How different passengers contribute to ridesharing scheduling?

Bachelor's Thesis Final Report

Qian Xie, supervised by Yang Yu
Institute of Interdisciplinary Information Sciences

Contents

1 Introduction

1.1 Background

Transportation is an indispensable part of a city. However, while more and more individuals or families can afford a private car, urban diseases such as traffic congestion, air pollution are disturbing citizens' daily life. Take Beijing as an example, it's normal to wait for a long time at a busy crossing. Getting a car plate needs waiting at a license-plate lottery system, which is really competitive. Needless to say the great number of vehicles on the road has contributed to the terrible air conditions.

Nevertheless, the emergence of ridesharing platforms such as Uber, Lyft and DiDi seems to alleviate the situation, since the number of cars on the road and even the ownership may have a decline. Besides dispatching nearby taxi or private car to pick up the rider, ridesharing platforms also provide pooling services that dispatching a driver to several riders in the neighborhood. Such pooling services can reduce the rate of empty seats and further benefiting the transportation and environment.

When it comes to pooling, things become more intricate. There are two and even more riders sharing a same car service. They have different origins and destinations (*OD pairs*) but supposed to share a similarity between routes. They have sequences calling the app and being dispatched. The drivers have to detour a short distance to pick up and drop off each rider. On the riders' side, they have to face a trade-off: on the one hand, choosing pooling service is usually cheaper; on the other hand, they have to wait a longer time to be picked up. On the platform's side, they are concerned with two aspects: dispatching and pricing, based on the spatial-temporal information (OD pairs, time, distance) of calling riders in the current time frame. Dispatching and pricing are also two hot topics discussed by researchers in various fields.

1.2 Objective

Here we are interested in a topic somewhat related to pricing: **how to evaluate the contribution of heterogeneous riders to the carpooling**. Thus we can charge them accordingly based on their contribution. Intuitively, some riders would start from or pass hot spots, so it's more likely to find nearby callers (riders who call the pooling service) during the trip. Some riders, on the contrary, may in a remote or unpopular area, lead to a longer pick-up distance and higher empty seat rate. It seems that the latter case takes up more resources than the former case and thus contributes less to the social welfare. To make our discussion more rigorous and convincing, we need to set up a model for analysis and quantify contribution in a mathematical way.

1.3 Overview

Fortunately, there is a concept in game theory named *Shapley Value* offering a powerful mathematical tool to answer our question. Shapley Value intends to provide a fair distribution of total surplus among players in a game. But it can also be phrased as a

characterization of each player's importance/contribution to the overall cooperation and thus derive each player's reasonable expected payoff.

To formulate the problem, we also need to simplify the intricate reality and list all possible pooling cases, that is, how are OD pairs distributed in an area, how platform dispatch cars to pick up riders and how drivers detour to pick up and drop off each rider. There are two approaches: one is to theoretically analyze the road network topology, one is to empirically run a simulation and inference the pattern. In our work, we plan to use both methods, or start from experiments to get inspiration for the theoretical analysis.

After measuring the contribution, we hope to apply our methods to real-world scenario and further discuss the **characteristics of carpooling in different cities**, that is, which city has a higher efficiency on carpooling. To achieve the goal, we need to obtain real-world data and play on it. Another goal is to devise the **pricing method based on riders' potential contribution**. Note that the platform has to display the estimated prices on the app for riders once they call the pooling service, no matter the cars dispatched for them have passengers already or not. So the estimated prices should be based on the historical results rather than their coming pooling trips. We may utilize data mining and unsupervised learning to accomplish the two tasks.

2 Literature Review

Shapley Value [10] was proposed by Lloyd Shapley in 1953 as an numerical evaluation of the "values" of players in a cooperative game and later explained in details in some books [11, 8]. The concept has a great social impact and was widely employed in various areas, like the Shapley-Shubik Index [12] discussing the voting rules of United Nations Security Council. Besides social issues, recently many researchers apply the solution concept to engineering areas like networking [13], cloud computing [3] and smart grid [4, 7].

There are also a volume of works focusing on pricing methods such as VCG mechanism [6] and carpooling scheduling methods such as online bipartite matching [1], partition-based matchmaking algorithm [9] and genetic algorithm [2].

The most related work is Li et al's paper [5] proposing a dynamic pricing method using Shapley Value that divides the payoff according to different riders' contribution. The difference is that

- The pricing method suggested in Li et al's paper is ex-post, that is, the calculation of division is based on the assumption that the payments are charged at the end of the trip. However, in the real-world, platform has to provide an estimated price right at the time the rider call the pooling service. So here we attempt to devise an ex-ante method.

- Li et al's paper narrows down the coalitional game for this problem on three-player case, that is, only a driver and two riders. However, in the real-world, it is possible that two to four riders on the same car. Besides, since we think the problem from an ex-ante angle, the potential number of riders to be picked up can be various. For example, in a popular area, there may be 8 people waiting for pooling service at the same time. That's why we would like to do a simulation on multiple OD pairs.

- There exists problems in topology of the detoured trips in Li et al's paper. The detour not only happens during the pick-up but also during the drop-off. Not only the riders have to deviate but also the drivers. We will avoid such problems and be more rigorous in our research.

3 Conceptual Framework

3.1 Key Concept

Definition 3.1. A **coalition** S is defined to be a subset of the set of players $N = \{1, 2, \cdots, n\}$, $S \subset N$. The set of all coalitions is denoted by 2^N.

Definition 3.2. The **coalitional form** of an n-person game is given by the pair (N, v) where $N = \{1, 2, \cdots, n\}$ is the set of players and v is a real-valued function, called **characteristic function** of the game, defined on the set of all coalitions 2^N, satisfying

- $v(\emptyset) = 0$
- (superadditivity) if S ad T are disjoint coalitions, i.e. $S \cap T = \emptyset$, then $v(S) + v(T) \leq v(S \cup T)$

Definition 3.3. A payoff vector $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ where x_i is the payoff that player i is to receive, is said to be **group rational** or **efficient** if $x_1 + x_2 + \cdots x_n = v(N)$, that is, the total amount received by the players should be $v(N)$.

Definition 3.4. A payoff $\mathbf{x}$ is said to be **individual rational** if $x_i \geq v(\{i\})$ for all $i = 1, 2, \cdots, n$, that is, no player would agree to receive less than that player could obtain acting alone.

Definition 3.5. An **imputation** is a payoff vector that is group rational and individually rational. The set of imputations can be written as

$$\{\mathbf{x} = (x_1, x_2, \cdots, x_n) : \sum_{i \in N} x_i = v(N) \text{ and } x_i \geq v(\{i\}) \text{ for all } i \in N\}$$

Definition 3.6. An imputation $\mathbf{x}$ is said to be **unstable** if there is a coalition S such that $v(S) > \sum_{i \in S} x_i$ and said to be **stable** otherwise. It means that if there exists coalition S whose total return from $\mathbf{x}$ is less than what the coalition can achieve acting by itself, then there will be a tendency for the coalition to upset the proposed $\mathbf{x}$.

Definition 3.7. The set C of imputations is called the **core** where

$$C = \{\mathbf{x} = (x_1, x_2, \cdots, x_n) : \sum_{i \in N} x_i = v(N) \text{ and } \sum_{i \in S} x_i \geq v(S) \text{ for all } S \subset N\}$$

Definition 3.8. A **value function** ϕ is an n-tuple of function that assigns to each possible characteristic function v of an n-person game as

$$\phi(v) = (\phi_1(v), \phi_2(v), \cdots, \phi_n(v))$$

where $\phi_i(v)$ represents the worth or value of player i in the game with characteristic function v.

Theorem 3.1. *There exists a unique function ϕ satisfying the following* ***Shapley axioms****.*

- *(Efficiency)* $\sum_i \phi_i(v) = v(N)$
- *(Symmetry) if i and j satisfy* $v(S \cup \{i\}) = v(S \cup \{j\}$ *for every coalition S not containing i and j, then* $\phi_i(v) = \phi_j(v)$
- *(Dummy Axiom) if i satisfies* $v(S) = v(S \cup \{i\})$ *for every coalition S not containing i, then* $\phi_i(v) = 0$
- *if u and v are characteristic functions, then* $\phi(u + v) = \phi(u) + \phi(v)$

Theorem 3.2. *The Shapley value is given by* $\phi = (\phi_1, \cdots, \phi_n)$ *where for* $i = 1, \cdots, n$,

$$\phi_i(v) = \sum_{S \subset N, i \in S} [(|S| - 1)!(n - |S|)!][v(S) - v(S - \{i\})]/n!$$

3.2 Problem Formulation

We formalize the carpooling as a game in coalitional form. There are two kinds of players in this game: **drivers** and **passengers**. Denote the set of drivers and the set of passengers as $D = \{d_1, d_2, \cdots\}$ and $P = \{p_1, p_2, \cdots\}$. The set of players is $N = D \cup P$. Each driver d_i has a location of l_i, each passenger j has an origin s_j and destination t_j.

Definition 3.9. A **route** is defined as

$$(l_i, s_j, s_k, t_j, t_k)$$

meaning that d_i picks up p_j first and p_k next then drops off p_j first and p_k next, or

$$(l_i, s_j, s_k, t_k, t_j)$$

meaning that d_i picks up p_j first and p_k next then drops off p_k first and p_j next, or just

$$(l_i, s_j, t_j)$$

meaning that d_i serves only for p_j.

Definition 3.10. The **pick-up distance** x_p is defined as

$$x_p(p_j) = d(l_i, s_j)$$

$$x_p(p_k) = d(l_i, s_j) + d(s_j, s_k)$$

where $d(x, y)$ is the distance between x and y.

Definition 3.11. The **sharing distance** x_s is defined as

$$x_s(d_i) = x_s(p_j) = x_s(p_k) = \min\{d(s_j, t_j), d(s_k, t_k), d(s_j, t_k), d(s_k, t_j)\}$$

Definition 3.12. The **traveling distance** x_t is defined as

$$x_t(p_j) = x_s(p_j) + \delta_1 * d(s_j, s_k) + \delta_2 * d(t_j, t_k)$$

where $\delta_1 = 1$ if p_j is the first to pick up and 0 otherwise, $\delta_2 = 0$ if p_k is the fist to drop off and 0 otherwise.

Definition 3.13. The **charging distance** x_c is defined as

$$x_c(d_i) = d(s_j, t_j) + d(s_k, t_k)$$

if succeed in carpooling, otherwise

$$x_c(d_i) = d(s_j, t_j)$$

All distances defined above is supposed to be 0 if the passenger fails to be dispatched a car.

Definition 3.14. The **utility function** is defined as

$$u(p_i) = \alpha * x_p(p_i) + \beta * x_t(p_i)$$

$$u(d_i) = \gamma * x_c(d_i)$$

where α, β, γ are elasticity, x_p and x_t are pick-up distance and traveling distance respectively. The sign of α and β are negative and the sign of γ is positive.

Definition 3.15. For any subset $S \subset N$, the characteristic function is constructed from the utility function as

$$v(S) = \sum_{i \in S} u(i)$$

Note that $v(\emptyset) = 0$ and $v(S) + v(T) \leq v(S \cup T)$, the conditions of characteristic function are satisfied. Now we can substitute the characteristic function into the formula of the Shapley value.

4 Carpooling Simulation

4.1 Simulation Setup

In the simulation, for simplicity we consider a typical time frame that represents a short period of time, say 5 minutes, in the real-world scenario and a square that represents a geographical area, say a city. I sample the locations of 40 cars and OD pairs of 30 passengers in the square from the uniform distribution. Each car has two available seats.

Sampled data of car locations	
Id	location
0	(14.952217507147058, 26.844565718046496)
1	(18.852794470205147, 87.74882490214252)
2	(28.75545934688396, 81.55792053550675)
3	(49.60700219206683, 58.23358995412835)

Sampled data of passenger OD pairs		
Id	origin	destination
0	(51.011893091339836, 24.52031316125599)	(42.365106568425794, 7.922640746665611)
1	(68.68188697548895, 81.27765091032052)	(72.31095644440236, 55.55788517723118)
2	(25.47031972460838, 36.1498662361893)	(42.518319217711905, 15.136998042362071)

The assumption we make here is that the car will always go in straight line between any two of the five locations: the car location, the first pick-up location, the second pick-up location, the first drop-off location, the second drop-off location. For a successful carpooling, there are four possible cases:

- pickup passenger 1, pickup passenger 2, drop off passenger 1, drop off passenger 2
- pickup passenger 1, pickup passenger 2, drop off passenger 2, drop off passenger 1
- pickup passenger 2, pickup passenger 1, drop off passenger 1, drop off passenger 2
- pickup passenger 2, pickup passenger 1, drop off passenger 2, drop off passenger 1

Therefore our carpooling algorithm used here works as the following. First, compute the bipartite matching between drivers and passengers that minimize the total distance of drivers traveling to pick-up passengers. Next, we pair up the passengers according to the distance matrix. Each pair of passengers who are close to each other is ready for carpooling. Finally, for each pair, we consider the two assigned cars with the four cases respectively and choose the car and the case that can optimize the driver's total traveling distance.

bipartite matching minimizing total pick-up distances		
car	passenger	pick-up distance
17	0	12.840967153620223
9	1	1.611361089514806
11	2	11.181213879644423

pick-up and drop-off orders for each carpooling				
car	1st pick-up	2nd pick-up	1st drop-off	2nd drop-off
24	17	6	6	17
9	23	1	1	23
34	9	21	21	9
28	15	4	4	15
19	26	22	26	22
23	3	16	3	16
12	13	5	5	13
37	19			19
8	14			14

4.2 Simulation Result

We can observe that some passengers (e.g. passenger 19 and passenger 14) fail to be matched with another passenger for carpooling. The platform has to dispatch a car for such passenger specifically (e.g. car 37 and car 8). Now we calculate the Shapley value, the quantified contribution to the overall carpooling, for each passenger.

5 Conclusion

5.1 Broader Impacts

The proposed research will help answer some interesting questions. What are the values of customers to the ridesharing considering their trip spatial-temporal information and interactions between each other? Are there any more reasonable and fairer pricing methods for the pricing platforms? Can the ridesharing platform have a better dispatching strategy based on the contribution information of passengers? These questions are seldom discussed before, so our research is original. The results may make an impact on reality and benefit both platform and riders.

5.2 Future Work

In this paper, I assume that each car has only two available seats and each passenger calls for only one seat and in the future we can extend to more seats cases. Once obtain the real-world data, we can study the characteristics of carpooling in a geographical area. We can also devise the pricing method based on the Shapley value.

References

[1] Z. Huang, N. Kang, Z. G. Tang, X. Wu, Y. Zhang, and X. Zhu. How to match when all vertices arrive online. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 17–29. ACM, 2018.

[2] M.-K. Jiau and S.-C. Huang. Services-oriented computing using the compact genetic algorithm for solving the carpool services problem. *IEEE Transactions on Intelligent Transportation Systems*, 16(5):2711–2722, 2015.

[3] R. Kaewpuang, D. Niyato, P. Wang, and E. Hossain. A framework for cooperative resource management in mobile cloud computing. *IEEE Journal on Selected Areas in Communications*, 31(12):2685–2700, 2013.

[4] W. Lee, L. Xiang, R. Schober, and V. W. Wong. Direct electricity trading in smart grid: A coalitional game analysis. *IEEE Journal on Selected Areas in Communications*, 32(7):1398–1411, 2014.

[5] S. Li, F. Fei, D. Ruihan, S. Yu, and W. Dou. A dynamic pricing method for carpooling service based on coalitional game analysis. In *High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2016 IEEE 18th International Conference on*, pages 78–85. IEEE, 2016.

[6] H. Ma, F. Fang, and D. C. Parkes. Spatio-temporal pricing for ridesharing platforms. *arXiv preprint arXiv:1801.04015*, 2018.

[7] P. H. Nguyen, W. L. Kling, and P. F. Ribeiro. A game theory strategy to integrate distributed agent-based functions in smart grids. *IEEE Transactions on Smart Grid*, 4(1):568–576, 2013.

[8] M. J. Osborne and A. Rubinstein. *A course in game theory*. MIT press, 1994.

[9] D. Pelzer, J. Xiao, D. Zehe, M. H. Lees, A. C. Knoll, and H. Aydt. A partition-based match making algorithm for dynamic ridesharing. *IEEE Transactions on Intelligent Transportation Systems*, 16(5):2587–2598, 2015.

[10] L. S. Shapley. A value for n-person games. *Contributions to the Theory of Games*, 2(28):307–317, 1953.

[11] L. S. Shapley, A. E. Roth, et al. *The Shapley value: essays in honor of Lloyd S. Shapley*. Cambridge University Press, 1988.

[12] L. S. Shapley and M. Shubik. A method for evaluating the distribution of power in a committee system. *American political science review*, 48(3):787–792, 1954.

[13] C. Singh, S. Sarkar, A. Aram, and A. Kumar. Cooperative profit sharing in coalition-based resource allocation in wireless networks. *Ieee/acm transactions on networking (ton)*, 20(1):69–83, 2012.

A Simulation Codes

```
import random
import math
from munkres import Munkres
import networkx as nx
import matplotlib.pyplot as plt
import networkx.algorithms as nxa

CAR_NUMBER = 40
PASSENGER_NUMBER = 30

LENX = 100
LENY = 100


class Point:
        def __init__(self, x, y):
                self.x = x
                self.y = y

        def getX(self):
                return self.x
        def getY(self):
                return self.y

        def print(self):
                print("(", self.x, ", ", self.y, ")", end="")

class Car:
        def __init__(self, carId):
                self.carId = carId
                self.p = Point(random.uniform(0,LENX), random.uniform(0,LEN

        def getDistance(self, p):
                x = self.p.getX() - p.getX()
                y = self.p.getY() - p.getY()
                return math.sqrt(x * x + y * y)

        def print(self):
                print("car ", self.carId, " position: ", end="")
                self.p.print()
                print("")
```

```
class Passenger:
        def __init__(self, passengerId):
                self.passengerId = passengerId
                self.s = Point(random.uniform(0,LENX), random.uniform(0,LEN
                self.e = Point(random.uniform(0,LENX), random.uniform(0,LEN

        def getStart(self):
                return self.s

        def getEnd(self):
                return self.e

        def print(self):
                print("passenger ", self.passengerId, " from: ", end="")
                self.s.print()
                print(" to ", end="")
                self.e.print()
                print("")

        def getDistance(self):
                x = self.s.getX() - self.e.getX()
                y = self.s.getY() - self.e.getY()
                return math.sqrt(x * x + y * y)

        def getId(self):
                return self.passengerId

def getDistance(p1, p2):
        x = p1.getX() - p2.getX()
        y = p1.getY() - p2.getY()
        return math.sqrt(x * x + y * y)

def getIndex(indexes, i):
        for row, column in indexes:
                if row == i:
                        return column

def getCarpooling(car, passenger1, passenger2):
        # first pick up 1, then pick up 2, then dropoff 2, then dropoff 1
        d1 = car.getDistance(passenger1.getStart()) + getDistance(passenger1
                        getDistance(passenger2.getStart(), passenger2.getEnd
        # first pick up 1, then pick up 2, then dropoff 1, then dropoff 2
        d2 = car.getDistance(passenger1.getStart()) + getDistance(passenger1
                        getDistance(passenger2.getStart(), passenger1.getEnd
        # first pick up 2, then pick up 1, then dropoff 1, then dropoff 2
```

```
        d3 = car.getDistance(passenger2.getStart()) + getDistance(passenger2
                        getDistance(passenger1.getStart(), passenger1.getEnd
        # first pick up 2, then pick up 1, then dropoff 2, then dropoff 1
        d4 = car.getDistance(passenger2.getStart()) + getDistance(passenger2
                        getDistance(passenger1.getStart(), passenger2.getEnd
        d = [d1, d2, d3, d4]
        index = d.index(min(d))
        return (index, d[index])

def getSeq(passenger1, passenger2, index):
        result = []
        if(index < 2):
                result.extend([passenger1.getId(), passenger2.getId()])
        else:
                result.extend([passenger2.getId(), passenger1.getId()])
        if(index == 0 or index == 3):
                result.extend([passenger2.getId(), passenger1.getId()])
        else:
                result.extend([passenger1.getId(), passenger2.getId()])
        return result

cars = []
passengers = []
carId = 0
passengerId = 0
for i in range(0, CAR_NUMBER):
        cars.append(Car(carId))
        carId = carId + 1
for i in range(0, PASSENGER_NUMBER):
        passengers.append(Passenger(passengerId))
        passengerId = passengerId + 1

for car in cars:
        car.print()
for passenger in passengers:
        passenger.print()

matrix = []
for passenger in passengers:
        line = []
        for car in cars:
                line.append(car.getDistance(passenger.getStart()))
        matrix.append(line)

m = Munkres()
```

```
indexes = m.compute(matrix)
# print(matrix)

print('Lowest cost through this matrix:')
total = 0
for row, column in indexes:
    value = matrix[row][column]
    total += value + passengers[row].getDistance()
    print('car ', column, ' picks up passenger ', row , ' takes', value, "
print('total cost: ', total)


FG=nx.Graph()
for i in range(0, passengerId):
        for j in range(i, passengerId):
                distance = getDistance(passengers[i].getStart(), passengers
                FG.add_weighted_edges_from([(i,j,400 -  distance)])
dc = nxa.max_weight_matching(FG)

indexes2 = []
total = 0
print(indexes)
for (i, j) in dc:
        car1 = getIndex(indexes, i)
        car2 = getIndex(indexes, j)
        od1 = matrix[i][car1] + passengers[i].getDistance()
        od2 = matrix[j][car2] + passengers[j].getDistance()
        od =  od1 + od2
        (id1, d1) = getCarpooling(cars[car1], passengers[i], passengers[j])
        (id2, d2) = getCarpooling(cars[car2], passengers[i], passengers[j])
        if(d1 > d2):
                if(d2 < od):
                        row = [car2]
                        row.extend(getSeq(passengers[i], passengers[j], id2
                        row.append(d2)
                        indexes2.append(row)
                        total += d2
                else:
                        indexes2.append([car1, passengers[i].getId(), -1, -
                        indexes2.append([car2, passengers[j].getId(), -1, -
                        total += od
        else:
                if(d1 < od):
                        row = [car1]
                        row.extend(getSeq(passengers[i], passengers[j], id1
```

```
                        row.append(d1)
                        indexes2.append(row)
                        total += d1
                else:
                        indexes2.append([car1, passengers[i].getId(), -1, -
                        indexes2.append([car2, passengers[j].getId(), -1, -
                        total += od
for row in indexes2:
        print(row)
        print('car ', row[0], end = "")
        print(' picks up passenger ', row[1], end ="")
        if(row[2] != -1):
                print(' picks up passenger ', row[2], ' dopoff passenger ',
        print(" dopoff passenger : ", row[4], ' takes ', row[5])
print('total cost: ', total)
```