Proceedings of the 3rd International Conference on Signal Processing and Machine Learning
DOI: 10.54254/2755-2721/6/20230389

An improved partition merging algorithm to account for maximum waiting time in carpools

Andrew Wen[1,6,†], Jinglin Xu[2,7,†], Shunchang Chen[3,8,†], Ryan Wu[4,9,†], and Yuntao Lin[5,10,†]

[1]Phillips Academy Andover, Andover, MA, USA
[2]University Hill Secondary School, Vancouver, BC, Canada, V6S 0C6
[3]College of Science and Technology, Wenzhou-Kean University, Wenzhou, Zhejiang, 325000, China
[4]Livingston High School, Livingston, NJ, 07039, USA
[5]Sun Yat-sen University, Guangzhou, Guangdong, 510006, China

[6]andwendrew@gmail.com
[7]xubenny666@gmail.com
[8]1162569@wku.edu.cn
[9]ryan.wu365@gmail.com
[10]linyt65@mail2.sysu.edu.cn
†Andrew Wen, Jinglin Xu, Ryan Wu, Kai Chen, and Yuntao Lin contributed equally to this work and should be considered co-first authors.

Abstract. Nowadays, traffic problems have become an issue that needs to be solved in every populous country. Especially in large urban areas, people face traffic problems such as road congestion and high exhaust emissions due to the rapid growth of vehicles on the roads. In this context, several researchers have shown that carpooling, i.e., vehicle sharing, is an attractive solution to effectively address the traffic stress that arises when a large number of cars are traveling at the same time. The goal of our carpool scheduling problem is to reduce the number of carpools required by all users while ensuring their waiting time. Previous studies on similar topics have introduced additional static capacity constraints to simplify the problem, which limits the number of carpooling users in a vehicle. However, this does not match most real-world situations. This is because when a passenger gets off the vehicle, their seat should also be vacated for the next user. In this paper, we eliminate the static capacity constraint to enable timely user turnover during the carpooling journey. A greedy algorithm based on iterative matching and summation is proposed. In addition, we introduce the concept of "user waiting time", i.e., the amount of time a user is willing to wait to be picked up. We apply our algorithm to synthetic and real-world data sets, and our experimental results show that our algorithm has better performance than existing methods.

Keywords: carpool, vehicle sharing, partition merging, greedy algorithm, user waiting time.

DOI: 10.54254/2755-2721/6/20230389

1. Introduction

With the passage of time, the current situation of the world has become multi-polar. Society has become more stable, and people's economic situation has improved. Especially in big cities, cars are a necessity for almost every household. Moreover, the development of the automobile business has grown rapidly in the past two or three decades, and the introduction of trams has effectively solved the problems of environmental pollution and fuel consumption. Vehicles have brought convenience to people's travel, but they have also caused many problems. According to a study by Meyer et al., by 2050, the global automobile population is expected to reach 2.8 billion [1]. It is hard to imagine what the traffic situation will be like then. The rapid growth of vehicles has led to an increase in carbon emissions, road congestion, and other problems. In order to relieve the traffic pressure caused by people's demand for cars, carpooling has become the direction for developing a new era of sharing economy. This is the process of traversing between the source and destination of two or more passengers. Multiple users share a car, which leads to a reduction of vehicles on the road. In addition, the dedicated carpool lane - the "diamond lane" - provides smoother traffic by avoiding congestion on the complex trip routes of carpooling vehicles [2]. But inevitably, users need to wait for the driver to finish the previous user's trip. Therefore, the objective of this paper is to propose a carpool scheduling algorithm that guarantees maximum waiting time for users.

During the user matching process, the maximum waiting time of the user will be captured. This is the maximum wait time that the passenger can tolerate, which is around 20 minutes. The driver needs to calculate the time it takes to reach the pickup point when going to pick up the next passenger, and if it is less than or equal to the maximum wait time, the user is picked. The user's maximum wait time ensures that the passenger's order is not picked up by a distant driver. Considering each user's origin, destination, vehicle capacity, maximum waiting time, and other user requirements, our algorithm is designed to reduce the number of carpooling required by all users. Several researchers have already solved similar carpooling problems. Carpooling problems with static constraints are categorized as NP-hard problems. A static constraint means that the driver's vehicle capacity is equal to the number of passengers i.e. after a passenger arrives at his destination, his seat is still not assigned to other users either.

The main contributions of this paper are as follows.

- Consider the maximum user waiting time during the user matching process.
- A partition merging algorithm based on the greedy algorithm is given.
- Experiments were conducted on synthetic and real data to verify the superiority of our algorithm over existing carpool scheduling algorithms in terms of consideration of waiting time and a similar number of carpools.

The results show that the total number of carpooling can be reduced while respecting user considerations by using the maximum waiting time and the partition merging algorithm. The rest of this paper is organized as follows. Section 2 investigates related work. Section 3 introduces our model and formulates and analyzes the problem. Section 4 proposes a greedy algorithm and its changes. Section 5 includes experiments. Finally, Section 6 concludes the paper.

2. Related work

The carpooling problem has been studied extensively already. Its impacts, such as urban congestion, pollution, and increased traffic have caused it to become widely studied from a variety of different angles in the existing literature.

Most classic carpool studies focus on creating routing for fixed passengers, in which the destination and route of the passengers are regular and fixed. He et al. focused on using GPS trajectories to find and match common passengers to find and suggest carpool paths. This way, passengers would be able to find a way to carpool together on a regular basis based on their set paths [3]. However, this situation may not be the most realistic. Schedules, routes, and timing are bound to change at any time. Analyzing fixed patterns may be helpful but are not always the most realistic way of approaching the problem.

DOI: 10.54254/2755-2721/6/20230389

To expand upon this, existing research has also looked into more dynamic approaches to the issue. Zhao et al. examined this problem with the purpose of minimizing the average travel distance of all passengers through a dynamic route matching approach. The authors proposed a heuristic algorithm named SimilarDirection with a 2c approximation ratio in the delivery order calculation phase, where c is the capacity of each vehicle [4].

Research has also addressed issues such as location privacy. Duan et al. utilized a cloaking zone to protect the exact locations of riders from service providers such as Uber or Didi. However, this method only gives the service provider a general region that the passengers are in, not allowing for the service provider to maximize the efficiency of the system. To solve this problem, the authors implemented an individual-loss-based discount allocation strategy. Using a weighted bipartite matching graph, the drivers would first be matched with passengers. They would then broadcast the driver's location to see how satisfied the passenger would be with that option. They would then be able to choose one option and contact that driver with their specific location [5].

Oda and Wong proposed a model-free approach called MOVI, a Deep Q-network-based framework that directly learns the optimal vehicle dispatch location and orders to attempt to predict where taxis or ride-sharing services will be needed [6].

Geisberger et al. [7] present an efficient method to calculate the detour between the path of a pair of drivers and passengers. For each request to n offers using $2n$+1 exact computations, and in the fraction of a microsecond, stores the shortest paths of all pairs.

Furthermore, Zhang et al. investigated a taxi-sharing service that is extremely similar to carpooling in order to maximize the driver's profit and user experience. QA Share, a QoSaware taxi-sharing system design, is proposed. The authors devised a heuristic algorithm, which has a significantly faster execution time than previous algorithms, in order to match large numbers of taxi requests. QA Share was shown to result in 38% more profit during a period of 3 months using taxis in a Chinese city [8].

Buchholz et al. [9] present the Strict Partitioning Algorithm (SPA) that split the set of users into *k*-partitions. In a *k*-partition there is no more than *k* users, where *k* represents the capacity constraint of the carpool. Buchholz et al. also prove that the problem is NP-hard for *k* greater than or equal to 3 and give a run-time of $O(n^2)$for the special case of $k = 2$. However, this capacity constraint was a simplification of the problem, as this constraint was assumed to be static. This means that a car, by their definition, could only carry however many people can fit into the car during an entire trip. Realistically, as passengers come on and off the car, the car is capable of carrying much more than the number of passengers that can fit in the car throughout the entire trip.

Duan et al. developed an algorithm that used a dynamic capacity constraint to build off of previous research into the Strict Partition Merging Algorithm [10]. Much of our current research builds off of their paper. Their algorithm takes into consideration geometric properties and improves the Strict Partitioning Algorithm (SPA) by uplifting the static capacity constraint. They develop a Partition Merging Algorithm (PMA), which is a greedy algorithm that they prove, using real and generated data sets, performs better than the SPA. However, the PMA has a runtime of $O(n!)$, so the model is difficult to implement for large samples of cars. Furthermore, their algorithm does not take into account the waiting time restrictions possibly imposed by passengers in the creation of carpools. We implement a more holistic model, where we include waiting-time restrictions in the merging of carpools [11].

3. Proposed model

3.1. Problem formulation

In our proposed model, there will be a set of N people $\mathrm{P} = \{p_1, p_2, p_3, \dots, p_N\}$. Person p_i will have a starting point s_i and a destination d_i. Each person p_i also has a maximum acceptable detour distance δ_i, a seating capacity σ_i, and a maximum waiting time τ_i. Let S and D denote the set of all starting points and destinations respectively. Since our goal is to merge different people into the same carpool, we will have carpool c made up of n different people, where $1 \leq n \leq N$. Therefore, $c \in U$, where $U = \bigcup_{i=0}^{N} P^i$ is the set of all possible carpools. Let $C = \{c\}$ denote the set of carpools that we are trying to merge and

DOI: 10.54254/2755-2721/6/20230389

minimize. The set of all starting points and destinations are denoted by $S_c \subseteq S$ and $D_c \subseteq D$ respectively. In each carpool c_j, one person will be the driver, and there must be at least one possible path χ_j such that all constraints can be met. A path is a ordered set of starting points and destinations of people in the carpool, so $\chi = \left(x_1, x_2, x_3, \ldots, x_{n_j}\right)$, where$\forall x_i \in \chi: x_i \in S_c \cup D_c$Let $f\left(x_i, x_j\right)$ be the function used to calculate the distance between location x_i and x_j, and let $g\left(x_i, x_j\right)$ be the function used to calculate the time needed to travel from x_i to x_j. Let $\Delta_{i,j}$ denote the detour distance in path χ between x_i and x_j, so $\Delta_{i,j} = \sum_{k=i}^{j-1} f(x_k, x_{k+1}) - f\left(x_i, x_j\right)$. Let ω_i be the number of people in the car right after visiting location x_i, so $\omega_i = \sum_{k \le i, x_k \in S_c} 1 - \sum_{k \le i, x_k \in D_c} 1$.

Now after setting up all the definitions, we will look at all the constraints that need to be met.

1. Order Constraint: The first and last locations of path χ must be the starting point and destination of the same person, the driver. For any other person, the starting point must appear in the path before the destination. Therefore, $x_1 = s_\gamma, x_n = d_\gamma, 1 \le \gamma \le N$. Also, $\forall p_i \in c: a < b \; if \; x_a = s_i, x_b = d_i$.
2. Detour Constraint: For each user, the detour distance cannot exceed the maximum acceptable detour distance. Therefore, $\forall p_i \in c: \Delta_{a,b} \le \delta_i \; if \; x_a = s_i, x_b = d_i$.
3. Capacity Constraint: At any time, the number of people in the car may not surpass the seating capacity of the car. Therefore, $\forall i \in [1 \,..\, 2n]: \omega_i \le \sigma_a \; if \; x_1 = s_a$.
4. Inclusion Constraint: All starting points and destinations of passengers in carpool *c* should be included in path χ. Therefore, $S_c \subset \chi, D_c \subset \chi$.
5. Waiting Time Constraint: The waiting time of every person in carpool c should not exceed the person's maximum waiting time constraint. Therefore$\forall p_i \in c: \sum_{j=1}^{k-1} g\left(x_j, x_{j+1}\right) \le \tau_i \; if \; x_k = s_i$

We define two carpools c_a and $c_b$to be merge-able if they exist at least one acceptable path χ for carpool $c_a \cup c_b$ that satisfies all the constraints.

Therefore, we can formulate our problems as below.

minimize	$\lvert C \rvert$	(1)
subject to	$\gamma = \gamma'$ if $\mathrm{x_1} = \mathrm{s_\gamma}$ *and* $\mathrm{x_n} = \mathrm{d_{\gamma'}}$	(2)
	a < b if $x_a = s_i$ and $x_b = d_i$	(3)
	$\forall p_i \in c: \Delta_{a,b} \le \delta_i$ if $x_a = s_i$ and $x_b = d_i$	(4)
	$\forall i \in [1 \,..\, 2n]: \omega_i \le \sigma_a$ if $x_1 = s_a$	(5)
	$S_c \subset \chi$ and $D_c \subset \chi$	(6)
	$\forall p_i \in c: \sum_{j=1}^{k-1} g\left(x_j, x_{j+1}\right) \le \tau_i$ if $x_k = s_i$	(7)

3.2. Problem hardness

The well-known Traveling Salesperson Problem (TSP), which is known to be NP-hard, is equivalent to a special case of the Carpool Scheduling Problem (CSP)

Theorem 1. *The Carpool Scheduling Problem (CSP) is NP-hard.*

Proof. It suffices to prove the equivalence between the TSP problem and some specific cases of the CSP problem. The goal of the TSP problem is to give a set of locations and a starting point, to find the shortest path starting and ending at the starting point that passes through each city exactly once. Suppose we set the maximum carrying capacity to 2 and then reduce the allowed detour to 0%. Suppose all passengers have the same starting and ending point; then, the driver performs the role of the salesman, having to cycle through all passengers (which perform the roles of the cities). The optimal path for the driver is the answer to the TSP problem in this setup.

Since finding the optimal path looping through all passengers requires a massive *O*(*n*!) runtime, in TSP we instead look for paths whose lengths are upper bounded by some limit (1.5 approximation ratio by Christofides). This situation is exactly the same as the CSP problem allowing certain detour percentages. This problem is still NP-hard, as desired.

Therefore, it follows that our Carpool Scheduling Problem is also NP-hard.

DOI: 10.54254/2755-2721/6/20230389

4. Algorithm

4.1. Improved Partition Merging Algorithm based on Tolerate-Time

Algorithm 1 Improved Partition Merging Algorithm based on Tolerate-Time (IPMAT)

Input: A set of users P = $\{p_i\}$, the starting points $\{s_i\}$, the destinations $\{d_i\}$, the maximum acceptable detour distances $\{\delta_i\}$, the seating capacity $\{\sigma_i\}$ and the maximum acceptable waiting times $\{\tau_i\}$.
Output: a set of carpools C.

1: Initialize $C \leftarrow \{\{p_1\}, \{p_2\}, \dots, \{p_N\}\}$.
2: In graph edge set E, build edge (c_i, c_j) iff c_i and c_j are mergeable for $\forall c_i, c_j \in C$.
3: Set $G \leftarrow (C, E)$.
4: **repeat**
5: $E_m \leftarrow maximum\ matching\ of\ G$.
6: Merge c_i and $c_j$if edge $(c_i, c_j) \in E_M, for\ \forall c_i, c_j$.
7: Reset $E \leftarrow \emptyset$.
8: **for** $\forall c_i, c_j \in C$ **do**
9: **if** $\exists p_a, p_b \in c_{i \cup c_j}, f(s_a, s_b) + f(s_b, d_a) - f(s_a, d_a) > \sigma_a\ or\ f(s_a, d_b) + f(d_b, d_a) - f(s_a, d_a) > \sigma_a$ **then**
10: Skip this round.
11: **end if**
12: Initialize a partial order set $S \leftarrow S_{c_i} \cup D_{c_i} \cup S_{c_j} \cup D_{c_j}$.
13: $R \leftarrow$ all topological sort of S.
14: **if** $\exists r \in R$ satisfy the waiting time constraint, detour constraint, and capacity constraint **then**
15: Build edge (c_i, c_j) in E.
16: **end if**
17: **end for**
18: **until** $\boldsymbol{E = \emptyset}$.
19: **return** $\boldsymbol{C}$ as the final carpool-set

This subsection presents the Improved Partition Merging Algorithm based on Tolerate-time (IPMAT). Partition Merging Algorithm (PMA) is a greedy algorithm considering the multi-round matching. Unlike Strict Partition Algorithm (SPA), it makes use of the dynamic capacity of each carpool to strengthen the merge-ability. If the starting points and destinations of users are too far from each other, without constructing any path, we already know that the two carpools cannot merge. PMA can be improved with these geometry properties. However, PMA and Improved Partition Merging Algorithm (IPMA) ignore the difference between waiting for being picked up and sitting in the car. That is to say, PMA and IPMA do not consider customers' waiting time. So IPMAT, considering the max waiting time, is proposed.

As shown in Algorithm 1, IPMAT will initialize the set of carpools *C* at first, making each user their own carpool in *C*. Then, the original carpools will construct the set of vertices for graph *G*. Next, build unweighted undirected edges (c_i, c_j) there exists an admissible path between carpools c_i and c_j. This shows that all users in carpool c_i and c_j can be together in the same carpool (line 2). After initialization, IPMAT tries to merge as many carpools as possible. And we use the Blossom algorithm [12] to achieve

DOI: 10.54254/2755-2721/6/20230389

the maximal number of merges (line 5, 6). If the start points and destinations of two carpools can meet the detour constraint, it is not possible to merge them. Therefore, we can simply skip this round (from line 9 to line 11). After all matching and merging, the graph *G* will shrink in size. Because of the change in the mergeability of the carpools, we need to recalculate E (from line 12 to line 16).

The mergeability check process aims to find one admissible path in the two carpools. Simply, it is a depth-first-search algorithm that checks each possible path with all starting points and destinations of users in both carpools until one admissible path is found. Based on the order constraint, we can use the topological sorting algorithm to decrease the number of possible paths to shrink the search space. Considering the user's waiting time, detour distance, and the capacity of the driver's car, we can also cut down some useless branches to lower the search time.

4.2. Algorithm complexity

Theorem 2. *The worst-case time complexity of IPMAT is* $O\left(n^3\sqrt{n}\right)$.

Proof. We first analyze the complexity of each iteration. Note that by nature of the algorithm, we keep iterating until there no longer exists a maximal matching, i.e. there is nothing left to merge. Indeed, there may be at most *N* iterations, since each iteration cannot do nothing and must thus reduce the total number of carpools by one - there are initially *N* carpools (the individual cars), so there can be at most *N* iterations. Note that in each iteration, there are three steps: merge checking, maximal matching, then physically merging.

- Mergeability Check: Suppose $\sigma = \max(\sigma_1, \sigma_2, \dots, \sigma_N)$. The worst case happens when checking the mergeability of two carpools, both of whom have the maximum allowable size σ. Nevertheless, the amount of time needed to check for this case is constant and of the order $O(\sigma!)$ based on the number of permutations allowable for the ordering of the $\leq\sigma$ cars in any carpool. Since the number of carpools is at most *N*. Since there can be at most $\binom{N}{2}$ possible edges drawn, there are at most $O(N^2)$ merges that need to be checked, so the absolute worst case in this scenario is $O(\sigma!) \times O(N^2) = O(N^2)$.
- Maximal matching: There are many algorithms for finding maximal matching on graphs $G = (V, E)$, of which there is the Blossom algorithm [12] which runs in $O(|E||V|^2)$ and the more advanced algorithm of Micali and Vazirani [13] which runs in $O\left(E\sqrt{|V|}\right)$. Due to the fact that $|E| \leq \binom{|V|}{2} = O(N^2)$ and $|V| \leq N = O(N)$, the worst complexity from Blossom is $O(N^5)$ and the worst complexity from Micali and

  Vazirani is $O\left(N^2\sqrt{N}\right)$.
- Merging: In a graph with $|V| \leq N$, the maximal matching will have size $\leq \frac{|V|}{2} = O(N)$. The actual details of the merging involving the reorganization of information is a constant depending on $\sigma_i, \delta_i, \tau_i$.

In summary, the worst-case complexity of a single iteration of IPMAT is $O(N^2) + O\left(N^2\sqrt{N}\right) + O(N) = O\left(N^2\sqrt{N}\right)$. As there are $\leq N$ iterations, the worst-case complexity is $O\left(N^3\sqrt{N}\right)$.

However, in our implementation of the algorithm, we use the Blossom algorithm, so the runtime of the code we use to generate simulations will have a runtime of around $O(N^6)$. If we take only the first k topological orders, then for k sufficiently smaller than N, the runtime is $O(N^5)$.

5. Experiment

5.1. Data sets

This subsection will introduce the data set used in our experiment. The data set used in this research was taken from the 2016 NYC yellow taxi trip duration data. The data was comprised of 1458644 trip records. Each trip record was labeled with a unique ID, including the start and end times of the trip, the number of passengers in the car during the trip, and the starting and ending locations expressed in longitude and latitude. We used this data set to evaluate our new algorithm. We use relative distance to show the detour

DOI: 10.54254/2755-2721/6/20230389

distance of each user, i.e., the percentage of each user's travel distance. After removing obvious outliers, we randomly choose 500 trips from all data sets multiple times as our input.

Here is some visual representation of the data:

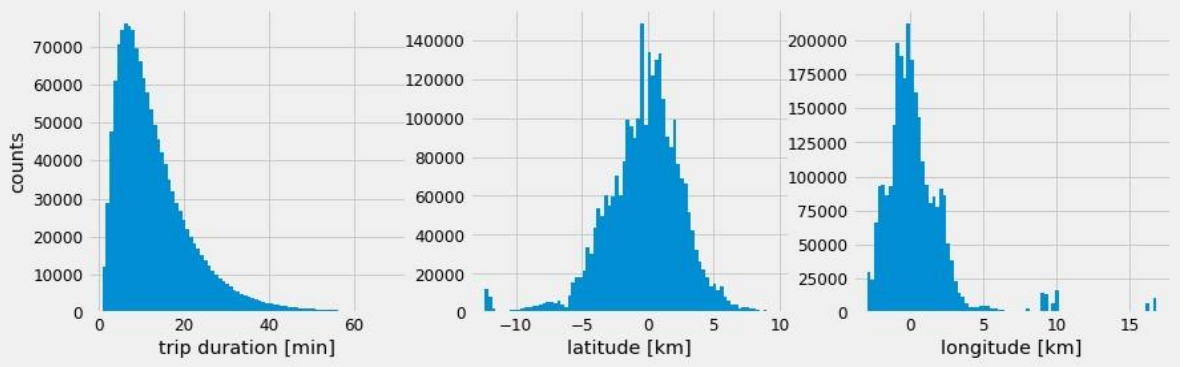

Figure 1. Trip Distance and Duration.

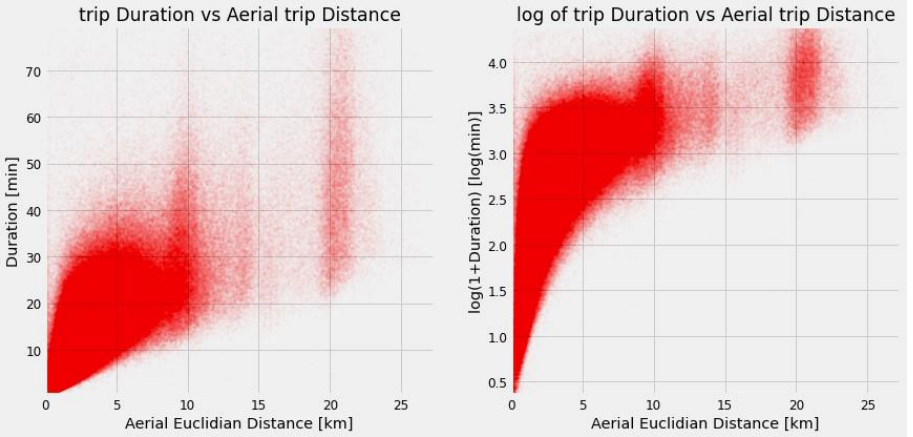

Figure 2. Data-set Distribution.

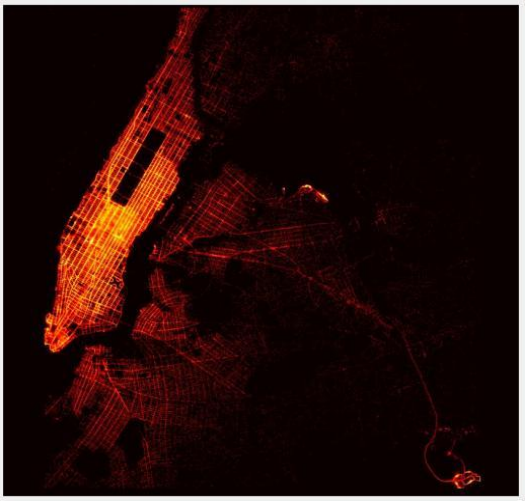

Figure 3. Hot map of all trips.

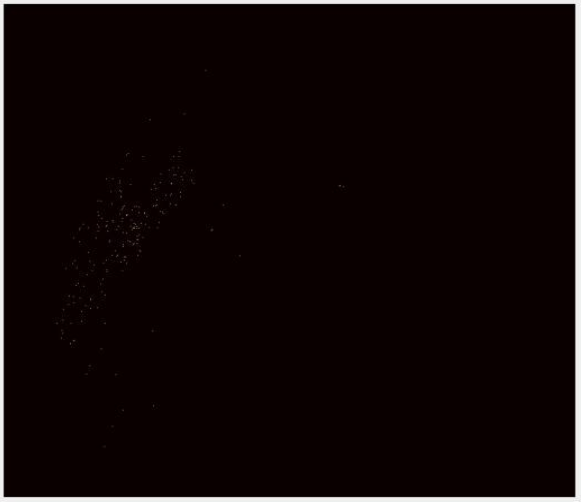

Figure 4. Hot map of sample trips.

5.2. Experimental settings

Since the number of possible topological orders varies intensely in different partial order sets, the time needed to apply complete IPMAT may fluctuate in different data sets. To control time consumption, we only iterate k times, using the first k topological sequences instead of using all the topological sequences

DOI: 10.54254/2755-2721/6/20230389

in IPMAT. And k is set to 1000 in this experiment to show the effect of detour better and waiting time. We use 3 different percentages, 5%, 10%, and 15% to imply detour distance in different scales. And we set 2 different constants, 100 and 200 to present the max waiting time for each user. The capacity of each user is set to 4 since a capacity of 4 is analogous to a real-world scenario. All results are averaged over 50 times for smoothness. In all, there are 500 user requests involved. Since we are trying to reflect a real-world situation, this experiment runs based on the NYC data set.

5.3. Evaluation results

The performances of IPMAT-100, IPMAT-200, IPMAT-1000 are shown in Figure 5, Figure 6 and Figure 7. All three figures reflect as the number of users and detour rate increases, the carpool number increases with 100 units maximum waiting time. The matching rete which defines as the number of carpools divided by the total number of users, floats around 35%. It shows that IPMAT's performance is approximately equivalent to the traditional algorithm, even though it considers extra factor. In addition, IPMAT has good stability when detour rate. When the number of iteration changes, the difference is quite slight. Thus, only using the algorithm with 100 iterations will be likely to generate significance carpool scheme in reality.

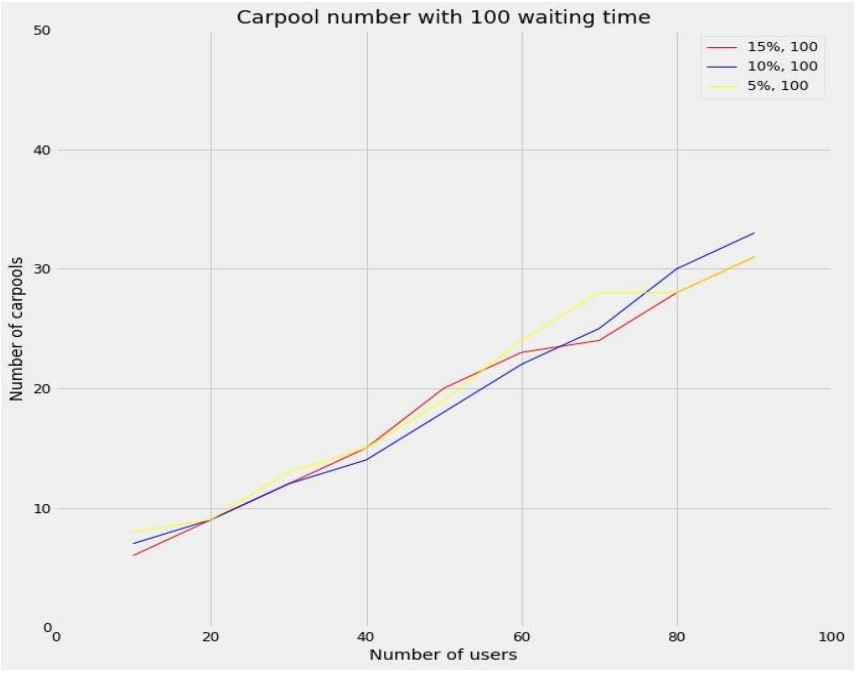

Figure 5. Carpool numbers with 100 units maximum waiting time.

DOI: 10.54254/2755-2721/6/20230389

Figure 6. Carpool numbers with 200 units maximum waiting time.

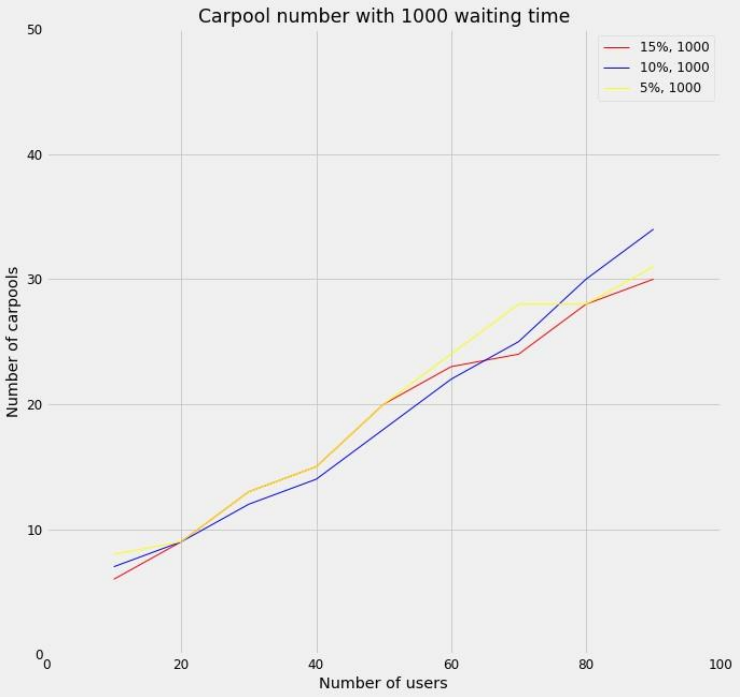

Figure 7. Carpool with 1000 units maximum waiting time.

6. Conclusion

In this paper, we discuss the problem of carpooling to reduce the number of cars on the road and consider the waiting time of users. We analyze the reasons for carpooling and the carpooling scheme. Finally, a greedy algorithm, IPMAT, is proposed to compute the locally optimal results of the carpooling problem.

DOI: 10.54254/2755-2721/6/20230389

We use the variable IPMAT in our experiments to reduce the algorithm time consumption. We do not use all topological sequences in IPMAT, but only the first *k* topological sequences. To assess the performance of our algorithm, we run our algorithm on real-world data sets. The results show that our method outperforms SPA in many cases. This paper does not include traffic conditions in the calculation of the time required for a trip, which can be investigated as future work.

Acknowledgements

We would like to thank our mentor Professor David Woodruff for his valuable insights and guidance throughout this effort, as well as TA Yunshan Guo for advice and direction in the editing process.
Andrew Wen, Jinglin Xu, Ryan Wu, Kai Chen, and Yuntao Lin contributed equally to this work and should be considered co-first authors.

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] 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.

[3] He, W, Kai, H, Li, D. Intelligent carpool routing for urban ridesharing by mining GPS trajectories. IEEE Trans Intell Transp Syst 2014; 15(5): 2286–2296.

[4] Zhao T, Yang Y, Wang E. Minimizing the average arriving distance in carpooling. International Journal of Distributed Sensor Networks. January 2020. doi:10.1177/1550147719899369

[5] Y. Duan, G. Gao, M. Xiao and J. Wu, "A Privacy-Preserving Order Dispatch Scheme for Ride-Hailing Services," 2019 IEEE 16th International Conference on Mobile Ad Hoc and Sensor Systems (MASS), 2019, pp. 118-126

[6] T. Oda and C. Joe-Wong, "Movi: A model-free approach to dynamic fleet management," arXiv preprint arXiv:1804.04758, 2018.

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

[8] 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.

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

[10] Y. Duan, T. Mosharraf, J. Wu, and H. Zheng, "Optimizing carpool scheduling algorithm through partition merging," in IEEE ICC, 2018.

[11] N. Agatz, A. Erera, M. Savelsbergh, and X. Wang, "Optimization for dynamic ridesharing: A review," European Journal of Operational Research, vol. 223, no. 2, pp. 295–303, 2012.

[12] A. Shoemaker, S. Vare, "Edmonds' Blossom Algorithm," Standford University, CME 323, 2016.

[13] S. Micali, V. Vazirani, "An $O(\sqrt{V}E)$ algorithm for finding maximum matching in general graphs," 21st IEEE FOCS, pp. 17–27, 1980.