A Greedy Approach for Carpool Scheduling Optimization in Smart Cities

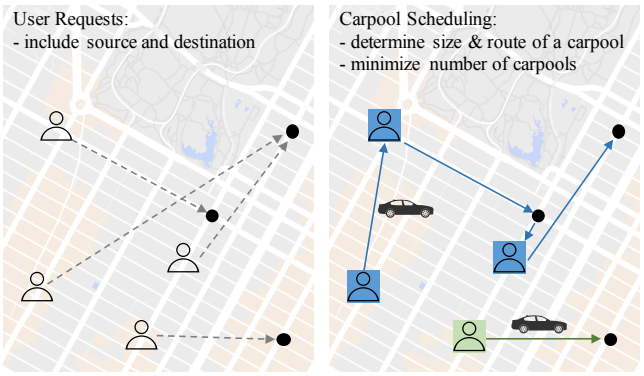
Yubin Duan, Jie Wu, and Huanyang Zheng

Center for Networked Computing, Temple University, USA

ARTICLE HISTORY
Compiled September 18, 2018

ABSTRACT

Nowadays, large cities, especially metropolitan areas, face numerous problems caused by the rapidly increasing number of vehicles on the road. Several researchers have shown that carpooling can be an efficient solution to the traffic pressure caused by large numbers of cars. The objective of our carpool scheduling problem is to minimize the number of carpools needed to transport all users. Previous research on the similar topic introduces static capacity constraint to the problem, which limits the carpool size to the vehicle's capacity. However, it is not necessary since a seat in the vehicle can be occupied by several passengers only if their routes are not overlapped. In this paper, we remove the static capacity constraint, and doing so allows a vehicle to carry more passengers than its capacity. We propose a greedy approach based on iterative matching and merging. Specifically, starting from a set of single-user carpools, the algorithm iteratively checks the merge-ability of each pair of carpools, and then applies the maximum matching algorithm to maximize the number of carpools to be merged.In addition, the merge-ability checking process is carefully studied and two methods are proposed to reduce the algorithm's time complexity. Furthermore, we improve the time efficiency of our algorithms by taking advantage of geometry properties. We apply our algorithms to both synthetic and real-world datasets, and experiment results show that our algorithms have better performances than existing approaches.



KEYWORDS
Carpool problem, greedy algorithm, cluster merging, maximum matching.

CONTACT Yubin Duan. Email: yubin.duan@temple.edu

1. Introduction

In recent years, the number of private vehicles on streets has skyrocketed, leading to numerous problems in cities, and especially in metropolitan areas. According to a study by Meyer et al. [1], the global car population is projected to reach 2.8 billion by 2050. The rapid increase of vehicles has led to environmental, economic, and social problems, including increased carbon emission, travel costs, and congestion [2]. To relieve the pressure caused by increasing demands for cars for transportation, carpools were proposed. The non-hoursehold carpool, which allows two or more commuters from different residences to travel together in the same private vehicle, reduces the number of single-occupied vehicles needed per journey [3]. According to research conducted by Roxana J. Javid et al. [4], under a hypothetical scenario where high-occupancy vehicle lanes (also known as carpool lanes) that encourage carpool are increased, the annual reduction in the CO_2e emissions of the 50 U.S states and the District of Columbia may reach up to 1.83 million metric tons. Cathy Wu et al. [5] has studied carpool algorithms for 3+ high-occupancy vehicle lanes. In addition to reducing environmental impact, carpooling also reduces the economic burdens of users. Driven by the advantages of carpooling, this paper proposes a carpool scheduling algorithm that could be used in carpool assignment procedures.

Given the starting points, destinations, the vehicle capacity constraint, the detour limitation, and other user requirements, our carpool scheduling problem is to select a minimum number of drivers who can serve all the user requests (without breaking anyone's requirement) and to calculate the service order (i.e pick up and drop off order) for each driver. Several researchers have addressed carpool problems or the similar taxi-sharing problems [6]. The carpooling problem with additional constraints has been classified as an NP-hard problem [7]. Additional constraint here means that when a passenger is picked up by a driver, the capacity of the driver's vehicle is filled by the same people for the whole trip, i.e., even if the passenger has arrived at his destination, the seat assigned to the passenger will not be released.

We use Fig. 1 to illustrate the process of carpool scheduling for a group of users: $\{s_1, d_1\}, \{s_2, d_2\}, \{s_3, d_3\}$, and $\{s_4, d_4\}$ represent the start and destination points of users p_1, p_2, p_3 and p_4, respectively. The numbers on edges show the distance between vertices. Assume that each user has a car with a seating capacity of 2, and all of them are willing to share their cars if it does not involve a detour of more than 20% of the shortest path from their starting points to corresponding destinations. To demonstrate the additional constraint in the previous problem, suppose p_4 is appointed as a driver and p_1 is in the same carpool with p_4. p_4 will start from s_4 and pick up p_1. After p_4 drops off p_1 at d_1, there will be one seat available in p_4's car. In this situation, p_4 actually could pick up more users only if the detour constraint is not violated.

However, with the additional constraint, the seat capability will still be filled even after p_1 is dropped off, and there will be no chance for p_2 and p_3 to join the carpool. Therefore, under this constraint, users will be divided into disjoint carpools where the size of each carpool is at most 2. As a consequence, the minimum number of carpools needed is 3 in the example. A possible arrangement of carpools could be: let p_4 be the driver and deliver p_1 from s_1 to d_1, and other users p_2, p_3 go with their own car. The number of carpools cannot be further reduced, since there is no way to build a carpool which contains any two users among p_1, p_2 and p_3. Otherwise, the detour constraint will be violated.

However, the additional constraint in the previous problem is unreasonable in reality since it is possible that drivers could take extra passengers while satisfying the detour

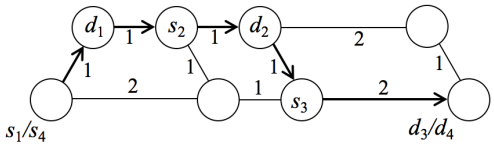


Figure 1. An illustration of the carpool scheduling problem scenario.

constraint. Use the same example in Fig. 1, the driver p_4 could transport both p_1, p_2 and p_3 to their destination under the 20% detour constraint. A possible path is shown in the figure, i.e. $s_4(s_1) \rightarrow d_1 \rightarrow s_2 \rightarrow d_2 \rightarrow s_3 \rightarrow d_4(d_3)$. The detour of p_4 is $6 - 5 = 1$, which is exactly 20% of the length of the shortest path from s_4 to d_4. We can find out that, without this constraint, the minimum number of drivers needed in Fig. 1 is 1. Therefore, the static constraint limits further optimization of carpool scheduling problems. Without this additional constraint the carpool problem becomes even harder since the search space is enlarged. A more formal proof of the NP-hardness of our problem will be introduced later.

Our main contributions are summarized as follows:

- The carpool problem without the previous additional constraint is addressed and analyzed, and we prove that our proposed problem is NP-hard.
- We provide a partition merging based on greedy algorithms for the carpool scheduling. It can reduce the number of carpools significantly.
- We further introduce two variations to improve time efficiency of the proposed algorithm. The first kind of variation focuses on merge-ability checking process and the other takes advantage of geometry properties.
- Experiments on both synthetic and real-world data are set up and validate the superiority of our algorithm over existing carpool scheduling algorithms in terms of total carpool numbers.

The remainder of the paper is organized as follows. Section 2 surveys related works. Section 3 describes our model, formulates and analyzes the problem. Section 4 proposes a greedy algorithm and its improvements. Section 5 includes the experiments. Finally, Section 6 concludes the paper.

2. Related Work

Several researches have been done on the carpool scheduling and related problems, including [8–17]. Significant results are reviewed in the rest of this section.

Based on different objectives, the carpool problem has several variations. Agatz et al. [8] and Amey [9] focused on minimizing the miles of vehicles participated in the carpool. Ghoseiri et al. [10] and Xing et al. [11] proposed to maximize the number of participants of carpool. Baldacci et al. [13] jointly considered milage cost and number of participants. They proposed to minimize the sum of the penalty for unserviced clients and the overall path costs. Different from their objectives, we aim at minimizing the number of carpools needed to carry all users, which also minimizes the number of private cars used in the carpool system. Our perspective is more useful when trying to release the traffic pressure caused by the huge amount of vehicles.

The carpool problems with the objective to minimize the number of cars used is considered by Buchholz et al. [12]. Although they share the same objective with our

Table 1. Table of Notations.

Notations	Description
$P = \{p_1, \ldots, p_n\}$	The user set
σ_i	The detour limitation of p_i
λ_i	The capacity limitation of p_i's vehicle
s_i / d_i	Source/Destination of user p_i
$C = \{c_1, \ldots, c_m\}$	The carpool set
r_i	The path for c_i
κ_r	The occupation of the path r

approach, Buchholz et al. [12] introduced the additional constraint that the size of carpool cannot exceed the capacity limitation. As a solution, they presented the Strict Partitioning Algorithm (SPA) which divides users into sets of k-partitions. Based on the additional constraint, the number of users in each partition is no more than k. They prove that the NP-hardness of their problem for $k \geq 3$. Besides, they propose an $O(n^2)$ solution for the simpler case when $k = 2$. As we mention in section 1, the additional constraint in their problem is not necessary. Also, the SPA only works for the $k = 2$ case, which means their solution allows only one person to share the car with the driver. In the paper, we remove the additional constraint and propose a heuristic algorithm which can deal with the case with any arbitrary positive k.

Another related topic is the taxi sharing problem. Its main difference from our carpool scheduling problem is that the requests of users are not static. It is to say, the route schedules cannot be computed in advance. Santi et al. [14] and Zhang et al. [15] addressed taxi-sharing problems. A method similar to [12] for taxi sharing is used by Santi et al. in [14]. Specifically, a share-ability network between individual passenger trips is built and they try to merge the trips based on the spatial and temporal proximity between them. According to their algorithm, a taxi with a seat capacity of 2 can combine at most k ($k > 2$) trips if there is no overlap between them. Their algorithm effectively overcomes the limitations of SPA. However, they assumed that a taxi is available anytime and anywhere. The model considers the start and end time of each trip along with the coordinates and builds a hyper graph to express the share-ability of different trips. The complexity of building such a share-ability network is $O(k!)$ because of checking all possible k-combinations of trips. Therefore, the proposed model is not scalable beyond $k = 3$. In our model, we consider individual trips as a primitive set of single clusters and merge them in different rounds. In each round, we consider all pairs (not k-combination) of clusters and merge them, if possible. Therefore, we do not need any hyper graphs and the complexity is significantly lower than [14].

3. Problem Formulation and Analysis

In this section, we first propose the model and mathematical formulation of our carpool scheduling problem, then we prove the np-hardness of the proposed problem.

3.1. Model and Problem Formulation

In our model, users are defined as a set of people $P = \{p_1, p_2, \ldots, p_n\}$, where each person can be either a driver or a passenger. The driver in carpool should be willing to share his vehicles, and the passengers should be glad to carpool with other users. Each

user is associated with a maximal acceptable detour distance σ_i, a seating capacity for his vehicle λ_i, a starting point s_i and a destination d_i. The set of s_i and d_i is denoted by S and D, respectively. A carpool c consists of a group of users who are going to travel in the same vehicle. Formally, $c \in P^*$, where $P^* = \bigcup_{i=0}^{|P|} P^i = P \cup P^2 \cup \ldots \cup P^{|P|}$. Let $C = \{c_i\}$ denote the set of all possible carpools. The starting points of the users in carpool c form a sub-set $S_c \subseteq S$, and the destinations form a sub-set $D_c \subseteq D$. Each carpool c_i will be associated with a driver and a path r, since our goal is not only minimizing the amount of carpools, but also providing a possible path with an appointed driver to achieve this amount. The path r_i is defined as an ordered set of locations (including both starting points and destinations). Formally, $r = (l_1, l_2, \ldots, l_m)$, where $\forall\, l_i \in r\ :\ l_i \in S_c \cup D_c$. The order in r represents sequence of visiting each location. Associated with two locations l_i and l_j in the path, we use $\delta_{i,j}$ to record the detour distance (extra distance compared with travel from l_i to l_j directly) between location l_i and l_j in path, and $\delta_{i,j} = \sum_{k=i}^{j-1} f(l_k, l_{k+1}) - f(l_i, l_j)$. The function $f(l_i, l_j) : r^2 \mapsto \mathbb{R}_+$ is used to calculate the mileages between two locations; this can be calculated based on actual map information or simply set to Euclidean distance using coordinates. Besides the length, we use κ_r to denote the current occupancy of the path, which is defined as $\kappa_r = |r \cap S_c| - |r \cap D_c|$.

Considering the constraints in the carpool scheduling problem, a path r is admissible for a carpool c with user p_γ as the driver if the following constraints are satisfied :

(1) Order constraint: the first element in r should be the starting point of p_γ (s_γ) and the final element should be the destination of p_γ (d_γ). Any other user's starting point should appear before the corresponding destination in r. Formally, the constraint can be expressed as $l_1 = s_\gamma$, $l_m = d_\gamma$, and $\forall\, p_i \in c\ :\ k < k'$, if $l_k = s_i$ and $l_{k'} = d_i$.
(2) Detour constraint: the detour limitation of each user in carpool c should be satisfied. Formally, $\forall\, p_i \in c\ :\ \delta_{k,k'} \leq \sigma_i$, if $l_k = s_i$ and $l_{k'} = d_i$.
(3) Capacity constraint: at any instant the number of users in the car may not exceed k_d, i.e., $\forall r' \subseteq r\ :\ \kappa_{r'} \leq \lambda_\gamma$.
(4) Inclusion constraint: all starting points and destinations in carpool c should be included in path r to make sure that the transportation demands of users in c are satisfied. Formally, $|r| = 2|c|$, since each user in c has one origin plus one destination.

Two carpools c_i and c_j are mergeable if at least one admissible path can be found for carpool $c_i \cup c_j$.

Based on these definitions, we formulate our problem as:

$$
\begin{aligned}
\text{minimize} \quad & |C| && (1)\\
\text{subject to} \quad & p_j = p_{j'} \text{ if } l_1 = s_j \text{ and } l_{|r_i|} = d_{j'} && (2)\\
& k < k' \text{ if } l_k = s_j \text{ and } l_{k'} = d_j && (3)\\
& \delta_{k,k'} \leq \sigma_j \text{ if } l_k = s_j \text{ and } l_{k'} = d_j && (4)\\
& \kappa_{r'} \leq \lambda_\gamma \text{ for } \forall r' \subseteq r_i && (5)\\
& |r_i| = 2|c_i| && (6)\\
& \text{for } \forall c_i \in C,\ \forall p_j, p_{j'} \in c_i,\ \forall l_k, l_{k'} \in r_i
\end{aligned}
$$

3.2. Problem Hardness

Under a special case, our Carpool Scheduling Problem (CSP) is equivalent to the well known traveling salesman problem (TSP) [18]. The TSP is proven to be NP-hard, and therefore, we can prove that our problem is also NP-hard.

Theorem 3.1. *The Carpool Scheduling Problem is NP-hard.*

Proof. The proof is done by revealing the equivalence of a special case of the CSP and the TSP [12]. Given a set of cities and a home city, the TSP aims to select the minimum distance path that starts and ends at home to cover all given cities. We start by converting the input of the TSP to the input of the CSP. The salesman can be considered the driver of a car with a seat capacity of 2, and each city can be seen as a passenger with the same starting and ending point. Then we set detour of passengers to 0. The minimum cost path (or the minimum detour path) that can cover all passengers is the optimal solution of the TSP. Instead of finding an optimal path by visiting each city, we now try to find any path that has its length bounded by a given upper limit. This problem is equivalent to the CSP if the limit of the maximum path length is set as the maximum allowable detour. This variation remains NP-hard. Therefore, it is proven that the CSP is NP-hard. □

4. Algorithmic Design

In this section, we first introduce our greedy approach for carpool scheduling problem. Then algorithm variations are introduced based on different strategy used in merging process. Furthermore, proposed algorithms are improved by taking advantage of geometry properties.

4.1. Partition Merging Algorithm

This subsection presents the Partition Merging Algorithm (PMA). Observe the example shown in Fig. 2. Strict Partition Algorithm (SPA) applies matching algorithm to construct carpools, but it only contains one round of matching. SPA can make sure the number of users in each carpool won't exceed the capacity limitation of drivers, but it ignores the fact that seat occupation will be released when users are dropped-off at their destinations. That is to say, once SPA calculates a possible arrangement of carpools, it stops and ignores the merge-ability between these carpools. Simply speaking, SPA only considers the merge-ability of users, which are one-user-carpools, instead of the merge-ability of multi-users-carpools. Therefore, PMA, an algorithm considering multi-round matching, is proposed.

Specifically, PMA is a greedy algorithm based on carpool graph G. After initializing the graph G, PMA will try to merge as many vertices as possible in each round until there are no edges in G, i.e., no more carpools can merge.

More specifically, as shown in Algorithm 1, PMA will first initialize the set of carpools C and then set each carpool in C to a carpool with only one user (line 1). Then, this set of carpools will be treated as the set of vertices for graph G. That is to say, $|P|$ vertices in total will be set. Then, unweighted undirected edges e_{ij} will be built if an admissible path can be found between carpools c_i and c_j. The existence of an edge between vertices indicates both users in c_i and c_j can travel together within the same carpool (line 2). When initialization is finished, PMA plans the merge so that maximal

Algorithm 1 Partition Merging Algorithm (PMA)

Input: A set of users $P = \{p_1\}$ associate with the starting points set $\{s_i\}$, the destinations set $\{d_i\}$, the detour limitation set $\{\sigma_i\}$ and the capacity limitation set $\{\lambda_i\}$

Output: A set of carpools C.

1: Initialize $C \leftarrow \big\{\{p_1\}, \{p_2\}, \ldots, \{p_{|P|}\}\big\}$.
2: Initialize graph edge set E. Build edge (c_i, c_j) iff c_i and c_j are mergeable, for $\forall c_i, c_j \in C$. Set $G \leftarrow (G, E)$
3: **repeat**
4: $E_M \leftarrow$ maximum matching of G.
5: Merge c_i and c_j if edge $(c_i, c_j) \in E_M$, for $\forall c_i, c_j$. Update C. Reset $E \leftarrow \varnothing$
6: **for** $\forall c_i, c_j \in C$ **do**
7: **for** $\forall p_k \in c_i \cup c_j$ **do**
8: Initialize a partial order set $S \leftarrow S_{c_i} \cup D_{c_i} \cup S_{c_j} \cup D_{c_j}$. Initialize partial order relationship based on order constraint.
9: $R \leftarrow$ all topological sort of S
10: **if** $\exists r \in R$ satisfy capacity constraint and detour constraint **then**
11: build edge (c_i, c_j) in E
12: **until** $E = \varnothing$.
13: **return** C as the final carpool-set.

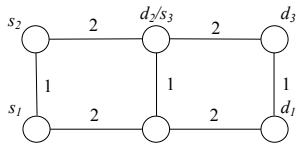


Figure 2. The example used to illustrate the Partition Merging Algorithm.

number of merges can be achieved by applying an efficient maximum matching algorithm [19] (lines 4 and 5). It is true that other efficient maximum matching algorithms like [20] can be used here. After matching and merging, graph G shrinks and contains fewer vertices, i.e., C can be updated with fewer carpools. Once a new set of carpools is found, the merge-ability of the carpools will change and must be recalculated (from line 6 to line 11). Repeating the steps mentioned above, the number of vertices in G decreases until no edge can be created; this means that the carpools cannot be merged any further, and a local optimal is reached. Finally, PMA returns C as the final result (line 13). Fig. 3 shows the merging situation in each round of PMA based on the scenario shown in Fig. 2. At the beginning of PMA, there are 3 one-user-carpools and $\{p_1\}$ and $\{p_2\}$ are mergeable. In the second round, even if carpool $\{p_1, p_2\}$ already contains 2 users (which is the capacity limitation of all users), it is still mergeable with $\{p_3\}$. They are mergeable because an admissible path can be found for carpool $\{p_1, p_2\} \cup \{p_3\}$ as shown in Fig. 4(b).

The merge-ability check process tries to find at least one admissible path to indicate merge-ability. Roughly speaking, it is a search algorithm that checks each possible path using the origins and destinations of users in carpool $c_i \cup c_j$ until an admissible one is found. However, checking all possible sequences with n nodes will cost $O(n!)$ time, which is not acceptable. Therefore, we use the order constraint of the problem to reduce the search space. Considering that the order constraint requires that the

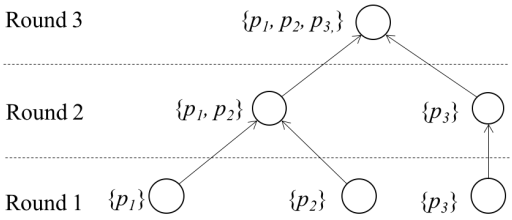
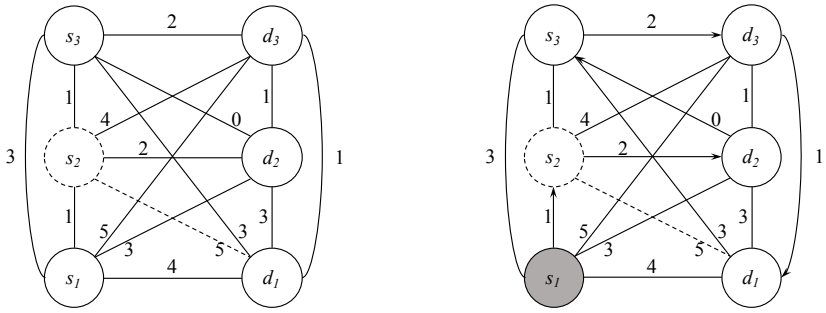


Figure 3. An illustration of the Partition Merging Algorithm.



(a) Locations and distances graph (b) An admissible path

Figure 4. Example of finding an admissible path.

origins of users appear before the corresponding destinations in an admissible path, we apply the topological sorting algorithm to eliminate paths that violate the order constraint. Considering that the driver's starting point must be the first location in the admissible path and his destination must be the last, we describe this constraint as an order constraint. Any other locations in the admissible path must appear after the driver's starting point and before his destination. This constraint can also be satisfied by applying the topological sorting algorithm. Line 7 in Algorithm 1 selects user p_k as a potential driver. After the driver is selected, a partial order set S is built in line 8. The partial order set should contain both origins and destinations of users in $c_i \cup c_j$ because of the inclusion constraint, and the partial order will be initialized based on the order constraint. After passing the capacity and detour constraint checks in line 10, an edge (c_i, c_j) is added in E to show that c_i, and c_j are mergeable.

Theorem 4.1. *The worst case complexity of PMA is* $O(n!)$.

Proof. The worst case occurs when PMA checks the merge-ability of two carpools in a carpool whose size is n. In this case, the PMA needs to test all possible partial orders of n nodes. The maximum possible number of these partial orders is $O(n!)$. Therefore, the worst case complexity is $O(n!)$. The time consumption of a $O(n!)$ algorithm may increase faster than a exponential growth $O(2^n)$, which is not affordable in read word applications. □

4.2. Insertion Methods

PMA explores all possible insertions in the merge-ability checking process, which is time consuming and is not applicable in real-world application. We propose two heuristic methods: driver-alone insertion and general insertion. When merging two carpools, both proposed insertion methods keep route schedule for one carpool and insert it

Algorithm 2 Partition Merging Algorithm with Driver-alone Insertion (PMADI)

Input: A set of users $P = \{p_1\}$ associate with the starting points set $\{s_i\}$, the destinations set $\{d_i\}$, the detour limitation set $\{\sigma_i\}$ and the capacity limitation set $\{\lambda_i\}$

Output: A set of carpools C.

1: Same as Algorithm 1, except change line 8 and 9 into:

2: $R \leftarrow$ insert c_i into c_j at points where there is only one person (the driver) in carpool c_j, and insert c_j into c_i at points where there is only one person in c_i.

entirely to the route for the other carpool. Both insertion methods can reduce the number of admissible paths checked.

Specifically, suppose we are merging two carpools c_i and c_j. Instead of applying the topological sort algorithm to eliminate paths which violate the order constraint, we first try to directly insert c_i into c_j without breaking c_i and then try from the other side. Driver-alone insertion only consider breaking the route when there is no passenger but only the driver on the route. General insertion explores more combinations, while it still keeps one route unaffected. PMA with Driver-along Insertion (PMADI) is shown in Algorithm 2. It is basically the same as PMA except the search space of admissible path is shrank. PMA with General Insertion (PMAGI) is similar as PMADI, except the path of a carpool can be interrupted in any point to insert the path of the other carpool. Insertion methods can greatly reduce the number of partial orders that should be checked for capacity constraint. They also provide a trade-off between time complexity and the algorithm performance.

We use Fig. 5 and Fig. 6 to illustrate these two insertion methods. In the example, there are two carpools $c_1 = \{s_1 - s_2 - d_2 - d_1\}$ and $c_2 = \{s_3 - s_4 - d_4 - d_3\}$. The driver-alone insertion first attempts to insert c_1 into c_2. The breaking point in c_2 can be the point between s_3 and s_4, or it can be the point between d_3 and d_4. In both of these points, there is only driver on the car and the insertion will increase the number of passengers on the vehicle to be the number of people in c_2 plus 1. The travel length for each insertion is recorded for further comparisons. Then the driver-alone insertion methods tries from the other side, which is to keep c_2 unchanged and insert c_2 into c_1. The possible breaking points on c_1 are the point between s_1 and s_2 as well as the point between d_1 and d_2. As for the general insertion, it also keeps one of c_1 and c_2 unchanged and breaks the other one. The difference is that it treats all points in the other route as potential breaking points. Fig. 6 illustrates the general insertion method. Similar to the driver-alone insertion, general insertion also entirely inserts c_i or c_j into the other part. The difference is that the general insertion explores all possible points of insertion instead of only considering the driver-alone case.

The driver-alone insertion can speed up the merge-ability checking process not only because it reduces the number of paths to be checked, it also saves time for capacity constraint checking. For any two carpools c_i and c_j in which the maximum number of users in the vehicle are κ'_i and κ'_j respectively, after inserting c_i into c_j with driver-alone insertion method, the maximum number of users in the vehicle is $\max\{\kappa'_i+1, \kappa'_j\}$. With the help of this property, whether the capacity constraint is violated or not can be checked by simply comparing the maximum number with the capacity limitation. For instance, the maximum number of users in the car for carpool c_2 in Fig. 5 is 2. Inserting c_2 into c_1 after the point s_3 generates the new carpool with path $s_3 \rightarrow s_1 \rightarrow s_2 \rightarrow d_2 \rightarrow d_1 \rightarrow s_4 \rightarrow d_4 \rightarrow d_3$. Following the path, the maximum number of people

(a) Inserting c_1 to c_2 (b) Inserting c_2 to c_1

Figure 5. Driver-alone insertion.

(a) Inserting c_1 to c_2 (b) Inserting c_2 to c_1

Figure 6. General insertion.

on the vehicle is 3.

The paths generated via both driver-alone and general insertion method are properly nested. The properly nested means that if passenger u_1 is picked up prior to u_2, then u_1 is dropped off before u_2. The paths in Fig.4 and Fig.5 can both illustrate this property. Specifically, u_3 is picked up at s_3 before u_4 boards the car, and u_3 is dropped off at d_3 which is prior to u_4's drop off location d_4. A path which is not properly nested can be generated when we use topological sort to generate all possible permutations. For example, a path which satisfies order constraint but is not properly nested can be: $s_1 \to s_2 \to s_3 \to d_2 \to d_3 \to s_4 \to d_4 \to d_1$. The property of properly nested helps us to track the number of users on the vehicle. It is not obvious to determine whether the property of properly nested is useful or not. Therefore, in the experiment section, we set up a series of experiments to evaluate the performance of PMADI, PMAGI and IPMA. The IPMA represents the algorithm exploring all possible topological sort.

Theorem 4.2. *The worst case complexity of PMAGI is* $O(n^{3.5})$.

Proof. The time complexity of PMA with General Insertion is $O(n^{3.5})$. The PMAGI iteratively merge carpools until there are no carpools to merge. We first focus on the time complexity of each iteration. Each iteration can be divided into three phases, including merge-ability checking phase, matching phase and merging phase. In the merge-ability checking phase, each carpool tests the merge-ability with other carpools. It is each carpool tries to insert into another carpool and checks the feasibility after insertion. The number of total possible insertion point is $O(n)$, and the total time consuming of checking is also $O(n)$. The reason is that n users can generate at most $O(n)$ intervals for insertion no matter how these n users are carpooled. In addition, the number of carpools is at most $O(n)$. Therefore, the time consumption of merge-ability checking phase is at most $O(n^2)$. The exact time complexity of the matching phase is hard to calculate since it is related to the number of edges generated. An $(O(\sqrt{V}E))$ algorithm for general matching algorithm is proposed in [21]. Therefore, the worst time consumption of matching is $O(n^{2.5})$. In the third phase, the merging phase, matched carpools are merged, and the time consumption is at most $O(n)$. Therefore, in each iteration, the time consumption is at most $O(n^2) + O(n^{2.5}) + O(n) = O(n^{2.5})$. The PMAGI at most has n iterations before convergence, since at least two carpools can be merged in each round before convergence. Therefore, the time complexity of PMAGI is $O(n^{3.5})$. The PMADI tries less insertions when checking the merge-ability,

Algorithm 3 Improved Partition Merging Algorithm (IPMA)

Input: A set of users $P = \{p_1\}$ associate with the starting points set $\{s_i\}$, the destinations set $\{d_i\}$, the detour limitation set $\{\sigma_i\}$ and the capacity limitation set $\{\lambda_i\}$

Output: A set of carpools C.

1: Same as Algorithm 1, except add a line after line 7:
2: **if** $\exists p_{k'} \in c_i \cup c_j$, $f(s_k, s_{k'}) + f(s_{k'}, d_k) - f(s_k, d_k) > \sigma_k$ or $f(s_k, d_{k'}) + f(d_{k'}, d_k) - f(s_k, d_k) > \sigma_k$ **then**
3: Skip this round.

and therefore the time complexity is also $O(n^{3.5})$. □

Admittedly, directly inserting may eliminate potential optimal choice for merging. It significantly reduces the time complexity of the algorithm and provides the system operator a choice to balance the running time and the optimality of the algorithm outcome. Besides, the partial orders generated by our insertion methods are properly nested.

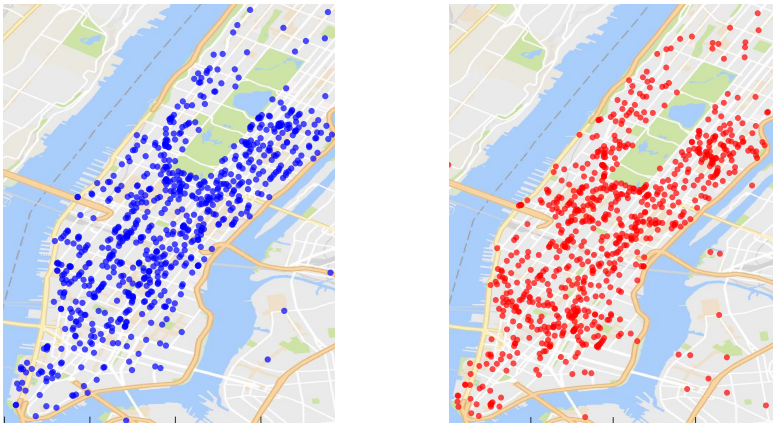
4.3. Improving the PMA with Geometry Properties

The time efficiency of PMA can be further improved by taking advantage of geometry properties. In PMA, the detour constraint is checked after paths are constructed by topological sorting. However, if the origins and destinations of users are too far from each other, we can conclude that these users cannot merge directly without constructing any paths. That is to say, the detour constraint can be pre-checked by calculating the total mileages starting from driver's origin and then passing by passenger's origin or destination to driver's destination. In this way, the time used to calculate the topological order of these nodes is reduced. More formally, when checking the merge-ability of c_i and c_j for each user p_k, if $\exists p_{k'}$ such that $f(s_k, s_{k'}) + f(s_{k'}, d_k) - f(s_k, d_k) > \sigma_k$ or $f(s_k, d_{k'}) + f(d_{k'}, d_k) - f(s_k, d_k) > \sigma_k$, then p_k cannot be the driver in the new carpool $c_i \cup c_j$.

The Improved Partition Merging Algorithm (IPMA) is shown in Algorithm 2. To illustrate the improvement, an example is shown in Fig. 4(a), for user p_2, the distance between the origin of p_2 and the destination of p_3 already exceeds the detour tolerant limitation of p_2. Therefore, there is no way to set p_2 as the driver. We do not have to waste time to find the admissible path. More intuitively, after plotting out all locations within the detour distance limitation of a user, we find that these locations form an ellipse. Any user whose origin or destination is located outside of the ellipse cannot travel with him or her. We can save running time in IPMA by skipping these users.

5. Experiment

In this section, simulated and real data-driven experiments are conducted to evaluate the performances of the proposed algorithms. After presenting the datasets and settings, the results are shown from different perspectives to provide insightful conclusions.



(a) Start location distribution (b) Destination distribution

Figure 7. Location distribution of the NYC dataset.

5.1. Synthetic and NYC Taxi Datasets

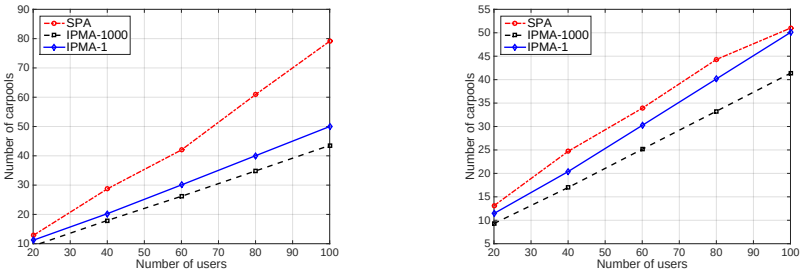
This subsection introduces the datasets used in our experiment. Both a synthetic dataset and a real-world dataset were used. For the synthetic dataset, we randomly created a user's request locations, including starting points and destinations. To fully test the performance of our algorithm, two different kinds of distributions are used: uniform distribution and normal distribution. In our uniform distribution dataset, the horizontal and vertical coordinates of each location are individual and range from 0-30 miles. What's more, the origin and destination of each user are also individual. In our normal distribution dataset, the independence of horizontal and vertical coordinates holds true, and so does the independence of origins and destinations. The mean of the normal distribution is set to 15 and the standard deviation is set to 5 to make sure that more than 99.73 % of coordinates are located in the same range of locations in the uniform distribution dataset. Under these parameter settings, 10,000 request locations are separately generated for the uniform distribution and normal distribution dataset.

The NYC dataset is extracted from yellow cab trace data in NYC. The yellow cab trace data in NYC contains each taxi service's start time, end time, the GPS coordinates of pickup and drop-off locations, travel distances. We analyzed the trace data of a day, and find that there are 743.9 requests per minute on average in the NYC area. We extracted 500 items from trace data where start times differ by less than 2 minutes to build our NYC dataset. The average user travel distance in our NYC dataset is 3.155 miles. The distribution of pickup and drop-off locations are shown in Fig. 7.

5.2. Experimental Settings

We conduct four sets of experiments to evaluate our proposed algorithms. We first compare IPMA with the Strict Partitioning Algorithm (SPA) in the synthetic dataset. In addition, we focus on the comparison between IPMA with its variation with driver-alone and general insertion methods, which are denoted as PMADI and PMAGI respectively. What's more, the IPMA is tested with different detour configurations. Finally, the IPMA is tested on the real-world dataset.

IPMA and SPA are evaluated in our first series of experiments. Considering the



(a) Uniform distribution dataset (b) Normal distribution dataset

Figure 8. Comparison between IPMA and SPA under different distribution.

number of possible topological orders fluctuates in different partial order sets, the time it takes to apply IPMA may vary intensely in different datasets. To control time consumption, we use the variant IPMA in our experiment. Instead of using all the topological orders in IPMA, we only use the top k topological orders. The modified version is referred to as IPMA-k in the following experiment. What's more, choosing small k may lead to less optimal results, and therefore, we plot different outcomes when k varies to visualize this effect. In the first series of experiments, both the uniform distribution dataset and the normal distribution dataset are used to compare the performances of IPMA-1, IPMA-1000, and the previous SPA. To maximize the difference between IPMA and SPA as much as possible, we set each user's capacity to 2. The detour distance of each user is set to relative distance, i.e., the percentage of the distance that a user travels. We use 5%, 10%, and 15% to imply small, middle and large detour distance respectively. The results are averaged over 50 times for smoothness.

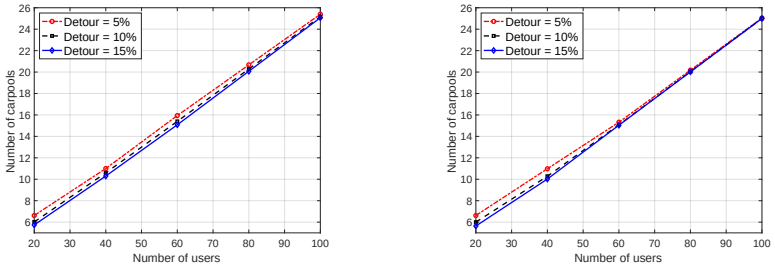
We also test the performance of IPMA with different detour distances. In this situation, we apply IPMA-1000 to our simulated data set. The capacity of each user is set to 4, since a capacity of 4 is more similar to a real-world scenario. The results are averaged over 50 times per simulation.

We further test the performance of two different insertion methods: driver-lone insertion and the general-insertion along with the IPMA-k. The experiment is used to check if our proposed insertion method will ignore too many possible paths and improve the performance slightly with large time consuming. As we mentioned earlier, both PMADI and PMAGI generate properly nested pickup and drop-off schedule for each carpool. Through this set of experiment, we try to understand if this property improves the performance of IPMA.

The last experiment is conducted on the real-world dataset aims to test the effect of k, i.e. the number of topological orders used in IPMA. We choose to test the effect of k based on the NYC Taxi dataset since we are trying to reflect the effect of k in the real-world situation. In all, there were 500 users' request involved. The capacity of each user is set to 5, and the middle detour distance is used, i.e., 10% detour distance.

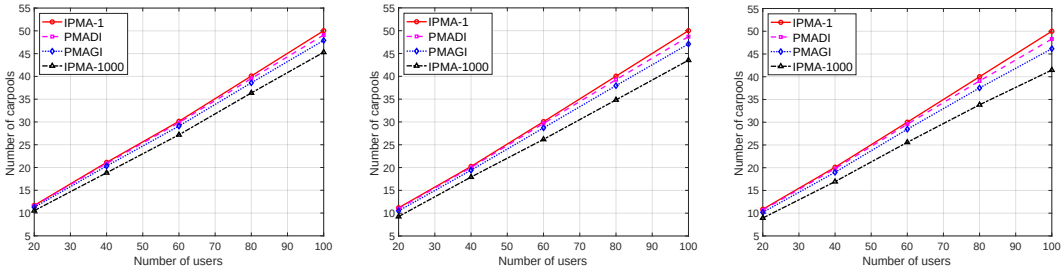
5.3. Evaluation Results

Fig. 8 shows the performance comparison between IPMA and SPA in both uniform and normal distribution datasets. Comparing results in Figs.8 (a) and (b), we can conclude that both IPMA-1 and IPMA-1000 outperformances than SPA. In the normal distribution dataset, the performance differences between IPMA and SPA are smaller than



(a) Uniform distribution dataset (b) Normal distribution dataset

Figure 9. Comparison of IPMA-1000 under different detour.



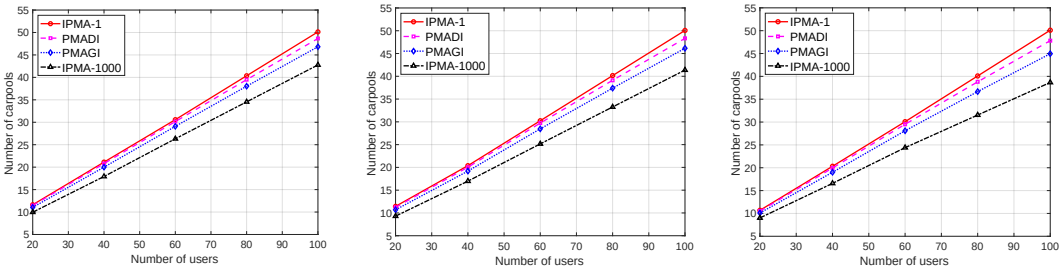
(a) Carpooling with 5% detour (b) Carpooling with 10% detour (c) Carpooling with 15% detour

Figure 10. Comparison among IPMA-1, PMADI, PMAGI and IPMA-1000 using uniform distribution.

than in the uniform distribution dataset. This is not because of the under-performance of IPMA, but rather due to the higher efficiency of SPA in the normal distribution dataset. The number of carpools in SPA decreases around 35%, which indicates that SPA has a better performance in centralized location distributions while IPMA has relatively stable performance. To sum up, IPMA always boasts better outcomes than SPA and IPMA's performance is more stable.

Fig. 9 represents the performance of IPMA-1000 under different detour configuration in both uniform and normal distribution datasets. Although the number of carpools decreases with a larger detour, the variation is very slight. What's more, when the number of users increases, there is nearly no difference between the number of carpools found by IPMA-1000 in either uniform or normal distribution dataset. The reason behind this might be that larger carpools can hardly be merged unless the detour limitation is greatly expanded. Specifically, larger carpools are more likely to be generated when number of users increases. Merging two large carpools may bring a huge increase on driver's and/or users' detours, and it is likely that no matter 5% or 15% detour capacity is not enough to support the merging. It may explain what we found from the comparison result.

The evaluation results of the performances of IPMA-1, IPMA-1000, PMADI and PMAGI on synthetic datasets are shown in Fig. 10 and Fig. 11. Fig. 10 corresponds to the uniform distribution dataset and Fig. 11 corresponds to the normal distribution dataset. From either Fig. 10 or Fig. 11, we can find out that both PMADI and PMAGI outperform the IPMA-1. It is easy to understand since both PMADI and PMAGI explore more than one permutation. Besides, we can also find out that in both distributions and for all different detour distances considered in our experiments, IPMA-1000 has better performances than other comparisons. It indicates that although the PMAGI and PMADI improves the performance of PMA, exploring more



(a) Carpooling with 5% detour (b) Carpooling with 10% detour (c) Carpooling with 15% detour

Figure 11. Comparison among IPMA-1, PMADI, PMAGI and IPMA-1000 using normal distribution.

Table 2. IPMA-k outcomes with different k.

k	10^0	10^1	10^2	10^3	10^4
Number of carpools	249	162	126	125	125

possible permutations can further improve the algorithm performance. The difference between IPMA-1000 and other algorithms becomes larger as the detour distance increases. Comparing the outcomes in Fig. 10 and Fig. 11, we conclude that the performance differences between IPMA-1000 and PMAGI/PMAGI are smaller in the uniform distribution dataset than in the normal distribution dataset.

Finally, we present the results on the NYC taxi dataset in Table I. We can see that k greatly influences the performance of IPMA-k. This makes sense since with a smaller k, fewer possible topological orders will be checked and the admissible path may be missed. For instance, when $k = 1$, IPMA will only choose one possible topological order to check its admissibility. If $k \to +\infty$, all possible topological orders are checked and no possible path can be lost. Although the outcome of IPMA-k is closer to optimal when k is larger, more running time is consumed. From the experiment result, we can see that $k = 100$ is large enough to give good results because when k keeps increasing, the number of carpools is nearly unchanged. Therefore, using hundreds of topological orders in IPMA is likely to produce a relatively good result when applying IPMA to a real world dataset.

6. Conclusion

This paper discusses the carpool problems in which a user shares his/her car with others in order to reduce the number of cars on the road. We discuss existing methods based on strict partitioning and provide evidence that strict partitioning cannot give an optimal result for a large number of real-life scenarios. Then, we propose a greedy algorithm, PMA, to calculate the local optimal result of out carpool problem. We propose the multi-round matching and merging methods based on a greedy approach. In addition, we study several checking methods for the merge-ability checking process and we propose driver-alone and general insertion methods besides the naive approach. Furthermore, the time efficiency of our algorithms is improved by taking advantage of geometry properties. To evaluate the performance of our approaches, we execute our algorithm on both synthetic and NTC Taxi datasets. The results from both datasets show that our algorithm based on partition merging outperform the SPA in a number of cases. The performances of PMAGI and PMADI are both greater than IPMA-1,

but they are not comparable with IPMA-k with larger k. Users' maximum waiting time is not considered in our paper, but it is also an important factor that should be studied more closely in future works.

7. Acknowledgment

This research was supported in part by NSF grants CNS 1629746, CNS 1564128, CNS 1449860, CNS 1461932, CNS 1460971, CNS 1439672, CNS 1301774, and ECCS 1231461. An earlier version of this paper was presented at the IEEE International Conference on Communications 2018 [22].

References

[1] Meyer I, Kaniovski S, Scheffran J. Scenarios for regional passenger car fleets and their co 2 emissions. Energy Policy. 2012;41:66–74.
[2] Gärling T, Steg L. Threats from car traffic to the quality of urban life: problems, causes and solutions. Emerald Group Publishing Limited; 2007.
[3] Neoh JG, Chipulu M, Marshall A. What encourages people to carpool? an evaluation of factors with meta-analysis. Transportation. 2017;44(2):423–447.
[4] Javid RJ, Nejat A, Hayhoe K. Quantifying the environmental impacts of increasing high occupancy vehicle lanes in the united states. Transp Res D. 2017;56:155–174.
[5] Wu C, Shankari K, Kamar E, et al. Optimizing the diamond lane: A more tractable carpool problem and algorithms. In: Proceedings of the 19th IEEE ITSC; 2016. p. 1389–1396.
[6] Silva E, Kokkinogenis Z, Câmara Á, et al. An exploratory study of taxi sharing schemas. In: Proceedings of the 19th IEEE ITSC; 2016. p. 247–252.
[7] Hartman IBA, Keren D, Dbai AA, et al. Theory and practice in large carpooling problems. Procedia Computer Science. 2014;32:339–347.
[8] Agatz N, Erera AL, Savelsbergh MW, et al. Dynamic ride-sharing: A simulation study in metro atlanta. Procedia-Social and Behavioral Sciences. 2011;17:532–550.
[9] Amey A. A proposed methodology for estimating rideshare viability within an organization, applied to the mit community. In: TRB Annual Meeting Procediings; 2011. p. 1–16.
[10] Ghoseiri K, Haghani AE, Hamedi M, et al. Real-time rideshare matching problem. Mid-Atlantic Universities Transportation Center Berkeley; 2011.
[11] Xing X, Warden T, Nicolai T, et al. Smize: a spontaneous ride-sharing system for individual urban transit. In: German Conference on Multiagent System Technologies; Springer; 2009. p. 165–176.
[12] Buchholz F. The carpool problem. Citeseer; 1997.
[13] Baldacci R, Maniezzo V, Mingozzi A. An exact method for the car pooling problem based on lagrangean column generation. Oper Res. 2004;52(3):422–439.
[14] Santi P, Resta G, Szell M, et al. Quantifying the benefits of vehicle pooling with shareability networks. Proceedings of the National Academy of Sciences. 2014;111(37):13290–13294.
[15] Zhang S, Ma Q, Zhang Y, et al. QA-share: Towards efficient qos-aware dispatching approach for urban taxi-sharing. In: Proceedings of the 12th IEEE SECON 2015; 2015. p. 533–541.
[16] Chang W, Zheng H, Wu J. On the RSU-based secure distinguishability among vehicular flows. In: Proceedings of the IEEE/ACM IWQoS 2017; 2017. p. 1–6.
[17] Wang N, Wu J, Ostovari P. Coverage and workload cost balancing in spatial crowdsourcing. In: Proceedings of the 14th IEEE UIC; 2017. p. 1–8.

[18] Lenstra JK, Kan A. Complexity of vehicle routing and scheduling problems. Networks. 1981;11(2):221–227.
[19] Galil Z. Efficient algorithms for finding maximum matching in graphs. ACM Computing Surveys. 1986 Mar;18(1):23–38.
[20] Bernstein A, Stein C. Faster fully dynamic matchings with small approximation ratios. In: Proceedings of the 27th ACM-SIAM SODA; 2016. p. 692–711.
[21] Micali S, Vazirani VV. An O($\sqrt{V}E$) algoithm for finding maximum matching in general graphs. In: 21st IEEE FOCS; 1980. p. 17–27.
[22] Duan Y, Mosharraf T, Wu J, et al. Optimizing carpool scheduling algorithm through partition merging. In: IEEE ICC; May; 2018. p. 1–6.