Optimizing Carpool Scheduling Algorithm through Partition Merging

Yubin Duan, Turash Mosharraf, Jie Wu, and Huanyang Zheng
Center for Networked Computing, Temple University, USA
Email: {yubin.duan, turash.mosharraf, jiewu, huanyang.zheng}@temple.edu

***Abstract*—The rapidly increasing number of vehicles in roads leads to numerous problems in metropolitan areas. Several researchers show that carpooling can be an efficient solution to relieve the pressures caused by large numbers of cars. Previous research on carpools introduces several additional constraints to simplify the problem, but some of them are unreasonable in reality. In this paper, we focus on removing the static capacity constraint. Doing so allows a vehicle to carry more passengers than vehicle's capacity, which is possible if some people are dropped off and new passengers take their places during the journey. A greedy approach based on multi-round matching is proposed, and it is further improved by taking advantage of geometry properties. We apply our algorithms to both simulated and real world datasets, and experiment results show that our algorithms have better performances than existing approaches.**

***Index Terms*—Carpool problem, hierarchical cluster merging, graph partitioning.**

I. 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 will grow to 2.8 billion with projection. The rapid increase of vehicles has led to environmental, economic, and social problems, including increased carbon emission, travel costs, and congestion [2]. To release the pressure caused by increasing demands for cars for transportation, carpools were proposed. Non-household carpools, where two or more commuters from different residences travel together in the same private vehicle reduce 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 reductions in the CO_2e emissions of the 50 U.S states and the District of Columbia achieve 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 starting points, the destination, 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 requested users (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 problems [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 arrives at his destination, the seat assigned to the passenger will not be released.

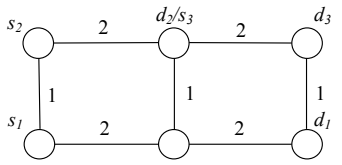


Fig. 1. An illustration of the carpool scheduling problem scenario.

Using Fig. 1 as an illustration, $\{s_1, d_1\}, \{s_2, d_2\}$, and $\{s_3, d_3\}$ represent the start and destination points of users p_1, p_2, and p_3, respectively. The numbers on edges show the distance between vertices. Assume that each of the three users 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 2 units. To demonstrate the additional constraint in the previous problem, suppose p_1 is appointed as a driver and p_2 is in the same carpool with p_1. p_1 will start from s_1 and pick up p_2 at s_2. After p_1 drops off p_2 at d_2, there will be one seat available in p_1's car, and in this situation, p_1 could pick up p_3 since the starting point s_3 of p_3 is the destination point d_2 of p_2. However, with the additional constraint, the seat capability will be fulfilled when p_2 gets into the car, and there will be no chance for 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. The minimum number of carpools for the existing scenario is 2, and 2 possible arrangements of carpools exists to achieve the minimum number. The first is setting p_1, p_2 in the same carpool and letting p_3 travel alone with the service schedule p_1: $s_1 \rightarrow s_2 \rightarrow d_2 \rightarrow d_1$, p_3: $s_3 \rightarrow d_3$. The other possible assignment is setting p_1, p_3 in the same carpool and let p_2 travel alone with the service schedule p_1: $s_1 \rightarrow s_3 \rightarrow d_3 \rightarrow d_1$, p_2: $s_2 \rightarrow d_2$.

However, the additional constraint in the previous problem is unreasonable in reality since it is possible that drivers could take extra passengers while maintaining the detour cost. As shown in Fig. 1, the driver p_1 could transport both p_2 and p_3

to their destination with same detour cost as traveling with p_2 via path $s_1 \rightarrow s_2 \rightarrow d_2(s_3) \rightarrow d_3 \rightarrow d_1$. That is to say, 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 its NP-hardness.
- We provide two time-window based greedy approximation algorithms. The first one can reducing the number of carpools significantly. We further improve it by taking advantage of geometry properties.
- Experiments on both simulated and real-world data are set up thereby validating 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 II surveys related works. Section III describes the model, formulates and analyzes the problem. Section IV proposes a greedy algorithm and its improvement. Section V includes the experiments. Finally, Section VI concludes the paper.

II. Related Work

Several researches have been done on the carpool problem, including [8–15]. Their work are reviewed in the rest part of this section.

Baldacci et. al [8] address the carpool problem as a transportation service organized by a large company which encourages its employees to pick up colleagues while driving to/from work to minimize the number of private cars travelling to/from the company site. Based on two integer programming formulations of the problem, an exact solution and a heuristic method is developed.

Buchholz et. al [9] present the Strict Partitioning Algorithm (SPA) that divides the set of users into several k-partitions. The number of members in a partition is no more than k, which represents the capacity constraint. What's more, they prove that the problem is NP-hard for $k \geq 3$. As long as $k = 2$, the solution has a time complexity of O(n^2). The limitation of the SPA is that for the case where $k = 2$, this approach allows only one person to share the car with the driver.

Geisberger [10] presents an efficient method for calculating the detour between the path of any driver-passenger pair in constant time using a pre-calculated distance table [11] that stores all pairs' shortest paths.

Santi et. al [12] and Zhang et. al [13] address taxi-sharing problems. Santi et. al use a method similar to [9] for taxi sharing. They build a share-ability network between individual passenger trips and 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 SP. 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 [12].

III. Problem Formulation and Analysis

A. 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 $P^* = \bigcup_{i=0}^{|P|} P^i = P \cup P^2 \cup P^3 \cup \ldots \cup P^{|P|}$. Let $C = \{c_i\}$ denotes the set of all possible carpools. The starting points of the users in carpool c construct a sub-set $S_c \subseteq S$, and the destinations construct 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 (i.e., 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 the visiting sequence of locations. 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 users' starting points should appear before their destinations 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 the definitions given above, our problem can be formulated as follows:

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

B. Problem Hardness

Under a special case, our Carpool Scheduling Problem (CSP) is equivalent to the well known traveling salesman problem (TSP) [16]. The TSP is proven to be NP-hard, and therefore, we can prove that our problem is also NP-hard.

***Theorem** 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[9]. 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 the each city can be seen as a passenger with the same starting and ending point. Then 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. ■

IV. Algorithmic Design

A. Partition Merging Algorithm

This subsection presents the Partition Merging Algorithm (PMA). Observe the example shown in Fig. 1. Strict Partition Algorithm (SPA) applying 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.

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.

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 number of merges can be achieved by applying an efficient maximum matching algorithm [17] (line 4 and 5). It is true that other efficient maximum matching algorithms like [18] can 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,

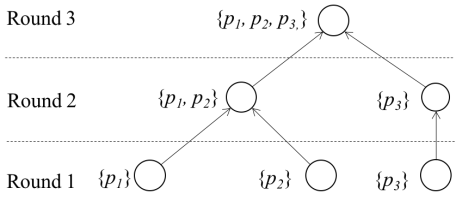
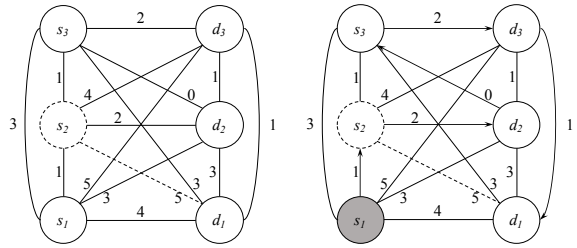


Fig. 2. An illustration of the Partition Merging Algorithm.



(a) Graph of locations and distances (b) An admissible path

Fig. 3. Example of finding an admissible path.

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 return C as the final result (line 13). Fig. 2 shows the merging situation in each round of PMA based on the scenario shown in Fig. 1. 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. 3(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 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.

Algorithm 2 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**
Skip this round.

B. Improving the PMA with geometry properties

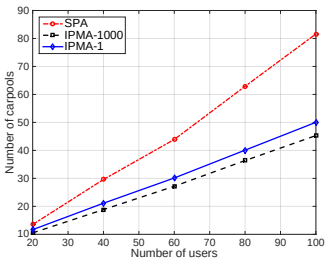
The time efficiency of PMA can be further improved by taking advantage of geometry properties. In PMA, the detour constraint is checked after paths are constructed by topological sorting. However, if the origins and destinations of users are too far from each other, we can conclude that these users cannot 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 then pass passenger's origin or destination to driver's destination. In this way, the time used to calculate the topological order of these nodes is saved. 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. 3(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.

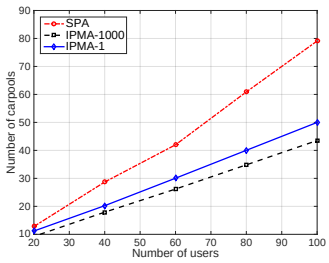
V. Experiment

A. Simulated and NYC taxi datasets

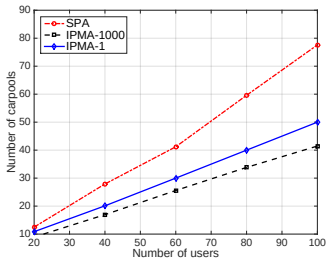
This subsection introduces the datasets used in our experiment. Both a simulated dataset and a real-world dataset are used. For the simulated dataset, we randomly create a user's request locations, including starting points and destinations. To fully test the performance of our algorithm, 2 different 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 is also individual. In our normal distribution dataset, the independence of horizontal and vertical coordinates remains, so as 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



(a) Carpool number with 5% detour

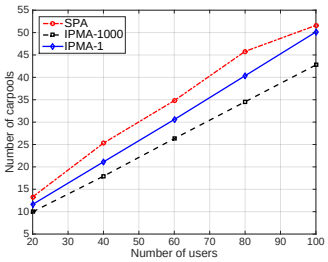


(b) Carpool number with 10% detour

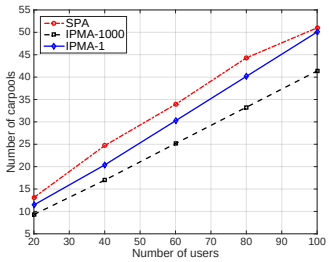


(c) Carpool number with 15% detour

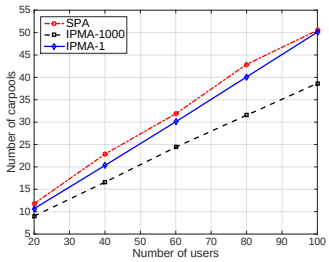
Fig. 4. Performance comparison between SPA, IPMA-1000, and IPMA-1 using uniform distribution.



(a) Carpool number with 5% detour



(b) Carpool number with 10% detour



(c) Carpool number with 15% detour

Fig. 5. Performance comparison between SPA, IPMA-1000, and IPMA-1 using normal distribution.

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, 10000 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 analyze the trace data of a day, and find that there are 743.9 requests per minute on average in the NYC area. We extract 500 items from trace data where start times differ by is less than 2 minutes to build our NYC dataset. The average user travel distance in our NYC dataset is 3.155 miles.

B. Experimental Settings

IPMA and SPA are evaluated in our first series of experiments. Considering the number of possible topological order fluctuates in different partial order set, 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.

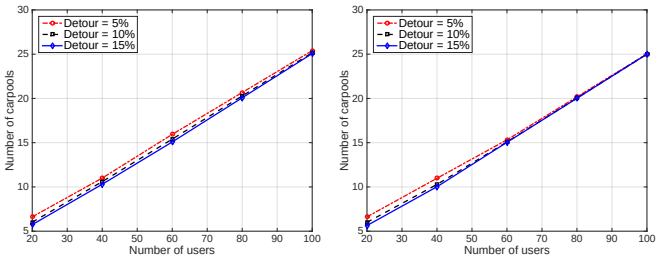
In the first series of experiments, both the uniform distribution dataset and the normal distribution dataset are used to compare the performances of IPMA-1, IPMA-1000, and the previous Strict Partitioning Algorithm (SPA). To maximize the difference of IPMA and SPA as much as possible, we set each user's 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.

The last experiment aims to test the effect of k, i.e. the number of topological orders used in IPMA. This experiment runs based on the NYC dataset since we are trying to reflect a real-world situation. In all, there are 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.

C. Evaluation Results

The evaluation results of the performances of SPA, IPMA-1, IPMA-1000 on simulated datasets are shown in Fig. 4 and Fig. 5. Fig. 4 corresponds to the uniform distribution dataset and Fig. 5 corresponds to the normal distribution dataset. Comparing the outcomes within Fig. 4 and Fig. 5, we see that in both distributions and for all different detour distances, IPMA-1000 and IPMA-1 has better performances than SPA. The difference between IPMA-1000 and IPMA-1 becomes larger along as the detour distance increases. Comparing the outcomes in Fig. 4 and Fig. 5, we conclude that the performance differences between IPMA and SPA are smaller in the normal distribution dataset than in the uniform distribution dataset. This is not because of the under-performance of IPMA, but rather due to the over-performance of SPA. Observing the values of the Y-axis, the numbers of carpools in IPMA-1000 and IPMA-1 decreases slightly when changing the uniform distribution dataset to the normal distribution. 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,



(a) Using uniform distribution dataset (b) Using normal distribution dataset

Fig. 6. Comparing outcomes of IPMA-1000 under different detour.

TABLE I
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

IPMA always boasts better outcomes than SPA and IPMA's performance is more stable.

The evaluation results that show the performance of IPMA with different detour distance are presented in Fig. 6. Although the number of carpools decrease with a larger detour, the variant is very slight. What's more, when the number of users increases, there is nearly no difference in the number of carpools found that IPMA find under either distribution.

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 kept increasing, the number of carpools is nearly unchanged. Therefore, using hundreds of topological orders in IPMA will likely produce a relatively good result when applying IPMA to a real world dataset.

VI. 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 then use geometry properties to further reduce the time consumption of PMA, and we propose IPMA accordingly. To evaluate the performance of our approaches, we execute our algorithm on both simulated and real world datasets. The results from both datasets show that our approaches outperform the SPA in a number of cases. Users' maximum waiting time is not considered in our paper, which can be studied as future work.

VII. 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.

References

[1] I. Meyer, S. Kaniovski, and J. Scheffran, "Scenarios for regional passenger car fleets and their co 2 emissions," *Energy Policy*, vol. 41, pp. 66–74, 2012.

[2] T. Gärling and L. Steg, *Threats from car traffic to the quality of urban life: problems, causes and solutions*. Emerald Group Publishing Limited, 2007.

[3] J. G. Neoh, M. Chipulu, and A. Marshall, "What encourages people to carpool? an evaluation of factors with meta-analysis," *Transportation*, vol. 44, no. 2, pp. 423–447, 2017.

[4] R. J. Javid, A. Nejat, and K. Hayhoe, "Quantifying the environmental impacts of increasing high occupancy vehicle lanes in the united states," *Transp. Res. D*, vol. 56, pp. 155–174, 2017.

[5] C. Wu, K. Shankari, E. Kamar, R. Katz, D. Culler, C. Papadimitriou, E. Horvitz, and A. Bayen, "Optimizing the diamond lane: A more tractable carpool problem and algorithms," in *Proceedings of the IEEE ITSC 2016*, pp. 1389–1396.

[6] E. Silva, Z. Kokkinogenis, Á. Câmara, J. Ulisses, J. Urbano, D. C. Silva, E. Oliveira, and R. J. Rossetti, "An exploratory study of taxi sharing schemas," in *Proceedings of the IEEE ITSC 2016*, pp. 247–252.

[7] I. B.-A. Hartman, D. Keren, A. A. Dbai, E. Cohen, L. Knapen, D. Janssens *et al.*, "Theory and practice in large carpooling problems," *Procedia Computer Science*, vol. 32, pp. 339–347, 2014.

[8] R. Baldacci, V. Maniezzo, and A. Mingozzi, "An exact method for the car pooling problem based on lagrangean column generation," *Oper. Res.*, vol. 52, no. 3, pp. 422–439, 2004.

[9] F. Buchholz, "The carpool problem," Citeseer, Tech. Rep., 1997.

[10] R. Geisberger, D. Luxen, S. Neubauer, P. Sanders, and L. Völker, "Fast detour computation for ride sharing." *OpenAccess Series in Informatics*, vol. 14, pp. 88–99, 01 2010.

[11] S. Knopp, P. Sanders, D. Schultes, F. Schulz, and D. Wagner, "Computing many-to-many shortest paths using highway hierarchies," in *Proceedings of the ACM-SIAM ALENEX 2007*, pp. 36–45.

[12] 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*, vol. 111, no. 37, pp. 13 290–13 294, 2014.

[13] S. Zhang, Q. Ma, Y. Zhang, K. Liu, T. Zhu, and Y. Liu, "Qashare: Towards efficient qos-aware dispatching approach for urban taxi-sharing," in *Proceedings of the IEEE SECON 2015*, pp. 533–541.

[14] W. Chang, H. Zheng, and J. Wu, "On the RSU-based secure distinguishability among vehicular flows," in *Proceedings of the IEEE/ACM IWQoS, 2017*, pp. 1–6.

[15] N. Wang, J. Wu, and P. Ostovari, "Coverage and workload cost balancing in spatial crowdsourcing," in *Proceedings of the IEEE UIC 2017*.

[16] J. K. Lenstra and A. Kan, "Complexity of vehicle routing and scheduling problems," *Networks*, vol. 11, no. 2, pp. 221–227, 1981.

[17] Z. Galil, "Efficient algorithms for finding maximum matching in graphs," *ACM Computing Surveys*, vol. 18, no. 1, pp. 23–38, Mar. 1986.

[18] A. Bernstein and C. Stein, "Faster fully dynamic matchings with small approximation ratios," in *Proceedings of the ACM-SIAM SODA 2016*, pp. 692–711.