Improved Algorithms for Trip-Vehicle Assignment in Ride-Sharing

Jingyang Zhao[1,2], **Mingyu Xiao**[1*], **Yonghang Su**[1]

[1]University of Electronic Science and Technology of China
[2]Kyung Hee University, Yongin-si, South Korea
jingyangzhao1020@gmail.com, myxiao@uestc.edu.cn, suyh_123@163.com

Abstract

The RIDE-SHARING ASSIGNMENT PROBLEM (AAAI 2018) is a fundamental problem in intelligent transportation systems, urban mobility, and algorithmic decision-making. Given a set of m vehicles with initial locations and n requests ($n \leq mk$), each with a specified origin and destination, the goal is to assign at most k requests to each vehicle and compute corresponding routes that minimize the total travel distance. The algorithmic approach depends on whether $n = mk$ or $n < mk$. In this paper, we present algorithms with provable approximation guarantees for both cases. When $n = mk$, we design a $\min\{\mathcal{O}(\sqrt{k}), \mathcal{O}(\sqrt{\frac{n}{k}})\}$-approximation algorithm, whereas previously the ratio $\mathcal{O}(\sqrt{k})$ was only proved for k being a power of 2. When $n < mk$, we achieve an approximation ratio of $\mathcal{O}(\sqrt{k}\log\max\{n, m\})$, breaking the natural $\mathcal{O}(k)$ barrier. We also conduct experiments to evaluate the empirical performance of our algorithms. The results show that our solutions consistently outperform those produced by the previous existing algorithm.

Code — https://github.com/Serryh1/RSAP

Introduction

As urbanization accelerates and private car ownership becomes increasingly unsustainable in densely populated areas (Clewlow and Mishra 2017), *ride-sharing* has emerged as a promising alternative, with the potential to alleviate traffic congestion, improve travel efficiency, and reduce environmental impact (Ho et al. 2018).

In recent years, many platforms, such as Uber, Lyft, and Didi Chuxing, have enabled passengers to share rides with others traveling in a similar direction (Wang and Yang 2019). The concept has also extended to long-distance travel, with services like BlaBlaCar facilitating intercity and even international carpooling (Shaheen, Stocker, and Mundler 2017). Moreover, ride-sharing has gained traction in corporate transportation and campus shuttle systems (Knopp, Biesinger, and Prandtstetter 2021; AlQuhtani 2022), offering cost-effective and resource-efficient alternatives for group transit.

*Corresponding author

The benefits of ride-sharing are well-documented. Passengers can reduce travel costs, and service providers can increase revenue by combining multiple requests into shared trips (Zeng, Tong, and Chen 2019; Luo and Spieksma 2022). Additionally, shared rides reduce total vehicle mileage, leading to lower fuel consumption and fewer carbon emissions (Coulombel et al. 2019; Cai et al. 2019; Yu et al. 2017).

However, realizing these benefits in practice hinges on the ability to efficiently match passengers with vehicles and plan feasible routes under real-world constraints. This core challenge gives rise to computationally complex optimization problems (Bei and Zhang 2018; Luo and Spieksma 2022).

In this paper, we study the RIDE-SHARING ASSIGNMENT PROBLEM (RSAP) (Bei and Zhang 2018), which formalizes this fundamental decision task: how to assign a set of transportation requests, each with pickup and drop-off locations, to a fleet of vehicles in a way that respects capacity constraints and minimizes total travel cost.

Formally, we are given a set of n requests $\overline{R} = \{(s_i, t_i) : i = 1, \ldots, n\}$, where each request involves transporting an agent (or object) from source s_i to destination t_i. There is also a set of m vehicles $U = \{u_1, \ldots, u_m\}$, each starting at location u_i (used to represent the vehicle itself). The objective is to assign requests to vehicles such that:

- Each vehicle serves at most k requests;
- Each request is served by exactly one vehicle;
- The total distance traveled by all vehicles is minimized.

That is, we seek a partition of requests into m disjoint groups $\mathcal{R} = \{R_u \subseteq \overline{R} : u \in U\}$ with $|R_u| \leq k$ for each $R_u \in \mathcal{R}$, and a feasible route I_u for each vehicle to serve the requests in R_u. A solution is represented by the pair $(\mathcal{R}, \mathcal{I})$, where $\mathcal{I} = \{I_u : u \in U\}$ denotes the set of routes. Vehicles are not required to return to their starting locations, and feasibility requires that $n \leq mk$.

As in prior work (Bei and Zhang 2018; Li, Li, and Lee 2020; Luo and Spieksma 2020, 2022; Luo et al. 2022), we focus on *non-preemptive* routes: once a vehicle picks up an object, it must deliver it directly to its destination without intermediate drop-offs.

We aim to design *approximation algorithms* for the RSAP. For minimization problems, a ρ-approximation algorithm is one that runs in polynomial time and returns a solution with

objective value at most ρ times the optimal, where $\rho \geq 1$ is called the *approximation ratio*.

Related Work

Bei and Zhang (2018) initiated the study of the RSAP, showing that the problem is NP-hard even for $k = 2$. They presented a 2.5-approximation algorithm for the case $k = 2$ and $n = 2m$. Li, Li, and Lee (2020) extended this result by giving a 2.5-approximation for $k = 2$ and $n < 2m$.

Using techniques from an assignment problem (Goossens et al. 2012), Luo and Spieksma (2020, 2022) improved the approximation to 2 for $k = 2$ and $n = 2m$. They also showed that the RSAP is APX-hard for any $k \geq 2$, and gave a simple $(2k - 1)$-approximation algorithm for general k.

Luo et al. (2022) later proposed a matching-based algorithm for the case $n = mk$, achieving an $\mathcal{O}(\sqrt{k})$ approximation when k is a power of 2. However, their analysis is incomplete for general k, and we provide an instance (see details in the full version) where the algorithm fails to achieve any constant-factor approximation for $k = 3$. Thus, for non-power-of-2 values of k, the best known ratio remains $2k - 1$ (Luo and Spieksma 2022).

It is worth noting that ride-sharing has also been studied under different objectives, such as maximizing revenue (Tong et al. 2018a; Jacob and Roet-Green 2021; Zheng, Chen, and Ye 2018), minimizing maximum vehicle distance (Zeng, Tong, and Chen 2019), and designing truthful mechanisms (Shen, Lopes, and Crandall 2016; Kleiner, Nebel, and Ziparo 2011; Protopapas et al. 2024).

Our Contributions

We introduce new approximation algorithms for the RSAP that significantly improve upon previous results for both cases $n = mk$ and $n < mk$ (see Table 1). Our main contributions are summarized below:

1. **Case $n = mk$.** We propose two new algorithms:
   - ALG.1 achieves an approximation ratio of $\mathcal{O}(\sqrt{k})$, which is $\mathcal{O}(\sqrt{n})$ in the worst case.
   - ALG.2 achieves an approximation ratio of $\mathcal{O}(\sqrt{\frac{n}{k}})$.

   Taking the better of the two, we obtain an approximation ratio of $\min\{\mathcal{O}(\sqrt{k}), \mathcal{O}(\sqrt{\frac{n}{k}})\}$, which is at most $\mathcal{O}(\sqrt[4]{n})$ in the worst case. Previously, the best ratio was $\mathcal{O}(\sqrt{k})$ when k is a power of 2, and $2k - 1$ otherwise.
2. **Case $n < mk$.** We present the first non-trivial approximation algorithm, ALG.3, which achieves an approximation ratio of $\mathcal{O}(\sqrt{k} \log \max\{n, m\})$, breaking the natural $\mathcal{O}(k)$ barrier for the first time.
3. **Experimental Results.** We implemented our algorithms for the $n = mk$ case. The results show that our methods outperform the algorithm in (Luo et al. 2022), especially when k is not a power of 2. Due to its complexity, we did not implement the algorithm for the $n < mk$ case, which may require further engineering effort to be practical.

Due to limited space, the proofs of lemmas marked with "*" were omitted, and they can be found in the full version of this paper.

n	Approximation Ratio	Reference
$n = mk$	$2k - 1$	Luo and Spieksma (2022)
	$\mathcal{O}(\sqrt{k})$ under $k = 2, 4, 8, \ldots$	Luo et al. (2022)
	$\min\{\mathcal{O}(\sqrt{k}), \mathcal{O}(\sqrt{\frac{n}{k}})\}$	**This Paper**
$n < mk$	$2k - 1$	Luo and Spieksma (2022)
	$\mathcal{O}(\sqrt{k} \log \max\{n, m\})$	**This Paper**

Table 1: A summary of previous approximation algorithms and our approximation algorithms for the RSAP.

Notation

Let $G = (V = U \cup S \cup T, E, w)$ denote the input complete graph, where $U = \{u_1, ..., u_m\}$ denotes the initial locations of the m vehicles, $S = \{s_1, ..., s_n\}$ (resp., $T = \{t_1, ..., t_n\}$) denotes the origins (resp., destinations) of the n requests, and w is a non-negative weight function defined on the edges in E. For convenience, we assume that V consists of $m+2n$ distinct vertices and that w is a metric satisfying $w(a, a) = 0$, $w(a, b) = w(b, a)$, and the triangle inequality $w(a, b) \leq w(a, c) + w(c, b)$ for all $a, b, c \in V$.

We let $\overline{R} = \{(s_i, t_i)\}_{i=1}^{n}$ denote the set of n requests.

Throughout the paper, we work with multi-edge sets, and the union of any two edge sets is taken with multiplicities. Given a subgraph S of G, we let $V(S)$ (resp., $E(S)$) denote its vertex (resp., edge) set.

A *walk* in a graph, denoted by $W = v_1 v_2 ... v_l$, is a sequence of vertices, where one vertex may appear more than once and each consecutive pair of vertices is connected by an edge. We denote the edge set of W by $E(W) = \{v_1 v_2, ..., v_{l-1} v_l\}$. A *path* in a graph is a walk where no vertex appears more than once, and a path on l distinct vertices is referred to as an l-path, whose *order* is l. Given a walk W, one can skip some vertices on the walk to obtain a new walk W', and such an operation is called *shortcutting*. For example, if $W = v_1 v_3 v_2 v_3 v_4$, by shortcutting the vertices v_3 and v_4, we obtain a new walk $W' = v_1 v_2$. If the edge weight function w satisfies the triangle inequality, we have $w(W') \leq w(W)$.

In a graph $G' = (V', E', w')$, an *l-path partition* $\mathcal{P}$ is a set of $\frac{|V'|}{l}$ vertex-disjoint l-paths that together cover all vertices in V'. Note that there exists an l-path partition in G' only if $|V'|$ is divisible by l. For any vertex $v \in V'$, the *degree* of v in G' is the number of edges in E' incident to v. If G' is connected and the number of odd-degree vertices in V' is either 0 or 2, then a walk W with $E'(W) = E'$ can be found in $\mathcal{O}(|E'|)$ time (Schrijver 2003). Note that when there are exactly two odd-degree vertices, the walk W starts and ends at those two vertices, respectively.

For any integer $t > 0$, we denote the set $\{1, ..., t\}$ by $[t]$. A *route* of a vehicle u that satisfies the requests in $\{(s_i, t_i)\}_{i=1}^{k}$ is a walk starting from u that visits all vertices in $\{s_i, t_i\}_{i=1}^{k}$, such that for each $i \in [k]$, s_i appears before t_i on the walk.

For the RSAP, we fix an optimal solution $(\mathcal{R}^*, \mathcal{I}^*)$, where we let $\mathcal{R}^* = \{R_u^* \mid u \in U\}$ and $\mathcal{I}^* = \{I_u^* \mid u \in U\}$. That is, for each vehicle $u \in U$, R_u^* denotes the set of requests assigned to u, and I^* denotes the corresponding route. We let $\text{OPT} = \sum_{I \in \mathcal{I}^*} w(I)$ denote the total weight of the routes in this optimal solution.

Algorithm 1: ALG.1

Input: An instance $G = (V = U \cup S \cup T, E, w)$.
Output: A feasible solution.

1: Construct a complete graph $H = (V_H, E_H, \hat{w})$, where $V_H = [n]$ and $\hat{w}(i,j) \coloneqq w(s_i, s_j) + \frac{3}{4}w(t_i, t_j)$ for any $i, j \in [n]$.
2: Compute a $4(1 - \frac{1}{k})$-approximate k-path partition $\mathcal{P}$ in H using GW.1 in (Goemans and Williamson 1995).
3: Obtain a set of m pairwise disjoint groups $\mathcal{R}$, w.r.t. $\mathcal{P}$, where each group contains exactly k requests.
4: Construct a complete bipartite graph $B = (U \cup \mathcal{R}, E_B, c)$, where, for any $u \in U$ and $R \in \mathcal{R}$, set the cost of the edge between them as $c(u, R) \coloneqq \min_{(s_i,t_i)\in R}(w(u, s_i) + w(s_i, t_i))$.
5: Compute a minimum cost perfect matching M^* in B using Edmonds' blossom algorithm (Schrijver 2003).
6: **for** each $(u, R_u) \in M^*$ **do**
7: Assign all requests in R_u to vehicle u.
8: Obtain two k-paths $P_u^s = s_{\sigma_1}...s_{\sigma_k}$ and $P_u^t = t_{\sigma_1}...t_{\sigma_k}$ in G, assuming that the k-path w.r.t. R_u in $\mathcal{P}$ is $P_u = \sigma_1...\sigma_k$.
9: Set $E_u \coloneqq E(P_u^s) \cup E(P_u^s) \cup E(P_u^t) \cup \{us_{\sigma_i}, s_{\sigma_i}t_{\sigma_i}\}$, assuming that $c(u, R_u) = w(u, s_{\sigma_i}) + w(s_{\sigma_i}, t_{\sigma_i})$.
10: Let $E_u \coloneqq E_u \cup \{t_{\sigma_1}t_{\sigma_i}\}$ if $w(t_{\sigma_1}, t_{\sigma_i}) \le w(t_{\sigma_i}, t_{\sigma_k})$, and $E_u \coloneqq E_u \cup \{t_{\sigma_i}t_{\sigma_k}\}$ otherwise.
11: Compute a walk W_u in $G[E_u]$ that traverses all edges in E_u (Schrijver 2003).
12: Construct a route I_u by shortcutting W_u.
13: **end for**
14: **return** $(\mathcal{R}, \mathcal{I})$, where $\mathcal{I} = \{I_u \mid u \in U\}$.

Algorithms for Case $n = mk$

In this section, we introduce two new algorithms (ALG.1 and ALG.2) for the case $n = mk$.

The First Algorithm

ALG.1 is based on the well-known $4(1 - \frac{1}{k})$-approximation algorithm, denoted as GW.1, for the *k-path partition problem* (Goemans and Williamson 1995).

The main idea of ALG.1 is to use GW.1 to partition all n requests in $\overline{R}$ into a set of m groups $\mathcal{R}$ with each containing exactly k requests. However, since each request consists of two locations, we cannot directly apply GW.1 to the original graph G to obtain the desired partition. To address this, we construct a complete graph $H = (V_H, E_H, \hat{w})$, where $V_H = [n]$ and $\hat{w}(i,j) \coloneqq w(s_i, s_j) + \frac{3}{4}w(t_i, t_j)$ for any $i, j \in [n]$. Note that $\hat{w}$ remains a metric. Then, we apply GW.1 to H to compute a k-path partition $\mathcal{P}$, where each k-path $P \in \mathcal{P}$ corresponds to a group of k requests. Therefore, based on $\mathcal{P}$, we obtain the desired set of m groups $\mathcal{R}$.

To assign the groups in $\mathcal{R}$ to the m vehicles, we use Edmonds' blossom algorithm (Schrijver 2003) to compute a minimum cost *bipartite* perfect matching M^* between the groups in $\mathcal{R}$ and the vehicles in U, where, for any $u \in U$ and $R \in \mathcal{R}$, the *cost* of the edge between them is defined as $c(u, R) \coloneqq \min_{(s_i,t_i)\in R}(w(u, s_i) + w(s_i, t_i))$.

Finally, for each pair $(u, R_u) \in M^*$, we design a route I_u for vehicle u to serve all requests in R_u, based on the use of the corresponding k-path $P_u \in \mathcal{P}$ in H.

The details of ALG.1 are presented in Algorithm 1.

Lemma 1. *For the RSAP with $n = mk$, ALG.1 computes in $\mathcal{O}(n^3)$ time a solution $(\mathcal{R}, \mathcal{I})$ with $w(\mathcal{I}) \le 2\hat{w}(\mathcal{P}) + c(M^*)$.*

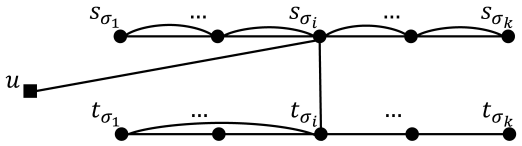

Figure 1: An illustration of the graph $G[E_u]$, where we assume that $w(t_{\sigma_1}, t_{\sigma_i}) \le w(t_{\sigma_i}, t_{\sigma_k})$. In this case, we also know that $W_u = us_{\sigma_i}...s_{\sigma_i}t_{\sigma_i}...t_{\sigma_k}$.

Proof. Since $\mathcal{I} = \{I_u \mid u \in U\}$, to prove $w(\mathcal{I}) \le 2\hat{w}(\mathcal{P}) + c(M^*)$, it suffices to prove that, for each pair $(u, R_u) \in M^*$, the inequality $w(I_u) \le 2\hat{w}(P_u) + c(u, R_u)$ holds, where P_u is the k-path in $\mathcal{P}$ w.r.t. R_u. We begin by analyzing $w(E_u)$.

By Lines 8 and 9, we have $P_u = \sigma_1...\sigma_k$, $P_u^s = s_{\sigma_1}...s_{\sigma_k}$, $P_u^t = t_{\sigma_1}...t_{\sigma_k}$, and $c(u, R_u) = w(u, s_{\sigma_i}) + w(s_{\sigma_i}, t_{\sigma_i})$. By Lines 9 and 10, we have

$$w(E_u) = 2w(P_u^s) + w(P_u^t) + c(u, R_u) + Z, \qquad (1)$$

where $Z = \min\{w(t_{\sigma_1}, t_{\sigma_i}), w(t_{\sigma_i}, t_{\sigma_k})\} \le \frac{1}{2}w(t_{\sigma_1}, t_{\sigma_i}) + \frac{1}{2}w(t_{\sigma_i}, t_{\sigma_k})$. An illustration of $G[E_u]$ is shown in Figure 1.

Since $w(t_{\sigma_1}, t_{\sigma_i}) + w(t_{\sigma_i}, t_{\sigma_k}) \le w(P_u^t)$ by the triangle inequality, we have

$$\begin{aligned} w(E_u) &\le 2w(P_u^s) + \frac{3}{2}w(P_u^t) + c(u, R_u) \\ &= 2\hat{w}(P_u) + c(u, R_u), \end{aligned} \qquad (2)$$

where the first inequality follows from (1), and the equality follows from the definition of $\hat{w}$ (see Line 1).

Next, we prove that I_u is a feasible route for vehicle u and its weight satisfies that $w(I_u) \le w(E_u)$.

Note that $G[E_u]$ is a connected graph with exactly 2 odd-degree vertices: u and t_{σ_k} if $w(t_{\sigma_1}, t_{\sigma_i}) \le w(t_{\sigma_i}, t_{\sigma_k})$, and u and t_{σ_1} otherwise (see Figure 1). Hence, a walk W_u with $w(W_u) = w(E_u)$ can be computed in Line 11. Note that for each $j \in [k]$, s_{σ_j} appears before t_{σ_j} in W_u. Thus, by Line 12, I_u is a feasible route for vehicle u that satisfies all requests in R_u. Moreover, by the triangle inequality, we have $w(I_u) \le w(W_u)$, and then we obtain $w(I_u) \le w(E_u)$.

Since $\mathcal{P}$ is a k-path partition in H, we have $\bigcup_{R\in\mathcal{R}} R = \overline{R}$. Then, since for each $R_u \in \mathcal{R}$, there is a route $I_u \in \mathcal{I}$ for vehicle u that satisfies all requests in R_u, we know $(\mathcal{R}, \mathcal{I})$ is a feasible solution for the RSAP.

For the running time of ALG.1, GW.1 in Line 2 runs in $\mathcal{O}(n^2 \log n)$ time (Goemans and Williamson 1995), the minimum cost perfect matching in Line 5 can be computed in $\mathcal{O}(m^3)$ time (Schrijver 2003), and the walk in Line 11 can be computed in $\mathcal{O}(k)$ time (Schrijver 2003). Therefore, ALG.1 takes $\mathcal{O}(n^3)$ time. □

Remark 1. *In a bipartite graph with n vertices, a minimum cost perfect matching can be computed in $\mathcal{O}(n^{2+\varepsilon})$-time for any $\varepsilon > 0$ (Chen et al. 2022). As $\mathcal{O}(n^2 \log n) \subseteq \mathcal{O}(n^{2+\varepsilon})$, the running time of ALG.1 can be improved to $\mathcal{O}(n^{2+\varepsilon})$.*

To prove the approximation ratio of ALG.1, by Lemma 1, we need to provide upper bounds for $\hat{w}(\mathcal{P})$ and $c(M^*)$.

We first consider $\hat{w}(\mathcal{P})$. We use the following key result.

Lemma 2 (*). *There exists a k-path partition $\mathcal{P}^*$ in the graph H such that $\hat{w}(\mathcal{P}^*) \leq \frac{7(k-1)\sqrt{k-1}}{k} \cdot OPT$.*

Since $\mathcal{P}$ is a $4(1-\frac{1}{k})$-approximate k-path partition in H, by Lemma 2, we obtain the following result.

Lemma 3. *We have $\hat{w}(\mathcal{P}) \leq \frac{28(k-1)^2\sqrt{k-1}}{k^2} \cdot OPT$.*

Then, we consider $c(M^*)$, which can be bounded by using the famous Hall's Marriage theorem (Schrijver 2003).

Lemma 4. *We have $c(M^*) \leq OPT$.*

Proof. First, we use the optimal solution $(\mathcal{R}^*, \mathcal{I}^*)$ and the set of groups $\mathcal{R}$ in ALG.1 to construct a k-regular bipartite graph $B' = (U \cup \mathcal{R}, E_{B'}, c)$.

Initially, $E_{B'} \coloneqq \emptyset$. Recall that for each vehicle $u \in U$, the assigned requests to u in the optimal solution is $R_u^* \in \mathcal{R}^*$. Then, for any vehicle $u \in U$ and any request $(s_i, t_i) \in R_u^*$, we add an edge (u, R) to $E_{B'}$, where $(s_i, t_i) \in R \in \mathcal{R}$. Note that this is always possible since $\bigcup_{R\in\mathcal{R}} R = \overline{R}$ by ALG.1. Moreover, since $w(I_u) \geq w(u, s_i) + w(s_i, t_i)$ by the triangle inequality and $c(u, R) = \min_{(s_i,t_i)\in R}(w(u, s_i) + w(s_i, t_i))$ by definition, we have $w(I_u) \geq c(u, R)$. Therefore, we have

$$\begin{aligned} c(E_{B'}) &= \sum_{u\in U} \sum_{(u,R)\in E_{B'}} c(u, R) \\ &\leq \sum_{u\in U} k \cdot w(I_u) = k \cdot \text{OPT}. \end{aligned} \tag{3}$$

Since exactly k requests are assigned to each vehicle $u \in U$ in the optimal solution, there are exactly k edges incident to u in B'. Moreover, since each $R \in \mathcal{R}$ consists of k requests, there are also exactly k edges incident to R in B'. Hence, B' forms a k-regular bipartite graph.

By Hall's Marriage theorem (Schrijver 2003), the edges in $E_{B'}$ can be decomposed into k perfect matchings in B' (see details in the full version). Thus, by (3), there exists a perfect matching M in B' with

$$c(M) \leq \frac{1}{k} c(E_{B'}) \leq \text{OPT}. \tag{4}$$

Note that M is also a perfect matching in the graph B.

Since M^* is a minimum cost perfect matching in B, by (4), we have $c(M^*) \leq c(M) \leq \text{OPT}$. □

By Lemmas 1, 3, and 4, we obtain the following theorem.

Theorem 1. *For the RSAP with $n = mk$, ALG.1 achieves an approximation ratio of $\frac{56(k-1)^2\sqrt{k-1}}{k^2} + 1$.*

The Second Algorithm

ALG.2 is based on the use of a *constrained spanning forest*, which is a spanning forest in the graph $G[S \cup T]$. Moreover, it requires that for any request $(s_i, t_i) \in \overline{R}$, both s_i and t_i lie in the same tree of the forest; and for any tree of the forest, the number of vertices it contains is divisible by $2k$. We remark that the weight of an optimal constrained spanning forest naturally provides a lower bound for OPT. We will show that a 2-approximate constrained spanning forest $\mathcal{F}$ can be computed in $\mathcal{O}(n^2 \log n)$ time using the primal-dual algorithm, denoted as GW.2, for the *general constrained spanning forest problem* (Goemans and Williamson 1995).

By definition, each tree $F \in \mathcal{F}$ contains a set of requests $R_F \subseteq \overline{R}$. By doubling all edges in $E(F)$ and shortcutting all t_i (resp., s_i), we can obtain a path P_F^s (resp., P_F^t). Here, the paths P_F^s and P_F^t are called *consistent*, i.e., the set of requests whose origins lie in $V(P_F^s)$ is the same as the set of requests whose destinations lie in $V(P_F^t)$.

ALG.2 aims to use the forest $\mathcal{F}$ to find two *consistent k-path partitions* $\mathcal{P}^s$ in $G[S]$ and $\mathcal{P}^t$ in $G[T]$ such that for every path $P^s \in \mathcal{P}^s$, which consists of the origins of a set of k requests R, there is a corresponding path $P^t \in \mathcal{P}^t$ consisting of the destinations of all requests in R, and vice versa. Thus, they correspond to a set of m groups $\mathcal{R}$, where each group contains exactly k requests. Then, similar to ALG.1, ALG.2 computes a minimum cost bipartite perfect matching M^* that assigns the groups in $\mathcal{R}$ to the vehicles in U. For each pair $(u, R_u) \in M^*$, it designs a route I_u for vehicle u to serve all requests in R_u, based on the corresponding k-paths $P_u^s \in \mathcal{P}^s$ and $P_u^t \in \mathcal{P}^t$.

To find the consistent k-path partitions $\mathcal{P}^s$ and $\mathcal{P}^t$, ALG.2 first uses the forest $\mathcal{F}$ to obtain two consistent sets of paths $\mathcal{P}_{\mathcal{F}}^s = \{P_F^s \mid F \in \mathcal{F}\}$ and $\mathcal{P}_{\mathcal{F}}^t = \{P_F^t \mid F \in \mathcal{F}\}$. If every path in $\mathcal{P}_{\mathcal{F}}^s \cup \mathcal{P}_{\mathcal{F}}^t$ has length k, then $\mathcal{P}_{\mathcal{F}}^s$ and $\mathcal{P}_{\mathcal{F}}^t$ are the desired k-path partitions. Otherwise, there must exist two consistent $l'k$-paths $P^s \in \mathcal{P}_{\mathcal{F}}^s$ and $P^t \in \mathcal{P}_{\mathcal{F}}^t$ for some $l' > 1$. In this case, ALG.2 invokes a subroutine, called SPLIT, to 'split' these two paths into two consistent sets of k-paths, $\mathcal{P}_k^s$ and $\mathcal{P}_k^t$, such that $V(P^s) = \bigcup_{P_k^s\in\mathcal{P}_k^s} V(P_k^s)$ and $V(P^t) = \bigcup_{P_k^t\in\mathcal{P}_k^t} V(P_k^t)$. By repeatedly applying this process, ALG.2 constructs the desired consistent k-path partitions.

The details of ALG.2 are presented in Algorithm 2, and its subroutine SPLIT is described in Algorithm 3. Note that SPLIT can be regarded as a *divide-and-conquer* algorithm.

Lemma 5. *In ALG.2, a 2-approximate constrained spanning forest $\mathcal{F}$ in $G[S \cup T]$ can be found in $\mathcal{O}(n^2 \log n)$ time.*

Proof. Let $G' = (V', E', w)$ denote the graph $G[S \cup T]$. The integer program of the minimum general constrained spanning forest problem in (Goemans and Williamson 1995) can be formalized as follows:

$$\begin{aligned} \text{minimize} \quad & \sum_{e\in E'} w(e) \cdot x_e && \text{(IP)} \\ & \sum_{e\in\delta(S)} x_e \geq f(S), \quad \forall\, \emptyset \neq S \subset V', \\ & x_e \in \{0, 1\}, \quad \forall\, e \in E', \end{aligned}$$

where $\delta(S)$ denotes the set of edges with one vertex in S and one vertex in $V' \setminus S$.

Goemans and Williamson (1995) proved that if the function f is a *proper* function, the minimum general constrained spanning forest problem w.r.t. f admits a 2-approximation algorithm with a running time of $\mathcal{O}(n^2 \log n)$.

In our constrained spanning forest problem, the spanning forest must satisfy that for any request $(s_i, t_i) \in \overline{R}$, both s_i and t_i lie in the same tree; and for any tree, the number of

Algorithm 2: ALG.2

Input: An instance $G = (V = U \cup S \cup T, E, w)$.
Output: A feasible solution.

1: Compute a 2-approximate constrained spanning forest $\mathcal{F}$ in $G[S \cup T]$ using GW.2 in (Goemans and Williamson 1995).
2: Initialize $\mathcal{P}^s_{\mathcal{F}} \coloneqq \emptyset$ and $\mathcal{P}^t_{\mathcal{F}} \coloneqq \emptyset$.
3: **for** each $F \in \mathcal{F}$ **do**
4: Construct a path P^s_F (resp., P^t_F) by doubling all edges in $E(F)$ and shortcutting all t_i (resp., s_i) in $V[F]$.
5: Update $\mathcal{P}^s_{\mathcal{F}} \coloneqq \mathcal{P}^s_{\mathcal{F}} \cup \{P^s_F\}$ and $\mathcal{P}^t_{\mathcal{F}} \coloneqq \mathcal{P}^t_{\mathcal{F}} \cup \{P^t_F\}$.
6: **end for**
7: Initialize $\mathcal{P}^s \coloneqq \mathcal{P}^s_{\mathcal{F}}$ and $\mathcal{P}^t \coloneqq \mathcal{P}^t_{\mathcal{F}}$.
8: **while** there exists an lk-path $P^s \in \mathcal{P}^s$ with $l > 1$ **do**
9: Let the consistent lk-path in $\mathcal{P}^t$ be P^t.
10: Call SPLIT on P^s and P^t to obtain k-path sets $\mathcal{P}^s_k$ and $\mathcal{P}^t_k$.
11: Update $\mathcal{P}^s \coloneqq \mathcal{P}^s \setminus \{P^s\} \cup \mathcal{P}^s_k$ and $\mathcal{P}^t \coloneqq \mathcal{P}^t \setminus \{P^t\} \cup \mathcal{P}^t_k$.
12: **end while**
13: Obtain a set of m pairwise disjoint groups $\mathcal{R}$ from $\mathcal{P}^s$ and $\mathcal{P}^t$, where each group contains exactly k requests.
14: Compute a minimum cost perfect matching M^* using Lines 4-5 of ALG.1.
15: **for** each $(u, R_u) \in M^*$ **do**
16: Assign all requests in R_u to vehicle u.
17: Let $P^s_u = s_{\sigma'_1} ... s_{\sigma'_k}$ and $P^t_u = t_{\sigma_1} ... t_{\sigma_k}$ be the corresponding k-paths from $\mathcal{P}^s$ and $\mathcal{P}^t$, respectively.
18: Construct a route I_u using Lines 9-12 of ALG.1.
19: **end for**
20: **return** $(\mathcal{R}, \mathcal{I})$, where $\mathcal{I} = \{I_u \mid u \in U\}$.

vertices it contains is divisible by $2k$. Hence, the integer program of our problem can be formalized into the (IP), where the function f satisfies that for any $S \subseteq V'$, $f(S) \in \{0, 1\}$, and $f(S) = 0$ if and only if

- $|S| \bmod 2k = 0$, and
- $|S \cap \{s_i, t_i\}| \neq 1$ for all $(s_i, t_i) \in \overline{R}$.

It can be verified that f forms a valid proper function (see details in the full version).

Therefore, a 2-approximate constrained spanning forest in the graph $G[S \cup T]$ can be found in $\mathcal{O}(n^2 \log n)$ time. □

Similar to Lemma 1, we have the following result.

Lemma 6 (*). *For the RSAP with $n = mk$, ALG.2 computes in $\mathcal{O}(n^3)$ time a solution $(\mathcal{R}, \mathcal{I})$ with $w(\mathcal{I}) \leq 2w(\mathcal{P}^s) + 2w(\mathcal{P}^t) + c(M^*)$.*

Similar to Remark 1, the running time of ALG.2 can also be improved to $\mathcal{O}(n^{2+\varepsilon})$ for any constant $\varepsilon > 0$.

Lemma 7 (*). *We have $w(\mathcal{P}^s) + w(\mathcal{P}^t) \leq \sqrt{72n/k} \cdot OPT$.*

By Lemmas 4, 6, and 7, we have the following result.

Theorem 2. *For the RSAP with $n = mk$, ALG.2 achieves an approximation ratio of $\sqrt{288n/k} + 1$.*

By Theorems 1 and 2, we obtain the following theorem.

Theorem 3. *For the RSAP with $n = mk$, there exists a $\min\{\mathcal{O}(\sqrt{k}), \mathcal{O}(\sqrt{\frac{n}{k}})\}$-approximation algorithm.*

Note that the ratio in Theorem 3 is at most $\mathcal{O}(\sqrt[4]{n})$.

Algorithm 3: SPLIT

Input: Two consistent lk-paths P^s and P^t, where $l > 1$.
Output: Two consistent sets of k-paths $\mathcal{P}^s_k$ and $\mathcal{P}^t_k$ with $V(P^s) = \bigcup_{P^s_k \in \mathcal{P}^s_k} V(P^s_k)$ and $V(P^t) = \bigcup_{P^t_k \in \mathcal{P}^t_k} V(P^t_k)$.

1: Initialize $\mathcal{P}^s_k \coloneqq \{P^s\}$ and $\mathcal{P}^t_k \coloneqq \{P^t\}$.
2: **while** there exists an $l'k$-path $P^s \in \mathcal{P}^s_k$ with $l' > 3$ **do**
3: Let the consistent path in $\mathcal{P}^t_k$ w.r.t. P^s be P^t.
4: Assume that $P^s = s_{\sigma_1} ... s_{\sigma_{l'k}}$, and let $h = \lceil \frac{l'}{2} \rceil$.
5: Obtain two paths P^s_1 and P^s_2 by deleting the *middle* edge $s_{\sigma_{hk}} s_{\sigma_{hk+1}}$ from P^s, i.e., $P^s_1 = s_{\sigma_1} ... s_{\sigma_{hk}}$, and $P^s_2 = s_{\sigma_{hk+1}} ... s_{\sigma_{l'k}}$.
6: Obtain two paths P^t_1 and P^t_2, which are consistent with P^s_1 and P^s_2 respectively, by shortcutting P^t.
7: Assume that $P^t_1 = t_{\sigma'_1} ... t_{\sigma'_{hk}}$, and $P^t_2 = t_{\sigma'_{hk+1}} ... t_{\sigma'_{l'k}}$.
8: Obtain two paths P^t_{11} and P^t_{12} by deleting the middle edge $t_{\sigma'_{\lceil \frac{h}{2} \rceil k}} t_{\sigma'_{\lceil \frac{h}{2} \rceil k+1}}$ from P^t_1.
9: Obtain two paths P^s_{11} and P^s_{12}, which are consistent with P^t_{11} and P^t_{12} respectively, by shortcutting P^s_1.
10: Analogously, obtain two paths P^t_{21} and P^t_{22} by deleting the middle edge $t_{\sigma'_{\lceil \frac{h+l'}{2} \rceil k}} t_{\sigma'_{\lceil \frac{h+l'}{2} \rceil k+1}}$ from P^t_2. Then, obtain two paths P^s_{21} and P^s_{22}, which are consistent with P^t_{21} and P^t_{22} respectively, by shortcutting P^s_2.
11: Let $\mathcal{P}^s_k \coloneqq \mathcal{P}^s_k \setminus \{P^s\} \cup \{P^s_{11}, P^s_{12}, P^s_{21}, P^s_{22}\}$ and $\mathcal{P}^t_k \coloneqq \mathcal{P}^t_k \setminus \{P^t\} \cup \{P^t_{11}, P^t_{12}, P^t_{21}, P^t_{22}\}$.
12: **end while**
13: **while** there exists an $l'k$-path $P^s \in \mathcal{P}^s_k$ with $2 \leq l' \leq 3$ **do**
14: Let the consistent path w.r.t. P^s in $\mathcal{P}^t_k$ be P^t.
15: Assume that $P^s = s_{\sigma_1} ... s_{\sigma_{l'k}}$.
16: Obtain a set of l' k-paths $\mathcal{P}^s = \{P^s_1, ..., P^s_{l'}\}$, where $P^s_i = s_{\sigma_{ik-k+1}} ... s_{\sigma_{ik}}$ for each $i \in [l']$.
17: Obtain a set of l' consistent k-paths $\mathcal{P}^t = \{P^t_1, ..., P^t_{l'}\}$ by shortcutting P^t according to each corresponding $P^s_i \in \mathcal{P}^s$.
18: Update $\mathcal{P}^s_k \coloneqq \mathcal{P}^s_k \setminus \{P^s\} \cup \mathcal{P}^s$ and $\mathcal{P}^t_k \coloneqq \mathcal{P}^t_k \setminus \{P^t\} \cup \mathcal{P}^t$.
19: **end while**
20: **return** $\mathcal{P}^s_k$ and $\mathcal{P}^t_k$.

An Algorithm for Case $n < mk$

In this section, we introduce our third algorithm (ALG.3) that works for the case $n < mk$.

Similar to ALG.1, ALG.3 begins by obtaining a complete graph $H = (V_H, E_H, \hat{w})$, where $V_H = U \cup [n]$. The function $\hat{w}$ is defined as follows: $\hat{w}(i, j) = w(s_i, s_j) + w(t_i, t_j)$ for any $i, j \in [n]$, $\hat{w}(u_i, j) = w(u_i, s_j) + w(u_i, t_j)$ for any $u_i \in U$ and $j \in [n]$, and $\hat{w}(u_i, u_j) = 2w(u_i, u_j)$ for any $u_i, u_j \in U$. Note that $\hat{w}$ remains a metric. Next, ALG.3 finds a forest $\mathcal{F}$ with possibly minimized $\hat{w}(\mathcal{F})$ in the graph H such that each tree contains exactly one distinct vehicle $u_F \in U$ and at most k vertices in $[n]$. We remark that each tree $F \in \mathcal{F}$ in H corresponds to a tree F' with $w(F') = \hat{w}(F)$ in G such that F' contains the vehicle u_F and the origins and the destinations of at most k requests. Moreover, by doubling all edges in $E(F')$ and then shortcutting, a feasible route I_{u_F} can be obtained with $w(I_{u_F}) \leq 2w(F') = 2\hat{w}(F)$.

It remains to show how to compute the forest $\mathcal{F}$ with minimized $\hat{w}(\mathcal{F})$ in the graph H. This problem is in fact a special case of the *airport and railway problem* (ARP) (Salavatipour and Tian 2025). In the ARP, we are given a metric graph

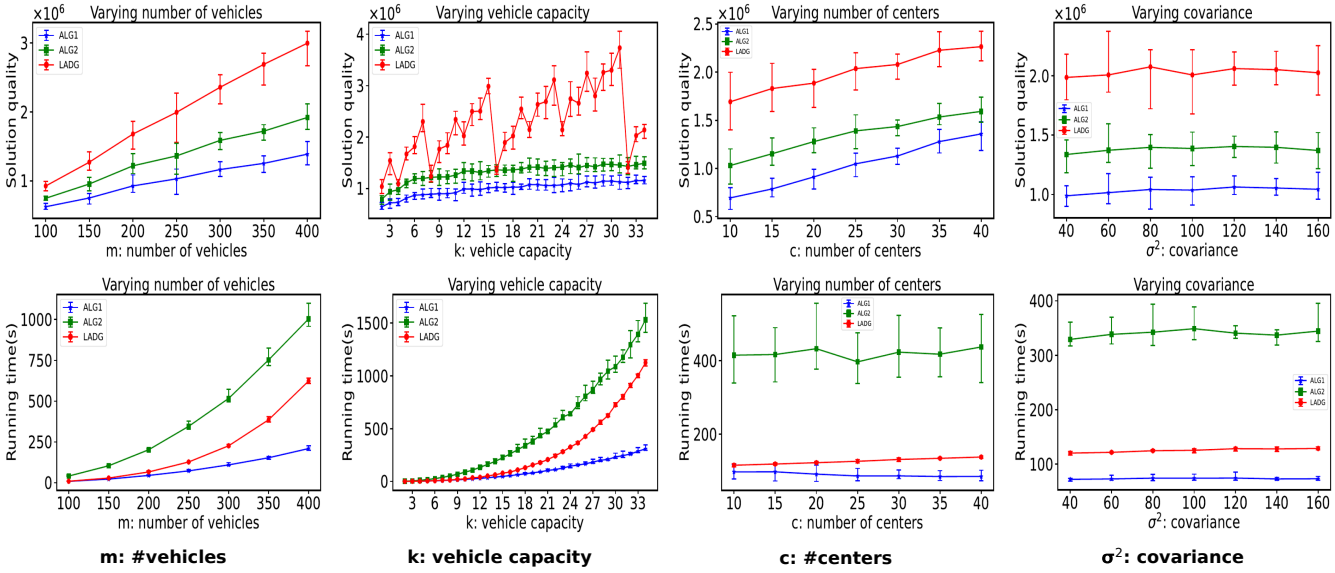

Figure 2: Experimental results on the datasets. Each row corresponds to one objective (from top to bottom): solution quality and running time. Each column corresponds to varying one parameter (from left to right): m, k, c, and σ^2. The results of ALG.1 (ours), ALG.2 (ours), and LADG (previous) are shown as blue, green, and red curves, respectively. Each point represents the average result over 10 instances, while the vertical line segment at each point indicates the minimum and maximum values.

Algorithm 4: ALG.3

Input: An instance $G = (V = U \cup S \cup T, E, w)$.
Output: A feasible solution.

1: Construct a complete graph $H = (V_H, E_H, \hat{w})$, where $V_H = U \cup [n]$. Define $\hat{w}(i,j) = w(s_i, s_j) + w(t_i, t_j)$ for any $i, j \in [n]$, $\hat{w}(u_i, j) = w(u_i, s_j) + w(u_i, t_j)$ for any $u_i \in U$ and $j \in [n]$, and $\hat{w}(u_i, u_j) = 2w(u_i, u_j)$ for any $u_i, u_j \in U$.
2: Compute an $\mathcal{O}(\log \max\{n, m\})$-approximate forest $\mathcal{F}$ in H using the algorithm in (Salavatipour and Tian 2025).
3: **for** each $F \in \mathcal{F}$ **do**
4: Assume that the vehicle in F is u_F.
5: Obtain the corresponding tree F' w.r.t. F in the graph G.
6: Let R_F denote the set of requests covered by F'.
7: Assign all requests in R_F to vehicle u_F.
8: Construct a route I_{u_F} by doubling all edges in $E(F')$ and then shortcutting.
9: **end for**
10: **return** $(\mathcal{R}, \mathcal{I})$, where $\mathcal{I} = \{I_{u_F} \mid F \in \mathcal{F}\}$.

along with a parameter $k \in \mathbb{N}^+$, where each edge has a weight and each vertex has a non-negative *opening cost*. The goal is to find a forest to cover all vertices, where each tree contains at least one opened vertex and has at most $k+1$ vertices. The objective is to minimize the total edge weight of the forest plus the total opening cost of the opened vertices.

To see why the problem of computing the forest $\mathcal{F}$ with minimized $\hat{w}(\mathcal{F})$ is a special case of the ARP, consider the graph H with the opening cost of each vertex in U set to 0, and each vertex in $[n]$ set to ∞. Then, any optimal (or approximate) solution to the ARP must be a forest in which each tree contains exactly one vehicle from U and at most k vertices from $[n]$. (Note that if a tree contains multiple distinct vehicles from U, one can always delete an edge on the unique path between any two of them to split the tree, repeating this process until each tree contains exactly one vehicle.) Since the ARP admits an $\mathcal{O}(\log |V(H)|)$-approximation algorithm and $|V(H)| = \mathcal{O}(n+m)$, an $\mathcal{O}(\log \max\{n, m\})$-approximate forest $\mathcal{F}$ can be obtained in polynomial time.

The details of ALG.3 are shown in Algorithm 4.

First, we analyze the solution weight of ALG.3.

Lemma 8 (*). *For the RSAP with $n < mk$, ALG.3 computes in $\mathcal{O}(n^4 + m^4)$ time a solution $(\mathcal{R}, \mathcal{I})$ with $w(\mathcal{I}) \le 2\hat{w}(\mathcal{F})$.*

By slightly modifying the proof of Lemma 2, we obtain

Lemma 9 (*). *There exists a forest $\mathcal{F}^*$ in the graph H such that $\hat{w}(\mathcal{F}^*) \le \frac{8k\sqrt{k}}{k+1} \cdot OPT$, where each tree in $\mathcal{F}^*$ contains only one vertex from U and at most k vertices from $[n]$.*

Since $\mathcal{F}$ is an $\mathcal{O}(\log \max\{n, m\})$-approximate forest in the graph H, by Lemma 9, we have the following result.

Lemma 10. *$\hat{w}(\mathcal{F}) \le \mathcal{O}(\sqrt{k} \log \max\{n, m\}) \cdot OPT$.*

By Lemmas 8 and 10, we have the following theorem.

Theorem 4. *For the RSAP with $n < mk$, ALG.3 achieves an approximation ratio of $\mathcal{O}(\sqrt{k} \log \max\{n, m\})$.*

Experiments

We evaluate our algorithms for the case $n = mk$ and compare our proposed algorithms, ALG.1 and ALG.2, with the LADG algorithm from (Luo et al. 2022). Notably, LADG has demonstrated strong empirical performance, outperforming several baselines—including adaptations of dial-a-ride algorithms (Tong et al. 2018b; Zeng, Tong, and Chen 2019) and a greedy heuristic—within the RSAP setting, as reported

m: #vehicles	100 150 200 **250** 300 350 400
k: vehicle capacity	2 3 $\cdots$ **18** $\cdots$ 33 34
c: #centers	10 15 20 **25** 30 35 40
σ^2: covariance	40 60 80 **100** 120 140 160

Table 2: Parameter settings for dataset generation. Bold values indicate fixed settings when varying the others.

in (Luo et al. 2022). We do not compare our algorithms with the previous best $(2k-1)$-approximation algorithm (Luo and Spieksma 2022) for non-power-of-2 values of k, as this algorithm is relatively simple and was also not included in the experimental evaluation of (Luo et al. 2022).

Implementation. Both ALG.1 and ALG.2 run in $\mathcal{O}(n^3)$ time, while LADG runs in $\mathcal{O}(n^3 \log n)$ time (Luo et al. 2022). Since LADG frequently requires computing minimum weight perfect matchings (Luo et al. 2022), we employ an optimized implementation of Edmonds' blossom algorithm (Schrijver 2003) to improve its efficiency.

In our implementation, we make a minor modification to the route selection step: when routing a vehicle u to serve its assigned requests R_u, instead of using the fixed edge pair $\{us_{\sigma_i}, s_{\sigma_i}t_{\sigma_i}\}$ yielding $c(u, R_u) = w(u, s_{\sigma_i}) + w(s_{\sigma_i}, t_{\sigma_i})$ (as in Line 9 of ALG.1), we evaluate all k such edge pairs $(s_{\sigma_i}, t_{\sigma_i}) \in R_u$ and choose the one with the minimum weight. This modification does not affect the theoretical approximation ratio and incurs negligible runtime overhead.

All algorithms are implemented in C++, compiled with `g++ -O3`, and executed on a server with an Intel Xeon Gold 6226R CPU @ 2.90GHz, 512 GB RAM, running Ubuntu Server 20.04 LTS.

Instances. To generate RSAP instances with m vehicles and $n = mk$ requests, we follow the approach in (Bei and Zhang 2018; Luo et al. 2022) within a 4000×4000 Euclidean square region $\rho = [0, 4000]^2$.

We first sample c cluster centers $\{\mu_1, \ldots, \mu_c\}$ uniformly from ρ. Then, for each vertex $v \in V$, we select a center μ_i uniformly at random and draw v from the 2D Gaussian distribution $N(\mu_i, \sigma^2\mathbf{I})$, where σ^2 is the variance and $\mathbf{I}$ is the 2×2 identity matrix. All sampling steps are independent.

Previously, both works (Bei and Zhang 2018; Luo et al. 2022) adopt an instance generation method characterized by four parameters: the number of vehicles m, the vehicle capacity k, the number of centers c, and the covariance σ^2. We adopt the same method, and our parameter settings are summarized in Table 2. When varying one parameter, the others are fixed to the boldface values. For each configuration (m, k, c, σ^2), we generate 10 independent instances and report average performance.

Results. Figure 2 summarizes our experimental results. Rows correspond to performance metrics (solution quality and running time), while columns reflect variations in each parameter: m, k, c, and σ^2. Note that the solution quality refers to the solution weight.

We observe that variations in c and σ^2 have a limited effect on performance, which is consistent with the findings of previous experiments (Bei and Zhang 2018; Luo et al. 2022). Hence, we focus our discussion on the impact of m and k.

Solution quality. ALG.1 consistently achieves the best solution quality, which aligns with its theoretical guarantee of outperforming ALG.2 for small values of k. In contrast, LADG performs the worst, even when k is a power of 2.

When varying k, the performance of ALG.1 and ALG.2 remains stable, whereas LADG exhibits pronounced fluctuations: its quality improves significantly when k is a power of 2 but deteriorates sharply when $k = 2^r - 1$ for some integer $r > 0$. These highlight why LADG cannot guarantee a good approximation ratio when k is not a power of 2.

In particular, LADG's performance tends to degrade when the binary representation of k contains many ones, as this may prevent it from finding perfect matchings during each round of its forward phase (see details of LADG). The worst performance is observed when $k = 2^r - 1$ for some integer $r > 0$, where the binary representation of k consists entirely of ones.

Running time. When varying m or k, ALG.1 and LADG have comparable running times, whereas ALG.2 is the slowest. This is because ALG.1 runs GW.1 on the graph H of size $|V_H| = n$, whereas ALG.2 runs GW.2 on $G[S \cup T]$ with $|S \cup T| = 2n$. Since both subroutines run in $\mathcal{O}(n^2 \log n)$ time, the runtime of ALG.2 is expected to be roughly four times that of ALG.1.

Given that LADG has a time complexity of $\mathcal{O}(n^3 \log n)$, we believe that optimizing the implementations of GW.1 and GW.2 can further speed up both ALG.1 and ALG.2.

Summary. ALG.1 not only outperforms LADG in solution quality but also runs faster. ALG.2 likewise achieves better solution quality than LADG, albeit with a slightly longer running time. Moreover, the running time of ALG.2 is roughly four times that of ALG.1.

Conclusion and Discussion

In this paper, we propose three novel algorithms for the RSAP, significantly improving the best-known approximation ratios for both $n = mk$ and $n < mk$. Experimental results for the case $n = mk$ demonstrate the strong empirical performance of our algorithms, consistently outperforming the previously best-known algorithm from (Luo et al. 2022).

From a technical perspective, our algorithms employ fundamentally different techniques compared to previous approximation approaches. These techniques show promise for adaptation to related problems, such as the dial-a-ride problem, opening up avenues for future development.

We did not implement ALG.3 for the case $n < mk$ in our experiments. This is because achieving the desired approximation ratio requires invoking the algorithm from (Salavatipour and Tian 2025), which is complex and hard to implement. ALG.3 is primarily designed to establish a theoretical approximation guarantee. For practical deployment, it may be necessary to design more efficient components or simplify certain intricate procedures to ensure scalability and performance in real-world applications. These aspects deserve further investigation.

Acknowledgments

The work is supported by the National Natural Science Foundation of China under the grants 62502078 and 62372095, and by the Postdoctoral Fellowship Program of CPSF under Grant Number GZC20251102.

References

AlQuhtani, S. 2022. Ridesharing as a potential sustainable transportation alternative in suburban universities: the case of Najran University, Saudi Arabia. *Sustainability*, 14(8): 4392.

Bei, X.; and Zhang, S. 2018. Algorithms for Trip-Vehicle Assignment in Ride-Sharing. In *AAAI 2018*, 3–9. AAAI Press.

Cai, H.; Wang, X.; Adriaens, P.; and Xu, M. 2019. Environmental benefits of taxi ride sharing in Beijing. *Energy*, 174: 503–508.

Chen, L.; Kyng, R.; Liu, Y. P.; Peng, R.; Gutenberg, M. P.; and Sachdeva, S. 2022. Maximum Flow and Minimum-Cost Flow in Almost-Linear Time. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*, 612–623. IEEE.

Clewlow, R. R.; and Mishra, G. S. 2017. Disruptive transportation: The adoption, utilization, and impacts of ride-hailing in the United States. Research Report UCD-ITS-RR-17-07, University of California, Davis, Institute of Transportation Studies, Davis, CA.

Coulombel, N.; Boutueil, V.; Liu, L.; Viguie, V.; and Yin, B. 2019. Substantial rebound effects in urban ridesharing: Simulating travel decisions in Paris, France. *Transportation Research Part D: Transport and Environment*, 71: 110–126.

Goemans, M. X.; and Williamson, D. P. 1995. A General Approximation Technique for Constrained Forest Problems. *SIAM J. Comput.*, 24(2): 296–317.

Goossens, D.; Polyakovskiy, S.; Spieksma, F. C.; and Woeginger, G. J. 2012. Between a rock and a hard place: the two-to-one assignment problem. *Mathematical methods of operations research*, 76(2): 223–237.

Ho, S. C.; Szeto, W. Y.; Kuo, Y.-H.; Leung, J. M.; Petering, M.; and Tou, T. W. 2018. A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research Part B: Methodological*, 111: 395–421.

Jacob, J.; and Roet-Green, R. 2021. Ride solo or pool: Designing price-service menus for a ride-sharing platform. *Eur. J. Oper. Res.*, 295(3): 1008–1024.

Kleiner, A.; Nebel, B.; and Ziparo, V. A. 2011. A Mechanism for Dynamic Ride Sharing Based on Parallel Auctions. In Walsh, T., ed., *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, 266–272. IJCAI/AAAI.

Knopp, S.; Biesinger, B.; and Prandtstetter, M. 2021. Mobility offer allocations in corporate settings. *EURO Journal on Computational Optimization*, 9: 100010.

Li, S.; Li, M.; and Lee, V. C. S. 2020. Trip-Vehicle Assignment Algorithms for Ride-Sharing. In *COCOA 2020*, volume 12577 of *Lecture Notes in Computer Science*, 681–696. Springer.

Luo, K.; Agarwal, C.; Das, S.; and Guo, X. 2022. The Multi-vehicle Ride-Sharing Problem. In Candan, K. S.; Liu, H.; Akoglu, L.; Dong, X. L.; and Tang, J., eds., *WSDM '22: The Fifteenth ACM International Conference on Web Search and Data Mining, Virtual Event / Tempe, AZ, USA, February 21 - 25, 2022*, 628–637. ACM.

Luo, K.; and Spieksma, F. C. 2022. Minimizing Travel Time and Latency in Multi-Capacity Ride-Sharing Problems. *Algorithms*, 15(2): 30.

Luo, K.; and Spieksma, F. C. R. 2020. Approximation Algorithms for Car-Sharing Problems. In Kim, D.; Uma, R. N.; Cai, Z.; and Lee, D. H., eds., *Computing and Combinatorics - 26th International Conference, COCOON 2020, Atlanta, GA, USA, August 29-31, 2020, Proceedings*, volume 12273 of *Lecture Notes in Computer Science*, 262–273. Springer.

Protopapas, N.; Yazdanpanah, V.; Gerding, E. H.; and Stein, S. 2024. Online Decentralised Mechanisms for Dynamic Ridesharing. In *AAMAS 2024*, 1602–1610. International Foundation for Autonomous Agents and Multiagent Systems / ACM.

Salavatipour, M. R.; and Tian, L. 2025. Approximation algorithms for the airport and railway problem. *J. Comb. Optim.*, 49(1): 8.

Schrijver, A. 2003. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer.

Shaheen, S.; Stocker, A.; and Mundler, M. 2017. *Online and app-based carpooling in France: Analyzing users and practices—A study of BlaBlaCar*. Springer.

Shen, W.; Lopes, C. V.; and Crandall, J. W. 2016. An Online Mechanism for Ridesharing in Autonomous Mobility-on-Demand Systems. In Kambhampati, S., ed., *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, 475–481. IJCAI/AAAI Press.

Tong, Y.; Zeng, Y.; Zhou, Z.; Chen, L.; Ye, J.; and Xu, K. 2018a. A Unified Approach to Route Planning for Shared Mobility. *Proc. VLDB Endow.*, 11(11): 1633–1646.

Tong, Y.; Zeng, Y.; Zhou, Z.; Chen, L.; Ye, J.; and Xu, K. 2018b. A unified approach to route planning for shared mobility. *Proceedings of the VLDB Endowment*, 11(11): 1633.

Wang, H.; and Yang, H. 2019. Ridesourcing systems: A framework and review. *Transportation Research Part B: Methodological*, 129: 122–155.

Yu, B.; Ma, Y.; Xue, M.; Tang, B.; Wang, B.; Yan, J.; and Wei, Y.-M. 2017. Environmental benefits from ridesharing: A case of Beijing. *Applied Energy*, 191: 141–152.

Zeng, Y.; Tong, Y.; and Chen, L. 2019. Last-Mile Delivery Made Practical: An Efficient Route Planning Framework with Theoretical Guarantees. *Proc. VLDB Endow.*, 13(3): 320–333.

Zheng, L.; Chen, L.; and Ye, J. 2018. Order Dispatch in Price-aware Ridesharing. *Proc. VLDB Endow.*, 11(8): 853–865.