Avg number of passengers served per interval	405.2	394.9	382.8	349.8
Total time saved of all served passengers	287628.5	331979.0	364905.9	379767.2
Avg time saved of served passengers per interval	3994.8	4610.8	5068.1	5274.5
Avg time saved per served passenger	9.86	11.68	13.24	15.08
Avg time saved per passenger	6.35	7.33	8.05	8.38
Total number of riders and public transit duration		45314 and 1384100.97 minutes		

Table 5.9: Overall solution comparison between different acceptance thresholds using Large3 Config. Every time unit is measured in minute.

ImpGreedy	AT:0.9	AT:0.8	AT:0.7	AT:0.6
Total number of passengers served	29657	28579	27218	24655
Avg number of passengers served per interval	411.9	396.9	378.0	342.4
Total time saved of all served passengers	295310.4	333864.3	361674.1	370964.7
Avg time saved of served passengers per interval	4101.5	4637.0	5023.3	5152.3
Avg time saved per served passenger	9.96	11.68	13.29	15.05
Avg time saved per passenger	6.52	7.37	7.98	8.19
Exact	AT:0.9	AT:0.8	AT:0.7	AT:0.6
Total number of passengers served	31168	30214	29122	26973
Avg number of passengers served per interval	432.9	419.6	404.5	374.6
Total time saved of all served passengers	305837.6	354127.2	386819.1	405339.6
Avg time saved of served passengers per interval	4247.7	4918.4	5372.5	5629.7
Avg time saved per served passenger	9.81	11.72	13.28	15.03
Avg time saved per passenger	6.75	7.81	8.54	8.95
Total number of riders and public transit duration		45314 and 1384100.97 minutes		

Table 5.10: Overall solution comparison between different acceptance thresholds using Huge3 Config. Every time unit is measured in minute.

a quicker acceptable route is required for a passenger. This may reduces the total number of passengers served but each served passenger saves more time. As a result, the average time saved per (served) passenger increases as AT decreases. Because the optimization goal of the MTR problem is to maximize the number of passengers served, the algorithms for MTR find solutions with more passengers served instead of focusing on more time saved even if the solutions for a smaller AT are solutions for a greater AT. From AT 0.8 to 0.7, the gap of inverse relation reduces to: 4.762% (3.164%) decreased in total number of passengers served and 8.330% (9.232%) increased in total time saved of all served passengers for ImpGreedy (Exact respectively); and the gap reduces further from AT 0.7 to 0.6. From the results, it seems that AT between 0.7-0.8 has a nice balance between the number of passengers served and time saved of served passengers. Although passengers can choose their own acceptance thresholds in practice, the system can suggest a default AT for all passengers which would balance between the chance of being served and the amount of time saved.

The occupancy rate and vacancy rate of drivers for Huge3 are depicted in Figure 5.10. The average occupancy and vacancy rates for both algorithms align with the result shown in Table 5.10. The average occupancy rate for Exact with AT 0.9 actually just exceeds 3 people per vehicle. As shown, the vacancy rate increases more as the acceptance threshold decreases. The average vacancy rate for Exact with AT 0.9 is 1.3%. If the main goal is to increase occupancy rate and decrease vacancy rate, using a centralized AT of 0.9 is effective. Bases on this and previous result (Table 5.10), an AT of 0.8 seems to be the most balanced. As a summary of the performance of Exact with AT 0.8 and Config Huge3, 66.68% of total passengers are assigned ridesharing routes and 25.59% of total time are saved; and passengers are able to reduce their average travel duration from 30.54 minutes to 22.73 minutes. If we consider only the served passengers (30214), the average origin public transit

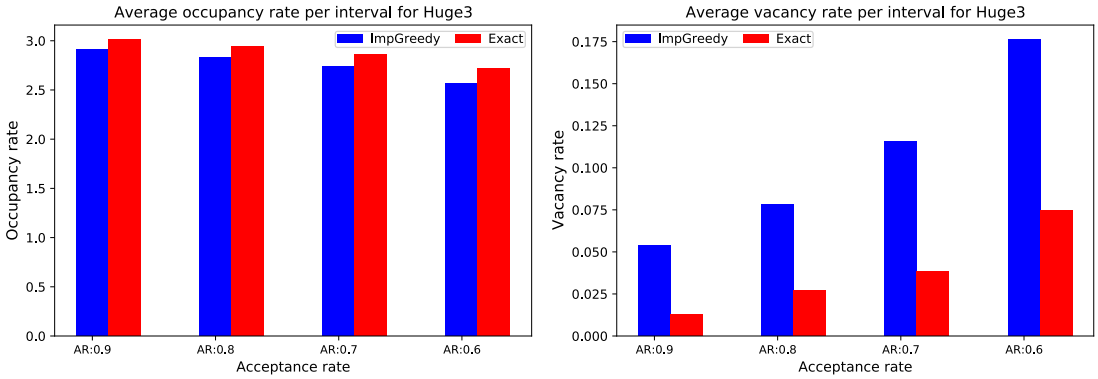


Figure 5.10: The average occupancy rate and vacancy rate per interval using Huge3 Config.

duration per served passenger is 30.29 minutes, and the average public transit + ridesharing duration per served passenger is 18.57 minutes.

5.5 Summary

We present an ILP formulation for the MTR problem. We prove that the MTR problem is NP-hard and developed an exact algorithm from the ILP (labeled as Exact) and two practical approximation algorithms for the problem: one (labeled as LPR) with $(1 - \frac{1}{e})$-approximation ratio and the other (labeled as ImpGreedy) with $\frac{1}{2}$-approximation ratio. Although Algorithm Exact may run in exponential time in a worst case, experiments show that it is efficient on practical data if the instance is not substantially large. Algorithm ImpGreedy outperforms Algorithm LPR in all instances tested for both performance and running time. Our base case experiments show that, on average, 61.7% and 58.7% of the passengers are assigned ridesharing routes and able to save 23.5% and 22.4% of travel time by Exact and ImpGreedy, respectively. Majority of the drivers are assigned at least one passenger, and vehicle occupancy rate has improved close to 3 (including the driver) on average. The number of passengers assigned to drivers and the time saved by ImpGreedy is about 95% of those by Exact. These results suggest that ridesharing can be an effective complement to public transit.

Chapter 6

Conclusion and Future Work

In this chapter, we summarize our results and discuss future work for the ridesharing minimization problems (RSOne and RSTwo, presented in Chapter 3), the ridesharing maximization problem (RPC, presented in Chapter 4), and the multimodal transportation with ridesharing maximization problem (MTR, presented in Chapter 5).

6.1 Ridesharing minimization problems

We have started a line of research that explores the hardness of the general ridesharing problem. Our results show that even restricted variants of RSOne and RSTwo are NP-hard. Specifically, if only one of the following Conditions C1-C5 is not satisfied (the four other conditions are satisfied), both RSOne (RSOne*) and RSTwo (RSTwo*) are still NP-hard.

C1. All trips have the same destination or all trips have the same origin, that is, $d_i = d_j$ for every pair of trip $i, j \in \mathcal{A}$ or $o_i = o_j$ for every pair of trip $i, j \in \mathcal{A}$.

C2. The individual of each trip can only serve others on the individual's preferred path only (without any detour), that is, $z_i = 0$ for every $i \in \mathcal{A}$.

C3. There is only one preferred path $\mathcal{P}_i = P_i$ for each trip, that is $|\mathcal{P}_i| \leq 1$ for every $i \in \mathcal{A}$. (note: if $\mathcal{P}_i = \emptyset$, a shortest path from o_i to d_i is computed by the system.)

C4. Each individual is willing to make at least $\delta_i \geq \lambda_i$ stops to either pick-up or drop-off passengers (or both), that is, $\delta_i \geq \lambda_i$ for pick-ups and/or drop-offs for every $i \in \mathcal{A}$.

C5. All trips have the same earliest departure time and same latest arrival time, that is, for every $i \in \mathcal{A}$, $\alpha_i = \alpha$ and $\beta_i = \beta$ for some $\alpha < \beta$.

Further, we show that it is NP-hard to approximate within a constant factor for both RSOne and RSTwo if one of Conditions C2-C5 is not satisfied. When all five Conditions C1-C5 and transitive serve relation are satisfied, we give a polynomial-time dynamic programming algorithm that can solve RSOne and RSTwo and a more time efficient algorithm that solves RSOne only. These results give a positive answer to the open problem:

Is there any variant of the ridesharing problem that can be solved in polynomial time?

However, it remains open that:

> whether RSOne or RSTwo is NP-hard or polynomial time solvable when the transitive serve relation is not satisfied and all five Conditions C1-C5 are satisfied.

We then investigate our second question:

> *Do there exist fast/efficient (practical enough) approximation algorithms for some variants of the ridesharing problem?*

We present three approximation algorithms for RSOneStop (an instance $(N, \mathcal{A})$ of RSOne that satisfies Conditions C1-C3 and C5). All three algorithms have an approximation ratio of $O(\frac{\lambda+2}{2})$, where λ is the maximum vehicle capacity of all trips in an $\mathcal{A}$. Two of these three approximation algorithms are modified from the approximation algorithms for the MCMP [65], called *EdgeSwap* and *StarImprove*. The third approximation algorithm is our novel algorithm that is more efficient than EdgeSwap and StarImprove. More precisely, the running times of EdgeSwap and StarImprove are $O(M+l^{2\lambda})$ and $O(M+\lambda \cdot l^3)$, respectively; our approximation algorithm has a running time of $O(M + l^2)$, where M is the size of the ridesharing instance which contains a road network N and l trips in $\mathcal{A}$.

An limitation of our exact and approximation algorithms for RSOne and RSTwo is that the algorithms are for some specific variants that must satisfy some of Conditions C1-C5 as mentioned above. That is not to say there are no applications for such variants. For example, school commute to a college/university and work commute to an office building align with some of the variants we studied. In other words, our algorithms can apply to these situations, and in fact, these scenarios have been studied in the literature for carpooling/ridesharing. For more general ridesharing cases that only satisfy one or two of Conditions C1-C5, one can first group the trips/individuals together who satisfy (or nearly satisfy) more conditions, and then apply our algorithms (as heuristics) to find a solution for each group. The drawback of this approach is that the approximation ratio is not retained. It is worth developing approximation algorithms for the NP-hard cases and exploring the algorithmic complexity of other simplified variants of ridesharing problem. It will be interesting to develop fast/efficient approximation algorithms for more general variants, anew or by extending our algorithms.

6.2 Ridesharing maximization problem

To reach the acclaimed potential of ridesharing, we need to fully adopt ridesharing in practice, especially for the existing ridehailing platforms (MoDs, such as Uber and Lyft). To promote ridesharing in practice, we study the RPC problem, which is to assign the maximum number of passengers to drivers for ridesharing service while satisfying an overall driver

profit constraint. Our model of the RPC problem provides a new framework to incorporate a flexible pricing scheme to maximize the number of passengers served while meeting a profit target. Our approach for solving the RPC problem allows personal/ad-hoc drivers and designated drivers to participate in the ridesharing system at the same time. Our solution approach for the RPC problem were developed by adopting the graph matching approach proposed by [95]. We create a hypergraph H to represent all feasible matches between the drivers of D and the passengers of R. We give an ILP formulation to the RPC problem, which is a variant of the weighted set packing formulation. This implies the NP-hardness of the RPC problem. Two variants of RPC are studied.

We first study a simpler variant of RPC (labeled as RPC1). In this variant, each driver can serve at most one passenger in any solution to RPC1. We present a polynomial-time exact algorithm framework (including two practical implementations of the algorithm) and a fast $\frac{1}{2}$-approximation algorithm for RPC1. The exact algorithm is based on finding a sequence of *minimum cost flows* in a flow network constructed from H. The $\frac{1}{2}$-approximation algorithm is based on finding a weighted bipartite matching in H.

The second variant of RPC studied is labeled as RPC+. In RPC+, drivers have arbitrary vehicle capacity, but only feasible matches with non-negative profit are considered in any solution to RPC+. We present a fast $\frac{1}{2\lambda}$-approximation algorithm for RPC+, which is a local search algorithm (labeled as LS2). If a specific condition on the profit target is met for an instance $(N, \mathcal{A})$, the approximation ratio of LS2 becomes $\frac{2}{3\lambda}$.

We create a simulation to evaluate our model and algorithms for both RPC1 and RPC+, based on a real-world ridesharing dataset in Chicago City. Because there is no practical test dataset publicly available for the RPC problem at the moment, we generate test instances (as realistic as possible) by using the ridesharing dataset and estimating Uber's profit model. Experimental results show that practical profit (price) schemes can be incorporated into our model and our algorithms are effective/efficient for the optimization goals.

The RPC+ variant considers only matches with non-negative profit, which may cover the MoD systems' profit-incentive, but it may impose a limit on improving the number of passengers served. It is worth developing algorithms for more general cases where matches with negative profit are also considered. A related optimization problem is to maximize the system-wide profit while a number of passengers must be served. Such an optimization problem may satisfy more demand compared to the RPC problem, which is important in reducing congestion and CO_2 emissions.

A limitation of our models and algorithms for different variants of the ridesharing problem (RPC, RSOne and RSTwo) presented in this thesis is that they focus on static ridesharing. To realize the full potential of ridesharing in practice, it would be important to extend these algorithms to dynamic ridesharing.

6.3 Multimodal transportation with ridesharing problem

We present an ILP formulation for maximizing the number of public transit users assigned to drivers in a transportation system that integrates public transit and ridesharing (the MTR problem). We prove that the MTR problem is NP-hard and develop an exact algorithm from the ILP (labeled as Exact) and two practical approximation algorithms for the problem: one (labeled as LPR) with $(1 - \frac{1}{e})$-approximation ratio using LP relaxation and rounding technique and the other (labeled as ImpGreedy) with $\frac{1}{2}$-approximation ratio using a greedy approach.

Based on real-world ridesharing and transit datasets in Chicago, we create datasets for an extensive computational study to evaluate our model and algorithms. Although Algorithm Exact may run in exponential time in a worst case, experiments show that its computational time is acceptable on practical data if the instance is not substantially large. Algorithm ImpGreedy runs much faster than Exact and has a performance close to Exact. Despite the theoretical approximation ratio of LPR is better than that of ImpGreedy, ImpGreedy outperforms LPR in all instances tested for both performance and running time. Algorithm ImpGreedy has a polynomial running time in the worst case and runs much faster than Exact for every instance tested. ImpGreedy can be a fallback plan for Exact when the latter cannot give a solution within a time limit in practice. Our simulation has shown that integrating public transit and ridesharing can benefit the transportation system as a whole.

It is worth improving the performance of ImpGreedy while keeping its time efficiency. The algorithms (exact and approximation) for both MTR and RPC rely on computing the feasible matches (Algorithm 7 + Algorithm 8) quickly for practical use. For the instances tested for MTR (Section 5.4) and for RPC (Section 4.5), the time to find all feasible matches (Algorithm 7 + Algorithm 8) is much longer than the computational times of the exact and approximation algorithms for either problem to assign passengers to drivers. It is important to develop faster algorithms for computing feasible matches for practice. There is a limitation of our simulation setup: we use a simplified transit system. It can be improved (for a more realistic experiment) by using the real transit schedule, as another future work. The delay of drivers and passengers due to traffic or any other reason is not exactly considered in our simulation either (vehicle speed due to traffic is estimated in the experiment for RPC). Since in practice, delay can happen; and it can cause a cascading effect on subsequent passenger pick-ups/drop-offs. Incorporating the delay of drivers and passengers into the algorithms and simulation would align with a more practical scenario.

Bibliography

[1] Niels Agatz, Alan Erera, Martin Savelsbergh, and Xing Wang. Dynamic ride-sharing: A simulation study in metro atlanta. *Transportation Research Part B: Methodological*, 45(9):1450–1464, 2011.

[2] Niels Agatz, Alan Erera, Martin Savelsbergh, and Xing Wang. Optimization for dynamic ride-sharing: A review. *European Journal of Operational Research*, 223(2):295–303, 2012.

[3] Ravindra Ahuja, Thomas Magnanti, and James Orlin. *Network Flows: theory, algorithms, and applications*. Prentice Hall, 1993.

[4] Kamel Aissat and Sacha Varone. Carpooling as complement to multi-modal transportation. In *Enterprise Information Systems (ICEIS 2015)*, pages 236–255, 2015.

[5] María Alonso-González, Theo Liu, Oded Cats, Niels Van Oort, and Serge Hoogendoorn. The potential of demand-responsive transport as a complement to public transport: an assessment framework and an empirical evaluation. *Transportation Research Record*, 2672(8):879–889, 2018.

[6] Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 114(3):462–467, 2017.

[7] Yasaman Amirkiaee and Nicholas Evangelopoulos. Why do people rideshare? an experimental study. *Transportation Research Part F: Traffic Psychology and Behaviour*, 55:9–24, 2018.

[8] Claudia Archetti, M. Grazia Speranza, and Daniele Vigo. *Chapter 10: Vehicle Routing Problems with Profits*, pages 273–297. Society for Industrial and Applied Mathematics, 2014.

[9] Vincent Armant and Kenneth N. Brown. Minimizing the driving distance in ride sharing systems. In *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*, pages 568–575, 2014.

[10] Mohammad Asghari and Cyrus Shahabi. Adapt-pricing: A dynamic and predictive technique for pricing to maximize revenue in ridesharing platforms. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '18, page 189–198. Association for Computing Machinery, 2018.

[11] American Automobile Association. Your driving costs fact sheet, August 2022.

[12] American Public Transportation Association. 2022 public transportation fact book. technical report, January 2023.

[13] Roberto Baldacci, Vittorio Maniezzo, and Aristide Mingozzi. An exact method for the car pooling problem based on lagrangean column generation. *Operation Research*, 52(3):422–439, 2004.

[14] Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. Route planning in transportation networks. In *Algorithm Engineering: Selected Results and Surveys*, volume 9220 of *LNCS*, pages 19–80, Springer, Cham, 2016.

[15] Xiaohui Bei and Shengyu Zhang. Algorithms for trip-vehicle assignment in ridesharing. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), 2018.

[16] Piotr Berman. A d/2 approximation for maximum weight independent set in d-claw free graphs. In *Algorithm Theory - SWAT 2000*, pages 214–219. Springer Berlin Heidelberg, 2000.

[17] Omar Besbes, Francisco Castro, and Ilan Lobel. Surge pricing and its spatial supply response. *Management Science*, 67(3):1350–1367, 2021.

[18] Sarra Bouhenni, Saïd Yahiaoui, Nadia Nouali-Taboudjemat, and Hamamache Kheddouci. A survey on distributed graph pattern matching in massive graphs. *ACM Comput. Surv.*, 54(2), 2021.

[19] Kris Braekers, Katrien Ramaekers, and Inneke Van Nieuwenhuyse. The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering*, 99:300–313, 2016.

[20] Bureau of Transportation Statistics, United States Department of Transportation. Average Cost of Owning and Operating an Automobile, National Transportation Statistics, August 2022.

[21] Juan Camilo Castillo, Dan Knoepfle, and Glen Weyl. Surge pricing solves the wild goose chase. In *Proceedings of the 2017 ACM Conference on Economics and Computation*, pages 241–242, 2017.

[22] Brian Caulfield. Estimating the environmental benefits of ride-sharing: A case study of dublin. *Transportation Research Part D: Transport and Environment*, 14(7):527–531, 2009.

[23] Center for Sustainable Systems, University of Michigan. Personal transportation factsheet, 2020.

[24] Barun Chandra and Magnús Halldórsson. Greedy local improvement and weighted set packing approximation. *Journal of Algorithms*, 39(2):223–240, 2001.

[25] Chicago Transit Authority. System-wide rail capacity study. Published June 2017; Revised February 2019.

[26] Christian Coester and Elias Koutsoupias. The online k-taxi problem. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1136–1147, 2019.

[27] Jean-François Cordeau. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54(3):573–586, 2006.

[28] Jean-François Cordeau and Gilbert Laporte. The dial-a-ride problem: models and algorithms. *Annals of Operations Research*, 153(1):29–46, 2007.

[29] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 2009.

[30] Milind Dawande, Jayant Kalagnanam, Pinar Keskinocak, Sibel Salman, and Ramamoorthi Ravi. Approximation algorithms for the multiple knapsack problem with assignment restrictions. *Journal of Combinatorial Optimization*, 4:171–186, 2000.

[31] Mi Diao, Hui Kong, and Jinhua Zhao. Impacts of transportation network companies on urban mobility. *Nature Sustainability*, 4:494–500, 2021.

[32] European Environment Agency. Greenhouse gas emissions from transport in europe, 2019.

[33] A. Di Febbraro, E. Gattorna, and N. Sacco. Optimization of dynamic ridesharing systems. *Transportation Research Record*, 2359(1):44–50, 2013.

[34] Sharon Feigon and Colin Murphy. Shared mobility and the transformation of public transit. TCRP Research Report, Transportation Research Board, 2016.

[35] Amos Fiat, Yuval Rabani, and Yiftach Ravid. Competitive k-server algorithms. *Journal of Computer and System Sciences*, 48(3):410–428, 1994.

[36] Andres Fielbaum, Xiaoshan Bai, and Javier Alonso-Mora. On-demand ridesharing with optimized pick-up and drop-off walking locations. *Transportation Research Part C: Emerging Technologies*, 126:103061, 2021.

[37] Lisa Fleischer, Michel X. Goemans, Vahab S. Mirrokni, and Maxim Sviridenko. Tight approximation algorithms for maximum general assignment problems. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, page 611–620. Society for Industrial and Applied Mathematics, 2006.

[38] Martin Fürer and Huiwen Yu. Approximating the k-set packing problem by local improvements. In *Combinatorial Optimization - Third International Symposium (ISCO 2014)*, LNCS, pages 408–420. Springer Verlag, 2014.

[39] Masabumi Furuhata, Maged Dessouky, Fernando Ordóñez, Marc-Etienne Brunet, Xiaoqing Wang, and Sven Koenig. Ridesharing: The state-of-the-art and future directions. *Transportation Research Part B: Methodological*, 57:28–46, 2013.

[40] Michael Garey and David Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[41] Keivan Ghoseiri, Ali Haghani, and Masoud Hamedi. Real-time rideshare matching problem. Final Report of UMD-2009-04, U.S. Department of Transportation, 2011.

[42] Qian-Ping Gu and Jiajian Liang. Multimodal transportation with ridesharing of personal vehicles. In *32nd International Symposium on Algorithms and Computation (ISAAC 2021)*, volume 212 of *LIPIcs*, pages 39:1–39:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

[43] Qian-Ping Gu, Jiajian Leo Liang, and Guochuan Zhang. Algorithmic analysis for ridesharing of personal vehicles. In *Proceedings of the 2016 International Conference on Combinatorial Optimization and Applications*, volume 10043 of *LNCS*, pages 438–452, 2016.

[44] Qian-Ping Gu, Jiajian Leo Liang, and Guochuan Zhang. Efficient algorithms for ridesharing of personal vehicles. In *Proceedings of the 2017 International Conference on Combinatorial Optimization and Applications*, volume 10627 of *LNCS*, pages 340–354, 2017.

[45] Qian-Ping Gu, Jiajian Leo Liang, and Guochuan Zhang. Algorithmic analysis for ridesharing of personal vehicles. *Theoretical Computer Science*, 749:36–46, 2018.

[46] Qian-Ping Gu, Jiajian Leo Liang, and Guochuan Zhang. Efficient algorithms for ridesharing of personal vehicles. *Theoretical Computer Science*, 788:79–94, 2019.

[47] Qian-Ping Gu, Jiajian Leo Liang, and Guochuan Zhang. Approximate ridesharing of personal vehicles problem. In *Proceedings of the 2020 International Conference on Combinatorial Optimization and Applications*, volume 12577 of *LNCS*, pages 3–18, 2020.

[48] Qian-Ping Gu, Jiajian Leo Liang, and Guochuan Zhang. Approximate ridesharing of personal vehicles problem. *Theoretical Computer Science*, 871:30–50, 2021.

[49] Aldy Gunawan, Hoong Chuin Lau, and Pieter Vansteenwegen. Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, 255(2):315–332, 2016.

[50] Irith Ben-Arroyo Hartman, Daniel Keren, Abed Abu Dbai, Elad Cohen, Luk Knapen, Ansar-Ul-Haque Yasar, and Davy Janssens. Theory and practice in large carpooling problems. *Procedia Computer Science*, 32:339–347, 2014.

[51] Elad Hazan, Shmuel Safra, and Oded Schwartz. On the complexity of approximating k-set packing. *Computational Complexity*, 15(1):20–39, 2006.

[52] Alejandro Henao and Weslet Marshall. The impact of ride-hailing on vehicle miles traveled. *Transportation*, 49:2173–2194, 2021.

[53] Wesam Herbawi and Michael Weber. The ridematching problem with time windows in dynamic ridesharing: A model and a genetic algorithm. In *2012 IEEE Congress on Evolutionary Computation*, pages 1–8, 2012.

[54] John Hopcroft and Richard Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[55] Hadi Hosni, Joe Naoum-Sawaya, and Hassan Artail. The shared-taxi problem: Formulation and solution methods. *Transportation Research Part B: Methodological*, 70:303–318, 2014.

[56] Fu-Shiung Hsieh. A comparative study of several metaheuristic algorithms to optimize monetary incentive in ridesharing systems. *ISPRS International Journal of Geo-Information*, 9(10):590, 2020.

[57] Bin Hu, Ming Hu, and Han Zhu. Surge pricing and two-sided temporal responses in ride hailing. *Manufacturing & Service Operations Management*, 24(1):91–109, 2022.

[58] Haosheng Huang, Dominik Bucher, Julian Kissling, Robert Weibel, and Martin Raubal. Multimodal route planning with public transport and carpooling. *IEEE Transactions on Intelligent Transportation Systems*, 20(9):3513–3525, 2019.

[59] Yan Huang, Ruoming Jin, Favyen Bastani, and Xiaoyang Sean Wang. Large scale real-time ridesharing with service guarantee on road networks. *Proceedings of the VLDB Endowment*, 7(14):2017–2028, 2014.

[60] Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM*, 24(1), 1977.

[61] Jaeyoung Jung, R. Jayakrishnan, and Ji Young Park. Dynamic shared-taxi dispatch algorithm with hybrid-simulated annealing. *Computer-Aided Civil and Infrastructure Engineering*, 31(4):275–291, 2016.

[62] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972.

[63] Antoon W.J. Kolen, Jan Karel Lenstra, Christos H. Papadimitriou, and Frits C.R. Spieksma. Interval scheduling: A survey. *Naval Research Logistics (NRL)*, 54(5):530–543, 2007.

[64] Pramesh Kumar and Alireza Khani. An algorithm for integrating peer-to-peer ridesharing and schedule-based transit system for first mile/last mile access. *Transportation Research Part C: Emerging Technologies*, page 122, 2021.

[65] Gilad Kutiel and Dror Rawitz. Local search algorithms for maximum carpool matching. In *Proceedings of 25th Annual European Symposium on Algorithms*, volume 87, pages 55:1–55:14, 2017.

[66] J. K. Lenstra and A. H. G. Rinnooy Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227, 1981.

[67] Manzi Li, Gege Jiang, and Hong K. Lo. Pricing strategy of ride-sourcing services under travel time variability. *Transportation Research Part E: Logistics and Transportation Review*, 159:102631, 2022.

[68] Jiajian Leo Liang. An algorithmic study on ridesharing problem. Master's thesis, Simon Fraser University, 2016.

[69] Todd A. Litman. *Transportation Cost and Benefit Analysis Techniques, Estimates and Implications*. Victoria Transport Policy Institute, 2009. (Updated October 2016).

[70] Hang Liu and H. Howie Huang. Graphene: Fine-Grained IO management for graph computing. In *15th USENIX Conference on File and Storage Technologies (FAST 17)*, pages 285–300, Santa Clara, CA, 2017.

[71] Mengyang Liu, Zhixing Luo, and Andrew Lim. A branch-and-cut algorithm for a realistic dial-a-ride problem. *Transportation Research Part B: Methodological*, 81:267–288, 2015.

[72] Yang Liu and Yuanyuan Li. Pricing scheme design of ridesharing program in morning commute problem. *Transportation Research Part C: Emerging Technologies*, 79:156–177, 2017.

[73] Roger Lloret-Batlle, Neda Masoud, and Daisik Nam. 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, 2017.

[74] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M. Hellerstein. Distributed graphlab: A framework for machine learning and data mining in the cloud. *Proc. VLDB Endow.*, 5(8):716–727, 2012.

[75] Meghna Lowalekar, Pradeep Varakantham, and Patrick Jaillet. Competitive ratios for online multi-capacity ridesharing. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, page 771–779, 2020.

[76] Qi Luo, Shukai Li, and Robert C. Hampshire. Optimal design of intermodal mobility networks under uncertainty: Connecting micromobility with mobility-on-demand transit. *EURO Journal on Transportation and Logistics*, 10:100045, 2021.

[77] Qi Luo, Viswanath Nagarajan, Alexander Sundt, Yafeng Yin, John Vincent, and Mehrdad Shahabi. Efficient algorithms for stochastic ridepooling assignment with mixed fleets, 2022.

[78] Shou Ma, Yu Zheng, and Ouri Wolfson. Real-time city-scale taxi ridesharing. *IEEE Transactions on Knowledge and Data Engineering*, 27(7):1782–1795, 2015.

[79] Tai-Yu Ma. On-demand dynamic bi-/multi-modal ride-sharing using optimal passenger-vehicle assignments. In *2017 IEEE International Conference on Environment and Electrical Engineering and 2017 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I CPS Europe)*, pages 1–5, 2017.

[80] Tai-Yu Ma, Saeid Rasulkhani, Joseph Y.J. Chow, and Sylvain Klein. A dynamic ridesharing dispatch and idle vehicle repositioning strategy with integrated transit transfers. *Transportation Research Part E: Logistics and Transportation Review*, 128:417–442, 2019.

[81] Leandro do Martins, Rocio de la Torre, Canan Corlu, Angel Juan, and Mohamed Masmoudi. Optimizing ride-sharing operations in smart sustainable cities: Challenges and the need for agile algorithms. *Computers & Industrial Engineering*, 153:107080, 2021.

[82] Neda Masoud, Daisik Nam, Jiangbo Yu, and R. Jayakrishnan. Promoting peer-to-peer ridesharing services as transit system feeders. *Transportation Research Record*, 2650(1):74–83, 2017.

[83] Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{|V|}|E|)$ algoithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science (SFCS 1980)*, pages 17–27, 1980.

[84] Yves Molenbruch, Kris Braekers, Patrick Hirsch, and Marco Oberscheider. Analyzing the benefits of an integrated mobility system using a matheuristic routing algorithm. *European Journal of Operational Research*, 290(1):81–98, 2021.

[85] Abood Mourad, Jakob Puchinger, and Chengbin Chu. A survey of models and algorithms for optimizing shared mobility. *Transportation Research Part B: Methodological*, 123:323–346, 2019.

[86] Ali Najmi, David Rey, and Taha H. Rashidi. Novel dynamic formulations for real-time ride-sharing systems. *Transportation Research Part E: Logistics and Transportation Review*, 108:122–140, 2017.

[87] Jishnu Narayan, Oded Cats, Niels van Oort, and Serge Hoogendoorn. Integrated route choice and assignment model for fixed and flexible public transport systems. *Transportation Research Part C: Emerging Technologies*, 115, 2020.

[88] Mehdi Nourinejad and Mohsen Ramezani. Ride-sourcing modeling and pricing in non-equilibrium two-sided markets. *Transportation Research Part B: Methodological*, 132:340–357, 2020. 23rd International Symposium on Transportation and Traffic Theory (ISTTT 23).

[89] Mehdi Nourinejad and Matthew J. Roorda. Agent based model for dynamic ridesharing. *Transportation Research Part C: Emerging Technologies*, 64, 2016.

[90] Xinwu Qian, Wenbo Zhang, Satish Ukkusuri, and Chao Yang. Optimal assignment and incentive design in the taxi group ride problem. *Transportation Research Part B: Methodological*, 103:208–226, 2017.

[91] Arvind U Raghunathan, David Bergman, John Hooker, Thiago Serra, and Shingo Kobori. Seamless multimodal transportation scheduling, 2018.

[92] Ulrike Ritzinger, Jakob Puchinger, and Richard F. Hartl. A survey on dynamic and stochastic vehicle routing problems. *International Journal of Production Research*, 54(1):215–231, 2016.

[93] Stefan Ropke, Jean-François Cordeau, and Gilbert Laporte. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks*, 49(4):258–272, 2007.

[94] Mauro Salazar, Nicolas Lanzetti, Federico Rossi, Maximilian Schiffer, and Marco Pavone. Intermodal autonomous mobility-on-demand. *IEEE Transactions on Intelligent Transportation Systems*, 21(9):3946–3960, 2020.

[95] Paolo Santi, Giovanni Resta, Michael Szell, Stanislav Sobolevsky, Steven H. Strogatz, and Carlo Ratti. Quantifying the benefits of vehicle pooling with shareability networks. *Proceedings of the National Academy of Sciences*, 111(37):13290–13294, 2014.

[96] A. Santos, N. McGuckin, H.Y. Nakamoto, D. Gray, and S. Liss. Summary of travel trends: 2009 national household travel survey. Technical report, US Department of Transportation Federal Highway Administration, 2011.

[97] Douglas Santos and Eduardo Xavier. Taxi and ride sharing: A dynamic dial-a-ride problem with money as an incentive. *Expert Systems with Applications*, 42(19):6728–6737, 2015.

[98] Grzegorz Sierpiński. Changes of the modal split of traffic in europe. *Archives of Transport System Telematics*, 6(1):45–48, 2013.

[99] Andrea Simonetto, Julien Monteil, and Claudio Gambella. Real-time city-scale ridesharing via linear assignment problems. *Transportation Research Part C: Emerging Technologies*, 101:208–232, 2019.

[100] Statistics Canada. Census: Main mode of commuting, 2016.

[101] Mitja Stiglic, Niels Agatz, Martin Savelsbergh, and Mirko Gradisar. The benefits of meeting points in ride-sharing systems. *Transportation Research Part B: Methodological*, 82:36–53, 2015.

[102] Mitja Stiglic, Niels Agatz, Martin Savelsbergh, and Mirko Gradisar. Enhancing urban mobility: Integrating ride-sharing and public transit. *Computers & Operations Research*, 90:12–21, 2018.

[103] Maxim Sviridenko and Justin Ward. Large neighborhood local search for the maximum set packing problem. In *Automata, Languages, and Programming (ICALP)*, pages 792–803, 2013.

[104] Amirmahdi Tafreshian, Neda Masoud, and Yafeng Yin. Frontiers in service science: ride matching for peer-to-peer ride sharing: a review and future directions. *Service Science*, 12(2-3):41–60, 2020.

[105] Mohammad Tamannaei and Iman Irandoost. Carpooling problem: A new mathematical model, branch-and-bound, and heuristic beam search algorithm. *Journal of Intelligent Transportation Systems*, 23(3):203–215, 2019.

[106] Vu Thi Thao, Sebastian Imhof, and Widar von Arx. Integration of ridesharing with public transport in rural switzerland: practice and outcomes. *Transportation Research Interdisciplinary Perspectives*, 10:100340, 2021.

[107] Ioannis Tikoudis, Luis Martinez, Katherine Farrow, Clara García Bouyssou, Olga Petrik, and Walid Oueslati. Ridesharing services and urban transport CO2 emissions: simulation-based evidence from 247 cities. *Transportation Research Part D: Transport and Environment*, 97, 2021.

[108] Alejandro Tirachini and Andres Gomez-Lobo. Does ride-hailing increase or decrease vehicle kilometers traveled (VKT)? a simulation approach for santiago de chile. *International Journal of Sustainable Transportation*, 14(3):187–204, 2020.

[109] Keval Vora. LUMOS: Dependency-Driven disk-based graph processing. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 429–442, Renton, WA, 2019.

[110] Keval Vora, Guoqing Xu, and Rajiv Gupta. Load the edges you need: A generic I/O optimization for disk-based graph processing. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, pages 507–522, Denver, CO, 2016.

[111] Hai Wang and Hai Yang. Ridesourcing systems: A framework and review. *Transportation Research Part B: Methodological*, 129:122–155, 2019.

[112] Zhengtian Xu, Yafeng Yin, and Jieping Ye. On the supply curve of ride-hailing systems. *Transportation Research Part B: Methodological*, 132:29–43, 2020. 23rd International Symposium on Transportation and Traffic Theory (ISTTT 23).

[113] Chiwei Yan, Helin Zhu, Nikita Korolko, and Dawn Woodard. Dynamic pricing and matching in ride-hailing platforms. *Naval Research Logistics*, 67:705–724, 2020.

[114] Hai Yang, Chaoyi Shao, Hai Wang, and Jieping Ye. Integrated reward scheme and surge pricing in a ridesourcing marke. *Transportation Research Part B: Methodological*, 134:126–142, 2020.

[115] Kenan Zhang and Yu Nie. To pool or not to pool: equilibrium, pricing and regulation. *Transportation Research Part B: Methodological*, 151:59–90, 2021.

[116] Yuanyuan Zhang and Yuming Zhang. Exploring the relationship between ridesharing and public transit use in the united states. *International Journal of Environmental Research and Public Health*, 15(8):1763, 2018.