

Approximate Ridesharing of Personal Vehicles Problem ¹

Qian-Ping Gu^a, Jiajian Leo Liang^a and Guochuan Zhang^b

^aSchool of Computing Science, Simon Fraser University, Canada
qgu@sfu.ca, leo.liang@sfu.ca

^bCollege of Computer Science and Technology, Zhejiang University, China
zgc@zju.edu.cn

Abstract. It is important to find ride matches for individuals who participate in ridesharing quickly, and it is equally important to minimize the number of drivers to serve all individuals and minimize the total travel distance of the vehicles. This paper considers the following ridesharing problem: given a set of trips, each trip consists of an individual, a vehicle of the individual and some requirements, select a subset of trips and use the vehicles of selected trips to deliver all individuals to their destinations while satisfying the requirements and achieving some optimization goal. Requirements of trips are specified by parameters including source, destination, vehicle capacity, preferred paths, detour distance and number of stops a driver is willing to make, and time constraints. We consider two optimization problems: minimizing the number of selected vehicles and minimizing total travel distance of the vehicles. We prove that it is NP-hard to approximate both minimization problems within a constant factor if any one of the requirements related to the detour distance, preferred paths, number of stops and time constraints is not satisfied. We give $\frac{K+2}{2}$ -approximation algorithms for minimizing the number of selected vehicles when the requirement related to the number of stops is not satisfied, where K is the largest capacity of all vehicles.

Keywords: Ridesharing problem, optimization problems, approximation algorithms, algorithmic analysis

1 Introduction

The use of shared mobility (carpooling/ridesharing) is becoming popular around the world. With recent advancement in communication technologies, ridesharing in large scale are emerging. Ridesharing services are enabling timely and convenient transportation to many people. The need of such services is increasing as the population grows in urban areas. At the same time, the number of cars on the road also increases. According to [34], personal vehicles are the main transportation mode in more than 200 European cities between 2001 and 2011. In the United States, personal vehicles are also the main transportation mode [9]. The occupancy rate of personal vehicles in the United States is 1.6 persons per vehicle in 2011 [15, 32]

¹A preliminary version of the paper appeared in the Proceeding of COCOA2020 [18].

(and decreased to 1.5 persons per vehicle in 2017 [9]), which can be a major cause for congestion, and the estimated cost of congestion is a round \$121 billion per [4]. Shared mobility (carpooling or ridesharing) is a promising effective way to increase the occupancy rate, which can reduce congestion [3, 14]. It is estimated that ridesharing to work in Dublin, Ireland, can reduce 12,674 tons of CO₂ emissions per year [8], and taxi-ridesharing in Beijing can reduce 120 million liters of gasoline annually [26]. A number of systems for ridesharing services are known, such as Uber, Lyft and DiDi. These systems are also called mobility-on-demand (MoD) systems and can support dynamic ridesharing: ridesharing requests arrive dynamically, and the system provides a service in real-time. In this paper, we study the static ridesharing problem but our algorithms can be applied to dynamic services. A sequence of dynamic ridesharing requests within a time interval can be viewed as a set of static requests and solved by a static ridesharing algorithm [33].

Due to the COVID-19 pandemic, traffic volume has decreased significantly in many areas [11]^{2,3}. The recent study [11] of pandemic impacts on road traffic found that one of the most effective ways to reduce traffic fuel consumption and emissions is indeed by reducing the number of vehicles on the road, and the authors suggested that the policy makers should encourage ridesharing to reduce the number of vehicles on the road after the pandemic. During the pandemic, in addition to reducing emissions, ridesharing may be safer than public transit for colleagues who work at the same place; such a type of ridesharing is also easier for contact tracing. A major lesson from the pandemic is that people should cooperate to protect our living environment, and ridesharing is an effective way to reduce the vehicles on the road, and thus reduce pollution [11, 14]. In this paper, we focus on this goal.

More specifically, we study two minimization problems in the following ridesharing problem: Given a set of trips (requests) in a road network, where each trip consists of an individual, a vehicle of the individual and some requirements, select a subset of trips to deliver the individuals of all trips to their destinations by the vehicles of the selected trips satisfying the requirements. We call the individual of a selected trip a *driver* and an individual other than a driver a *passenger*. The parameters specifying the requirements of a trip include the source and destination of the trip, the vehicle capacity (number of seats to serve passengers), the preferred paths of individual when selected as a driver, the detour distance and number of stops the driver willing to make to serve passengers, and time constraints such as departure/arrival times. Our optimization goals are to minimize the number of selected vehicles (or equivalently number of drivers) and to minimize the total travel distance of selected vehicles (drivers) to serve all trips.

In general, the ridesharing problem is a generalization of the vehicle routing problem

²Google COVID-19 Community Mobility Reports, 2021-02. <https://google.com/covid19/mobility>

³Geotab Blog, 2021-02. <https://geotab.com/blog/congestion-and-commercial-traffic>

(VRP) and Dial-A-Ride problem (DARP) [29], and thus, it is NP-hard. Mixed Integer Programming (MIP) formulation combined with exact methods or heuristics to solve the MIP formulation have been used to solve the ridesharing problem [4, 20, 22]. The MIP based approach is time consuming and not practical for the ridesharing of large scale. Most previous works use (meta)heuristics or study the simplified ridesharing for large instances [1, 23, 31, 33]. There are two types of optimization goals in ridesharing: *operational objectives* and *quality-related objectives* [29]. The operational objectives are to optimize system-wide optimization performances such as maximizing the number of matched/served trips and minimizing the total travel distance/time of all vehicles. The quality-related objectives are to improve the satisfactions of individuals (drivers/passengers), such as minimizing the waiting time of each individual passenger and maximizing the cost saving of the drivers/passengers. This is done in first-come first-serve manner for ridesharing service users and is usually achieved by agent-based or decentralized approach by simulating the interaction between passenger ride-requests and driver ride-services (e.g. [5, 13, 30]). The decentralized approach may be good for providing service to users, but it lacks the ability to achieve operational objectives which optimize system performance. The centralized approach, using MIP or heuristics, can achieve system-wide optimization goals as a whole (approximately for large instances). A number of variants and mathematical formulations of the ridesharing problem are derived from DARP and a review on DARP can be found in [28]. Readers may refer to [2, 14, 29] for literature surveys and reviews on the ridesharing problem.

Previous works mainly focus on empirical studies of the ridesharing problem. Recently, a model for analyzing the relations between the computational complexity of the ridesharing problem and its parameters was introduced by Gu. et al [16]; an algorithmic analysis for the simplified ridesharing problem with parameters of source, destination, vehicle capacity, detour distance limit and preferred paths only. It is shown in [16] that if one of the following conditions is not satisfied then both minimizing the number of drivers and minimizing the total travel distance of the drivers are NP-hard:

- (1) All trips have the same destination or all trips have the same source.
- (2) Detour is not allowed for every trip (zero detour condition).
- (3) Each trip has a unique preferred path (fixed path condition).

When all of Conditions (1)-(3) are satisfied, the following exact algorithms are given in [17]: $O(M + l^3)$ time dynamic programming algorithms for both minimization problems, where M is the size of road map and l is the number of trips, and an $O(M + l \cdot \log l)$ time exact algorithm for minimizing the number of drivers.

Kuteil and Rawitz [24] studied the maximum carpool matching problem (MCMP), which is closely related to the ridesharing problem. An instance of MCMP consists of a directed graph $H(V, E)$, where the vertices of V represent the individuals and an arc $(u, v) \in E$ denotes v can serve u . Every $v \in V$ is flexible to be a driver or passenger. The goal of MCMP is to select a set of drivers $S \subseteq V$ to serve all trips of V such that the number of passengers is maximized; MCMP is NP-hard [19]. Approximation algorithms are proposed in [24], and these algorithms can be modified into $\frac{K+2}{2}$ -approximation algorithms for minimizing the number of drivers in the ridesharing problem, where K is the largest capacity of all vehicles.

In this paper, we extend the computational complexity analysis of the simplified ridesharing problem to more generalized problems with three additional parameters: the number of stops a driver willing to make to serve passengers, departure time and arrival time of each trip. Two more conditions related to the three parameters are considered:

- (4) Each driver is willing to stop as many times as its vehicle capacity to pick-up passengers.
- (5) All trips have the same earliest departure time and same latest arrival time.

We call Condition (4) the *stop constraint condition* and Condition (5) the *time constraint condition*. Our results in this paper are:

1. We prove that it is NP-hard to approximate within a constant factor for each problem of minimizing the number of drivers and minimizing the total travel distance of drivers if stop constraint or time constraint condition is not satisfied.
2. We further show that it is NP-hard to approximate within a constant factor for each problem of minimizing the number of drivers and minimizing the total travel distance of drivers if Condition (2) (zero detour) or Condition (3) (fixed path) is not satisfied.
3. We give two $\frac{K+2}{2}$ -approximation algorithms for minimizing the number of drivers when the input instances satisfy Conditions (1-3) and (5), where K is the largest capacity of all vehicles. For a ridesharing instance of a road network with size M and l trips, our first algorithm, which is a modification of an approximation algorithm (StarImprove) for the MCMP problem in [24], runs in $O(M + K \cdot l^3)$ time. Our second algorithm is more practical and runs in $O(M + l^2)$ time.

Application of ridesharing In practice, our algorithms may apply to the following described scenario: A ridesharing scenario in regular school commute may be represented by an instance satisfying Conditions (1)-(3) and (5). In the morning, many students and staffs go to the same university/college campus (Condition (1), same destination) around the same time (Condition (5)). Each driver always wants to use a fixed path (usually the fastest route)

from home to school (Condition (3)) and does not want to detour (Condition (2)) because time constraints may be tight and traffic can be unpredictable during the peak hours. Then in the afternoon, staffs and students leave from the same school (Condition (1), same source) around the same time; from the similar reasons, drivers may want to use a fixed path from school to home and do not want to detour. Depending on the time constraints, a driver may want to only stop once or twice to pick-up passengers such that the driver's travel duration is not prolonged; on the other hand, if a driver wants to stop many times, Condition (4) is satisfied. There are studies focus on ridesharing for university commute, such as [7, 10, 12]. The work commute scenario is similar to school commute, except the destinations may be scattered around an office area. In this case and a more dynamic setting for ridesharing, one can apply our algorithms by grouping users together who satisfy (or nearly satisfy) Conditions (1-3) and (5). This grouping technique has been applied to solving the traveling salesman problem with time windows problem (TSPTW) [6], which may be useful in minimizing the total travel distance of drivers for the ridesharing problem.

Other studies have shown the possible potential of ridesharing involving autonomous vehicles and (autonomous) taxis [4, 25]. It can further benefit the use of autonomous vehicles by computing solution with minimum number of vehicles or minimum total travel distance of vehicles. Another interesting application is multimodal transportation with ridesharing (integrating public and private transportation). This area of research has gained some attraction recently (e.g. [21, 27, 35]). It may be possible to apply our algorithms to this area to satisfy public transportation users demand.

The rest of the paper is organized as follows. We give in Section 2 the preliminaries of the paper. We prove the inapproximability results for stop constraint condition and time constraint condition in Sections 3 and 4, respectively. The inapproximability results for Conditions (2) and (3) are given in Section 5. We modify an approximation algorithm for the MCMP problem into an approximation algorithm for minimizing the number of drivers in Section 6 and present a more practical approximation algorithm for the same minimization problem in Section 7. The final section concludes the paper.

2 Preliminaries

A (undirected) graph G consists of a set $V(G)$ of vertices and a set $E(G)$ of edges, where each edge $\{u, v\}$ of $E(G)$ is a (unordered) pair of vertices in $V(G)$. A digraph H consists of a set $V(H)$ of vertices and a set $E(H)$ of arcs, where each arc (u, v) of $E(H)$ is an ordered pair of vertices in $V(H)$. A graph G (digraph H) is weighted if every edge of G (arc of H) is assigned a real number as the edge length. A *path* between vertex v_0 and vertex v_k in graph G is a sequence e_1, \dots, e_k of edges, where $e_i = \{v_{i-1}, v_i\} \in E(G)$ for $1 \leq i \leq k$ and $v_i \neq v_j$ for

$i \neq j$ and $0 \leq i, j \leq k$. A path from vertex v_0 to vertex v_k in a digraph H is defined similarly with each $e_i = (v_{i-1}, v_i)$ an arc in H . The *length* of a path P is the sum of the lengths of edges (arcs) in P . For simplicity, we express a road network by a weighted undirected graph $G(V, E)$ with non-negative edge length: $V(G)$ is the set of locations in the network, an edge $\{u, v\}$ represents the road segment between u and v .

In the ridesharing problem, we assume that the individual of every trip can be assigned as a driver or passenger. In addition to a vehicle and individual, each trip has a source, a destination, a capacity of the vehicle, a set of preferred (optional) paths (e.g., shortest paths) to reach the destination, a limit (optional) on the detour distance/time from the preferred path to serve other individuals, a limit (optional) on the number of stops a driver wants to make to pick-up passengers, an earliest departure time, and a latest arrival time. Each trip in the ridesharing problem is expressed by an integer label i and specified by parameters $(s_i, t_i, n_i, d_i, \mathcal{P}_i, \delta_i, \alpha_i, \beta_i)$, which are defined in Table 1.

Parameter	Definition
s_i	The source (start location) of i (a vertex in G)
t_i	The destination of i (a vertex in G)
n_i	The number of seats (capacity) of i available for passengers
d_i	The detour distance limit i willing to make for offering services
\mathcal{P}_i	The set of preferred paths of i from s_i to t_i in G
δ_i	The maximum number of stops i willing to make to pick-up passengers
α_i	The earliest departure time of i
β_i	The latest arrival time of i

Table 1: Parameters for a trip i .

When the individual of trip i delivers (using i 's vehicle) the individual of a trip j , we say trip i *serves* trip j and call i a *driver* and j a *passenger*. The *serve relation* between a driver i and a passenger j is defined as follows. A trip i can serve i itself and can serve a trip $j \neq i$ if i and j can arrive at their destinations by time β_i and β_j respectively such that j is a passenger of i , the detour of i is at most d_i , and the number of stops i has to make to serve j is at most δ_i . When a trip i *can serve* another trip j , it means that i - j is a feasible assignment of a driver-passenger pair. We extend this notion to a set $\sigma(i)$ of passenger trips that can be served by a driver i ($i \in \sigma(i)$). A driver i can serve all trips of $\sigma(i)$ if the total detour of i is at most d_i , the number of stops i have to make to pick-up $\sigma(i)$ is at most δ_i , and every $j \in \sigma(i)$ arrives at t_j before β_j . At any specific time point, a trip i can serve at most $n_i + 1$ trips. If trip i serves some trips after serving some other trips (known as *re-take passengers* in previous studies), trip i may serve more than $n_i + 1$ trips. In this paper, we

study the ridesharing problem in which no re-taking passenger is allowed. A serve relation is *transitive* if i can serve j and j can serve k imply i can serve k . Let (G, R) be an instance of the ridesharing problem, where G is a road network (weighted graph) and $R = \{1, \dots, l\}$ is a set of trips. (S, σ) , where $S \subseteq R$ is a set of trips assigned as drivers and σ is a mapping $S \rightarrow 2^R$, is a partial solution to (G, R) if

- for each $i \in S$, i can serve $\sigma(i)$,
- for each pair $i, j \in S$ with $i \neq j$, $\sigma(i) \cap \sigma(j) = \emptyset$, and
- $\sigma(S) = \cup_{i \in S} \sigma(i) \subseteq R$.

When $\sigma(S) = R$, (S, σ) is called a solution of (G, R) . For a (partial) solution (S, σ) we sometimes simply call S a (partial) solution when σ is clear from the context or not related to the discussion.

We consider the problem of minimizing $|S|$ (the number of drivers) and the problem of minimizing the total travel distance of the drivers in S . To investigate the relations between the computational complexity and problem parameters, Gu et al. [16] introduced the simplified minimization (ridesharing) problems with parameters $(s_i, t_i, n_i, d_i, \mathcal{P}_i)$ only and the following conditions:

- (1) All trips have the same destination or all trips have the same source, that is, $t_i = D$ for every $i \in R$ or $s_i = \chi$ for every $i \in R$.
- (2) Zero detour: each trip can only serve others on his/her preferred path, that is, $d_i = 0$ for every $i \in R$.
- (3) Fixed path: \mathcal{P}_i has a unique preferred path P_i .

It is shown in [16] that if any one of Conditions (1), (2) and (3) is not satisfied, both minimization problems are NP-hard. Polynomial time exact algorithms are given in [17] for the simplified minimization problems if all of Conditions (1-3) and transitive serve relation are satisfied. In this paper, we study more generalized minimization problems with all parameters in Table 1 considered. To analyze the computational complexity of the more generalized minimization problems, we introduce two more conditions:

- (4) The number of stops each driver is willing to make to pick-up passengers is at least its capacity, that is, $\delta_i \geq n_i$ for every $i \in R$ (stop constraint).
- (5) All trips have the same earliest departure time and same latest arrival time, that is, for every $i \in R$, $\alpha_i = \alpha$ and $\beta_i = \beta$ for some $\alpha < \beta$ (time constraint).

The polynomial-time exact algorithms in [17] can still apply to any ridesharing instance when all of Conditions (1-5) and transitive serve relation are satisfied.

3 Inapproximabilities for stop constraint condition

We first show the NP-hardness results for the stop constraint condition, that is, when Conditions (1)-(3) and (5) are satisfied but Condition (4) is not. When Condition (1) is satisfied, we assume all trips have the same destination (since it is symmetric to prove the case that all trips have the same source). If all trips have distinct sources, one can solve both minimization problems by using the polynomial-time exact algorithms in [17]: when Conditions (1-3) are satisfied and each trip has a distinct source s_i , each trip is represented by a distinct vertex i in the serve relation graph in [17]. Each time a driver i serves a trip j , i must stop at $s_j \neq s_i$ to pick-up j . When Condition (4) is not satisfied ($\delta_i < n_i$), i can serve at most δ_i passengers. Therefore, we can set the capacity n_i to $\min\{n_i, \delta_i\}$ and apply the exact algorithms to solve the minimization problems. In what follows, we assume trips have arbitrary sources (multiple trips may have a same source).

3.1 Both minimization problems are NP-hard

We first prove both minimization problems are NP-hard. These proofs will provide a base for proving the inapproximabilities for both minimization problems. The NP-hardness proofs are a reduction from the 3-partition problem. The decision problem of 3-partition is that given a set $A = \{a_1, a_2, \dots, a_{3r}\}$ of $3r$ positive integers, where $r \geq 2$, $\sum_{i=1}^{3r} a_i = rM$ and $M/4 < a_i < M/2$, whether A can be partitioned into r disjoint subsets A_1, A_2, \dots, A_r such that each subset has three elements of A and the sum of integers in each subset is M . Given a 3-partition instance $A = \{a_1, \dots, a_{3r}\}$, construct a ridesharing problem instance (G, R_A) as follows (also see Figure 1).

- G is a graph with $V(G) = \{D, u_1, \dots, u_{3r}, v_1, \dots, v_r\}$ and $E(G)$ having edges $\{u_i, v_1\}$ for $1 \leq i \leq 3r$, edges $\{v_i, v_{i+1}\}$ for $1 \leq i \leq r-1$ and $\{v_r, D\}$. Each edge $\{u, v\}$ has weight of 1, representing the travel distance from u to v . It takes $r+1$ units of distance traveling from u_i to D for $1 \leq i \leq 3r$.
- $R_A = \{1, \dots, 3r + rM\}$ has $3r + rM$ trips. Let α and β be valid constants representing time.
 - Each trip i , $1 \leq i \leq 3r$, has source $s_i = u_i$, destination $t_i = D$, $n_i = a_i$, $d_i = 0$, $\delta_i = 1$, $\alpha_i = \alpha$ and $\beta_i = \beta$. Each trip i has a preferred path $\{u_i, v_1\}, \{v_1, v_2\}, \dots, \{v_r, D\}$ in G .
 - Each trip i , $3r + 1 \leq i \leq 3r + rM$, has source $s_i = v_j$, $j = \lceil (i - 3r)/M \rceil$, destination $t_i = D$, $n_i = 0$, $d_i = 0$, $\delta_i = 0$, $\alpha_i = \alpha$, $\beta_i = \beta$ and a unique preferred path $\{v_j, v_{j+1}\}, \{v_{j+1}, v_{j+2}\}, \dots, \{v_r, D\}$ in G .

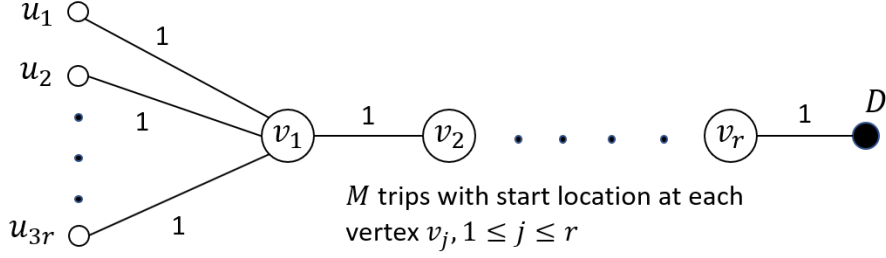


Figure 1: Ridesharing instance based on a given 3-partition problem instance.

Lemma 3.1. *Any solution for the instance (G, R_A) has every trip i , $1 \leq i \leq 3r$, as a driver and total travel distance at least $3r \cdot (r + 1)$.*

Proof. Since Condition (2) is satisfied (detour is not allowed), every trip i , $1 \leq i \leq 3r$, must be a driver in any solution. A solution with exactly $3r$ drivers has total travel distance $3r \cdot (r + 1)$, and any solution with a trip i , $3r + 1 \leq i \leq 3r + rM$, as a driver has total travel distance greater than $3r \cdot (r + 1)$. \square

Lemma 3.2. *Minimizing the number of drivers in the ridesharing problem is NP-hard when Conditions (1-3) and (5) are satisfied, but Condition (4) is not.*

Proof. We prove the lemma by showing that an instance $A = \{a_1, \dots, a_{3r}\}$ of the 3-partition problem has a solution if and only if the ridesharing problem instance (G, R_A) has a solution of $3r$ drivers.

Assume that instance A has a solution A_1, \dots, A_r where the sum of elements in each A_j is M . For each $A_j = \{a_{j_1}, a_{j_2}, a_{j_3}\}$, $1 \leq j \leq r$, assign the three trips whose $n_{j_1} = a_{j_1}$, $n_{j_2} = a_{j_2}$ and $n_{j_3} = a_{j_3}$ as drivers to serve the M trips with sources at vertex v_j . Hence, we have a solution of $3r$ drivers for (G, R_A) .

Assume that (G, R_A) has a solution of $3r$ drivers. By Lemma 3.1, every trip i , $1 \leq i \leq 3r$, is a driver in the solution. Then, each trip j for $3r + 1 \leq j \leq 3r + rM$ must be a passenger in the solution, total of rM passengers. Since $\sum_{1 \leq i \leq 3r} a_i = rM$, each driver i , $1 \leq i \leq 3r$, serves exactly $n_i = a_i$ passengers. Since $a_i < M/2$ for every $a_i \in A$, at least three drivers are required to serve the M passengers with sources at each vertex v_j , $1 \leq j \leq 3r$. Due to $\delta_i = 1$, each driver i , $1 \leq i \leq 3r$, can only serve passengers with the same source. Therefore, the solution of $3r$ drivers has exactly three drivers j_1, j_2, j_3 to serve the M passengers with sources at vertex v_j , implying $a_{j_1} + a_{j_2} + a_{j_3} = M$. Let $A_j = \{a_{j_1}, a_{j_2}, a_{j_3}\}$, $1 \leq j \leq r$, we get a solution for the 3-partition instance.

The size of (G, R_A) is polynomial in r . It takes a polynomial time to convert a solution of (G, R_A) to a solution of the 3-partition instance and vice versa. \square

Lemma 3.3. *Minimizing the total travel distance of drivers in the ridesharing problem is NP-hard when Conditions (1-3) and (5) are satisfied but Condition (4) is not.*

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

Assume that the 3-partition instance has a solution. Then there is a solution of $3r$ drivers for (G, R_A) as shown in the proof of Lemma 3.2. The total travel distance of this solution is d_{sum} .

Assume that (G, R_A) has a solution with total travel distance d_{sum} . Trips i with $1 \leq i \leq 3r$ must be drivers. From this, there is a solution for the 3-partition instance as shown in the proof of Lemma 3.2. \square

3.2 Inapproximability results

Based on the results in Section 3.1, we extend our reduction to further show that it is NP-hard to approximate both minimization problems within a constant factor if Condition (4) is not satisfied. Let (G, R_A) be the ridesharing problem instance constructed based on a given 3-partition instance A as described above for Lemma 3.2. We modify (G, R_A) to get a new ridesharing instance (G, R') as follows. For every trip i , $1 \leq i \leq 3r$, we multiply n_i with rM , that is, $n_i = a_i \cdot rM$, where r and M are given in instance A . There are now rM^2 trips with sources at vertex v_j for $1 \leq j \leq r$, 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 (G, R') is polynomial in r and M . Note that Lemma 3.1 holds for (G, R') and $\sum_{i=1}^{3r} n_i = rM \sum_{i=1}^{3r} a_i = (rM)^2$.

Lemma 3.4. *Let (G, R') be a ridesharing problem instance constructed above from a 3-partition problem instance $A = \{a_1, \dots, a_{3r}\}$. The 3-partition problem instance A has a solution if and only if the ridesharing problem instance (G, R') has a solution (σ, S) s.t. $3r \leq |S| < 3r + rM$, where S is the set of drivers.*

Proof. Assume that instance A has a solution A_1, \dots, A_r where the sum of elements in each A_j is M . For each $A_j = \{a_{j_1}, a_{j_2}, a_{j_3}\}$, $1 \leq j \leq r$, we assign the three trips whose $n_{j_1} = a_{j_1} \cdot rM$, $n_{j_2} = a_{j_2} \cdot rM$ and $n_{j_3} = a_{j_3} \cdot rM$ as drivers to serve the rM^2 trips with sources at vertex v_j . Hence, we have a solution of $3r$ drivers for (G, R') .

Assume that (G, R') has a solution with $3r \leq |S| < 3r + rM$ drivers. Let $R'(1, 3r)$ be the set of trips in R' with labels from 1 to $3r$. By Lemma 3.1, every trip $i \in R'(1, 3r)$ is a driver in S . Since $a_i < M/2$ for every $a_i \in A$, $n_i < rM \cdot M/2$ for every trip $i \in R'(1, 3r)$. From

this, it requires at least three drivers in $R'(1, 3r)$ to serve the rM^2 trips with sources at each vertex v_j , $1 \leq j \leq r$. For every trip $i \in R'(1, 3r)$, i can only serve passengers with the same source due to $\delta_i = 1$. There are two cases: (1) $|S| = 3r$ and (2) $3r < |S| < 3r + rM$.

(1) It follows from the proof of Lemma 3.2 that every three drivers j_1, j_2, j_3 of the $3r$ drivers serve exactly rM^2 passengers with sources at vertex v_j . Then similarly, let $A_j = \{a_{j_1}, a_{j_2}, a_{j_3}\}$, $1 \leq j \leq r$, we get a solution for the 3-partition problem instance.

(2) For every vertex v_j , let X_j be the set of trips with source v_j not served by drivers in $R'(1, 3r)$. Then $0 \leq |X_j| < rM$ due to $|S| < 3r + rM$. For every trip $i \in R'(1, 3r)$, $n_i = a_i \cdot rM$ is a multiple of rM . Hence, the sum of capacity for any trips in $R'(1, 3r)$ is also a multiple of rM , and further, $n_i + n_{i'} = (a_i + a_{i'}) \cdot rM < rM \cdot (M - 1)$ for every $i, i' \in R'(1, 3r)$ because $a_i < M/2$ and $a_{i'} < M/2$. From these and $|X_j| < rM$, there are 3 drivers $j_1, j_2, j_3 \in R'(1, 3r)$ to serve trips with source v_j and $n_{j_1} + n_{j_2} + n_{j_3} \geq rM^2$. Because $n_{j_1} + n_{j_2} + n_{j_3} \geq rM^2$ for every $1 \leq j \leq r$ and $\sum_{1 \leq i \leq 3r} n_i = (rM)^2$, $n_{j_1} + n_{j_2} + n_{j_3} = rM^2$ for every j . Thus, we get a solution with $A_j = \{a_{j_1}, a_{j_2}, a_{j_3}\}$, $1 \leq j \leq r$, for the 3-partition problem.

It takes a polynomial time to convert a solution of (G, R') to a solution of the 3-partition instance and vice versa. \square

Theorem 3.1. *Let (G, R') be the ridesharing instance stated above based on a 3-partition instance. Approximating the minimum number of drivers for (G, R') within a constant factor is NP-hard. This implies that it is NP-hard to approximate the minimum number of drivers within a constant factor for a ridesharing instance when Conditions (1-3) and (5) are satisfied and Condition (4) is not.*

Proof. Assume that there is a polynomial time c -approximation algorithm C for instance (G, R') for any constant $c > 1$. This means that C will output a solution (σ_C, S_C) for (G, R') such that $OPT(R') \leq |S_C| \leq c \cdot OPT(R')$, where $OPT(R')$ is the minimum number of drivers for (G, R') . When the 3-partition instance is a “No” instance, the optimal value for (G, R') is $OPT(R') \geq 3r + rM$ by Lemma 3.4. Hence, algorithm C must output a value $|S_C| \geq 3r + rM$. When the 3-partition instance is a “Yes” instance, the optimal value for (G, R') is $OPT(R') = 3r$. For any constant $c > 1$, taking M such that $c < M/3 + 1$. The output $|S_C|$ from algorithm C on (G, R') is $3r \leq |S_C| \leq 3rc < 3r + rM$ for a 3-partition “Yes” instance. Therefore, by running the c -approximation algorithm C on any ridesharing instance (G, R') 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 $P = NP$. \square

Theorem 3.2. *It is NP-hard to approximate the total travel distance of drivers within any constant factor for a ridesharing instance when Conditions (1-3) and (5) are satisfied and*

Condition (4) is not.

Proof. Let (G, R') be the ridesharing problem instance used in Theorem 3.1, based on a given 3-partition instance $A = \{a_1, \dots, a_{3r}\}$. Let $d(S)$ be the sum of travel distances for a set S of drivers. Let $R'(1, 3r)$ be the set of trips in R' with labels from 1 to $3r$. By Lemma 3.1, all of $R'(1, 3r)$ must be drivers in any solution for (G, R') and $d(R'(1, 3r)) = 3r(r + 1)$. Assume that there is a polynomial time c -approximation algorithm C for the ridesharing problem (G, R') for any constant $c > 1$. This means that C will output a solution (σ_C, S_C) for (G, R') such that $OPT(R') \leq d(S_C) \leq c \cdot OPT(R')$, where $OPT(R')$ is the minimum total travel distance of drivers for (G, R') . By Lemma 3.4, when the 3-partition instance is a “No” instance, the number of drivers in any solution for (G, R') is at least $3r + rM$. All rM trips (of the $3r + rM$) can have source at vertex v_r , so $d(S_C) \geq 3r(r + 1) + rM$. When the 3-partition instance is a “Yes” instance, the optimal value for (G, R') is $OPT(R') = 3r(r + 1)$. For any constant $c > 1$, taking M and r such that $c < \frac{M}{3(r+1)} + 1$. The output $d(S_C)$ from algorithm C on (G, R') is $3r(r + 1) \leq d(S_C) \leq 3r(r + 1)c < 3r(r + 1) + rM$ for a 3-partition “Yes” instance. Therefore, by running the c -approximation algorithm C on any ridesharing instance (G, R') 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 $P = NP$. \square

4 Inapproximabilities for time constraint condition

Assume that Conditions (1-4) are satisfied but Condition (5) is not, that is, trips can have arbitrary departure time and arrival time. Recall that we assume all trips have the same destination when Condition (1) is satisfied (the same reduction with simple modifications can also be applied to all trips have the same source).

4.1 NP-hardness results

We first show that both minimization problems are NP-hard, and these proofs serve as part of the inapproximability proofs. The NP-hardness proofs are a reduction from 3-partition problem, which is similar to the one used in Lemma 3.2. Given a 3-partition minimization problem instance, construct a ridesharing instance (G, R_A) with G shown in Figure 1. The only differences are the values of α_i , β_i and δ_i .

- For trips i , $1 \leq i \leq 3r$, source $s_i = u_i$, destination $t_i = D$, $n_i = a_i$, $d_i = 0$, $\delta_i = n_i$, $\alpha_i = 0$, $\beta_i = 2r$. Each trip i has a preferred path $\{u_i, v_1\}, \{v_1, v_2\}, \dots, \{v_r, D\}$ in G .

- For trips i , $3r + 1 \leq i \leq 3r + rM$, source $s_i = v_j$, $j = \lceil (i - 3r)/M \rceil$, destination $t_i = D$, $n_i = 0$, $d_i = 0$, $\delta_i = 0$. Each trip i has a unique preferred path $\{v_j, v_{j+1}\}, \{v_{j+1}, v_{j+2}\}, \dots, \{v_r, D\}$ in G , $\alpha_i = r$ and $\beta_i = 2r - j + 1$, where $j = \lceil (i - 3r)/M \rceil$.

Note that every trip $i \in R_A$ has the same travel distance from s_i to t_i as previous construction in Section 3. Since they have the same construction, Lemma 3.1 also holds for this ridesharing instance (G, R_A) .

Lemma 4.1. *In any solution for the instance (G, R_A) , all trips served by a driver $i \in R_A$ (other than i itself), must have the same source v_j , for some $j \in [1, \dots, 3r]$.*

Proof. By Lemma 3.1, every trip i , $1 \leq i \leq 3r$, is a driver in any solution. Thus, only trips with source v_j , $1 \leq j \leq 3r$, can be passengers. Let j be a trip with source v_j . The travel time from v_j to D is $r - j + 1$. Since $\beta_j = 2r - j + 1$, j must be picked-up no later than time r . Otherwise, j cannot arrive at $t_j = D$ by time β_j . From this and the fact that $\alpha_j = r$, j must be picked-up at time r exactly. Suppose that driver i serves trip j . The travel time from s_i to $s_j = v_j$ is $j \leq r$. i can arrive at D (after delivering j) no later than time $2r = \beta_i$.

Let j_1 and j_2 be two trips with $s_{j_1} = v_{j_1}$, $s_{j_2} = v_{j_2}$ and $j_1 < j_2$. Then any trip i with $1 \leq i \leq 3r$ can serve only one of j_1 and j_2 due to the following reasons. Suppose i picks-up j_1 first. By the time i reaches v_{j_2} after picking-up j_1 , it will pass time r , and from above, i can no longer serve j_2 . Otherwise, j_2 will not be arrive on time. Suppose i picks-up j_2 first. When i reaches v_{j_1} by going back, it will also pass time r . Hence, i cannot serve j_1 in this case. Therefore, if i decides to serve a trip j with source v_j , the only other trips i can serve must also have source v_j . \square

Lemma 4.1 actual implies that every driver i ($1 \leq i \leq 3r$) in any solution for (G, R_A) will only make at most one stop, effectively making $\delta_i = 1$.

Lemma 4.2. *Minimizing the number of drivers in the ridesharing problem is NP-hard when Conditions (1-4) are satisfied but Condition (5) is not.*

Proof. Assume that the 3-partition instance has a solution A_1, \dots, A_r where the sum of elements in each A_j is M . For each $A_j = \{a_{j_1}, a_{j_2}, a_{j_3}\}$, $1 \leq j \leq r$, we assign the three trips whose $n_{j_1} = a_{j_1}$, $n_{j_2} = a_{j_2}$ and $n_{j_3} = a_{j_3}$ as drivers to serve the M trips with sources at vertex v_j . Hence, we have a solution of $3r$ drivers for (G, R_A) .

Assume that (G, R_A) has a solution of $3r$ drivers. By Lemma 3.1, every trip i , $1 \leq i \leq 3r$, is a driver in the solution. Similarly, each driver i serves exactly $n_i = a_i$ passengers, and at least three drivers are required to serve the M passengers with sources at each vertex v_j , $1 \leq j \leq 3r$. By Lemma 4.1, each driver i , $1 \leq i \leq 3r$, can only serve passengers with

the same source. Therefore, the solution of $3r$ drivers has exactly three drivers j_1, j_2, j_3 to serve the M passengers with sources at the vertex v_j , implying $a_{j_1} + a_{j_2} + a_{j_3} = M$. Let $A_j = \{a_{j_1}, a_{j_2}, a_{j_3}\}$, $1 \leq j \leq r$, we get a solution for the 3-partition problem instance.

The size of (G, R_A) is polynomial in r . It takes a polynomial time to convert a solution of (G, R_A) to a solution of the 3-partition instance and vice versa. \square

With a similar argument to that of Lemma 3.3, we have the following lemma.

Lemma 4.3. *Minimizing the total travel distance of drivers in the ridesharing problem is NP-hard when Conditions (1-4) are satisfied but Condition (5) is not.*

4.2 Inapproximability results

It is NP-hard to approximate the minimum number of drivers and total travel distance of drivers within a constant factor for the ridesharing problem when Conditions (1-4) are satisfied but condition (5) is not. The proofs are identical to the inapproximability proof of Theorem 3.1 and Theorem 3.2 respectively for each minimization problem. Let (G, R_A) be the ridesharing problem instance constructed based on a given 3-partition instance as described in Section 4. Construct a ridesharing instance (G, R') from (G, R_A) as described in Section 3.2. Then Lemma 3.4 and Lemma 4.1 can be applied to (G, R') . From this, the analysis of Theorem 3.1 and Theorem 3.2 can be applied to (G, R') , and we have the following theorems.

Theorem 4.1. *It is NP-hard to approximate the minimum number of drivers within any constant factor for a ridesharing instance satisfying Conditions (1-4) but not Condition (5).*

Theorem 4.2. *It is NP-hard to approximate the minimum total travel distance of drivers within any constant factor for a ridesharing instance satisfying Conditions (1-4) but not Condition (5).*

5 Inapproximabilities for Conditions (2) and (3)

It is proved in [16] that the ridesharing minimization problems are NP-hard when all conditions except Condition (2) or Condition (3) are satisfied. In this section, we show that for each case, it is NP-hard to approximate both minimization problems within any constant factor. The proof uses a similar method as described in Section 3.2.

5.1 Inapproximability results for non-zero detour

Recall that the NP-hardness proof for this case is a reduction from the 3-partition problem too [16]. For completeness, we show the construction of the slightly modified (G, R_A) and the inapproximability proof. Given a 3-partition instance $A = \{a_1, \dots, a_{3r}\}$, the ridesharing instance (G, R_A) is constructed as follows (see Figure (2a)):

- G is a graph with $V(G) = \{I, D, u_1, \dots, u_r, v_1, \dots, v_{3r}\}$ and $E(G)$ having edges $\{u_i, I\}$ of weight rM for $1 \leq i \leq r$, edges $\{v_i, I\}$ of weight a_i for $1 \leq i \leq 3r$ and edge $\{I, D\}$ of weight rM .
- $R_A = \{1, \dots, r + 3rrM\}$ has $r + 3rrM$ trips. Let α and β be valid constants representing time.
 - Each trip i , $1 \leq i \leq r$, has source $s_i = u_i$, destination $t_i = D$, $n_i = 3rM$, $d_i = 2M$, $\delta_i = n_i$, $\alpha_i = \alpha$ and $\beta_i = \beta$. Each trip i has a preferred path $\{u_i, I\}, \{I, D\}$ in G .
 - Each trip i , $r + 1 \leq i \leq r + 3rrM$, has source $s_i = v_j$, $j = \lceil (i - r)/rM \rceil$, destination $t_i = D$, $n_i = 0$, $d_i = 0$, $\delta_i = 0$, $\alpha_i = \alpha$, $\beta_i = \beta$ and a unique preferred path $\{v_j, I\}, \{I, D\}$ in G .

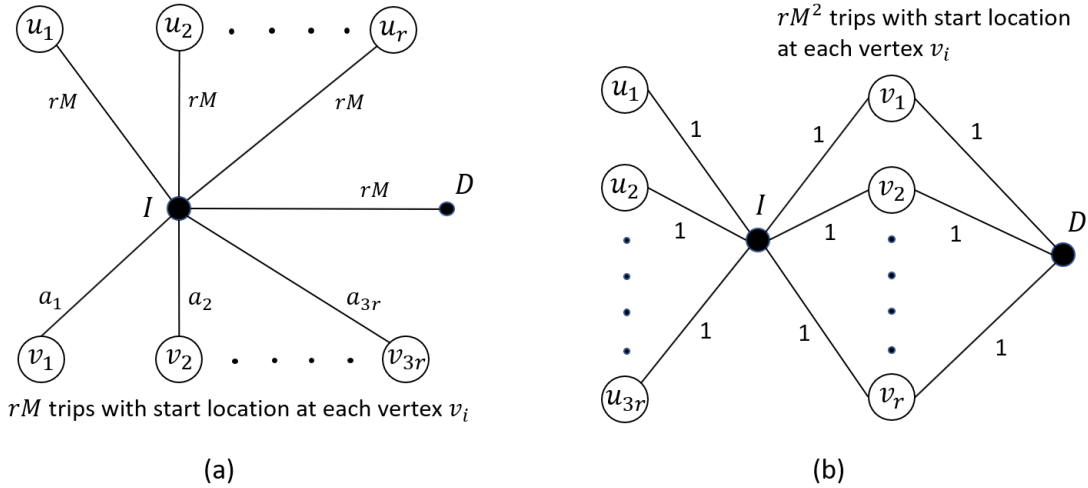


Figure 2: Ridesharing instance based on a given 3-partition problem instance: (a) Conditions (1) and (3-5) are satisfied but not Condition (2); (b) Conditions (1-2) and (4-5) are satisfied but not Condition (3).

First we re-state Lemma 3.1 in [16] as the following lemma.

Lemma 5.1. *Any solution for the instance (G, R_A) has every trip i with $1 \leq i \leq r$ as a driver and total travel distance at least $2rM(r+1)$.*

Proof. Any solution for the instance must have every trip i with $1 \leq i \leq r$ as a driver because of the detour distance limit and the travel distance between the trips $1 \leq i \leq r$. Let $S = \{i \mid 1 \leq i \leq r\}$ be the set of driver. The total travel distance of S is at least $2rM$. For each trip j with source $s_j = v_k$ ($1 \leq k \leq 3r$), the total travel distance of drivers in S and trip j is at least $2rrM + 2a_k$ if j is served by a driver in S , otherwise is at least $2rrM + a_k + rM$. Since $a_k < rM$, the minimum total travel distance of any solution is to have every j with source at v_k , $1 \leq k \leq 3r$, as a passenger served by S with total distance $2rrM + \sum_{1 \leq k \leq 3r} 2a_k = 2rrM + 2rM = 2rM(r+1)$. \square

Lemma 5.2. *Let (G, R_A) be a ridesharing problem instance constructed above from a 3-partition problem instance $A = \{a_1, \dots, a_{3r}\}$. The 3-partition problem instance A has a solution if and only if the ridesharing problem instance (G, R_A) has a solution (σ, S) s.t. $r \leq |S| < r + rM$, where S is the set of drivers.*

Proof. Assume that instance A has a solution A_1, \dots, A_r where the sum of elements in each A_i is M . For each $A_i = \{a_{i_1}, a_{i_2}, a_{i_3}\}$, $1 \leq i \leq r$, we say trips with sources at vertex v_k ($1 \leq k \leq 3r$) correspond to A_i if the edge $\{v_k, I\}$ 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 $3rM$ corresponding trips with sources at three different vertices v_k . Recall that every trip i with source at u_i has detour distance limit $d_i = 2M$ and capacity $n_i = 3rM$. We assign a trip i with source at u_i as a driver to serve the corresponding trips of A_i for $1 \leq i \leq r$. It requires exactly $2M$ detour distance for i to serve the $3rM$ corresponding trips of A_i . Hence, we have a solution of r drivers for (G, R_A) .

Assume that (G, R_A) has a solution with $r \leq |S| < r + rM$ drivers. Let $R(1, r)$ be the set of trips in R_A with labels from 1 to r . By Lemma 5.1, every trip $i \in R(1, r)$ is a driver in S (trips with source at u_i are drivers). For every vertex v_k , let X_k be the set of trips with source at v_k not served by drivers in $R(1, r)$. Then $0 \leq |\cup_{1 \leq k \leq 3r} X_k| < rM$ due to $|S| < r + rM$. From this and there are rM trips with source at each vertex v_k , every driver in $R(1, r)$ must detour to some vertex v_k ($1 \leq k \leq 3r$) to pick-up some passengers. In other words, every vertex v_k for $1 \leq k \leq 3r$ must have been visited by at least one driver in $R(1, r)$. Assume some driver $i \in R(1, r)$ has detour distance less than $2M$ (i detours to at least 1 vertex and at most 3 vertices because $M/4 < a_k < M/2$ for $1 \leq k \leq 3r$). Then from the fact that the sum of elements in A is rM ($\sum_{1 \leq k \leq 3r} a_k = rM$), some driver i' must has detour distance greater than $2M$ so that all vertices can be visited. This is a contradiction to $d_{i'} = 2M$. Hence, the detour distance of every driver in $R(1, r)$ is exactly $2M$. For each driver i , $1 \leq i \leq r$, let A_i

be the subset of the three integers of A corresponding to the detour distance traveled by i (one way). Then A_1, \dots, A_r is a solution for the 3-partition problem instance.

It takes a polynomial time to convert a solution of (G, R_A) to a solution of the 3-partition instance and vice versa. \square

With Lemma 5.2, the analysis of Theorem 3.1 can be applied to (G, R_A) , and we have the following theorem.

Theorem 5.1. *It is NP-hard to approximate the minimum number of drivers within any constant factor for a ridesharing instance satisfying Conditions (1) and (3-5) but not Condition (2).*

Theorem 5.2. *It is NP-hard to approximate the minimum total travel distance of drivers within any constant factor for a ridesharing instance satisfying Conditions (1) and (3-5) but not Condition (2).*

Proof. With a similar argument to that of Theorem 3.2, we have the theorem. \square

5.2 Inapproximability results for no fixed preferred path

The NP-hardness proof for this case is also a reduction from the 3-partition problem [16]. Given a 3-partition instance $A = \{a_1, \dots, a_{3r}\}$, the ridesharing instance (G, R_A) is constructed as follows (see Figure (2b)):

- G is a graph with $V(G) = \{D, u_1, \dots, u_{3r}, v_1, \dots, v_r\}$ and $E(G)$ having edges $\{u_i, I\}$ for $1 \leq i \leq 3r$, edges $\{I, v_i\}$ for $1 \leq i \leq r$ and edges $\{v_i, D\}$ for $1 \leq i \leq r$. Every edge in $E(G)$ has weight of 1.
- $R_A = \{1, \dots, 3r + (rM)^2\}$ has $3r + (rM)^2$ trips. Let α and β be valid constants representing time.
 - Each trip i , $1 \leq i \leq 3r$, has source $s_i = u_i$, destination $t_i = D$, $n_i = a_i \cdot rM$, $d_i = 0$, $\delta_i = n_i$, $\alpha_i = \alpha$ and $\beta_i = \beta$. Each trip i has r preferred paths $\{u_i, I\}, \{I, v_i\}, \{v_i, D\}$ in G , $1 \leq i \leq r$.
 - Each trip i , $3r+1 \leq i \leq (rM)^2$, has source $s_i = v_j$, $j = \lceil (i-3r)/rM^2 \rceil$, destination $t_i = D$, $n_i = 0$, $d_i = 0$, $\delta_i = 0$, $\alpha_i = \alpha$, $\beta_i = \beta$ and a unique preferred path $\{v_j, D\}$ in G .

Lemma 3.2 in [16] also holds for the instance (G, R_A) , which is stated as the following lemma.

Lemma 5.3. *Any solution for the instance (G, R_A) has every trip i with $1 \leq i \leq r$ as a driver and total travel distance at least $9r$.*

Lemma 5.4. *Let (G, R_A) be a ridesharing problem instance constructed above from a 3-partition problem instance $A = \{a_1, \dots, a_{3r}\}$. The 3-partition problem instance A has a solution if and only if the ridesharing problem instance (G, R_A) has a solution (σ, S) s.t. $3r \leq |S| < 3r + rM$, where S is the set of drivers.*

Proof. With Lemma 5.3 and Theorem 3.3 in [16], a similar analysis of Lemma 3.4 can be applied to this lemma. \square

From Lemma 5.4, the analysis of Theorem 3.1 and Theorem 3.2 can be applied to (G, R_A) , and we have the following theorems.

Theorem 5.3. *It is NP-hard to approximate the minimum number of drivers within any constant factor for a ridesharing instance satisfying Conditions (1-2) and (3-4) but not Condition (5).*

Theorem 5.4. *It is NP-hard to approximate the minimum total travel distance of drivers within any constant factor for a ridesharing instance satisfying Conditions (1-2) and (3-4) but not Condition (5).*

6 Approximation algorithms based on MCMP

For short, we call the ridesharing problem with all conditions satisfied except Condition (4) as *ridesharing problem with stop constraint*. Let $K = \max_{i \in R} n_i$ be the largest capacity of all vehicles. Kutiel and Rawitz [24] proposed two $\frac{1}{2}$ -approximation algorithms for the maximum carpool matching problem. We show in this section that the algorithms in [24] can be modified to $\frac{K+2}{2}$ -approximation algorithms for minimizing the number of drivers in the ridesharing problem with stop constraint. Then in the next section, we propose a more practical $\frac{K+2}{2}$ -approximation algorithm for the minimization problem.

An instance of the maximum carpool matching problem (MCMP) consists of a directed graph $H(V, E)$, a capacity function $c : V \rightarrow \mathbb{N}$, and a weight function $w : E \rightarrow \mathbb{R}^+$, where the vertices of V represent the individuals and an arc $(u, v) \in E$ implies v can serve u . We are only interested in the unweighted case, that is, $w(u, v) = 1$ for every $(u, v) \in E$. Every $v \in V$ 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 solution to MCMP is a set \mathcal{S} of vertex-disjoint stars in H . 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 $S_v \in \mathcal{S}$, vertex v has out-degree

of 0 and every leave in P_v has only one out-edge towards v . The center vertex of each star S_v is assigned as a driver and the leaves are assigned as passengers. The set of edges in \mathcal{S} is called a matching M . An edge in M is called a *matched* edge. Notice that $|M|$ equals to the number of passengers. For an arc $e = (u, v)$ in H , vertices u and v are said to be *incident to e* . For a matching M and a set $V' \subseteq V$ of vertices, let $M(V')$ be the set of edges in M incident to V' . The *in-neighbors* of a vertex v is defined as $N^{in}(v) = \{u \mid (u, v) \in E\}$, and the set of arcs entering v is defined as *in-arcs* $E^{in}(v) = \{(u, v) \mid (u, v) \in E\}$. Table 2 lists the basic notation and definition for this section.

Notation	Definition
\mathcal{S}	A set of vertex-disjoint stars in H (solution to MCMP)
S_v and P_v	A Star S_v rooted at center 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 n_v in Table 1)
Matching M	The set of edges in \mathcal{S} , namely $E(\mathcal{S})$
$M(V')$	The set of edges in M incident to a set V' of vertices
$N^{in}(v)$	The set of <i>in-neighbors</i> of v , $N^{in}(v) = \{u \mid (u, v) \in E\}$
$E^{in}(v)$	The set of arcs entering v , <i>in-arcs</i> $E^{in}(v) = \{(u, v) \mid (u, v) \in E\}$
δP_v	The number of stops required for v to pick-up P_v

Table 2: Common notation and definition used in this section.

Two approximation algorithms (StarImprove and EdgeSwap) are presented in [24]; both can achieve $\frac{1}{2}$ -approximation ratio, that is, the number of passengers found by the algorithm is at least half of that for the optimal solution.

EdgeSwap The EdgeSwap algorithm requires the input instance to have a bounded degree graph (or the largest capacity K is bounded by a constant) to have a polynomial running time. The idea of EdgeSwap is to swap i matched edges in M with $i + 1$ edges in $E \setminus M$ for $1 \leq i \leq k$ and k is a constant integer. The running time of EdgeSwap is in the order of $O(|E|^{2k+1})$. EdgeSwap can directly apply to the minimization problem to achieve $\frac{K+2}{2}$ -approximation ratio in $O(l^{2K})$ time, which may not be practical even if K is a small constant.

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

- check if there exists a star S_v with $E(S_v) \cap M = \emptyset$ such that the resulting set of stars

$\mathcal{S} \setminus M(V(S_v)) \cup S_v$ gives a larger matching.

Such a star S_v is called an *improvement* and $|P_v| \leq c(v)$. Given a ridesharing instance (G, R) satisfying all conditions, except Condition (4). The StarImprove algorithm cannot apply to (G, R) directly because the algorithm assumes a driver v can serve any combination of passengers corresponding to vertices adjacent to v up to $c(v)$. This is not the case for (G, R) in general. For example, suppose v can serve u_1 and u_2 with $n_v = 2$ 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 sources ($s_v \neq s_{u_1} \neq s_{u_2}$), this assignment is not valid for (G, R) . Hence, we need to modify StarImprove for computing a star. For a vertex v and star S_v , let $N_{-M}^{in}(v) = \{i \mid i \in N^{in} \setminus V(M)\}$ and δP_v be the number of stops required for v to pick-up P_v . Suppose the in-neighbors $N_{-M}^{in}(v)$ are partitioned into $g_1(v), \dots, g_m(v)$ groups such that trips with same source are grouped together. When stop constraint is considered, finding a star S_v with maximum $|P_v|$ is similar to solving a fractional knapsack instance using a greedy approach as shown in Figure 3. The idea is, in each iteration, to select the largest group of in-neighbors $N_{-M}^{in}(v)$ until the capacity $c(v)$ is reached.

Algorithm 1 Greedy algorithm

- 1: $P_v = \emptyset$; $c = c(v)$; $\delta P_v = 0$;
- 2: **if** \exists a group $g_j(v)$ s.t. $s_u = s_v$ for any $u \in g_j(v)$ **then**
- 3: Select $g_i(v) = \max_{1 \leq i \leq m: s_u = s_v, u \in g_i(v)} \{|g_i(v) \setminus P_v|\}$;
- 4: Let $g'_j(v) \subseteq g_j(v)$ be a maximum subset of $g_j(v)$ such that $|g'_j(v)| \leq c$.
- 5: $P_v = P_v \cup g'_j(v)$; $c = c - |g'_j(v)|$;
- 6: **end if**
- 7: **while** $c > 0$ and $\delta P_v < \delta_v$ **do**
- 8: Select $g_i(v) = \max_{1 \leq i \leq m} \{|g_i(v) \setminus P_v|\}$;
- 9: Let $g'_i(v) \subseteq g_i(v)$ be a maximum subset of $g_i(v)$ such that $|g'_i(v)| \leq c$.
- 10: $P_v = P_v \cup g'_i(v)$; $c = c - |g'_i(v)|$; $\delta P_v = \delta P_v + 1$;
- 11: **end while**
- 12: **return** the star S_v induced by $P_v \cup \{v\}$;

Figure 3: Greedy algorithm for computing S_v .

Lemma 6.1. *Let v be the trip being processed and S_v be the star found by Algorithm 1 w.r.t. current matching M . Then $|P_v| \geq |P'_v|$ for any star S'_v s.t. $P'_v \cap M = \emptyset$.*

Proof. Assume for contradiction, $|P'_v| > |P_v|$ for some star S'_v s.t. $P'_v \cap M = \emptyset$. Since $|P'_v| > |P_v|$, $c(v) > |P_v|$. For any trip $u \in N_{-M}^{in}(v)$, 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 $s_u \neq s_v$; 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, 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, $|P_v \cap g_i(v)| \geq |P'_v \cap g_i(v)|$ 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. \square

Definition 6.1. A star S_v rooted at v is an improvement with respect to matching M if $|P_v| \leq c(v)$, $\delta P_v \leq \delta_v$ and $|S_v| - \sum_{(u,v) \in E(S_v)} |M(u)| > |M(v)|$.

Definition 6.1 is equivalent to the original definition in [24], except the former is for the unweighted case and stop constraint. When an improvement is found, the current matching M is increased by exactly $|S_v| - \sum_{(u,v) \in E(S_v)} |M(u)|$ edges. For a vertex v and a subset $S \subseteq E^{in}(v)$, let $N_S^{in}(v) = \{u \mid (u, v) \in S\}$.

Lemma 6.2. Let M be the current matching and v be a vertex with no improvement. Let $S_v \subseteq E^{in}(v)$ s.t. $|S_v| \leq c(v)$ and $\delta P_v \leq \delta_v$, then $|S_v| \leq |M(v)| + |M(N_{S_v}^{in}(v))|$. Further, if the star S_v found by Algorithm 1 w.r.t. M is not an improvement, then no other S'_v is an improvement.

Proof. When no improvement exists for a vertex v , we get $|S_v| - |M(N_{S_v}^{in}(v))| = |S_v| - \sum_{(u,v) \in S_v} |M(u)| \leq |M(v)|$ by Definition 6.1.

To maximize $|S_v|$, we need to maximize $|S_v| - \sum_{(u,v) \in S_v} |M(u)|$, which can be done by selecting only in-neighbors of v that are not in matching M . This is because for any $(u, v) \in S_v$ s.t. u is incident to a matched edge, $|M(u)| \geq 1$. In other words, including such a vertex u cannot increase $|S_v| - \sum_{(u,v) \in S_v} |M(u)|$. Algorithm 1 considers only in-neighbors $N_{-M}^{in}(v) = \{i \mid i \in N^{in} \setminus V(M)\}$. By Lemma 6.1, $|P_v|$ is maximized among all stars rooted at v w.r.t. M . Hence, lemma holds. \square

Lemma 6.2 is equivalent to Lemma 5 of [24], except the former is for the unweighted case and stop constraint. By Lemma 6.2 and the same argument of Lemma 6 in [24], we have the following lemma.

Lemma 6.3. The modified StarImprove algorithm computes a solution to an instance of ridesharing problem with stop constraint with $\frac{1}{2}$ -approximation.

Theorem 6.1. Let (G, R) be a ridesharing instance satisfying all conditions, except condition (4). Let $|S^*|$ be the minimum number of drivers for (G, R) , $l = |R|$ and $K = \max_{i \in R} n_i$. Then,

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

Proof. First, we need to construct a directed graph H_R to represent the serve relation of the trips in R as described in [17], which takes $O(M)$ time. Then reverse the direction of all arcs in H_R , 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 above. The rest of the proof is for the second bullet point.

By Lemma 6.3, the modified *StarImprove* algorithm finds a solution to (G, R) with at least $(l - |S^*|)/2$ passengers, and hence, at most $|S| \leq l - (l - |S^*|)/2 = (l + |S^*|)/2$ drivers. There are $l - |S^*|$ passengers in the optimal solution, implying $|S^*| \geq (l - |S^*|)/K = l/(K+1)$, so $l \leq (K+1)|S^*|$. Therefore,

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

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 $|M|$ exists, where M is the current matching. In total, the for-loop takes $O(l^2)$ time. Then for the modified *StarImprove*, it takes $O(K \cdot l^2)$ time; $O(K \cdot l)$ time for computing S_v if it exists. 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 $O(M + K \cdot l^3)$. \square

7 A more practical new approximation algorithm

In this section, we present our new approximation algorithm for the ridesharing problem with stop constraint; our algorithm has a better running time than the approximation algorithms based on MCMP. 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 . In general, if each trip has a unique preferred path and trip i can serve trip j implies j 's preferred path is a subpath of i 's preferred path, then the serve relation is transitive. For the scenarios we described in the paragraph Application of ridesharing in Section 1, each driver always wants to use a unique preferred path P . In most cases, such a path P is a shortest path. For school commute, the

length of P is usually not too long. There is a high chance that the shortest path of any two points within P is a subpath of P . In practice, drivers may not even specify a preferred path, so a ridesharing instance satisfying the transitive serve relation may consist of trips with the preferred paths computed by a coordinator, or the road network has a unique shortest path between every pair of nodes and each trip uses the shortest path from the source to destination as the preferred path.

7.1 New approximation algorithm

Given a ridesharing instance (G, R) , we construct a directed meta graph $\Gamma(V, E)$ to express the serve relation, where $V(\Gamma)$ represents the start locations of all trips in (G, R) . Each node μ of $V(\Gamma)$ contains all trips with the same start location μ . There is an arc (μ, ν) in $E(\Gamma)$ if a trip in μ can serve a trip in ν . Since Conditions (1-3) and (5) 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 arc (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 graph Γ may contain a number of connected components. However, a trip in a node μ from one component cannot serve a trip in ν from another component and vice versa. Hence, the solution for a component is independent from another component in Γ . In what follows, we assume Γ is a single connected component and use Γ for the simplified meta graph. Notice that Γ is an inverse tree and for every pair of nodes μ and ν in Γ , if there is a path from μ to ν then μ can serve ν . We label the nodes of Γ as $V(\Gamma) = \{\mu_p, \mu_{p-1}, \dots, \mu_1\}$, where $p = |V(\Gamma)|$, in such a way that for every arc (μ_b, μ_a) of Γ , $b > a$, and we say μ_b has a larger label than μ_a . The labeling is done by the procedure in [17] (see Figure 4). Figure 5 shows an example of a graph $\Gamma(V, E)$. Each node in Γ without an incoming arc is called an *origin*, and μ_1 is the unique sink. For a node μ in $V(\Gamma)$, the set of trips contained in node μ is denoted by $R(\mu)$. For a set U of nodes in $V(\Gamma)$, $R(U) = \bigcup_{\mu \in U} R(\mu)$. 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 $i \in R$, the node that contains i is denoted by $\text{node}(i)$, that is, if $i \in R(\mu)$ then $\text{node}(i) = \mu$. Table 3 contains the basic notation and definition for this section.

We divide all trips of R into two sets W and X as follows:

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

For a node μ in Γ , let $X(\mu) = X \cap R(\mu)$ and $W(\mu) = W \cap R(\mu)$. 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

Procedure Label-Inverse-Tree

Input: An inverse tree Γ of l trips and p nodes.

Output: Distinct integer labels μ_p, \dots, μ_1 for nodes in Γ .

begin

let ST be a stack and push the sink of Γ into ST;

$i := p$ and mark every arc in Γ un-visited;

while ST $\neq \emptyset$ **do**

let μ be the node at the top of ST;

if there is an arc (ν, μ) in Γ un-visited **then**

push ν into ST and mark (ν, μ) visited;

else

remove μ from ST; assign μ integer label μ_i ; $i := i - 1$;

endif

endwhile

end.

Figure 4: Procedure for assigning integer labels to nodes in Γ [17].

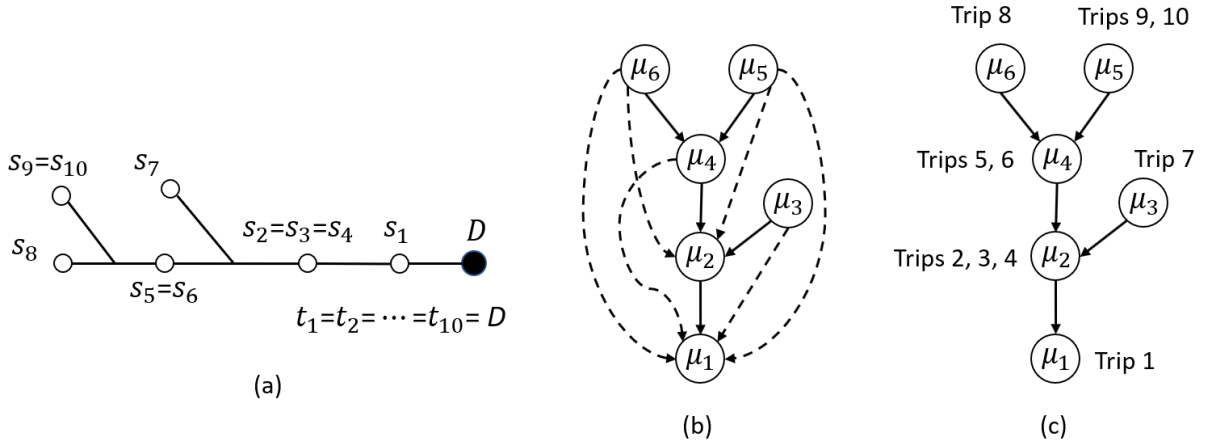


Figure 5: (a) A set R of 10 trips with same destination D in the road network graph G . (b) The directed meta graph expressing the serve relation of these trips with shortcuts in dashed arcs. (c) The simplified meta graph, which is an inverse tree.

Notation	Definition
$\Gamma(V, E)$ and p	A directed graph 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
$R(\mu)$ and $R(U)$	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
$\text{node}(i)$	The node that contains trip i (if $i \in R(\mu)$ then $\text{node}(i) = \mu$)
$\text{free}(i)$	The remaining seats (capacity) of i w.r.t. current solution (S, σ)
$\text{stop}(i)$	The number of stops i has to made to serve all trips assigned to i

Table 3: Basic notation and definition used in this section.

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 $i \in R$ be a driver. Denoted by $\text{free}(i) = n_i - |\sigma(i)| + 1$ is the remaining seats (capacity) of i with respect to solution (S, σ) . Denoted by $\text{stop}(i)$ is the number of stops i has to made in order to serve all trips in $\sigma(i)$ w.r.t. (S, σ) . For the initial solution $(S, \sigma) = (\emptyset, \emptyset)$, $\text{free}(i) = n_i$ and $\text{stop}(i) = 0$ for all $i \in R$. For a driver i and node μ , we define $R(i, \mu, S)$ as the set of $\min\{\text{free}(i), |R(\mu) \setminus \sigma(S)|\}$ trips in $R(\mu) \setminus \sigma(S)$ 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 (Algorithm 2) are described in following, and the pseudo code is given in Figure 6.

(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 $\bar{X} = \{i \in X \cap R(A_\mu^*) \setminus \sigma(S) \mid \text{stop}(i) < \delta_i \vee i \in R(\mu)\}$. The algorithm finds and assigns a trip x as a driver to serve $W(x, \mu, S)$ such that $x = \text{argmin}_{x \in \hat{X}_1 \cup \bar{X} : n_x \geq |W(\mu) \setminus \sigma(S)|} \delta_x - \text{stop}(x)$.
 - If such a trip x does not exist, it means that $n_x < |W(\mu) \setminus \sigma(S)|$ 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 $Z = \{i \in R \setminus \sigma(S) \mid \delta_i = 0\}$ are served. Let $\Gamma(Z) = \{\mu \in \Gamma \mid Z(\mu) = (Z \cap R(\mu)) \neq \emptyset\}$. Each node μ of $\Gamma(Z)$ is processed in the decreasing order of their node labels.

- If $|Z(\mu)| \geq 2$, trip $x = \operatorname{argmax}_{x \in Z(\mu)} n_x$ is assigned as a driver and serves $Z(x, \mu, S)$ consists of trips 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 non-empty node μ of $\Gamma(Z)$ is processed again; note that every μ contains exactly one $z \in Z(\mu)$ now, that is, $|Z(\mu)| = 1$.

- A driver $x \in \hat{X}_2 = \{i \in S(A_\mu^*) \mid \text{free}(i) > 0 \wedge (\text{stop}(i) < \delta_i \vee i \in R(\mu))\}$ 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 $x \in \bar{X} = \{i \in X \cap R(A_\mu^*) \setminus \sigma(S) \mid \text{stop}(i) < \delta_i \vee i \in R(\mu)\}$ 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 $R(\mu_j)$ that have not be served, that is, $R(\mu_j) \not\subseteq \sigma(S)$.

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

7.2 Analysis of new approximation algorithm

A driver in a solution is called a *solo* driver if it serves only itself. Algorithm 2 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 $R(\text{node}(i)) \setminus \{i\}$

Algorithm 2 New approximation algorithm

Input: A ridesharing instance (G, R) and the meta graph Γ (inverse tree) for (G, R) .

Output: A solution (S, σ) for (G, R) with $\frac{K+2}{2}$ -approximation ratio.

```

1:  $(S, \sigma) = (\emptyset, \emptyset)$ . Let  $\Gamma(W) = \{\mu \in V(\Gamma) \mid W(\mu) \setminus \sigma(S) \neq \emptyset\}$ .
2: while  $\Gamma(W) \neq \emptyset$  do /* Beginning of Phase-I */
3:   Compute  $\mu = \operatorname{argmax}_{\mu \in \Gamma(W)} |W(\mu) \setminus \sigma(S)|$ . Let  $\hat{X}_1 = \{i \in S(A_\mu) \mid \text{free}(i) > 0 \wedge \text{stop}(i) < \delta_i\}$ 
4:   and  $\bar{X} = \{i \in X \cap R(A_\mu^*) \setminus \sigma(S) \mid \text{stop}(i) < \delta_i \vee i \in R(\mu)\}$ .
5:   if  $\hat{X}_1 \cup \bar{X} \neq \emptyset$  then
6:     Compute  $x = \operatorname{argmin}_{x \in \hat{X}_1 \cup \bar{X} : n_x \geq |W(\mu) \setminus \sigma(S)|} \delta_x - \text{stop}(x)$ .
7:     if  $x == \emptyset$  then  $x = \operatorname{argmax}_{x \in \hat{X}_1 \cup \bar{X}} \text{free}(x)$  with the smallest  $\delta_x - \text{stop}(x)$ .
8:     if  $x \notin S$  then  $S = S \cup \{x\}$ ;  $\sigma(x) = \{x\}$ ;
9:      $\sigma(x) = \sigma(x) \cup W(x, \mu, S)$ ; update  $\text{free}(x)$  and  $\text{stop}(x)$ ;
10:   else
11:     for each  $w \in W(\mu) \setminus \sigma(S)$ ,  $S = S \cup \{w\}$ ,  $\sigma(w) = \{w\}$ ; update  $\text{free}(w)$ ;
12:   end if
13: end while /* End of Phase-I. Below is Phase-II */
14: Let  $Z = \{i \in R \setminus \sigma(S) \mid \delta_i = 0\}$  and  $\Gamma(Z)$  be the set of nodes containing  $Z$ .
15: for each node  $\mu \in \Gamma(Z)$  in decreasing order of the node labels do
16:   while  $|Z(\mu)| \geq 2$  do
17:     Compute  $x = \operatorname{argmax}_{x \in Z(\mu)} n_x$ .  $S = S \cup \{x\}$ ;  $\sigma(x) = \{x\}$ ;  $\sigma(x) = \sigma(x) \cup Z(x, \mu, S)$  where
18:      $Z(x, \mu, S)$  consists of trips with smallest capacity; update  $\text{free}(x)$  and  $\text{stop}(x)$ ; update  $Z$ .
19:   end while
20: end for
21: for each node  $\mu \in \Gamma(Z)$  in decreasing order of node labels do /* implying  $|Z(\mu)| = 1$  */
22:   Let  $\hat{X}_2 = \{i \in S(A_\mu^*) \mid \text{free}(i) > 0 \wedge (\text{stop}(i) < \delta_i \vee i \in R(\mu))\}$ .
23:   if  $\hat{X}_2 \neq \emptyset$  then Compute  $x = \operatorname{argmax}_{x \in \hat{X}_2} \text{free}(x)$ .
24:   else Let  $\bar{X} = \{i \in X \cap R(A_\mu^*) \setminus \sigma(S) \mid \text{stop}(i) < \delta_i \vee i \in R(\mu)\}$ . Compute  $x = \operatorname{argmax}_{x \in \bar{X}} \delta_x$ .
25:   if  $x \notin S$  then  $S = S \cup \{x\}$ ;  $\sigma(x) = \{x\}$ ;
26:    $\sigma(x) = \sigma(x) \cup Z(x, \mu, S)$ ; update  $\text{free}(x)$  and  $\text{stop}(x)$ ;
27: end for /* End of Phase-II. Below is Phase-III */
28: for each node  $\mu$  from  $\mu_p$  to  $\mu_1$  do
29:   while  $R(\mu) \not\subseteq \sigma(S)$  do
30:     Let  $\hat{X}_2 = \{i \in S(A_\mu^*) \mid \text{free}(i) > 0 \wedge (\text{stop}(i) < \delta_i \vee i \in R(\mu))\}$ .
31:     if  $\hat{X}_2 \neq \emptyset$  then Compute  $x = \operatorname{argmax}_{x \in \hat{X}_2} \text{free}(x)$ .
32:     else Compute  $x = \operatorname{argmax}_{x \in X(\mu) \setminus \sigma(S)} n_x$  (with smallest  $\delta_x$  as a tiebreaker)
33:     if  $x \notin S$  then  $S = S \cup \{x\}$ ;  $\sigma(x) = \{x\}$ ;
34:      $\sigma(x) = \sigma(x) \cup R(x, \mu, S)$ ; update  $\text{free}(x)$  and  $\text{stop}(x)$ ;
35:   end while
36: end for

```

Figure 6: Algorithm for approximating the minimum number of drivers.

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 need to treat Z separately. Let λ be the number of solo drivers in a solution computed by Algorithm 2 and λ^* be the number of solo drivers in any optimal solution. Then there are at most $(|R| - \lambda)/2 + \lambda$ drivers in the solution computed by Algorithm 2 and at least $(|R| - \lambda^*)/(K + 1) + \lambda^*$ drivers in the optimal solution. A central line of the analysis is to show that λ^* is close to λ which guarantees the approximation ratio of Algorithm 2

We now introduce some notation used in our analysis. Denoted by (S, σ) is the complete solution computed by Algorithm 2. Denoted by (S_I, σ_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 $i \in S_I$, $(\sigma_I(i) \setminus \{i\}) \cap (R \setminus W) = \emptyset$. Let $S_I(X) = S_I \cap X$ and $S_I(W) = S_I \cap W = S_I \setminus S_I(X)$. Note that each driver $i \in S_I(X)$ must serve at least one trip from W and $\sigma_I(S_I(X)) \setminus S_I(X) \subseteq W$ if $S_I(X) \neq \emptyset$. Let $W = \{W_1, \dots, W_e\}$ such that each W_j ($1 \leq j \leq e$) is the set of trips served by a driver (or drivers when $W_j \subseteq W$) 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 $i \in S_j$, $\text{free}_j(i) = n_i - |\sigma_j(i)| + 1$ 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 serve all trips in $\sigma_j(i)$ w.r.t. (S_j, σ_j) .

Property 7.1. *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 $W_j \subseteq S_I(W)$. For clarity, we denote each set $W_j \subseteq S_I(W)$ 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 7.2. *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 (G, R) with $|S^*|$ minimized. We first show, in Lemma 7.1, that the number of trips in $S_I(W)$ served by S^* is at most that of the passengers served by $\sigma_I(S_I(X))$. The proof idea is as follows. Let $U \subseteq S^*$ be the set of drivers such that for every $u \in U$, $\sigma^*(u) \cap S_I(W) \neq \emptyset$ and $U \cap W = \emptyset$. We prove that U are also drivers in S_I (specifically, $U \subseteq S_I(X)$) and $\sigma_I(u)$ serves at least $|\sigma^*(u) \cap S_I(W)|$ passengers for each $u \in U$.

Lemma 7.1. *Let (S^*, σ^*) be an optimal solution for (G, R) 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 $|\sigma_I(S_I(X)) \setminus S_I(X)| \geq |\sigma^*(U) \cap S_I(W)|$.*

Proof. Let U_j be the set of drivers in S^* that serve $(W_1 \cup \dots \cup W_j) \setminus S^*(W)$ for $1 \leq j \leq e$. Note that $W = W_1 \cup \dots \cup W_e$ and $U_e = U$. Let $\tilde{W}_{a_1}, \dots, \tilde{W}_{a_d}$ be the sets computed by the algorithm such that $\tilde{W}_{a_b} \subseteq S_I(W)$, $1 \leq b \leq d$, 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) $U'_{a_b} = \{u \in U_{a_b} \mid \sigma^*(u) \cap \tilde{W}_{a_b} \neq \emptyset \text{ and } u \in R(\mu_{a_b})\}$ and (2) $U''_{a_b} = \{u \in U_{a_b} \mid \sigma^*(u) \cap \tilde{W}_{a_b} \neq \emptyset \text{ and } u \in R(A_{\mu_{a_b}})\}$. 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 μ_{a_b} is 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 $\text{free}_{a_{b-1}}(u) = 0$. Otherwise, $\sigma_{a_{b-1}}(u)$ would have served trips from \tilde{W}_{a_b} , a contradiction to the algorithm. From $\text{free}_{a_{b-1}}(u) = 0$, $|\sigma_{a_b}(u) \cap W| \geq |\sigma^*(u) \cap W|$ 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 $\text{free}_{a_b}(u) < n_u$. 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: $U''_{a_b}(0) = \{u \in U''_{a_b} \mid \text{free}_{a_b}(u) = 0\}$ and $U''_{a_b}(1) = \{u \in U''_{a_b} \mid \text{free}_{a_b}(u) \geq 1\}$. 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 $\text{free}_{a_b}(u) = 0$. 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 $\text{free}_{j-1}(u) > \text{free}_{a_b}(u) > 0$, $W_j = W(\mu_{a_j}) \setminus \sigma_{j-1}(S_{j-1})$ must be served by $\sigma_j(u)$, implying $|W_j| < \text{free}_{j-1}(u)$. Since W_j is computed before \tilde{W}_{a_b} , $|\tilde{W}_{a_b}| \leq |W(\mu_{a_b}) \setminus \sigma_{j-1}(S_{j-1})| \leq |W(\mu_{a_j}) \setminus \sigma_{j-1}(S_{j-1})| < \text{free}_{j-1}(u)$, 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 7.3. *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 $\text{free}_{a_c}(u) > 0$, $\text{stop}_{a_c}(u) = \delta_u$. It must be that $\text{free}_{a_b}(u) \geq \text{free}_{a_c}(u) > 0$ and $\text{stop}_{a_b}(u) = \delta_u$ (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 7.2), $\delta_u \geq 2$ 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 $\tilde{W}_{a_1}, \dots, \tilde{W}_{a_d} \subseteq S_I(W)$ such that 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, $\bigcup_{u \in U'_{a_b} \cup U''_{a_b}, 1 \leq b \leq d} \sigma_{a_b}(u) \cap W = \sigma_I(U) \setminus U$ and $\bigcup_{u \in U'_{a_b} \cup U''_{a_b}, 1 \leq b \leq d} \sigma^*(u) \cap \tilde{W}_{a_b} = \sigma^*(U) \cap S_I(W)$. Since $U \subseteq S_I(X)$ (Property 7.3), $|\sigma_I(S_I(X)) \setminus S_I(X)| \geq |\sigma_I(U) \setminus U| \geq |\sigma^*(U) \cap S_I(W)|$. \square

Lemma 7.2. *Let (S^*, σ^*) be any optimal solution for (G, R) . Let F_I be the set of drivers in S^* that serve all trips of $\sigma_I(S_I)$ in (S^*, σ^*) . Then, $|F_I| \geq \frac{2|S_I \cup F_I|}{K+2}$.*

Proof. Three sets U, B_1, B_2 of drivers in S^* are considered, each of which serves a portion of trips of $\sigma_I(S_I)$ in (S^*, σ^*) , and altogether $\sigma^*(U \cup B_1 \cup B_2) \cup S^*(W) \supseteq \sigma_I(S_I)$, where $S^*(W) = S^* \cap W$. Let U be the set of drivers in S^* that serve all trips of $S_I(W) \setminus S^*(W)$ in (S^*, σ^*) . By Property 7.3 and Lemma 7.1, all of U must be drivers in $S_I(X)$ and $|\sigma_I(U) \setminus U| \geq |\sigma^*(U) \cap S_I(W)|$. In this proof, the drivers in S_I are partitioned into three sets: $S_I(W), U$, and $S_X = S_I \setminus (S_I(W) \cup U)$.

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

Consider the remaining set of drivers $S_X = S_I \setminus (S_I(W) \cup U)$. For each driver $x \in S_X$, $\sigma_I(x)$ must serve at least one trip from W , meaning $|\sigma_I(x)| \geq 2$ and $|\sigma_I(S_X)| \geq 2|S_X|$. Let B_2 be the set of drivers in S^* that serve all trips of $\sigma_I(S_X)$ in (S^*, σ^*) . 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 $S \setminus S_I$ s.t. v serves itself only. Hence, the ratio between the number of drivers in S that serve $\sigma_I(S_X) \cup B_2$ 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 $|\sigma^*(v) \cap \sigma_I(S_X)| \leq K$ for each driver in $v \in B_2$, $|B_2| \geq 2|S_X|/K$. Since $\sigma_I(S_X) \cap \sigma_I(U) = \emptyset$ and $\sigma_I(S_X) \cap \sigma_I(W) = \emptyset$, $|(S_I(W) \setminus S^*(W)) \cup (\sigma_I(U) \setminus U) \cup \sigma_I(S_X)| \geq 2|S_I(W) \setminus S^*(W)| + 2|S_X|$. Thus, $|U \cup B_1 \cup B_2| \geq 2(|S_I(W) \setminus S^*(W)| + |S_X|)/K$.

Let $F_I = U \cup B_1 \cup B_2 \cup S^*(W)$, which is the set of drivers in S^* required to serve all of $\sigma_I(S_I)$ in (S^*, σ^*) . Then $|F_I| = |U \cup B_1 \cup B_2| + |S^*(W)| \geq 2(|S_I(W) \setminus S^*(W)| + |S_X|)/K + |S^*(W)|$. Recall that $S_I = S_I(W) \cup U \cup S_X$. The ratio between the number of drivers in S to serve

$\sigma_I(S_I) \cup B_1 \cup B_2$ and F_I is

$$\begin{aligned}
\frac{|S_I \cup B_1 \cup B_2|}{|F_I|} &\leq \frac{|S_I \cup F_I|}{|F_I|} \leq \frac{|S_I(W) \setminus S^*(W)| + |S_X| + |U \cup B_1 \cup B_2 \cup S^*(W)|}{|U \cup B_1 \cup B_2 \cup S^*(W)|} \\
&\leq \frac{|S_I(W) \setminus S^*(W)| + |S_X|}{2(|S_I(W) \setminus S^*(W)| + |S_X|)/K + |S^*(W)|} + 1 \\
&\leq \frac{|S_I(W) \setminus S^*(W)| + |S_X|}{2(|S_I(W) \setminus S^*(W)| + |S_X|)/K} + 1 \\
&= \frac{K}{2} + 1 = \frac{K+2}{2}.
\end{aligned} \tag{7.1}$$

Hence, it requires at least $|F_I| \geq 2|S_I \cup F_I|/(K+2)$ drivers in S^* to serve all trips of $\sigma_I(S_I)$. \square

Notice that Equation (7.1) holds when each driver in $u \in F_I$ serves at most K trips of $\sigma_I(S_I)$, that is, $|\sigma^*(u)| \leq K+1$. Next, we consider the minimum number of drivers in S^* that is required to serve all trips of $\sigma_{II}(S_{II})$ in (S^*, σ^*) , where (S_{II}, σ_{II}) is the partial solution computed at the end of Phase-II. Recall that $Z = \{i \in R \setminus \sigma_I(S_I) \mid \delta_i = 0\}$ and all of Z are served in $\sigma_{II}(S_{II})$.

Lemma 7.3. *Let (S^*, σ^*) be any optimal solution for (G, R) . Let F_{II} be the set of drivers in S^* that serve all trips of $\sigma_{II}(S_{II})$ in (S^*, σ^*) . Then, $|F_{II}| \geq \frac{2|S_{II} \cup F_{II}|}{K+2}$.*

Proof. We consider $F_{II} = F_I \cup C' \cup V' \cup C''$, each of C', V' and C'' is a set of drivers in S^* that serves a portion of trips of $\sigma_{II}(S_{II} \setminus S_I)$ in (S^*, σ^*) . Let $S' = \{x \in S_{II} \setminus S_I \mid |\sigma_{II}(x)| = 1\}$ be the set of solo drivers in $S_{II} \setminus S_I$. Since $S' \subseteq X$, $n_x > 0$ 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^* that serve all of S' in (S^*, σ^*) , 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 $C'_1 \cap R(A_{\text{node}(z)}^*)$ must have been assigned as drivers in S_z . Otherwise, any $v \in C'_1 \cap R(A_{\text{node}(z)}^*)$ would have been assigned as a driver in S_{II} to serve z when $\text{node}(z)$ is processed. Hence, $C'_1 \subseteq S_{II}$, and for every driver $v \in C'_1 \cap R(A_{\text{node}(z)}^*)$, $\text{free}_z(v) = 0$ or $\text{free}_z(v) > 0$ with $\text{stop}_z(v) = \delta_v$. 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 $|C'_1| \geq |S'_1|/K$ 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 $\sigma_{II}(v)$ is either in W or Z . For any $v \in C'_1$ such that $\sigma_{II}(v) \cap W \neq \emptyset$, $v \in S_I$ and v is included in the calculation of Equation (7.1). For any such v (regardless if $\sigma_{II}(v) \cap Z \neq \emptyset$), the ratio

$|S_{\text{II}} \setminus S_{\text{I}}|/|F_{\text{II}}|$ decreases because $v \in S_{\text{I}}$ and $v \in F_{\text{II}}$. To get the approximation ratio for the worst case, we assume that all $C'_1 \subseteq (S_{\text{II}} \setminus S_{\text{I}})$, that is, $\sigma_{\text{II}}(v) \cap W = \emptyset$ and $\sigma_{\text{II}}(v) \cap Z \neq \emptyset$ for all $v \in C'_1$. Let V' be the set of drivers in S^* that serve all of $\bigcup_{v \in C'_1} \sigma_{\text{II}}(v) \cap Z$ in (S^*, σ^*) . By the algorithm (Phase-II part 2 specifically), each passenger $z \in \sigma_{\text{II}}(v) \cap Z$ belongs to a distinct node, implying $|\bigcup_{v \in C'_1} \sigma_{\text{II}}(v) \cap Z| \geq |S'_1|$. From these, each driver in $V' \subseteq S^*$ can serve at most K trips of $\bigcup_{v \in C'_1} \sigma_{\text{II}}(v) \cap Z$, and hence, $|V'| \geq |S'_1|/K$. Since $S'_1 \cap (\bigcup_{v \in C'_1} \sigma_{\text{II}}(v) \cap Z) = \emptyset$, $|V' \cup C'_1| \geq 2|S'_1|/K$. 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,

$$\frac{|S' \cup C' \cup V'|}{|C' \cup V'|} \leq \frac{|C'_0| + |S'_1|}{|C'_0| + |C'_1 \cup V'|} + 1 \leq \frac{|S'_1|}{|C'_1 \cup V'|} + 1 \leq \frac{|S'_1|}{2|S'_1|/K} + 1 = \frac{K+2}{2}. \quad (7.2)$$

Consider the remaining drivers in $S'' = S_{\text{II}} \setminus (S_{\text{I}} \cup S' \cup C')$. Since each driver $x \in S''$ serves at least one passenger, $|\sigma_{\text{II}}(S'')| \geq 2|S''|$. Let $C'' = C''_0 \cup C''_1$ be the set of drivers in S^* that serve all of $\sigma_{\text{II}}(S'')$ in (S^*, σ^*) , where $C''_0 = \{v \in S^* \mid \sigma^*(v) \cap \sigma_{\text{II}}(S'') \neq \emptyset \text{ and } v \in Z\}$ and $C''_1 = \{v \in S^* \mid \sigma_{\text{II}}(S'') \neq \emptyset \text{ and } v \in X \setminus Z\}$. Note that $C''_0 \subseteq \sigma_{\text{II}}(S_{\text{II}})$ by definition. From the algorithm (Phase-II), $\sigma_{\text{II}}(S'') \subseteq Z$ and $\sigma_{\text{II}}(S'') \cap \sigma_{\text{II}}(S') = \emptyset$. In the worst case, each trip $v \in C''$ can be assigned as a driver in S such that v serves itself only. From this, $C''_0 \subseteq \sigma_{\text{II}}(S_{\text{II}})$ and that every $v \in C''$ can serve at most K trips of $\sigma_{\text{II}}(S'')$, the ratio between the number of drivers in S that serve $\sigma_{\text{II}}(S'') \cup C''$ and C'' is

$$\frac{|S'' \cup C''|}{|C''|} \leq \frac{|S''| + |C''|}{|C''|} \leq \frac{|S''|}{2|S''|/K} + 1 \leq \frac{K}{2} + 1 = \frac{K+2}{2}. \quad (7.3)$$

Next, we combine Equations (7.2) and (7.3) with Equation (7.1). Let $F_{\text{II}} = F_{\text{I}} \cup C' \cup V' \cup C''$, which is the set of drivers in S^* required to serve all trips of $\sigma_{\text{II}}(S_{\text{II}})$ in (S^*, σ^*) . Note that $F_{\text{I}} \subseteq S^*$ is the minimum set of drivers that serve all of $\sigma_{\text{I}}(S_{\text{I}}) \subseteq W$ and $(C' \cup V') \subseteq S^*$ is the minimum set of drivers that serve all of $S' \cup (\bigcup_{v \in S'} \sigma_{\text{II}}(v) \cap Z)$, and $C'' \subseteq S^*$ is the minimum set of drivers that serve all of $\sigma_{\text{II}}(S'') \subseteq Z$ such that $\sigma_{\text{II}}(S'') \cap \sigma_{\text{II}}(S') = \emptyset$. The minimum number of drivers in each set of F_{I} , $C' \cup V'$ and C'' is calculated based on each driver $u \in F_{\text{II}}$ serving K trips of $\sigma_{\text{II}}(S_{\text{II}})$, as stated in Equations (7.1), (7.2) and (7.3). Hence,

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

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

$$\begin{aligned}
\frac{|S_{\text{II}} \cup F_{\text{II}}|}{|F_{\text{II}}|} &\leq \frac{|S_{\text{I}} \cup S' \cup S'' \cup F_{\text{I}} \cup C' \cup V' \cup C''|}{|F_{\text{I}} \cup C' \cup V' \cup C''|} \\
&\leq \frac{|S_{\text{I}} \cup F_{\text{I}}| + |S' \cup C' \cup V'| + |S'' \cup C''|}{2(|S_{\text{I}} \cup F_{\text{I}}| + |S' \cup C' \cup V'| + |S'' \cup C''|)/(K+2)} \\
&= \frac{K+2}{2}.
\end{aligned} \tag{7.4}$$

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

Again, Equation (7.4) holds if each driver $u \in F_{\text{II}}$ serves at most K trips of $\sigma_{\text{II}}(S_{\text{II}})$, that is, $|\sigma^*(u)| \leq K+1$. Recall that B_1 and B_2 are subsets of F_{I} defined in the proof of Lemma 7.2, and C' , V' and C'' are subsets of F_{II} defined in the proof of Lemma 7.3. 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, $R(D_{\text{node}(v)}^*) \setminus \{v\} \subseteq \sigma_v(S_v)$ for $\delta_v > 0$ or $R(\text{node}(v)) \setminus \{v\} \subseteq \sigma_v(S_v)$ 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 7.1. Let B_1 and B_2 be the set of trips defined in the proof of Lemma 7.2. Let S' and S'' be the sets of drivers defined in the proof of Lemma 7.3. 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 (7.4) holds.

From Remark 7.1, let $B'_1 \cup B'_2 \subseteq B_1 \cup B_2$ be the trips assigned as drivers in Phase-III. Let $\bar{S} = S \setminus (S_{\text{II}} \cup B'_1 \cup B'_2 \cup C' \cup V' \cup C'')$ be the set of drivers found during Phase-III of the algorithm.

Lemma 7.4. It requires at least $\frac{2|S|}{K+2}$ drivers in S^* to serve all trips of $\sigma(S)$ in (S^*, σ^*) .

Proof. Any trip x in $R \setminus \sigma_{\text{II}}(S_{\text{II}})$ has $n_x > 0$ and $\delta_x > 0$ since all of W and Z are served in $\sigma_{\text{II}}(S_{\text{II}})$. 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 $n_x > 0$ and $\delta_x > 0$, x will serve at least one passenger ($|\sigma(x)| \geq 2$) if there exists an un-assigned trip in $R(D_{\text{node}(x)}^*) \setminus \{x\}$, that is, $R(D_{\text{node}(x)}^*) \setminus \{x\} \not\subseteq \sigma_x(S_x)$. Let $X(1) = \{x \in \bar{S} \mid |\sigma(x)| = 1\}$. For every pair $x, x' \in X(1)$, $x \notin R(D_{\text{node}(x')}^*) \cup R(A_{\text{node}(x')}^*)$. Otherwise, one of them can serve the other. For every $x \in X(1)$, any driver $x' \in \bar{S}(A_{\text{node}(x)}^*) \setminus \{x\}$ must serve at least two trips, where $\bar{S}(A_{\text{node}(x)}^*) = \bar{S} \cap R(A_{\text{node}(x)}^*)$. For any $x \in X(1)$, let Y_x be the set of drivers in S^* that serve

all of $\bigcup_{x' \in \bar{S}(A_{\text{node}(x)}^*)} \sigma(x')$ in (S^*, σ^*) . For a driver $y \in Y_x$, $\sigma^*(y) \setminus \{y\}$ can contain at most K trips of $\bigcup_{x' \in \bar{S}(A_{\text{node}(x)}^*)} \sigma(x')$. If $y \in \bigcup_{x' \in \bar{S}(A_{\text{node}(x)}^*)} \sigma(x')$, y can serve at most $K + 1$ trips of $\bigcup_{x' \in \bar{S}(A_{\text{node}(x)}^*)} \sigma(x')$. Hence, $|Y_x| \geq (2|\bar{S}(A_{\text{node}(x)})| + 1)/(K + 1)$. For any pair $x, x' \in X(1)$ with $x \neq x'$, since drivers in Y_x cannot serve any trip of $\bigcup_{x'' \in \bar{S}(A_{\text{node}(x')}^*)} \sigma(x'')$, it must be that $Y_x \cap Y_{x'} = \emptyset$. Let $Y = \bigcup_{x \in X(1)} Y_x$, $\bar{S}_{X(1)} = \bigcup_{x \in X(1)} \bar{S}(A_{\text{node}(x)}^*)$ and $\sigma(\bar{S}_{X(1)}) = \bigcup_{x \in \bar{S}_{X(1)}} \sigma(x)$. Note that $|Y| \geq |X(1)|$. Then,

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

The above can be rewritten as

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

and hence,

$$\sum_{x \in X(1)} |\bar{S}(A_{\text{node}(x)}^*)| \leq \frac{(K + 1) \cdot |Y| + |X(1)|}{2}.$$

Consider the remaining drivers in $\bar{S}_{\bar{X}(1)} = \bar{S} \setminus \bar{S}_{X(1)}$. Each driver $x \in \bar{S}_{\bar{X}(1)}$ serves at least two trips, implying $|\sigma(\bar{S}_{\bar{X}(1)})| \geq 2|\bar{S}_{\bar{X}(1)}|$. Let Y' be the set of drivers in S^* that serve all of $\bigcup_{x \in \bar{S}_{\bar{X}(1)}} \sigma(x)$ in (S^*, σ^*) . Any driver $x \in \bar{S}_{\bar{X}(1)}$ is not in $\sigma(\bar{S}_{X(1)})$ by the definition of $\bar{S}_{\bar{X}(1)}$, and x is not in $R(D_{\text{node}(x')}^*)$ for any $x' \in X(1)$ since otherwise, x' would have served x . From these, for every $y' \in Y'$, $y' \notin R(D_{\text{node}(x)}^*) \cup R(A_{\text{node}(x)}^*)$ 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 $K + 1$ trips of $\bigcup_{x \in \bar{S}_{\bar{X}(1)}} \sigma(x)$. Hence, $|Y'| \geq 2|\bar{S}_{\bar{X}(1)}|/(K + 1)$, implying $((K + 1) \cdot |Y'|)/2 \geq |\bar{S}_{\bar{X}(1)}|$. Each $y \in Y \cup Y'$ must be in either $\sigma(S \setminus S_{\text{II}})$ or $\sigma(S_{\text{II}})$ 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 (7.4). This means that we only need to consider \bar{S} , and the ratio between $|\bar{S}|$ and $|Y \cup Y'|$ is

$$\begin{aligned} \frac{|\bar{S}|}{|Y \cup Y'|} &= \frac{|\bar{S}_{X(1)}| + |\bar{S}_{\bar{X}(1)}|}{|Y| + |Y'|} \leq \frac{((K + 1) \cdot |Y| + |X(1)|)/2 + ((K + 1) \cdot |Y'|)/2}{|Y| + |Y'|} \\ &= \frac{(K + 1) \cdot (|Y| + |Y'|) + |X(1)|}{2(|Y| + |Y'|)} \\ &= \frac{K + 1}{2} + \frac{|X(1)|}{2(|Y| + |Y'|)} \\ &\leq \frac{K + 1}{2} + \frac{1}{2} = \frac{K + 2}{2}. \end{aligned} \tag{7.5}$$

Finally, we calculate the ratio between S and S^* , where $F_{\text{II}} \cup Y \cup Y' \subseteq S^*$. Recall that $S = \bar{S} \cup S_{\text{II}} \cup F_{\text{II}}$ 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 7.4 for Equation (7.4), the minimum number of drivers in each set of F_{II} , Y and Y' is calculated based on using all capacity K of every driver $u \in F_{\text{II}} \cup Y \cup Y'$. Hence, with Equations (7.4) and (7.5),

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

The ratio between S and S^* is

$$\begin{aligned} \frac{|S|}{|S^*|} &\leq \frac{|S|}{|F_{\text{II}} \cup Y \cup Y'|} \leq \frac{|\bar{S}| + |S_{\text{II}} \cup F_{\text{II}}|}{|F_{\text{II}} \cup Y \cup Y'|} \\ &\leq \frac{|\bar{S}| + |S_{\text{II}} \cup F_{\text{II}}|}{2(|S_{\text{II}} \cup F_{\text{II}}| + |\bar{S}|)/(K+2)} \\ &= \frac{K+2}{2} \end{aligned} \tag{7.6}$$

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

Next, we show that Algorithm 2 always computes a valid solution to any instance of the ridesharing problem with stop constraint, followed by its running time. Let (S', σ') be the partial solution computed by Algorithm 2 for a given time point.

Lemma 7.5. *Let (S, σ) be a solution found by Algorithm 2 after processing all trips in R . Then for each pair $i, j \in S$, $\sigma(i) \cap \sigma(j) = \emptyset$ and $\sigma(S) = R$, implying (S, σ) is indeed a valid solution to the ridesharing instance (G, R) .*

Proof. Phase-I of the algorithm ends until all trips of W are served, that is, $\Gamma(W) = \emptyset$. In each iteration of Phase-I, a node $\mu \in \Gamma(W)$ containing trips of W is chosen w.r.t. (S', σ') . A trip x is selected from $\hat{X}_1 \cup \bar{X}$, where $\hat{X}_1 = \{i \in S'(A_\mu) \mid \text{free}'(i) > 0 \text{ and } \text{stop}'(i) < \delta_i\}$ and $\bar{X} = \{i \in X \cap R(A_\mu^*) \setminus \sigma'(S') \mid \text{stop}'(i) < \delta_i \text{ or } i \in R(\mu)\}$. By the definition of \hat{X}_1 and \bar{X} , x is either a driver or an un-assigned trip that can still serve other trips in $R(D_\mu^*)$. 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 $Z = \{i \in R \setminus \sigma'(S') \mid \delta_i = 0\}$. Since all of W are served before Phase-II starts, $n_i \geq 1$ for every $i \in R \setminus \sigma'(S')$, 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 $Z(x, \text{node}(x), S')$ 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 $\hat{X}_2 = \{i \in S(A_\mu^*) \mid \text{free}(i) > 0 \text{ and } (\text{stop}(i) < \delta_i \text{ or } i \in R(\mu))\}$ 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 $R(\mu_j)$ 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 $R(x, \mu, S')$. The assignment of x is valid as mentioned above. Therefore, Algorithm 2 produces a valid solution after all trips in R are processed. \square

Theorem 7.1. *Given a ridesharing instance (G, R) of size M and l trips satisfying Conditions (1-3) and (5). Algorithm 2 computes a solution (S, σ) for (G, R) such that $|S^*| \leq |S| \leq \frac{K+2}{2}|S^*|$, where (S^*, σ^*) is any optimal solution and $K = \max_{i \in R} n_i$, with running time $O(M + l^2)$.*

Proof. By Lemma 7.4 and Lemma 7.5, Algorithm 2 computes a solution (S, σ) for R with $\frac{K+2}{2}$ -approximation ratio. It takes $O(M)$ time to construct the meta graph $\Gamma(V, E)$ using the preprocessing described in [17]. The labeling of nodes in Γ takes $O(l)$ time. Sorting the trips in a node μ according to their capacity takes $O(K \cdot |R(\mu)|)$ time for each node μ , so in total $O(K \cdot l)$ to sort all trips in R . The total time for the preprocessing is $O(M + K \cdot l)$; we assume $K < l$. 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 $R(\mu)$ 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 $O(l + K)$ time to serve $R(x, \mu, S')$. 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 2 is $O(M + l^2)$. \square

8 Conclusion

We proved that it is NP-hard to approximate with a constant factor each problem of minimizing the number of drivers and minimizing the total travel distance of drivers if one of

Conditions (2)-(5) is not satisfied. Our results together with the results in [16] imply that both minimization problems are NP-hard if one of Conditions (1)-(5) is not satisfied. We also presented $\frac{K+2}{2}$ -approximation algorithms for minimizing number of drivers for problem instances satisfying all conditions except Condition (4), where K is the largest capacity of all vehicles. It is worth developing approximation algorithms for other NP-hard cases; for example, two or more of the five conditions are not satisfied. It is interesting to study applications of the approximation algorithms for other related problems, such as multimodal transportation with ridesharing (integrating public and private transportation).

References

- [1] N. Agatz, A. Erera, M. Savelsbergh and X. Wang. Dynamic ride-sharing: A simulation study in metro Atlanta. *Transp. Research Part B*, 45(9):1450-1464, 2011.
- [2] N. Agatz, A. Erera, M. Savelsbergh, and X. Wang. Optimization for dynamic ride-sharing: A review. *European Journal of Operational Research*, 223:295-303, 2012.
- [3] S. Y. Amirkiaee and N. Evangelopoulos. Why do people rideshare? An experimental study. *Transp. Research Part F*, 55:9-24, 2018.
- [4] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli and D. Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proc. of the National Academy of Sciences (PNAS)*, 114(3):462-467, 2017.
- [5] N.V. Bozdog, M.X. Makkes, A. van Halteren and H. Bal. RideMatcher: Peer-to-peer matching of passengers for efficient ridesharing. 2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pp 263-272, 2018.
- [6] A. Bretin, G. Desaulniers and L.-M. Rousseau. The traveling salesman problem with time windows in postal services. *Journal of the Operational Research Society*, Taylor & Francis, 1-15, 2020.
- [7] M. Bruglieri, D. Ciccarelli, A. Colorni and A. Luè. PoliUniPool: a carpooling system for universities. *Procedia - Social and Behavioral Sciences*, 20:558-567, 2011.
- [8] B. Caulfield. Estimating the environmental benefits of ride-sharing: A case study of Dublin. *Transp. Research Part D*, 14(7):527-531, 2009.
- [9] Center for Sustainable Systems, University of Michigan. Personal Transportation Factsheet. Pub. No. CSS01-07, 2020.

- [10] A. Dehghanmongabadi and Ş. Hoşkara. Challenges of Promoting Sustainable Mobility on University Campuses: The Case of Eastern Mediterranean University. *Sustainability*, 10(12):4842, 2018.
- [11] J. Du, H.A. Rakha, F. Filali, H. Eldardiry. COVID-19 pandemic impacts on traffic system delay, fuel consumption and emissions. In Press: *International Journal of Transportation Science and Technology*, 2020.
- [12] S. Erdoğan, Cinzia Cirillo and J.-M. Tremblay. Ridesharing as a Green Commute Alternative: A Campus Case Study. *International Journal of Sustainable Transportation*, 9(5):377-388, 2015.
- [13] D. Fagnant and K.M. Kockelman. The travel and environmental implications of shared autonomous vehicles, using agent-based model scenarios. *Transp. Research Part C*, 40:1-13, 2014.
- [14] M. Furuhata, M. Dessouky, F. Ordóñez, M. Brunet, X. Wang and S. Koenig. Ridesharing: The state-of-the-art and future directions. *Transp. Research Part B*, 57:28-46, 2013.
- [15] K. Ghoseiri, A. Haghani, and M. Hamedi. Real-time rideshare matching problem. Final Report of UMD-2009-05, U.S. Department of Transportation, 2011.
- [16] Q. Gu, J. L. Liang and G. Zhang. Algorithmic analysis for ridesharing of personal vehicles. *Theoretical Computer Science*, 749:36-46, 2018
- [17] Q. Gu, J. L. Liang and G. Zhang. Efficient algorithms for ridesharing of personal vehicles. *Theoretical Computer Science*, 788:79-94, 2019.
- [18] Q. Gu, J. L. Liang and G. Zhang. Approximate Ridesharing of Personal Vehicles Problem. *Proc. of the 2020 International COCOA. LNCS 12577:3-18*, 2020.
- [19] I. B.-A. Hartman, D. Keren, A. A. Dbai, E. Cohen, and L. Knapen, A.-U.-H. Yasar, and D. Janssens. Theory and practice in large carpooling problems. *Proc. of the 5th International Conf. on ANT*, pages 339-347, 2014.
- [20] W. Herbawi and M. Weber. The ridematching problem with time windows in dynamic ridesharing: A model and a genetic algorithm. *Proc. of ACM Genetic and Evolutionary Computation Conference (GECCO)*, pp 1-8, 2012.
- [21] H. Huang, D. Bucher, J. Kissling, R. Weibel and M. Raubal. Multimodal route planning with public transport and carpooling. *IEEE Transactions on Intelligent Transportation Systems*, pp 1-13, 2019.
- [22] Y. Huang, F. Bastani, R. Jin and X. S. Wang. Large scale real-time ridesharing with service guarantee on road networks. *Proc. of the VLDB Endowment*, 7(14):2017-2028, 2014.

- [23] J. Jung, R. Jayakrishnan, J. Y. Park. Dynamic shared-Taxi dispatch algorithm with hybrid-Simulated annealing. *Computer-Aided Civil and Infrastructure Engineering*, 31(4):275-291, 2016.
- [24] G. Kutiel and D. Rawitz Local Search Algorithms for Maximum Carpool Matching. *Proc. of 25th Annual European Symposium on Algorithms*, pp 55:1-55:14, 2017.
- [25] M. Lokhandwala, H. Cai. Dynamic ride sharing using traditional taxis and shared autonomous taxis: A case study of NYC. *Transp. Research Part C*, 97:45-60, 2018.
- [26] S. Ma, Y. Zheng and O. Wolfson. Real-time city-scale taxi ridesharing. *IEEE Transactions on Knowledge and Data Engineering*. 27(7):1782-1795, 2015.
- [27] T. Ma, S. Rasulkhani, J.Y.J. Chow and S. Klein. A dynamic ridesharing dispatch and idle vehicle repositioning strategy with integrated transit transfers. *Transp. Research Part E*, 128:417-442, 2019.
- [28] Y. Molenbruch, K. Braekers and A. Caris. Typology and literature review for dial-a-ride problems. *Annals of Operations Research*, 259:295-325, 2017.
- [29] A. Mourad, J. Puchinger and C. Chu. A survey of models and algorithms for optimizing shared mobility. *Transportation Research Part B*, 123:323-346, 2019.
- [30] M. Nourinejad and M.J. Roorda. Agent based model for dynamic ridesharing. *Transp. Research Part C*, 64:117-132, 2016.
- [31] P. Santi, G. Resta, M. Szell, S. Sobolevsky, S.H. Strogatz and C. Ratti. Quantifying the benefits of vehicle pooling with shareability networks. *Proceedings of the National Academy of Sciences (PNAS)*, 111(37):13290-13294, 2014.
- [32] 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.
- [33] D.O. Santos and E.C. Xavier. Taxi and ride sharing: A dynamic dial-a-ride problem with money as an incentive. *Expert Systems with Apps.*, 42(19):6728-6737, 2015.
- [34] G. Sierpiński. Changes of the modal split of traffic in Europe. *Archives of Transport System Telematics*, 6(1):45-48, 2013.
- [35] M. Stiglic, N. Agatz, M. Savelsbergh, and M. Gradisar. Enhancing urban mobility: Integrating ride-sharing and public transit. *Computers & Operations Research*, 90:12-21, 2018.