The Simpler The Better: An Indexing Approach for Shared-Route Planning Queries

Yuxiang Zeng [†] Yongxin Tong [‡] Yuguang Song [‡] Lei Chen [†]
[†] The Hong Kong University of Science and Technology, Hong Kong SAR, China
[‡] SKLSDE Lab, BDBC and IRI, Beihang University, China
[†] {yzengal, leichen}@cse.ust.hk [‡] {yxtong, songyuguang}@buaa.edu.cn

ABSTRACT

Ridesharing services have gained global popularity as a convenient, economic, and sustainable transportation mode in recent years. One fundamental challenge in these services is planning the shared-routes (*i.e.*, sequences of origins and destinations) among the passengers for the vehicles, such that the platform's total revenue is maximized. Though many methods can solve this problem, their effectiveness is still far from optimal on either empirical study (*e.g.*, over 31% lower total revenue than our approach) or theoretical study (*e.g.*, arbitrarily bad or impractical theoretical guarantee). In this paper, we study the shared-route planning queries in ridesharing services and focus on designing efficient algorithms with good approximation guarantees. Particularly, our idea is to iteratively search the most profitable route among the unassigned requests for each vehicle, which is simpler than the existing methods. Unexpectedly, we prove this simple method has an approximation ratio of 0.5 to the optimal result. Moreover, we also design an index called *additive tree* to improve the efficiency and apply randomization to improve the approximation guarantee. Finally, experimental results on two real datasets demonstrate that our additive-tree-based approach outperforms the state-of-the-art algorithms by obtaining up to 31.4%-127.4% higher total revenue.

PVLDB Reference Format:
Yuxiang Zeng, Yongxin Tong, Yuguang Song, and Lei Chen. The Simpler The Better: An Indexing Approach for Shared-Route Planning Queries. *PVLDB*, 13(13): 3517-3530, 2020.
DOI: https://doi.org/10.14778/3424573.3424574

1. INTRODUCTION

Ridesharing has gained global popularity in urban transportation, such as carsharing, vanpooling, and food delivery. By sharing the rides in unoccupied vehicles, it not only provides a convenient trip with a lower price, but also mitigates traffic congestion, saves energy, and reduces pollution emissions in our daily lives.

Proceedings of the VLDB Endowment, Vol. 13, No. 13
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3424573.3424574

One fundamental problem in ridesharing platforms is planning the routes shared among the requests (*e.g.*, passengers) for the vehicles (*e.g.*, drivers). Different from other trip planning queries in spatial databases [17, 31, 11, 34, 26], the *shared-route* here is a sequence of origins (*e.g.*, pickup locations) and destinations (*e.g.*, delivery locations), which also satisfies the constraints (*e.g.*, deadline constraint) set by the platform. Moreover, these shared-routes are usually planned based on certain optimization objectives.

The main objectives in existing studies include maximizing the revenue of the platform [38, 6, 48, 5, 49] and minimizing the travel time of vehicles [24, 13, 25, 7]. To simultaneously consider both objectives, our paper focuses on planning the shared-route with the *shortest travel time* for each vehicle such that the platform's *total revenue is maximized*. Though many existing algorithms can be used to solve this problem, they usually have the following limitations.

Limitation 1. Though these methods are tested and compared with others in the real datasets, existing studies [38, 6, 48, 24, 13, 25] usually lack *theoretical and empirical comparison* with the optimal result. As a result, an algorithm, which obtains higher total revenue than the others, may still be worse than the optimal result. For example, the algorithms pruneGreedyDP [38] and PBM [48] *at least* have 29% and 10% lower revenue than our approach in real datasets, respectively. In other words, *the total revenue of these methods may be notably worse than the optimal result.*

Limitation 2. Others have *arbitrarily bad or impractical* approximation guarantees in the effectiveness. For instance, Zheng *et al.* [49] show their method has an approximation ratio of $O(1/m)$. The theoretical guarantee of [49] will be arbitrarily bad, if m (*i.e.*, the number of vehicles) is large. Though Bei *et al.* [7] also propose an approximation solution based on graph matching, their assumptions are not practical. For example, their approximation analysis [7] requires that any vehicle's capacity is 2 and the number of requests is exactly twice of the number of vehicles. Other matching based solutions [48, 30, 5] have no theoretical analysis to prove the approximation ratios. Thus, *existing algorithms still have no constant and practical theoretical guarantees.*

In this paper, we study shared-route planning queries and design solutions with constant approximation ratios to maximize the platform's total revenue. Specifically, our *main idea* is to iteratively find the most profitable route among the unassigned requests for each vehicle. Though the idea is **simple**, the approximation ratio (0.5) is much **better**. However, when the number of requests is large, it becomes time-consuming to search the most profitable route among

all these requests. To improve the *efficiency*, for each vehicle, we first filter out the infeasible requests, then search the most profitable route among the feasible ones using an index called the *additive tree*. Moreover, we also prove that a **simple** randomization strategy (*i.e.*, randomly picking the vehicles) can improve the approximation ratio to $1/(2-0.5/C) > 0.5$, where C denotes the vehicle's capacity. Finally, we conduct extensive experiments on real datasets.

Our main **contributions** are summarized as follows.

- We are the first to propose an approximation solution with a constant ratio (0.5) to solve the shared-route planning queries for maximizing the platform's revenue, based on these surveys [32, 12, 42].
- To improve the efficiency of this simple idea, we design a novel index called *additive tree* and propose several optimization strategies (*e.g.*, randomization and pruning). As a result, the approximation ratio is improved. We can save up to 747.2× time cost and 80.2× memory usage at the same time.
- Extensive experiments show that our solutions always outperform the state-of-the-art algorithms [48, 38] by having up to 31.4%-127.4% higher total revenue.

In the rest of this paper, we first present the problem definition in Sec. 2. Then we introduce a general framework in Sec. 3, which summarizes both existing baselines and our simpler solution. Next, we propose our indexing based optimization techniques in Sec. 4. Finally, we conduct experiments in Sec. 5, review related work in Sec. 6, and conclude in Sec. 7.

2. PROBLEM STATEMENT

In this section, we introduce the basic concepts in Sec. 2.1 and present the problem definition and hardness in Sec. 2.2.

2.1 Preliminaries

Definition 1 (Shortest Travel Time). *Given a set $\mathcal{V}$ of locations, the shortest travel time between any two locations is denoted by a function $\mathsf{dist} : \mathcal{V} \times \mathcal{V} \to [0, +\infty)$, which satisfies the triangle inequality (i.e., for any locations $x, y, z \in \mathcal{V}$, $\mathsf{dist}(x, y) + \mathsf{dist}(y, z) \geq \mathsf{dist}(x, z)$).*

The function dist can be the shortest travel time on either graphs [13, 38] or Euclidean spaces [40, 7].

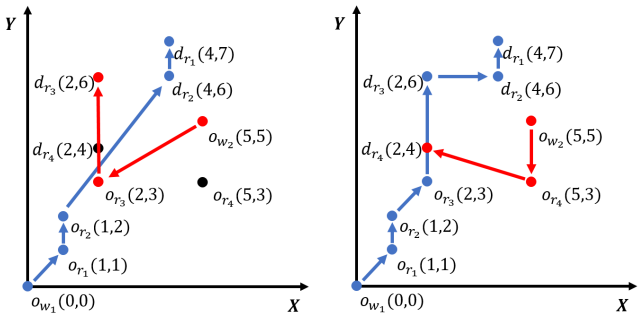
Definition 2 (Vehicle). *A vehicle is denoted by $w = \langle o_w, c_w \rangle$, which is initially at location o_w with a capacity c_w.*

The capacity indicates that a vehicle w can take at most c_w requests. In real applications, it represents the number of passengers/parcels that a taxi/courier can carry. Thus, the vehicle's capacity is usually a small constant [48, 24, 21, 28, 33]. We use $W = \{w_1 \cdots w_m\}$ to denote a set of m vehicles.

Definition 3 (Request). *A request is denoted by $r = \langle t_r, o_r, d_r, e_r, p_r \rangle$, which is released at time t_r with the origin o_r, destination d_r and deadline e_r. The fare/payment of this request is p_r. The request is **served** if (1) it is first picked up by one vehicle at the origin o_r; and (2) it is then delivered by the same vehicle at the destination d_r before the deadline e_r. If r is served, the platform will receive a payment p_r from the requester. Otherwise, the platform rejects the request.*

Table 1: The information of the requests ($\forall r_i, t_{r_i} = 0$)

Request	Origin	Destination	Deadline	Payment
r_1	(1,1)	(4,7)	10	6
r_2	(1,2)	(4,6)	9	5
r_3	(2,3)	(2,6)	7	4
r_4	(5,3)	(2,4)	8	3



(a) One feasible solution **(b)** The optimal solution

Figure 1: An illustration of the toy example

We use $R = \{r_1 \cdots r_n\}$ to denote a set of n requests and R_w ($\subseteq R$) to denote the requests assigned to the vehicle w.

Definition 4 (Route). *A shared-route (a.k.a route) of a vehicle w for serving requests R_w is denoted by $s_w = \langle l_w^0, l_w^1, l_w^2, \cdots, l_w^k \rangle$, which is an ordered sequence of vehicle's initial location (i.e., $l_w^0 = o_w$), and the requests' origins and destinations (i.e., $\forall i > 0, l_w^i \in \{o_r | r \in R_w\} \cup \{d_r | r \in R_w\}$). A route is **feasible** if (1) all the requests R_w can be successfully served by this route; (2) the number of requests $|R_w|$ is no more than the vehicle's capacity c_w.*

The travel time of a route s_w is defined as $\sum_i \mathsf{dist}(l_w^{i-1}, l_w^i)$.

2.2 Problem Definition and Hardness

In this subsection, we first present the definition of the *Shared-Route Planning Query* (SRPQ) problem as follows.

Definition 5 (SRPQ Problem). *Given a set R of n requests and a set W of m vehicles, the SRPQ problem is to find a route s_w for each vehicle $w \in W$ such that the total revenue of the platform $\mathit{OBJ}(R, W)$ is maximized*

$$\mathit{OBJ}(R, W) = \sum_{w \in W} \sum_{r \in R_w} p_r \tag{1}$$

and meets the following constraints:

- *Feasibility constraint: each vehicle is scheduled with a feasible route;*
- *Shortest travel time constraint: each scheduled route takes the shortest total travel time.*

For simplicity, we call the route with the shortest travel time as the "fastest route" in the rest of this paper. We then illustrate the SRPQ problem by the following example.

Example 1. *Suppose there are 2 vehicles w_1, w_2 and 4 requests r_1-r_4 in a ridesharing platform. Fig. 1 shows their locations (e.g., vehicles' initial location, requests' origins and destinations) and the travel time between any two locations is calculated by Euclidean distance (e.g., speed is 1). We also assume that the capacity of w_1 is 3 and the capacity of w_2*

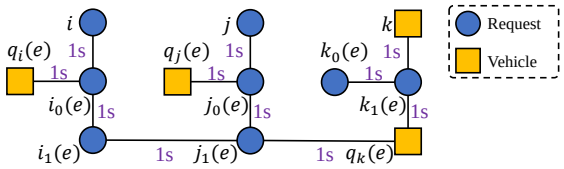


Figure 2: Vehicles and requests that correspond to any triple $e = (i, j, k)$ of the 3DM in the reduction [7]

is 1. The other information of requests is listed in Table 1. In the SRPQ problem, one possible solution is to assign the requests r_1, r_2 to w_1 and r_3 to w_4 (see Fig. 1a). Then we can calculate the fastest routes for these two vehicles, i.e., $s_{w_1} = \{o_{w_1}, o_{r_1}, o_{r_2}, d_{r_2}, d_{r_1}\}$ and $s_{w_2} = \{o_{w_2}, o_{r_3}, d_{r_3}\}$. Accordingly, the total revenue of this routing plan is $p_{r_1} + p_{r_2} + p_{r_3} = 6 + 5 + 4 = 15$. However, this plan is not optimal since 15 is not the maximum total revenue. Instead, the maximum revenue is $\sum_{i=1}^{4} p_{r_i} = 6 + 5 + 4 + 3 = 18$ and the corresponding routes are illustrated in Fig. 1b, i.e., $s_{w_1} = \{o_{w_1}, o_{r_1}, o_{r_2}, o_{r_3}, d_{r_3}, d_{r_2}, d_{r_1}\}$ and $s_{w_2} = \{o_{w_2}, o_{r_4}, d_{r_4}\}$.

Hardness. For the hardness of this problem, [48] has proved that it is NP-hard when the vehicle's capacity is as large as n (*i.e.*, the number of requests). However, since capacity is usually a constant (≥ 2), we analyze the hardness of the problem under the practical cases in Theorem 1.

THEOREM 1. *The SRPQ problem is NP-hard and APX-hard, when vehicle's capacity is a constant (≥ 2).*

PROOF. We prove the hardness of the SRPQ problem by reducing from the 3-dimensional perfect matching (3DM) problem, which is NP-hard and APX-hard [10]. An instance of 3DM is denoted by $\langle I, J, K, E\rangle$. Here, I, J, K are three disjoint sets with equal sizes n. E is a set of m triples $e = (i, j, k)$ such that $i \in I, j \in J, k \in K$. The 3DM problem decides if there exists a subset $M \subseteq E$ with n triples, such that every element in $I \cup J \cup K$ occurs in exactly once in M. Given such an instance, we use a similar reduction procedure as in [7] to generate the vehicles and requests.

(1) We generate one vehicle (k) for each element in K and three vehicles $(q_i(e), q_j(e), q_k(e))$ for each triple $e \in E$. The $n + 3m$ vehicles are denoted by yellow squares in Fig. 2.

(2) We generate one request (i/j) for each element in $I \cup J$ and six requests $(i_0(e), i_1(e), j_0(e), j_1(e), k_0(e), k_1(e))$ for each triple $e \in E$. The $2n + 6m$ requests are denoted by blue circles in Fig. 2.

(3) The travel time of any edge in Fig. 2 is 1s. The capacity of each vehicle is 2. The payment of each request is 1. Since a vehicle takes 2s to serve all the assigned requests in the reduction of [7], we set the deadline of each request as 2s. Besides, the destination of the request is also its origin.

According to [7], a perfect matching exists in the 3DM problem *if and only if* our SRPQ problem has a maximum total revenue of $2n + 6m$ (see our full paper [45] for more detailed explanation). Thus, we complete the proof. □

Table 2 lists the major notations used in this paper.

3. A TWO-PHASE FRAMEWORK

In this section, we first present a general framework to solve this problem in Sec. 3.1. Next, we introduce the existing baseline in Sec. 3.2 and our simple algorithm in Sec. 3.3, which are based on this framework. We summarize the comparisons between representative baselines and our proposed solutions in Table 3.

Table 2: Summary of major notations

Notation	Description
R, W	a set of n requests and m vehicles
o_r, d_r	origin and destination of the request r
t_r, e_r, p_r	release time, deadline and payment of the request r
o_w, c_w	initial location and capacity of the vehicle w
R_w, s_w	assigned requests and route of the vehicle w
$\mathsf{dist}(\cdot, \cdot)$	shortest travel time between two locations

3.1 Overview

Background. Existing solutions can be classified into two kinds, *i.e.*, *insertion-based* solutions [38, 48] and *grouping-based* solutions [5, 7, 48]. An insertion-based solution usually sequentially assigns (inserts) *one request* into the current route of one suitable vehicle. Differently, the *grouping-based* solution first determines *a group of requests* that can be shared together, and then picks a suitable group for each vehicle. Though insertion-based solutions are efficient, they are usually heuristics without theoretical guarantees (see [38] for details). Thus, we focus on designing a *grouping-based* solution with a constant approximation guarantee.

Main Idea. A *grouping-based* framework usually consists of *two phases*, whose main ideas are elaborated as follows.

(1) In the *first* phase, we determine all possible groups of requests, where these requests can be shared together.

(2) In the *second* phase, we schedule each vehicle with one group of requests and plan the fastest routes to serve them, in order to maximize the platform's total revenue.

Basic Concepts. According to the main idea, we introduce a concept of *subpath* to define the *request group* (*i.e.*, a group of requests that can be shared together).

DEFINITION 6 (SUBPATH). *A subpath (denoted by ps) is a subsequence of the route excluding the vehicle's initial location, i.e., $ps = \{l_w^1, l_w^2, \cdots, l_w^k\}$, where l_w^i is either a request's origin or a request's destination.*

Apparently, the first location l_w^1 of the subpath must be a request's origin, where the request is denoted by $\mathsf{headReq}(ps)$. A subpath is *feasible* if a hypothetical vehicle, who is at $o_w = l_w^1$ with a large enough capacity, can successfully serve these requests by following the subpath.

DEFINITION 7 (REQUEST GROUP). *A request group is denoted by $g = \langle k_g, R_g, PS_g, U_g\rangle$, which represents a set R_g of k_g requests. PS_g denotes a set of feasible subpaths that contain all the origins and destinations of the requests R_g. U_g denotes the profit of this group, i.e., the total payments of the requests R_g ($U_g = \sum_{r \in R_g} p_r$).*

Accordingly, a request group is *feasible* if its set of subpaths is non-empty, *i.e.*, $PS_g \neq \emptyset$. For instance, as shown in the first three rows of Table 4, the request group g_6 contains two requests (*i.e.*, $k_{g_6} = 2$, $R_{g_6} = \{r_1, r_3\}$) and there exists one feasible subpath in PS_{g_6}, *i.e.*, $\{o_{r_1}, o_{r_3}, d_{r_3}, d_{r_1}\}$. The profit of this group g_6 is $U_{g_6} = p_{r_1} + p_{r_3} = 6 + 4 = 10$.

3.2 Existing Baselines

Existing studies [30, 5, 7, 48] on this framework mainly focus on the *bipartite matching based* solutions and we summarize their common idea as follows.

Basic Idea. In the *first* phase, they generate all the feasible request groups by enumerating each combination of requests. In the *second* phase, they construct a bipartite

Table 3: The comparisons between existing solutions and our proposed solutions

Compared Algorithm	Greedy	pruneGreedyDP	MWBM	GAS	GAS-O1	GAS-O2
Reference	[48]	[38]	[48]	this paper		
Approximation Ratio	heuristic	heuristic	no guarantee[b]	0.5	0.5	$1/(2-0.5/C)$
Time Complexity[a]	$O(mn)$	$O(mn)$	$O(n^C + m^2\|G\|)$	$O(n^C + m\|G\|)$	$O(T_c + mT_s)$	$O(m(T_c^- + T_s^-))$
Space Complexity	$O(m+n)$	$O(m+n)$	$O(m\|G\|)$	$O(\|G\|)$	$O(\|G\|)$	$O(\|G^-\|)$

[a] C is the maximum vehicle's capacity. n^- is the maximum number of requests that satisfy the range filtering of a vehicle. $|G|$ ($|G^-|$) is the number of request groups for n ($n^- < n$) requests. T_c and T_c^- are the time complexities to construct our index with n and n^- requests. T_s and T_s^- are the time complexities to search our index with n and n^- requests.
[b] The approximation ratio is not proved to be guaranteed in [48].

Table 4: The details of the generated request groups and the plans of the introduced algorithms

#(Requests)		1	2				3
Request Group		$g_i = \{r_i\}$	$g_5 = \{r_1, r_2\}$	$g_6 = \{r_1, r_3\}$	$g_7 = \{r_2, r_3\}$	$g_8 = \{r_3, r_4\}$	$g_9 = \{r_1, r_2, r_3\}$
Subpath set		o_{r_i}, d_{r_i}	$o_{r_1}, o_{r_2}, d_{r_2}, d_{r_1}$ ... $o_{r_2}, o_{r_1}, d_{r_2}, d_{r_1}$	$o_{r_1}, o_{r_3}, d_{r_3}, d_{r_1}$	$o_{r_2}, o_{r_3}, d_{r_3}, d_{r_2}$ $o_{r_3}, o_{r_2}, d_{r_3}, d_{r_2}$	o_{r_4}, o_{r_3}, d_{r_3}, d_{r_4}	$o_{r_1}, o_{r_2}, o_{r_3}, d_{r_3}, d_{r_1}, d_{r_2}$ ... $o_{r_2}, o_{r_1}, o_{r_3}, d_{r_3}, d_{r_2}, d_{r_1}$
headSlack		3.3, 4.0, 4.0, 4.8	3.0, ..., 2.2	1.8	2.6, 1.5	1.0	0.6, ..., 0.8
Plan	MWBM	Assignment: (g_9, w_1), $(\emptyset, w_2)$; Total revenue: 15					
	GAS	Assignment: (g_9, w_1), (g_4, w_2); Total revenue: 18					

Algorithm 1: Existing Baseline MWBM [48]

input : the requests R and vehicles W
output: the planned routes $\{s_w | w \in W\}$
`/* Phase 1: Generation */`
1 The maximum vehicle's capacity $C \leftarrow \max_w c_w$;
2 A set of request groups $G \leftarrow \emptyset$;
3 **for** *size* $k \leftarrow 1$ **to** *the maximum capacity* C **do**
4 $G' \leftarrow$ the set of request groups containing k different requests in R, $G \leftarrow G \cup G'$;
`/* Phase 2: Schedule */`
5 Construct a weighted bipartite graph (G, W, E), where edge weight denotes the payments to serve the requests $g \in G$ by the vehicle $w \in W$;
6 $M \leftarrow$ the maximum weighted bipartite matching;
7 **foreach** *vehicle* w *and its assigned request group* g **do**
8 $s_w \leftarrow$ the fastest route to serve all the unassigned requests in g;
9 **return** $\{s_w | w \in W\}$;

graph between request groups and vehicles. The edge weight between a request group and a vehicle is defined as follows: (1) if the vehicle can serve this group of requests, the weight is the profit of the request group; (2) otherwise, the weight is 0. Next, they calculate the maximum weighted bipartite matching (MWBM) of this bipartite graph. Finally, they plan the fastest routes based on the matching result.

Algorithm Details. Algo. 1 illustrates the detailed procedure of the algorithm in [48]. Specifically, the maximum vehicles' capacity is denoted by C and the set of possible routes is denoted by S (lines 1-2). In the *first* phase, existing solutions enumerate all possible request groups by brute-force (lines 3-4). If the selected k requests can be served by any route, we add these routes into the set S. In the *second* phase, it first constructs a bipartite graph between the request groups and vehicles (line 5), and then obtains the maximum weighted bipartite matching m (line 6). Finally, in lines 7-8, they iteratively pick a vehicle w and plan the fastest route to serve all the unassigned requests in the request group g, where g is matched to w in M.

Example 2. *Back to our example. In the first phase of Algo. 1, the generated request groups are shown in Table 4. For instance, the requests r_1, r_3 of the request group g_6 can be shared together by the subpath $\{o_{r_1}, o_{r_3}, d_{r_3}, d_{r_1}\}$. Thus, we can construct a bipartite graph between the request groups and vehicles. For example, the edge weight between w_1 and g_6 is $U_{g_6} = 10$, since w_1 can serve the requests R_{g_6}. The edge weight between w_2 and g_6 is 0 since w_2 cannot serve R_{g_6}. In line 6, the maximum weighted matching of this bipartite graph is $\{(g_9, w_1), (g_3, w_2)\}$. Then, in lines 7-8, Algo. 1 will first plan the fastest route for vehicle w_1, i.e., $s_{w_1} = \{o_{w_1}, o_{r_1}, o_{r_2}, o_{r_3}, d_{r_3}, d_{r_2}, d_{r_1}\}$. Since r_3 has already been assigned to w_1, it cannot be allocated to w_2 again and hence $s_{w_2} = \{o_{w_2}\}$. The total revenue of these routes (15) is 16.7% lower than the optimal result (18).*

Complexity Analysis. In Algo. 1, The first phase takes $O(n^C)$ time and $O(|G|)$ space to generate G. The second phase takes $O(m^2|G|)$ time and $O(m|G|)$ space to obtain the maximum weighted bipartite matching (*e.g.*, Kuhn-Munkres algorithm [27]). Overall, the time complexity is $O(n^C + m^2|G|)$ and the space complexity is $O(m|G|)$.

Discussion. Based on our experiments (see Sec. 5), Algo. 1 in this framework has bad effectiveness and low efficiency (compared with our solutions). Since other existing solutions [30, 5, 7] also have similar steps as the first phase of Algo. 1, their time cost will also be high in large-scale datasets. To overcome these limitations, we first propose an effective solution in Sec. 3.3, and then design efficient optimization techniques in Sec. 4. Note that we compare with [48] instead of [30, 5, 7], because (1) Ref. [48] is the only collaborative work with a real industry (*i.e.*, Didi Chuxing [1]) among these studies, (2) Ref. [48] is more recent than [30, 5], and (3) Ref. [7] requires that any vehicle's capacity is 2, but a vehicle's capacity is usually no smaller than 3 in real platforms (*e.g.*, Didi Chuxing [1]).

3.3 Our Effective Solution

Basic Idea. Our idea is to pick the most profitable request group (*i.e.*, the one with the highest profit U_g) for each vehicle. Though the idea is simpler, we will later prove its approximation ratio (0.5) is much better.

Algorithm 2: Our Effective Solution GAS

input : the requests R and vehicles W
output: the planned routes $\{s_w | w \in W\}$

1 Execute the first phase of Algo. 1 (lines 1-4);
/* Phase 2: Schedule */
2 **foreach** *vehicle* $w \in W$ **do**
3 $g^* \leftarrow$ the most profitable request group in G that w can serve its requests, $s_w \leftarrow$ the fastest route for serving R_{g^*}, $G \leftarrow$ remove the request groups that have common requests in R_{g^*};
4 **return** $\{s_w | w \in W\}$;

Algorithm Details. Algo. 2 illustrates the detailed procedure. Specifically, we also first generate possible request groups (line 1). In lines 2-3, for each vehicle w, we find the most profitable request group g^* such that w can serve all the requests in g^*. Specifically, we enumerate each request group $g \in G$ and check whether there exists a subpath $ps \in PS_g$, such that w can serve all the requests R_g by following ps. Then we maintain g^* to be the one with the highest profit. If g^* exists, we then plan the fastest route for serving its requests R_{g^*}. After that, we remove any request group $g \in G$ that contains at least one request in R_{g^*}.

Example 3. *Back to our example. In Algo. 2, the results of the first phase are shown in Table 4, which is the same as Algo. 1. In the second phase, we first search the most profitable request group for vehicle w_1, which is $g_9 = \{r_1, r_2, r_3\}$. Then we can plan the fastest route for serving r_1-r_3 (e.g., by brute-force enumeration), i.e., $s_{w_1} = \{o_{w_1}, o_{r_1}, o_{r_2}, o_{r_3}, d_{r_3}, d_{r_2}, d_{r_1}\}$. After that, we need to remove request groups g_1-g_3, g_5-g_8 from G since they contain some requests in r_1-r_3. Similarly, in the next iteration, we assign r_4 to vehicle w_2 and plan the route $s_{w_2} = \{o_{w_2}, o_{r_4}, d_{r_4}\}$ for it. Based on these routes, the platforms' total revenue is 18, which is optimal.*

Complexity Analysis. In Algo. 2, line 2 has $O(m)$ iterations and each iteration takes $O(|G|)$ time. Thus, the time complexity is $O(n^C + m|G|)$ and the space complexity is $O(|G|)$, which is more efficient than Algo. 1.

Approximation Analysis. We next analyze the approximation ratio of Algo. 2 in Theorem 2. Based on the theoretical results, Algo. 2 should be also effective in practice.

Theorem 2. *The approximation ratio of Algo. 2 is 0.5.*

Proof. Let g_{w_i} denote the request group assigned to vehicle w_i by Algo. 2 and $g^*_{w_i}$ denote the request group assigned to vehicle w_i in the optimal result. We use $\mathsf{rev}(g)$ to denote the total payments of the requests R_g, *i.e.*, $\mathsf{rev}(g) = \sum_{r \in R_g} p_r$. It is obvious that the total revenue of the optimal result (denoted by OPT) is

$$OPT = \sum_{w_i \in W} \mathsf{rev}(g^*_{w_i}) \tag{2}$$

To bound the total revenue of our algorithm, we need to consider the following two cases.

(1) If $\mathsf{rev}(g_{w_i}) \geq \mathsf{rev}(g^*_{w_i})$, we only charge $\mathsf{rev}(g^*_{w_i})$ into the lower bound of our revenue.

(2) If $\mathsf{rev}(g_{w_i}) < \mathsf{rev}(g^*_{w_i})$, it indicates that some requests in $g^*_{w_i}$ must have been assigned to other vehicles in the previous iterations of line 2. Otherwise, $g^*_{w_i}$ should be assigned to w_i in line 3. Thus, we use a request group $\overline{g^*_{w_i}}$ to denote all the requests in $g^*_{w_i}$ that have already been assigned, *i.e.*, $R_{\overline{g^*_{w_i}}} \subseteq R_{g^*_{w_i}}$. If $\overline{g^*_{w_i}}$ is assigned to only one vehicle, this vehicle may have also been scheduled with some other requests that are not in $g^*_{w_i}$. As $\mathsf{rev}(g) = \sum_{r \in R_g} p_r$, we can infer that $\mathsf{rev}(\overline{g^*_{w_i}}) \leq \mathsf{rev}(g^*_{w_i})$. Besides, as the request group g_{w_i} is picked by our algorithm, it indicates that the g_{w_i} should be more profitable than the total payments of serving the remaining requests in $g^*_{w_i}$, *i.e.*,

$$\mathsf{rev}(g_{w_i}) \geq \mathsf{rev}(g^*_{w_i}) - \mathsf{rev}(\overline{g^*_{w_i}}) \tag{3}$$

So we charge the RHS of Eq. (3) into the lower bound.

For the proof simplification, we assume that $\overline{g^*_{w_i}} = \emptyset$ and $\mathsf{rev}(\overline{g^*_{w_i}}) = 0$ in the first case. Thus, we can use the RHS of Eq. (3) as the lower bound of the total revenue (denoted by ALG) by Algo. 2 as follows.

$$ALG = \sum_{w_i \in W} \mathsf{rev}(g_{w_i}) \geq \sum_{w_i \in W} \big(\mathsf{rev}(g^*_{w_i}) - \mathsf{rev}(\overline{g^*_{w_i}})\big) \tag{4}$$

Since a request can be assigned to only one vehicle, we also know that the requests in $\overline{g^*_{w_i}}$ are disjoint with the requests in $\overline{g^*_{w_j}}$ (when $i \neq j$). Thus, we have

$$\bigcup_{w_i} R_{\overline{g^*_{w_i}}} \subseteq \bigcup_{w_i} R_{g_{w_i}} \implies \sum_{w_i} \mathsf{rev}(\overline{g^*_{w_i}}) \leq \sum_{w_i} \mathsf{rev}(g_{w_i}) \tag{5}$$

Based on Eq. (2), Eq. (4), and Eq. (5), we have

$$ALG \geq \big(\sum_{w_i \in W} \mathsf{rev}(g^*_{w_i})\big) - \big(\sum_{w_i \in W} \mathsf{rev}(\overline{g^*_{w_i}})\big) \geq OPT - ALG$$

Finally, we can derive that the approximation ratio is 0.5. □

4. OUR INDEXING APPROACH

This section presents our indexing approach to reduce the high complexity of our simple idea. Specifically, we first define our index in Sec. 4.1, and then introduce its construction method (Sec. 4.2) and search method (Sec. 4.3). Next, we present the details of our indexing approach in Sec. 4.4. Finally, we discuss extensions to practical issues in Sec. 4.5.

4.1 Definition of Additive Tree

Basic Idea. Our index is motivated by Lemma 1.

Lemma 1. *If a request group g^+ is feasible, then any request group g, which contains a subset of requests R_{g^+}, must be also feasible.*

Proof. The statement is true since any subpath of g^+ is also feasible to serve the requests $R_g \subseteq R_{g^+}$. □

For instance, if there exists a feasible request group $g^+ = \{r_1, r_2, r_3\}$, request groups like $g_1 = \{r_1, r_2\}$, $g_2 = \{r_1, r_3\}$ are also feasible. As a result, we can generate the request group g^+ by *adding* one more request in g_1 or g_2. Accordingly, we define our index *additive tree* as follows.

Definition 8 (additive tree). *An additive tree T is an unweighted tree that satisfies the following properties:*
(1) The height of T is C, where $C = \max_{w_i} c_{w_i}$.

(2) At the k-th level, each node v represents a different request group (denoted by g(v)) which has a set of k requests. Particularly, the root represents an empty group.

(3) For each node v and any of its child node u ∈ child(v), the request group g(u) contains one more request than g(v), i.e., $R_{g(v)} \subset R_{g(u)}$ and $|R_{g(u)} \setminus R_{g(v)}| = 1$.

EXAMPLE 4. *An instance of the additive tree is illustrated in Fig. 3b, where each node represents a unique request group in our toy example. For instance, at the second level, node u_5 represents the request group g_5, i.e., $R_{g_5} = \{r_1, r_2\}$. Besides, at the third level, node u_9 is the child node of u_5 since u_9 represents a request group $R_{g_9} = \{r_1, r_2, r_3\}$, i.e., $R_{g_9} \subset R_{g_5}$ and $|R_{g_5} \setminus R_{g_9}| = 1$.*

In the following, we propose solutions to efficiently *(1)* construct the additive tree and *(2)* search the most profitable request group for a vehicle. Here we omit the deletion method since it is similar to other tree-based indexes.

4.2 Efficient Construction Method

4.2.1 Main Idea

Challenge. To construct the index, a straightforward solution is to hierarchically construct the node ($O(|G|) \gg O(n)$) and enumerating the additive requests for each node ($O(n)$). This *enumerating strategy* is inefficient since its enumerated request group is $O(n|G|) \gg O(n^2)$. Especially, many requests cannot be shared together due to the constraints.

Another issue is that the existing solutions overlook the time consumption of the *checking strategy*, *i.e.*, checking the feasibility of the enumerated request group. They usually first generate all the subpaths of origins and destinations and then check which satisfies the constraints. However, the number of possible subpaths is a large constant ($\frac{(2k_g)!}{2^{k_g}}$ [28]) in practice, where k_g is the number of requests in the request group. For instance, when $k_g = 5$, it is over 1.13×10^5.

By comparison, our following strategies are more efficient.

Enumerating Strategy. Based on the third property of the additive tree, each two sibling nodes (*e.g.*, nodes u_5 and u_6 in Fig. 3b) have exactly one different request from the other since they both have one more request than their parent node (*e.g.*, node u_1). As a result, to create the child node of the sibling nodes, we can generate its request group by joining the request set of these two sibling nodes. For instance, the requests of node u_9 in Fig. 3b can be generated by joining the request sets of nodes u_5 and u_6. Overall, our enumerating strategy is as follows.

(1) For the first level, each request group (node) is generated by a single request in the request set R.

(2) For the other level i, each request group (node) is generated by joining the request sets of any two sibling nodes at the $(i-1)$-th level.

The correctness of our strategy is a corollary of Lemma 1.

Checking Strategy. To check the feasibility of an enumerated request group, our checking strategy is as follows.

LEMMA 2. *Assume a request group g is generated by joining two feasible request groups g_1 and g_2. The request group g is feasible if the following conditions are satisfied:*

(1) $\forall r \in R_g$, any request group with requests $R_g \setminus \{r\}$ must be feasible, i.e., the corresponding node has been created.

Algorithm 3: Construct the index

```
input : the requests R and maximum capacity C
output: the index additive tree T
1 rt ← the root of T which contains no requests;
2 Create each child node u_i of rt, where R_{g(u_i)} = {r_i};
3 for size k ← 2 to the maximum capacity C do
4     U ← the nodes in T at the (k − 1)-th level;
5     foreach node u_i ∈ U do
6         foreach node u_j (j > i) of u_i 's siblings do
7             Create a request group g(v), where
                k_{g(v)} ← k, R_{g(v)} ← R_{g(u_i)} ∪ R_{g(u_j)};
8             if g(v) is feasible by Lemma 2 then
9                 v ← create a child node of u_i, which
                    represents the request group g(u);
```

(2) there exists a subpath $ps \in PS_{g_1}$ such that it is feasible to add (insert) the request r into ps, where $r \in R_{g_2} \setminus R_{g_1}$.

PROOF. The first condition is derived from Lemma 1.

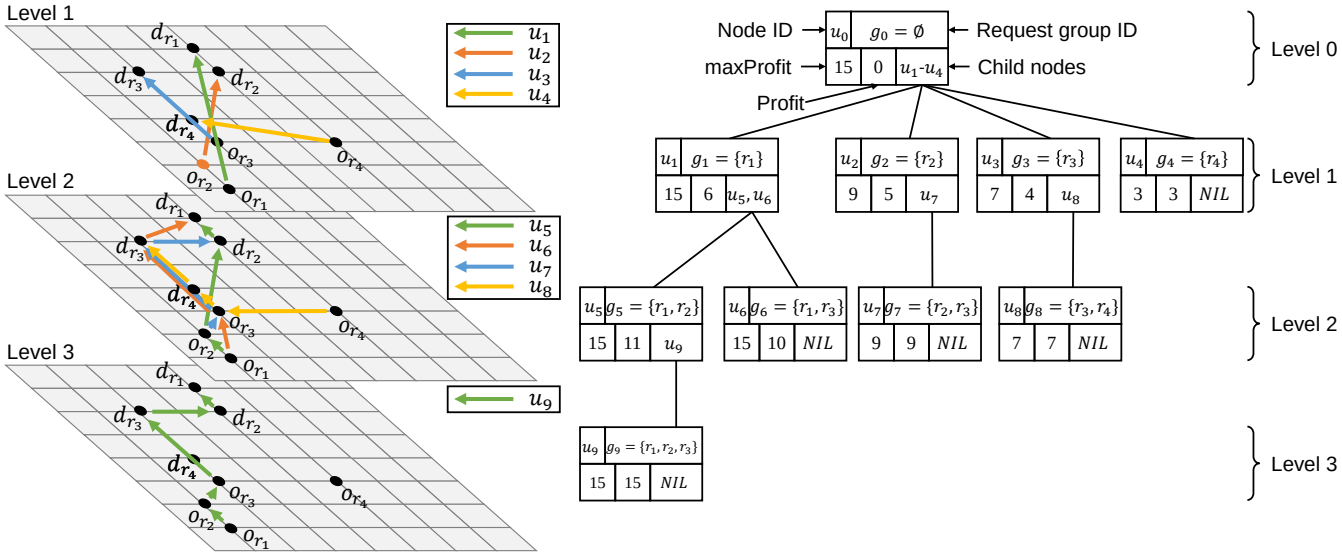
For the second condition, the request r is exactly the additive request, *i.e.*, $R_g = R_{g_1} \cup \{r\}$. Based on the definition of request group, g is feasible if there exists a subpath ps that can serve all the requests R_g. In other words, such a subpath can be also used to serve the requests $R_{g_1} \subset R_g$. Besides, the set PS_{g_1} contains all the subpaths that can serve the requests R_{g_1}. Therefore, we can generate each subpath ps by inserting the new request into each subpath $ps_1 \in PS_{g_1}$, where a new subpath is generated by putting the origin and destination of the new request into each position of ps_1. □

EXAMPLE 5. *As shown in Fig. 3b, the node u_9 ($g_9 = \{r_1, r_2, r_3\}$) is generated by joining the request sets of nodes u_6 and u_5. To test the feasibility of g_9, we first check whether request sets $\{r_1, r_2\}$, $\{r_1, r_3\}$ and $\{r_2, r_3\}$ exist (i.e., nodes u_5-u_7). We then try to insert the additive request r_2 into the subpath set PS_{g_6}. When $ps_1 = \{o_{r_1}, o_{r_3}, d_{r_3}, d_{r_1}\}$ (see Fig. 3a), the possible subpaths are $\{\boldsymbol{o_{r_2}}, \boldsymbol{d_{r_2}}, o_{r_1}, o_{r_3}, d_{r_3}, d_{r_1}\}$, $\{o_{r_1}, \boldsymbol{o_{r_2}}, \boldsymbol{d_{r_2}}, o_{r_3}, d_{r_3}, d_{r_1}\}$, $\cdots$, $\{o_{r_1}, \boldsymbol{o_{r_2}}, \boldsymbol{d_{r_2}}, o_{r_3}, d_{r_3}, d_{r_1}\}$, $\{o_{r_1}, o_{r_3}, \boldsymbol{o_{r_2}}, \boldsymbol{d_{r_2}}, d_{r_3}, d_{r_1}\}$, $\cdots$, $\{o_{r_1}, o_{r_3}, d_{r_3}, d_{r_1}, \boldsymbol{o_{r_2}}, \boldsymbol{d_{r_2}}\}$, where the origin and destination of r_2 are marked by bold.*

4.2.2 Our Construction Algorithm

Algorithm Details. Our construction method is illustrated in Algo. 3. Specifically, we create a root rt with no requests in line 1. In line 2, we create n child nodes u_i of the root, where u_i represents a request group of only one request $r_i \in R$. In lines 3-9, we hierarchically create the other nodes from top to bottom. Specifically, we use U to denote the set of nodes at the $(k-1)$-th level, *i.e.*, the nodes created in the last iteration. Then we generate the possible request groups in lines 5-7 by our *enumerating strategy*. In line 8, we test the feasibility of these request groups by our *checking strategy*. If the request group $g(v)$ is feasible, we create a child node v for its parent node u_i and update its subpath set $PS_{g(v)}$ accordingly (line 9).

EXAMPLE 6. *Back to our example ($C = 3$). We construct the additive tree in Fig. 3b to represent the request groups among r_1-r_4. We first create the root u_0 with no requests and add four child nodes u_1-u_4, where each child node contains one request in r_1-r_4. Then we create the nodes at the*



(a) The subpath of each node

(b) The detailed tree structure

Figure 3: An illustration of our index *additive tree*

levels 2-3. For example, when $k = 2$, $U = \{u_1, \cdots, u_4\}$ (line 4). We then pick $u_i = u_1$ and iterate its sibling nodes $u_j \in \{u_2, u_3, u_4\}$ (lines 5-6). As a result, we create child nodes u_5 and u_6 of u_1. At level 3, there is only one possible request group by joining the request sets of nodes u_5 and u_6. Accordingly, we create the child node u_9 of u_5. By our enumerating strategy, we can directly prune the request groups like $\{r_1, r_2, r_4\}$, $\{r_1, r_3, r_4\}$ and $\{r_2, r_3, r_4\}$.

Complexity Analysis. We use deg_i to denote the maximum degree of the nodes at the i-th level. In line 2, $deg_0 = n$ since the root has n child nodes. Line 3 has $C-1$ iterations. In each iteration, the number of nodes in U is bounded by $\prod_{i=0}^{k-2} deg_i$, which is also the number of iterations in line 5. Line 6 only has deg_{k-2} iterations since we only enumerate the sibling nodes. Since lines 7-9 take constant time, the time complexity is $O\big(\sum_{k=2}^{C}\big(deg_{k-2} \times (\prod_{i=0}^{k-2} deg_i)\big)\big)$, where $deg_0 = n$ and $deg_i < deg_{i-1}$. The space complexity is $O(\prod_{i=0}^{C-1} deg_i)$. In practice, deg_i becomes smaller than deg_{i-1} with the increase of level, because more requests are usually more difficult to be shared together.

4.3 Efficient Search Method

In our approximation solution (*i.e.*, Algo. 2 in Sec. 3), one fundamental operation is to *search the most profitable request group* for a vehicle. As each request group is represented by a node in our index, we discuss the search method in the following. Specifically, we introduce the basic concepts in Sec. 4.3.1, elaborate the main idea in Sec. 4.3.2, and present the detailed algorithm in Sec. 4.3.3

4.3.1 Preliminary

To check the feasibility of a vehicle for serving a request group, we borrow the concept of *slack time* [38] as follows.

Definition 9 (Slack Time). *Given a vehicle's route $s_w = \{l_w^0, l_w^1, \cdots, l_w^k\}$, the slack time slack_i of each location l_w^i $(i > 0)$ is defined as the maximal tolerable time for detouring between l_w^{i-1} and l_w^i while satisfying the deadline constraints of all the requests, i.e., $\mathsf{slack}_i = \min_{j \geq i}\{\mathsf{ddl}_j - \mathsf{arr}_j\}$, where arr_j is the arrival time at location l_w^j and ddl_j is the deadline of the request at location l_w^j.*

Slack time is widely used to check the violation of the deadline constraint. For instance, if the travel time between l_w^0 (*i.e.*, vehicle's initial location) and l_w^1 is no longer than the slack time slack_1, *i.e.*, $\mathsf{dist}(l_w^0, l_w^1) \leq \mathsf{slack}_1$, the deadlines of all the requests will be satisfied (*i.e.*, $\forall i > 1, \mathsf{dist}(l_w^0, l_w^1) \leq \mathsf{slack}_i$). This is because $\mathsf{slack}_1 \leq \mathsf{slack}_2 \cdots \leq \mathsf{slack}_k$ (by Definition 9). Otherwise, some request's deadline is violated.

For a request group g, we use $\mathsf{headSlack}(g, r)$ to denote the maximum slack time of the origin o_r among the subpaths PS_g whose first locations are also the origin o_r, *i.e.*,

$$\mathsf{headSlack}(g, r) = \max\{\mathsf{slack}_1 | ps \in PS_g, \mathsf{headReq}(ps) = r\}, \quad (6)$$

where $\mathsf{headReq}(ps)$ denotes the firstly picked request in ps.

4.3.2 Main Idea

To search the index, we need a *checking strategy* to test the feasibility of a vehicle for serving a request group. Besides, we also need a *pruning strategy* to accelerate the process by efficiently filtering impossible request groups.

Checking Strategy. Based on the concept of slack time, our checking strategy is summarized in Lemma 3.

Lemma 3. *A vehicle w can serve the requests R_g in the request group g if (1) $k_g \leq c_w$ and (2) there exists a request $r \in R_g$ such that $\mathsf{dist}(o_w, o_r) \leq \mathsf{headSlack}(g, r)$.*

Proof. The first condition is due to the capacity constraint. For the other condition, the vehicle w needs to serve the requests before their deadlines. Based on the definition of slack time, the travel time $\mathsf{dist}(o_w, o_r)$ must be shorter than the slack time of the firstly picked request r. Though PS_g stores all the subpaths (*i.e.*, the routes excluding the vehicle's initial location), we only need to check the maximum slack time of the origin o_r among these subpaths, whose first locations are all o_r, *i.e.*, $\mathsf{headSlack}(g, r)$. □

Pruning Strategy. In the worst case, the searching process (without pruning) has to traverse the subpaths of all the

Algorithm 4: Search the index Search

```
input : vehicle w, current node u, the currently
        most profitable node u*
1 if u.maxProfit < u*.Profit then return;
2 if vehicle w can serve g(u) by Lemma 3 then
3     if u.Profit > u*.Profit then u* ← u;
4     foreach child node v of the node u do
5         Search(w, v, u*);
```

nodes in the index. Thus, we also prune some impossible request groups to improve the efficiency by Lemma 4.

LEMMA 4. *Let $LB[r]$ to denote the travel time between the origin of request r and its nearest vehicle, i.e., $LB[r] = \min_{w_i} \mathsf{dist}(o_{w_i}, o_r)$. For each node u, we can remove every subpath $ps \in PS_{g(u)}$ such that $\mathsf{slack}_1 < LB[\mathsf{headReq}(ps)]$, where slack_1 denotes the slack time of the first location in ps. If $PS_{g(u)}$ becomes empty, we can remove the node.*

PROOF. Assume to the contrary. A vehicle w can serve the request group $g(u)$ by the subpath ps, even if $\mathsf{slack}_1 < LB[r]$, where $r = \mathsf{headReq}(ps)$. Thus, we have $\mathsf{dist}(o_w, o_r) \leq slack_1$ by the definition of slack time. Thus, $\mathsf{dist}(o_w, o_r) < LB[r]$, which contradicts the definition of $LB[r]$. □

4.3.3 Our Search Algorithm

Algorithm Details. Algo. 4 illustrates our algorithm to search the most profitable request group for a vehicle. We use u.Profit to denote the profit of node u (*i.e.*, the profit of the corresponding request group) and u.maxProfit to denote the maximum profit among all the nodes in the subtree rooted at u. We use u^* to denote the currently most profitable node during the search process. In line 1, we will stop searching the subtree of u if all the nodes in the subtree have less profit than u^*. In line 2, we check whether vehicle w can serve the current request group $g(u)$. If w cannot serve $g(u)$, it cannot serve any descendant node of u either. This is because the requests $R_{g(u)}$ are also contained in any descendant node of u. Otherwise, we may replace u^* with u (line 3) and recursively search its child nodes (lines 4-5).

EXAMPLE 7. *As shown in Fig. 3b, we want to search the most profitable request group for the vehicle w_1. Specifically, we first set u^* as the root (i.e., $u^* = u_0$) and search the subtree rooted at u_1. Since w_1 can serve the request group g_1, we further change u^* into u_1 and search the subtree rooted at u_5. Similarly, u^* will be changed into u_5 and then u_9. After that, we will search the subtree rooted at u_6. Since $u_6.maxProfit = 10$ is smaller than the current bound (i.e., $u_9.Profit = 15$), we will skip the subtree. In the end, the search algorithm will return u_9 (i.e., g_9) as the final result.*

Complexity Analysis. Both time complexity and space complexity equal to the number of nodes in the index.

4.4 Indexing-based Approximation Solution

To improve the efficiency of our simple idea, one can directly apply the construction and search methods of our index in Algo. 2 (this method is named as GAS-O1). However, we find that it becomes inefficient when there are a large number of requests in our experiments. Thus, we propose a slightly different algorithm (GAS-O2) to solve this issue.

Algorithm 5: Indexing based Solution GAS-O2

```
input : the requests R and vehicles W
output: the planned routes {s_w | w ∈ W}
1 foreach randomly picked vehicle w ∈ W do
2     R' ← filter the unassigned requests in R that
       cannot be served by w;
3     T' ← construct the additive tree of R' by Algo. 3;
4     u* ← search the most profitable node for vehicle
       w in T' by Algo. 4;
5     if such node u* exists then
6         s_w ← the fastest route to serve requests in u*;
7 return {s_w | w ∈ W};
```

Basic Idea. We still iteratively pick the most profitable request group for each vehicle, but we do not have to construct an index of *all the requests*. Instead, for each vehicle, we first filter a subset of requests that can be served by it, then construct the index of the filtered requests, and finally search the most profitable request group. Besides, we also use *randomization* to improve the approximation guarantee.

Algorithm Details. Algo. 5 illustrates the algorithm GAS-O2. In line 1, we first randomly pick a vehicle w to determine its route. Specifically, we first filter a subset R' from the currently unassigned requests (*e.g.*, by range filtering), where each request in R' can be served by the vehicle w (line 2). In line 3, we construct the additive tree T' of these requests R' by Algo. 3. We next search the most profitable node u^* in the index by Algo. 4 (line 4). If such u^* exists, we can plan the fastest route for this vehicle w (lines 5-6).

EXAMPLE 8. *Back to our example and we assume the vehicles are iterated by this order: w_1, w_2. For the vehicle w_1, we first filter the requests $R' = \{r_1, r_2, r_3\}$ since w_1 cannot serve r_4. We then construct the index for r_1-r_3, which is the tree in Fig. 3b excluding the nodes u_4, u_8. Thus, we will obtain u_9 as the most profitable node in line 4 and plan the fastest route $s_{w_1} = \{o_{w_1}, o_{r_1}, o_{r_2}, o_{r_3}, d_{r_3}, d_{r_2}, d_{r_1}\}$ in line 6. Similarly, we will obtain u_4 as the most profitable node for vehicle w_2 and plan the fastest route $s_{w_2} = \{o_{w_2}, o_{r_4}, d_{r_4}\}$.*

Complexity Analysis of GAS-O2. In Algo. 5, line 1 has $O(m)$ iterations and we denote the maximum size of R' is $n^- \ll n$. Line 3 takes $O\big(\sum_{k=2}^{C} \big(deg_{k-2}^- \times (\prod_{i=0}^{k-2} deg_i^-)\big)\big)$ time, where deg_i^- is the maximum degree of the nodes at the i-th level. Line 4 takes $O(|T^-|)$ time, where $|T^-| = O(\prod_{i=0}^{C-1} deg_i^-)$. Lines 5-6 take constant time. Overall, the time complexity is $O\big(m \sum_{k=2}^{C} \big(deg_{k-2}^- \times (\prod_{i=0}^{k-2} deg_i^-)\big)\big)$ and its space complexity is $O(|T^-|)$.

Approximation Analysis. Based on Theorem 2, Algo. 5 also has an approximation ratio of 0.5 in the worst case. However, as Algo. 5 is a randomized algorithm (*i.e.*, line 1 randomly picks a vehicle). we prove it has an (expected) approximation ratio of $1/(2 - 0.5/C)$ in Theorem 3. When $C = 2, 3$ and 4, the ratio is 0.57, 0.54 and 0.53. In other words, its approximation ratio is strictly better than 0.5.

THEOREM 3. *The expected approximation ratio of Algo. 5 is $1/(2 - 0.5/C)$.*

PROOF. We use the same notations in the proof of Theorem 2. For each vehicle w, we can still bound the total revenue of Algo. 5 based on the two cases in Theorem 2.

(1) It gets at least the same revenue as the optimal solution, *i.e.*, $\mathsf{rev}(g_{w_i}) \geq \mathsf{rev}(g^*_{w_i})$. Thus, we have $ALG \geq OPT$.

(2) It gets less revenue than the optimal solution, *i.e.*, $\mathsf{rev}(g_{w_i}) < \mathsf{rev}(g^*_{w_i})$. Based on the proof of Theorem 2, we have $ALG \geq OPT - ALG$ in this case.

In Algo. 5, since the vehicle is randomly picked, both cases will occur with some probabilities. We use X_i to denote the probability of the case (i). Accordingly, the (expected) total revenue of our algorithm can be bounded by Eq. (7).

$$\mathbb{E}[ALG] \geq X_1 \cdot OPT + X_2 \cdot (OPT - ALG) \tag{7}$$

To bound the probability X_1 ($X_2 = 1 - X_1$), we assume that the vehicles are picked by a permutation π. The revenue of a vehicle w, which is at the i-th place in π, belongs to the case (2). It indicates that at least one vehicle before w is scheduled with some requests in g^*_w. We use bi to denote the minimum rank of such vehicles in π. We can create a new permutation $\pi(i)$, where the vehicle w is removed at the position of i and all the other vehicles in π remain in their original positions. In this permutation $\pi(i)$, we know that:

(a) When $i \leq bi$, the vehicle w in $\pi(i)$ will satisfy the first case since no request in g^*_w has been assigned.

(b) When $i > bi$, the vehicle w will satisfy the second case.

In other words, an event of the second case in the permutation π corresponds to bi events of the first case in the other permutations $\pi(i)$. In the worst case, bi equals to 1. If we treat the bi of each vehicle w_i is all 1, we can only get the approximation ratio of 0.5 (*i.e.*, $X_1 = X_2$).

Therefore, we use the following fact: for each integer $j = 1, \cdots, m/C$, there are at most C vehicles whose bi equals to j. The statement is *true* since the vehicle at the position bi can be scheduled with at most C requests, while each request belongs to one distinct vehicle of the second case. Thus, we can infer the probability X_1 as follows.

$$X_1 = \underbrace{\frac{1}{m}}_{\#(\pi(i))} \times \overbrace{\frac{1}{m}}^{\#(\text{vehicles})} \times (\sum_{j=1}^{m/C} (C \times j)) = \frac{C+m}{2Cm} \tag{8}$$

By substituting Eq. (8) into Eq. (7) ($X_2 = 1 - X_1$), we can infer the (expected) approximation ratio as follows.

$$\frac{\mathbb{E}[ALG]}{OPT} = \frac{1}{2 - X_1} = \frac{1}{2 - \frac{C+m}{2Cm}} = O(\frac{1}{2 - 0.5/C}).$$

□

4.5 Extension

We also extend our methods to the following settings.

(1) The capacity of a request (*e.g.*, the number of passengers) may be larger than 1. To consider this practical issue, we use c_r to be the capacity of a request r. Thus, when checking the feasibility of a route, a subpath or a request group, the capacity of the vehicle should be no smaller than the total capacity of the requests, *i.e.*, $c_w \geq \sum_{r \in R_w} c_r$ (in Definition 3) and $C \geq \sum_{r \in R_g} c_r$ (in Lemma 2 and Lemma 3).

(2) Some work [13] also considers the constraint of *detour-ratio* for each request. For example, in a feasible route, the distance from a request's origin to its destination should be shorter than a given threshold. Our methods can also support this constraint by directly considering it in the feasibility test of a route or a subpath.

Moreover, our approximation ratios still hold in these settings, because the analysis in Theorem 2 and Theorem 3 will not change when considering the aforementioned issues.

Table 5: Statistics of the real datasets

Dataset	#(Requests)	#(Vertices)	#(Edges)
Chengdu	from 11.01 to 11.30	214,440	466,330
Haikou	from 09.01 to 09.30	42,542	89,206

Table 6: Parameter settings

Parameters	Settings
Number of requests n	700, 900, **1100**, 1300, 1500
Number of vehicles m	100, 200, **300**, 400, 500
Deadline e_r (second)	**600**, 750, 900, 1050, 1200
Vehicle's capacity c_w	2, **3**, 4, 5, 6
Scalability n	1k, 2k, 4k, 6k, $\cdots$, 20k

5. EXPERIMENTAL STUDY

In the following, we introduce our experimental setup in Sec. 5.1 and the experimental results in Sec. 5.2.

5.1 Experimental Setup

Datasets. We evaluate the proposed algorithms on two real datasets [2]. They are published by Didi Chuxing [1], the largest ridesharing company in China. The first one was collected in Chengdu in November 2016 and the other one was collected in Haikou in September 2017. Table 5 summarizes the road networks of these two cities, which are extracted from OpenStreetMap [4]. Both datasets contain 30 days of taxi requests in Didi Chuxing. Thus, we use these real-word origins and destinations, and generate the other parameters as shown in Table 6, where default settings are marked in bold. Specifically, we sample a certain number of requests (n) from the real datasets. Since these datasets do not have the information of the deadline, we set any request's deadline by adding the value in Table 6 with the shortest travel time between its origin and destination (*e.g.*, $e_r = t_r + \mathsf{dist}(o_r, d_r) + 600$ by default). The payment of each request is calculated by the pricing strategy in Didi Chuxing [48]. For the vehicles, we randomly generate their initial locations from the vertices of the road network and vary their capacities. Our parameter settings are also used in existing work (*e.g.*, [48, 38, 40, 13, 37, 44, 47, 36]). Finally, we test the scalability by varying n from $1k$ to $20k$. Since there are around $80k$ requests per day in *Haikou* dataset, the size of scalability test is up to six hours of requests, which is 133×, 432× and 1080× larger than the largest test used in [7] ($n = 150$), [5] (50s of requests) and [7] (20s of requests).

Compared Algorithms. We compare the following state-of-the-art algorithms in the experiments.

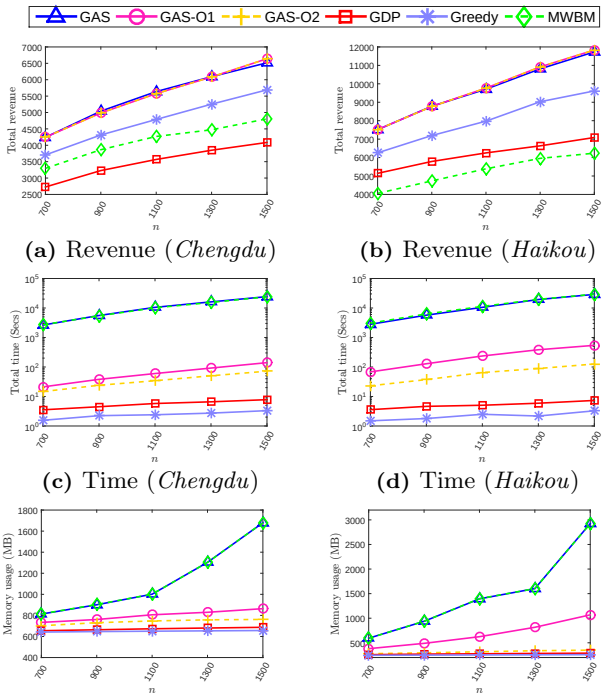
(1) GAS (this paper). It is the basic implementation of our approximation solution (*i.e.*, Algo. 2).

(2) GAS-O1 (this paper). It is the implementation of GAS by only using our index *additive tree*.

(3) GAS-O2 (this paper). It is the indexing approach (*i.e.*, Algo. 5), which applies the index and randomization.

(4) GAS-O3 (this paper). We apply a data-driven strategy to improve the time cost of GAS-O2 in scalability tests. Specifically, in line 2 of Algo. 5, we sample 4% of the requests in R' to execute the lines 3-6. We first select the top 2% of the requests that are sorted by their payments in a descending order, where the parameter 2% is fine-tuned. For each of the top 2% requests, we pick another request in R', which can be shared and has the highest payment.

(5) pruneGreedyDP [38] (GDP for short). It sequentially assigns each request to the vehicle, which has the minimum increased travel time to insert this request.

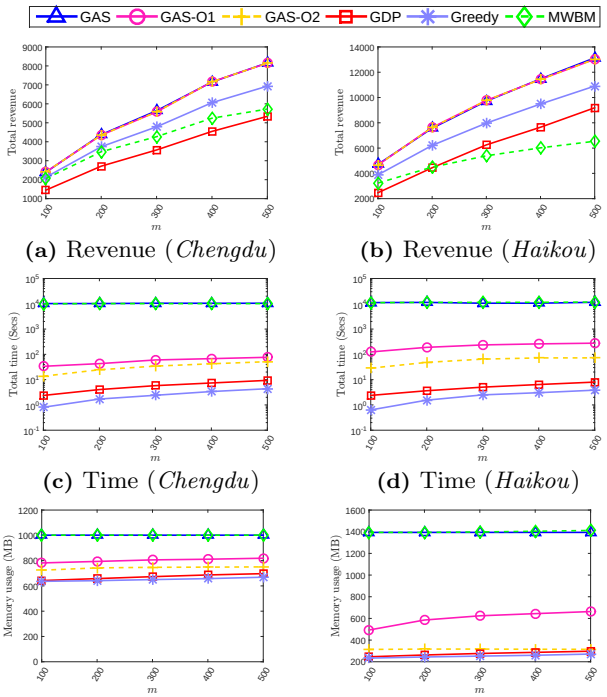


(a) Revenue (*Chengdu*) **(b)** Revenue (*Haikou*)

(c) Time (*Chengdu*) **(d)** Time (*Haikou*)

(e) Memory (*Chengdu*) **(f)** Memory (*Haikou*)

Figure 4: Performance of varying n



(a) Revenue (*Chengdu*) **(b)** Revenue (*Haikou*)

(c) Time (*Chengdu*) **(d)** Time (*Haikou*)

(e) Memory (*Chengdu*) **(f)** Memory (*Haikou*)

Figure 5: Performance of varying m

(6) Greedy [48]. It iteratively inserts the currently most profitable request to one feasible vehicle.

(7) MWBM [48]. It is the implementation of Algo. 1, which is the same as algorithm PBM in [48]. We compare with [48] instead of [30, 5, 7], since Ref. [48] is a more recent work than [30, 5] and Ref. [7] requires that any vehicle's capacity is 2. Moreover, Ref. [48] is the only collaborative work with a real industry (*i.e.*, Didi Chuxing [1]) among [48, 30, 5, 7].

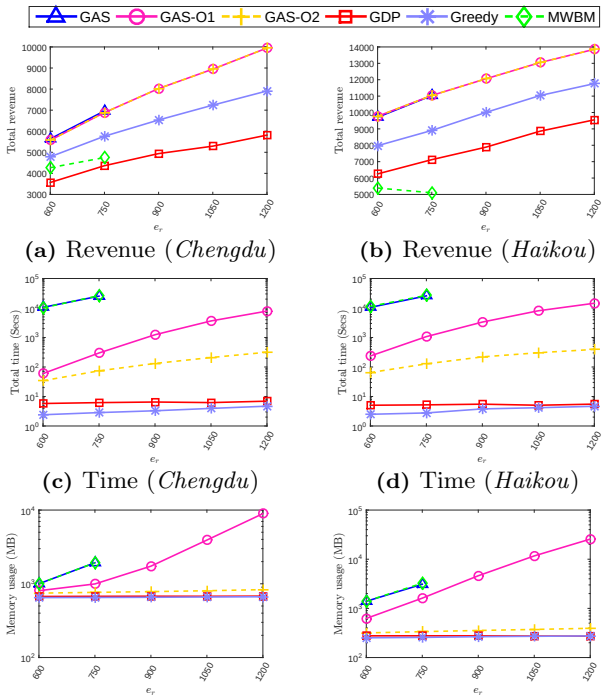
Implementation and Metrics. The experiments are conducted on a server with 40 Intel(R) E5 2.30GHz processors with 128GB memory. In each experiment, these algorithms use the same method SHP [20] to query the shortest travel time on road networks. We implement an LRU cache to maintain the results of the recent distance queries as in [13]. We also apply the grid index ($1km \times 1km$) to conduct the range filtering in these methods. For all compared algorithms, the results on different grid lengths within a practical range (*e.g.*, from 1km to 5km [40, 38, 13]) are relatively stable (see [45] for more details due to space limitations). All the algorithms are implemented in C++ and are evaluated in terms of *total revenue* ("revenue" for short), *total running time* ("time" for short) and *memory usage* ("memory" for short). Each experimental setting is repeated 30 times and the average results are reported. In some cases (*e.g.*, scalability tests), the algorithms MWBM, GAS and GAS-O1 are too inefficient in time (>10 hours) and space (>80GB) to be terminated, and hence we cannot show these results.

5.2 Experimental Result

Impact of the number of requests. Fig. 4 presents the results of varying the number of requests. Specifically, Fig. 4a and Fig. 4b illustrate the total revenue of the compared algorithms. In both datasets, our proposed algorithms GAS, GAS-O1, GAS-O2 are more effective than the existing methods. For instance, they obtain up to 89.7%, 66.9% and 23.1% higher revenue than MWBM, GDP and Greedy in the *Haikou* dataset, respectively. Greedy is more effective than other existing solutions, while MWBM is sometimes the least effective. In terms of total running time, Greedy is the most efficient, GDP is the first runner-up, and GAS-O2 is the second runner-up. Compared with the results of GAS-O1 and GAS, our index improves the running time by up to 175.4 times. MWBM is the least efficient, which is 54.2×-332.7× slower than GAS-O1 and GAS-O2. In terms of memory usage, Greedy and GDP are the most efficient, while MWBM and GAS are the least efficient.

Impact of the number of vehicles. Fig. 5 shows the results of varying the number of vehicles. In both *Chengdu* and *Haikou* datasets, our algorithms still obtain the highest total revenue, which is at least 16.6% higher than the existing methods. MWBM is still notably less effective than the others. Though both GDP and Greedy use the same insertion operator [38], Greedy is better since it inserts the more profitable request with higher priority, while GDP sequentially inserts the requests without considering their payments. As for total running time, Greedy is always the most efficient and GDP is the runner-up. GAS-O1 is up to 302.6× faster than GAS by the index. GAS-O2 is up to 4.4× faster than GAS-O1, since GAS-O2 constructs a small index for each vehicle instead of constructing a large index for all the vehicles. MWBM is still inefficient, *e.g.*, by up to 4645×, 294.6×, 727.5× slower than GDP, GAS-O1, and GAS-O2, respectively. As for memory usage, all the algorithms take no more than 1.5GB space.

Impact of the length of deadlines. Fig. 6 presents the results of varying the length of requests' deadlines. As shown in Fig. 6a and Fig. 6b, the total revenue of all the algorithms usually increases when increasing the length of deadlines. However, MWBM gets less revenue with the increase of the deadline in *Haikou* dataset. Overall, MWBM is the least effective, and our algorithms GAS, GAS-O1 and GAS-

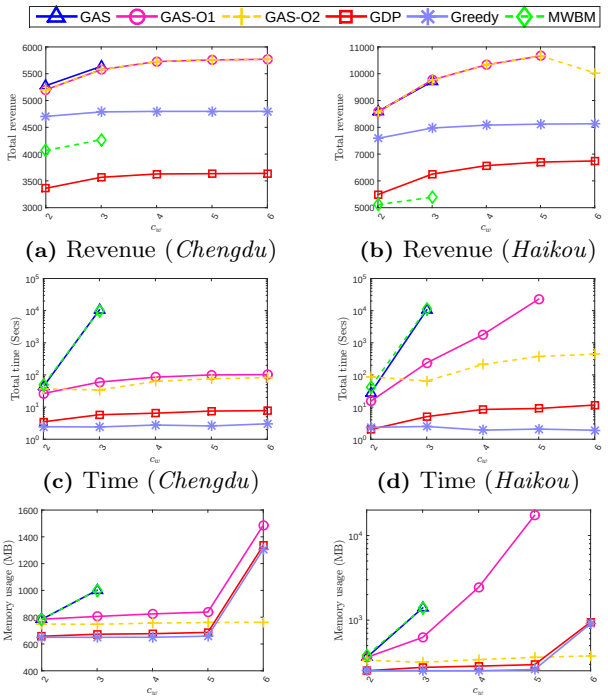


(a) Revenue (*Chengdu*) (b) Revenue (*Haikou*)

(c) Time (*Chengdu*) (d) Time (*Haikou*)

(e) Memory (*Chengdu*) (f) Memory (*Haikou*)

Figure 6: Performance of varying deadline



(a) Revenue (*Chengdu*) (b) Revenue (*Haikou*)

(c) Time (*Chengdu*) (d) Time (*Haikou*)

(e) Memory (*Chengdu*) (f) Memory (*Haikou*)

Figure 7: Performance of varying capacity

GAS-O2 are notably better than the others. For instance, our solutions obtain up to 116.4% higher total revenue than the existing algorithms in both datasets. Greedy is still more effective than GDP and MWBM. In terms of total running time, Greedy is still the most efficient, GDP is the runner-up, and MWBM is the least efficient. Besides, GAS-O1 is up to 175.4× faster than GAS and 36.7× slower than GAS-O2. As for memory cost, we observe that GAS-O1 consumes less space (by up to 1.6GB) than MWBM and GAS due to our index. Moreover, GAS-O2 has 65.2× lower memory cost than GAS-O1.

Impact of the size of capacities. Fig. 7 illustrates the results of varying the vehicles' capacities. We can observe that the total revenue increases with the increase of capacities. Our algorithms are notably more effective than the existing solutions by having up to 81.2% higher total revenue. As for time cost, both GAS and MWBM become inefficient when the vehicle's capacity becomes large. For example, MWBM and GAS are 21.1× slower than GDP in the default setting. The results are consistent with the time complexities $O(n^C)$ of their first phases, where C is the vehicle's capacity. However, by our index, they can potentially improve the time cost by up to 175.4× since GAS-O1 can be 175.4× faster than GAS. We also observe GAS-O2 is slower than GAS-O1 when $c_w = 2$. When $c_w = 2$, the time complexity of GAS-O2 is $O(2mn^-(1 + deg_1))$ and the time complexity of GAS-O1 is $O(mn^-(1 + deg_1) + n(1 + deg_1))$, where the meanings of these notations are shown in Table 3. Since $mn^- > n$, GAS-O2 is slightly slower than GAS-O1. As for memory usage, we observe similar patterns with the previous results.

Scalability tests. Fig. 8 shows the results of scalability tests. In terms of total revenue, GAS-O2, GAS-O1 and GAS have up to 31.4%, 102.8% and 127.4% higher total revenue than Greedy, MWBM and GDP, respectively. The total revenue of GAS-O3 is also notably better than existing methods, which is slightly lower than GAS-O2. As for total running time, GAS and MWBM are the least efficient methods. Greedy is still the most efficient, GDP is the first runner-up, and GAS-O3 is the second runner-up. Overall, GAS-O3 is comparatively efficient. For instance, GAS-O3 is less than 8.6× slower than GDP. As for memory cost, MWBM, GAS and GAS-O1 need extremely large spaces to store all the request groups, when there are a large number of requests. Greedy, GDP and GAS-O3 are more efficient than others. GAS-O2 consumes 80.2× less space than GAS-O1, which is efficient enough for a modern server (*e.g.*, <5GB space).

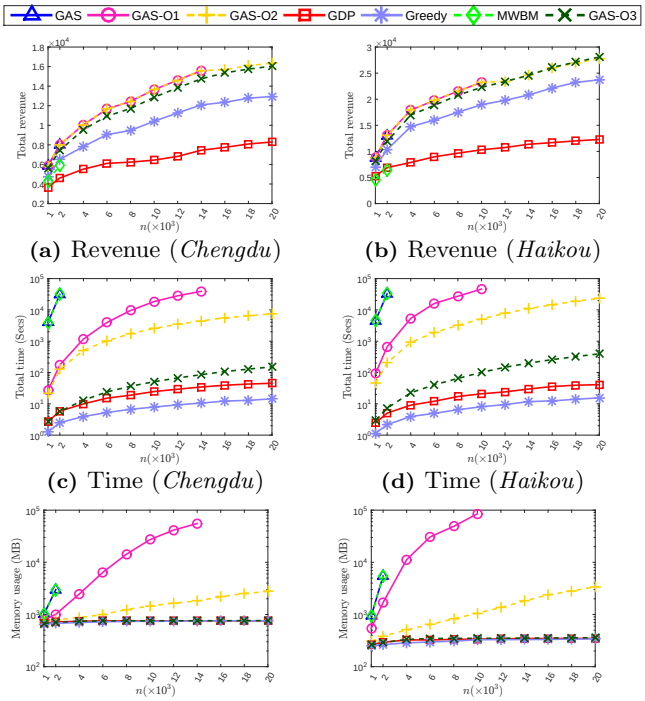
Summary of results. We summarize the major experimental results in the following.

(1) In terms of the platform's total revenue, our algorithms GAS, GAS-O1 and GAS-O2 always have higher revenue than the state-of-the-art algorithms. For instance, they have 17%-31%, 55%-127% and 31%-116% higher total revenue than Greedy, GDP and MWBM respectively. In other words, *the platform may lose a huge amount of money* by the existing methods.

(2) Greedy and GDP are more efficient than our methods, while MWBM takes the highest time cost and space cost.

(3) Our optimizations strategies are effective in improving the efficiency. For instance, GAS-O1 is up to 302.6× faster than GAS, and it also saves up to 3.2× space than GAS. GAS-O2 is up to 60.6× faster than GAS-O1, and it also saves up to 80.2× space than GAS-O1.

(4) Overall, we trade efficiency for *total revenue*, because *a higher total revenue* is always one of the most important concerns in real platforms (*e.g.*, Didi Chuxing [29, 19]), and it often indicates a higher total income of the drivers [35]. Besides, data-driven tricks can be also used in GAS-O2 to accelerate the running time. For example, GAS-O3 not only has similar total revenue as GAS-O2, but also processes six hours of requests within 6.5 minutes, which meets the real-



(a) Revenue (*Chengdu*) **(b)** Revenue (*Haikou*)

(c) Time (*Chengdu*) **(d)** Time (*Haikou*)

(e) Memory (*Chengdu*) **(f)** Memory (*Haikou*)

Figure 8: Performance of scalability tests

world requirement [12]. Parallelization can also accelerate GAS-O2, *e.g.*, implementing Algo. 3 by OpenMP [3].

6. RELATED WORK

Our paper is related to *trip planning queries in spatial databases* and *route planning in ridesharing services*.

Trip planning queries in spatial databases. The trip planning query is an important research direction in spatial databases. It usually aims to find a trip starting from a given point through multiple Point-of-Interests (PoIs), such that the users' requirement is satisfied, *e.g.*, optimal sequenced route queries [17, 31, 22, 8, 18], group trip planning queries [11, 34, 26], and route skyline queries [15, 41].

Among these problems, our paper is closely related to *group trip planning queries* [11, 34, 26] and *optimal route queries with arbitrary order constraints* [18]. The major differences are summarized as follows.

(1) They do not support the *deadline constraint*, which is widely applied in ridesharing service to ensure the passengers' user experience (see surveys [32, 12, 42, 39]).

(2) Most of them [11, 34, 26] focus on planning the trips consisting of multiple PoIs, where these PoIs can be in arbitrary orders. However, in ridesharing services, the origin of a request must exist before its destination.

(3) Though [18] considers such order constraints, its studied problem can only find *one* optimal route for *one vehicle* instead of *multiple* routes for *multiple vehicles*. Besides, it does not support the *deadline constraint* either.

Thus, these methods cannot be applied to our problem.

Route planning in ridesharing services. In recent years, ridesharing services have been studied in many communities (*e.g.*, database, data mining and AI). One of the fundamental problems in these studies is route planning, *i.e.*, planning a shared-route for each vehicle to optimize certain objectives. For instance, [24, 13, 25] focus on maximizing the total number of served requests, which is a special case of our objective (*i.e.*, each request has a payment of 1). [38, 6, 48, 49] focus on maximizing the platform's total revenue, and [46, 7] aim to minimize the travel cost. Our SRPQ problem considers both the total revenue (*i.e.*, the objective) and the travel cost (*i.e.*, the shortest travel time constraint).

Moreover, their solutions can be classified into two kinds: *insertion-based* [24, 13, 25, 38] and *grouping-based* [46, 6, 7, 48, 49]. The insertion operation was first proposed in [14], which finds a new route by adding (inserting) a new request into the current route of a vehicle. Though insertion-based algorithms are mostly heuristics, they have been tested to be effective and efficient in real datasets [38, 40]. Grouping-based solutions usually first generate a set of request group and then assign each group to a suitable vehicle, *i.e.*, the two-phase framework in Sec. 3.1. Compared with insertion-based algorithms (*e.g.*, Greedy and GDP in our experiments), the grouping-based solutions (*e.g.*, MWBM in our experiments) are less efficient but likely to have theoretical guarantees. Overall, none of these methods have constant approximation ratios for our SRPQ problem.

There are some other important objectives, including maximizing the social utility [9], maximizing the shared-route percentage [33], minimizing the requester's waiting time [43], maximizing the satisfaction rate of requesters while minimizing the distance of vehicles [23], and balancing the efficiency and fairness [16].

7. CONCLUSION

In this paper, we study the shared-route planning queries (SRPQ) problem in ridesharing services. Though existing methods can solve this problem, they are not effective enough in either theoretical study or empirical study. Thus, we focus on designing efficient solutions with constant approximation guarantees. Specifically, our main idea is to iteratively search the most profitable route among the unassigned requests for each vehicle, which is much simpler than the existing ones. Unexpectedly, we prove that it has an approximation ratio of 0.5. Furthermore, we also design an index called *additive tree* to improve the efficiency and apply the randomization technique to further improve the approximation ratio. Finally, experimental results on real datasets demonstrate that our indexing approach always outperforms the state-of-the-art algorithms in terms of effectiveness (*i.e.*, the platform's total revenue) by a large margin.

8. ACKNOWLEDGMENTS

We are grateful to anonymous reviewers for their constructive comments. This work is partially supported by the National Key Research and Development Program of China under Grant No. 2018AAA0101100. Yuxiang Zeng and Lei Chen's works are partially supported by the Hong Kong RGC GRF Project 16209519, CRF Project C6030-18G, C1031-18G, C5026-18G, AOE Project AoE/E-603/18, China NSFC No. 61729201, Guangdong Basic and Applied Basic Research Foundation 2019B151530001, Hong Kong ITC ITF grants ITS/044/18FX and ITS/470/18FX, Microsoft Research Asia Collaborative Research Grant, Didi-HKUST joint research lab project, and Wechat and Webank Research Grants. Yongxin Tong and Yuguang Song's works are partially supported by the National Science Foundation of China (NSFC) under Grant No. 61822201 and U1811463. Yongxin Tong and Lei Chen are the corresponding authors in this paper.

9. REFERENCES

[1] Didi Chuxing. http://www.didichuxing.com/, 2020.

[2] GAIA. https://outreach.didichuxing.com/research/opendata/, 2020.

[3] OpenMP. https://www.openmp.org/, 2020.

[4] OpenStreetMap. http://www.openstreetmap.com/, 2020.

[5] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 114(3):462–467, 2017.

[6] M. Asghari, D. Deng, C. Shahabi, U. Demiryurek, and Y. Li. Price-aware real-time ride-sharing at scale: an auction-based approach. In *SIGSPATIAL*, page 3, 2016.

[7] X. Bei and S. Zhang. Algorithms for trip-vehicle assignment in ride-sharing. In *AAAI*, pages 3–9, 2018.

[8] X. Cao, L. Chen, G. Cong, and X. Xiao. Keyword-aware optimal route search. *PVLDB*, 5(11):1136–1147, 2012.

[9] X. Fu, C. Zhang, H. Lu, and J. Xu. Efficient matching of offers and requests in social-aware ridesharing. *GeoInformatica*, 23(4):559–589, 2019.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[11] T. Hashem, T. Hashem, M. E. Ali, and L. Kulik. Group trip planning queries in spatial databases. In *SSTD*, pages 259–276, 2013.

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

[13] Y. Huang, F. Bastani, R. Jin, and X. S. Wang. Large scale real-time ridesharing with service guarantee on road networks. *PVLDB*, 7(14):2017–2028, 2014.

[14] J.-J. Jaw, A. R. Odoni, H. N. Psaraftis, and N. H. Wilson. A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research Part B: Methodological*, 20(3):243–257, 1986.

[15] H. Kriegel, M. Renz, and M. Schubert. Route skyline queries: A multi-preference path planning approach. In *ICDE*, pages 261–272, 2010.

[16] N. S. Lesmana, X. Zhang, and X. Bei. Balancing efficiency and fairness in on-demand ridesourcing. In *NeurIPS*, pages 5310–5320, 2019.

[17] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S. Teng. On trip planning queries in spatial databases. In *SSTD*, pages 273–290, 2005.

[18] J. Li, Y. D. Yang, and N. Mamoulis. Optimal route queries with arbitrary order constraints. *IEEE Transactions on Knowledge and Data Engineering*, 25(5):1097–1110, 2013.

[19] M. Li, Z. Qin, Y. Jiao, Y. Yang, J. Wang, C. Wang, G. Wu, and J. Ye. Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning. In *WWW*, pages 983–994, 2019.

[20] Y. Li, L. H. U, M. L. Yiu, and N. M. Kou. An experimental study on hub labeling based shortest path algorithms. *PVLDB*, 11(4):445–457, 2017.

[21] Y. Li, J. Wan, R. Chen, J. Xu, X. Fu, H. Gu, P. Lv, and M. Xu. Top-k vehicle matching in social ridesharing: A price-aware approach. *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[22] H. Liu, C. Jin, B. Yang, and A. Zhou. Finding top-k optimal sequenced routes. In *ICDE*, pages 569–580, 2018.

[23] H. Luo, Z. Bao, F. Choudhury, and S. Culpepper. Dynamic ridesharing in peak travel periods. *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[24] S. Ma, Y. Zheng, and O. Wolfson. T-share: A large-scale dynamic taxi ridesharing service. In *ICDE*, pages 410–421, 2013.

[25] 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.

[26] M. T. Mahin and T. Hashem. Activity-aware ridesharing group trip planning queries for flexible pois. *ACM Transactions on Spatial Algorithms and Systems*, 5(3):1–41, 2019.

[27] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38, 1957.

[28] J. Pan, G. Li, and J. Hu. Ridesharing: Simulator, benchmark, and evaluation. *PVLDB*, 12(10):1085–1098, 2019.

[29] Z. T. Qin, J. Tang, and J. Ye. Deep reinforcement learning with applications in transportation. In *KDD*, pages 3201–3202, 2019.

[30] 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*, 111(37):13290–13294, 2014.

[31] M. Sharifzadeh, M. R. Kolahdouzan, and C. Shahabi. The optimal sequenced route query. *The VLDB Journal*, 17(4):765–787, 2008.

[32] B. Shen, Y. Huang, and Y. Zhao. Dynamic ridesharing. *Sigspatial Special*, 7(3):3–10, 2016.

[33] N. Ta, G. Li, T. Zhao, J. Feng, H. Ma, and Z. Gong. An efficient ride-sharing framework for maximizing shared route. *IEEE Transactions on Knowledge and Data Engineering*, 30(2):219–233, 2018.

[34] A. Tabassum, S. Barua, T. Hashem, and T. Chowdhury. Dynamic group trip planning queries in spatial databases. In *SSDBM*, pages 38:1–38:6, 2017.

[35] X. Tang, Z. T. Qin, F. Zhang, Z. Wang, Z. Xu, Y. Ma, H. Zhu, and J. Ye. A deep value-network based approach for multi-driver order dispatching. In *KDD*, pages 1780–1790, 2019.

[36] Q. Tao, Y. Zeng, Z. Zhou, Y. Tong, L. Chen, and K. Xu. Multi-worker-aware task planning in real-time spatial crowdsourcing. In *DASFAA*, pages 301–317, 2018.

[37] Y. Tong, Y. Zeng, B. Ding, L. Wang, and L. Chen. Two-sided online micro-task assignment in spatial

crowdsourcing. *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[38] Y. Tong, Y. Zeng, Z. Zhou, L. Chen, J. Ye, and K. Xu. A unified approach to route planning for shared mobility. *PVLDB*, 11(11):1633–1646, 2018.

[39] Y. Tong, Z. Zhou, Y. Zeng, L. Chen, and C. Shahabi. Spatial crowdsourcing: a survey. *The VLDB Journal*, 29(1):217–250, 2020.

[40] Y. Xu, Y. Tong, Y. Shi, Q. Tao, K. Xu, and W. Li. An efficient insertion operator in dynamic ridesharing services. In *ICDE*, pages 1022–1033, 2019.

[41] B. Yang, C. Guo, C. S. Jensen, M. Kaul, and S. Shang. Stochastic skyline route planning under time-varying uncertainty. In *ICDE*, pages 136–147, 2014.

[42] X. Yi, T. Yongxin, and L. Wei. Recent progress in large-scale ridesharing algorithms. *Journal of Computer Research and Development*, 57(1):32–52, 2020.

[43] Y. Zeng, Y. Tong, and L. Chen. Last-mile delivery made practical: An efficient route planning framework with theoretical guarantees. *PVLDB*, 13(3):320–333, 2019.

[44] Y. Zeng, Y. Tong, L. Chen, and Z. Zhou. Latency-oriented task completion via spatial crowdsourcing. In *ICDE*, pages 317–328, 2018.

[45] Y. Zeng, Y. Tong, Y. Song, and L. Chen. The simpler the better: An indexing approach for shared-route planning queries in ridesharing (full paper). `http://home.cse.ust.hk/%7Eyzengal/simple.pdf`, 2020.

[46] L. Zhang, Z. Ye, K. Xiao, and B. Jin. A parallel simulated annealing enhancement of the optimal-matching heuristic for ridesharing. In *ICDM*, pages 906–915, 2019.

[47] B. Zhao, P. Xu, Y. Shi, Y. Tong, Z. Zhou, and Y. Zeng. Preference-aware task assignment in on-demand taxi dispatching: An online stable matching approach. In *AAAI*, pages 2245–2252, 2019.

[48] L. Zheng, L. Chen, and J. Ye. Order dispatch in price-aware ridesharing. *PVLDB*, 11(8):853–865, 2018.

[49] L. Zheng, P. Cheng, and L. Chen. Auction-based order dispatch and pricing in ridesharing. In *ICDE*, pages 1034–1045, 2019.