12-2016

Scheduling for Timely Passenger Delivery in a Large Scale Ride Sharing System

Yang Zhang
University of Tennessee, Knoxville, yzhan157@vols.utk.edu

To the Graduate Council:

I am submitting herewith a thesis written by Yang Zhang entitled "Scheduling for Timely Passenger Delivery in a Large Scale Ride Sharing System." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Engineering.

Hairong Qi, Lee Han, Major Professor

We have read this thesis and recommend its acceptance:

Husheng Li

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Scheduling for Timely Passenger Delivery in a Large Scale Ride Sharing System

A Thesis Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Yang Zhang
December 2016

DEDICATION

This thesis is dedicated to my parents, Shuzhi Liu and Dejing Zhang,
for bringing me to the world,
for raising me, loving me,
for the support in my education,
to my sister Yi Zhang
for her love.

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to Dr. Hairong Qi, Dr. Lee. Han, Dr. Husheng Li for their continuous support and guidance, for all the talks and emails, and their precious time. Special thanks to Ed Forshee for taking care of me in life. Many thanks to all my other friends who have loved and supported me!

ABSTRACT

Taxi ride sharing is one of the most promising solutions to urban transportation issues, such as traffic congestion, gas insufficiency, air pollution, limited parking space and unaffordable parking charge, taxi shortage in peak hours, etc. Despite the enormous demands of such service and its exciting social benefits, there is still a shortage of successful automated operations of ride sharing systems around the world. Two of the bottlenecks are: (1) on-time delivery is not guaranteed; (2) matching and scheduling drivers and passengers is a NP-hard problem, and optimization based models do not support real time scheduling on large scale systems.

This thesis tackles the challenge of timely delivery of passengers in a large scale ride sharing system, where there are hundreds and even thousands of passengers and drivers to be matched and scheduled. We first formulate it as a mixed linear integer programming problem, which obtains the theoretical optimum, but at an unacceptable runtime cost even for a small system. We then introduce our greedy agglomeration and Monte Carlo simulation based algorithm. The effectiveness and efficiency of the new algorithm are fully evaluated: (1) Comparison with solving optimization model is conducted on small ride sharing cases. The greedy agglomerative algorithm can always achieve the same optimal solutions that the optimization model offers, but is three orders of magnitude faster. (2) Case studies on large scale systems are also included to validate its performance. (3) The proposed greedy algorithm is straightforward for parallelization to utilize distributed computing resources. (4) Two important details are discussed: selection of the number of Monte Carlo simulations and proper calculation of delays in the greedy agglomeration step. We find out from experiments that the sufficient number of simulations to achieve a “sufficiently optimal solution” is linearly related to the product of the number of vehicles and the number of passengers. Experiments also show that enabling margins and counting early delivery as negative delay leads to more accurate solutions than counting delay only.

TABLE OF CONTENTS

Chapter One Introduction 1
Ride Sharing Services 1
Chapter Two Literature Review On Solution Methods 4
History and Variations of Vehicle Routing Problems 4
Capacitated Vehicle Routing Problem (CVRP) 4
Vehicle Routing Problem with Pickup and Delivery (VRPPD) 5
Vehicle Routing Problem with Time Window (VRPTW) 5
Vehicle Routing Problem with Pickup and Delivery with Time Window (VRPPDTW) 6
Timely Ridesharing Problem 6
Chapter Three A Mixed-Integer Linear Programming Formulation 8
Model Formulation 8
Terminology 9
Objective Function and Constraints 10
Linearization 11
A Case Study 12
Chapter Four Greedy Agglomerative Clustering and Monte Carlo Simulation Based Method 14
Algorithm Development 14
Complexity Analysis 17
Case Studies 18
Parallelization 28
Discussion 29
Selection of the number of simulations 29
Proper calculation of delays 32
Chapter Five Travel Time Extraction From Real Road Network 37
Chapter Six Conclusion and Future Work 41
List of References 44
Appendix 49
Vita 53

LIST OF TABLES

Table 3.1. Symbols and their definitions. 9
Table 4.1. Runtime Comparison of solving optimization model and greedy agglomeration algorithm on small cases. 19
Table 4.2. 95% quantiles of sufficient number of simulations 32

LIST OF FIGURES

Figure 1.1. Example: a three-vehicle-five-passenger ride sharing system. 1
Figure 1.2. Examples of people's unpleasant experience of using taxi ride-sharing service. 3
Figure 3.1. A 3-vehicle-6-passenger case study. 13
Figure 4.1. Monte Carlo simulation and passenger list shuffling 16
Figure 4.2. Greedy Agglomeration step to find the best matching and service order to achieve minimum system-wide service delay 16
Figure 4.3. Find the best service order and delay increase after adding a new passenger to a vehicle. 17
Figure 4.4. Relationship between achieved minimum system delay and the number of simulations. 19
Figure 4.5. Setups and optimal scheduling of the 6 test cases. (Notation: left – setup (travel time/requested arrival time), right - optimal scheduling (requested arrival time/actual arrival time/delay)) 20
Figure 4.6. Histograms of the system delays in the 200 simulations. 23
Figure 4.7. A 50-passenger-20-vehicle case. 24
Figure 4.8. A 100-passenger-50-vehicle case 25
Figure 4.9. A 200-passenger-100-vehicle case 26
Figure 4.10. A 300-passenger-150-vehicle case 27
Figure 4.11 Runtime test of a 100-passenger-50-vehicle system using different numbers of processors. 28
Figure 4.12. Improved solution values and increasing number of simulations 30
Figure 4.13. Histograms of the number of simulations to hit sufficiently optimal solution evaluated by Definition 1, 200 trials in total. 31
Figure 4.14. Histograms of the number of simulations to hit sufficiently optimal solution evaluated by Definition 2, 200 trials in total. 31
Figure 4.15. Relationship between sufficient number of simulations and the product of number of passengers (NP) and number of vehicles (NV). 32
Figure 4.16. Comparison of minimum delay value changes using two different ways of calculating delays 34
Figure 5.1. Multi-source multi-target shortest path algorithm based on Dijkstra's. 38
Figure 5.2. Multi-source multi-target shortest path algorithm based on A*. 39
Figure 5.3. Runtime comparison of Dijkstra's and A* many-to-many shortest path algorithms. 40

CHAPTER ONE
INTRODUCTION

Ride Sharing Services

Ride sharing, a travel mode initially launched by the U. S. government for the purpose of reducing fuel consumption during WWII and the 1970s fuel crisis [1, 2], has seen its growing necessity over the past few decades. Expanding urban population and increasing vehicle ownerships [3] conflict with limited capacities of roadways, causing severe congestions especially during peak hours [4, 5]. A study conducted by Schrank et al. revealed that congestions in 498 selected US urban areas in 2011 cost people 5.5 billion hours of extra waiting time in traffic, led to 56 billion pounds of extra greenhouse emission, 2.9 billion gallons of wasted fuel, and many other massive costs [6]. The equivalent financial cost due to congestions was 121 billion US dollars in 2011, compared to 94 billion in 2000 and 24 billion in 1982 [6]. Emission by vehicles causes air pollutions and hampers public health. Traffic related air pollutions have been confirmed to be related to increased infant mortality rates [7], complicated respiratory diseases [8], children obesity [9], childhood cancers [10], brain tumors [11], among many other health issues.

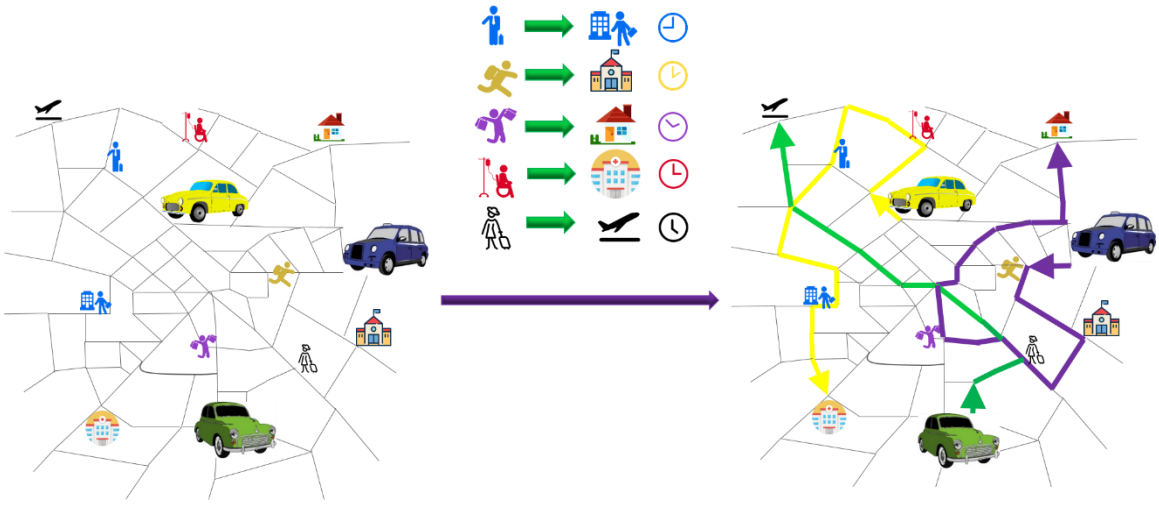


Figure 1.1. Example: a three-vehicle-five-passenger ride sharing system.

Ride sharing can potentially offer efficient solutions to shortage of fuels, congestion, air pollutions and related public health problems, limited parking and high parking charges [12, 13], etc. Besides these, taxi ride sharing is among the most promising solutions to reduce the unusual transportation pressures that big metropolitan areas are suffering, such as shortage of taxicabs during rush hours, and/or anytime in certain areas such as airports, central business districts [14, 15]. Many other studies have looked into transportation expenses the household are paying. The 2009 National Household Travel Survey (NHTS) indicated that an average American household travels about 20,000 miles per year with a total road transportation cost of about 15%-18% of their annual income [16, 17]. A mature and trustworthy ride sharing system will attract more people to do carpooling with their neighbors, colleagues and other participants to cut their commuting expenses [18].

Not only is the needs for ride sharing growing, but also the enabling technologies. Unlike a decade ago when people had to call a scheduling center to make the appointment long time ahead, smart phones nowadays and the applications can gather the spatial and time information from the drivers and ride requesters, and algorithms supporting real time scheduling can match the drivers and passengers instantaneously. Although some literatures have tried to classify ride sharing systems into single-vehicle-single-passenger type, single-vehicle-multiple-passenger type, multiple-vehicle-multiple-passenger type, this thesis will focus on the general multiple-vehicle-multiple-passenger system. A simple three-vehicle-five-passenger example is shows in Figure 1.1.

Although the needs for ride sharing is huge, the benefits are exciting, and the supporting communication technologies are in place, there is still a shortage of successful automated ride sharing operations. On one hand, as explained by Agatz [19], ride sharing involves many social factors. Passengers have different preferences when deciding who to share a vehicle with and which vehicle to pick, age, gender, profession of the co-riders can all play into it, as well as the model, color, year, entertainment facility of the car. It is complicated and hard to come up with a uniform model to accommodate all the factors. On the other hand, as we will show in next chapter, most scheduling algorithms are not able to handle large quantities of passengers and vehicles and make it hard for real time system deployment.

This study is motivated by one of the concerns taxi-ride sharing users have, the punctuality problem. In a setting where ride requesters have preferred arrival time to their destination, (i.e. getting to the airport at 4 p.m.,) it happens that a taxi serving more than one passengers fails to deliver all passengers on time. According to the feedbacks people gave in Google Play Store to the ride-hailing companies (e.g. Uber, Lyft, Didi Chuxing, Sidecar, etc.), delay issue is among the most frequently mentioned unsatisfactory experiences of their customers, as one

can see from some examples in Figure 1.2. Unpredictable waiting and delay makes it unpleasant and even frustrating to use ride sharing cab services, and in the long run will hurt the reputation of these companies if not effectively addressed.

This thesis focuses on scheduling passengers and taxis towards timely delivery in a large ride sharing system. In Chapter 2, we review a few classes of vehicle routing problems and their solution methods. In Chapter 3, we formulate timely delivery as a mixed linear integer programming problem, with the total system delay as the objective. In Chapter 4, we design a greedy agglomeration and Monte Carlo simulation based heuristic algorithm to quickly find the "sufficiently optimal" solution for a large system, followed by case studies on both small and large system, and the parallel implementation. We then discuss two problems: the selection of number of simulations, and proper calculation of delays in the greedy agglomeration algorithm. In Chapter 5, we look into another problem that is necessary for ride sharing system design, extracting multi-origin-multi-destination travel time matrix from a real road network. We modified single-source-single-target Dijkstra's and A* algorithms to serve for multiple-to-multiple purpose. Experiments are designed to compare the two. Chapter 6 concludes the thesis, with summarization of the contribution of our work, limitations and future extensions.

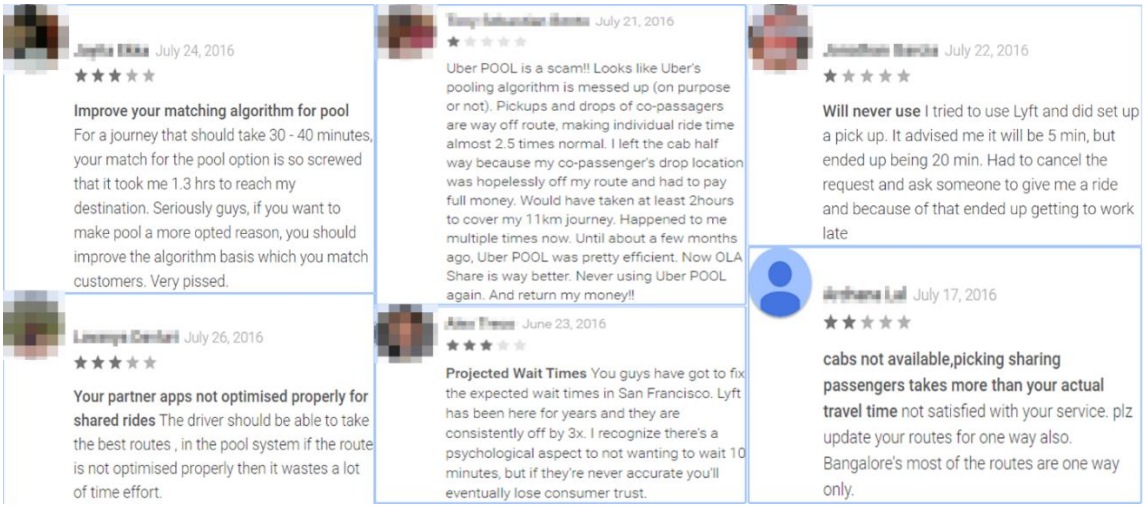


Figure 1.2. Examples of people's unpleasant experience of using taxi ride-sharing service.

CHAPTER TWO

LITERATURE REVIEW ON SOLUTION METHODS

In this chapter, we overview the history various vehicle routing problems, and famous solution methods. At the end we talk about how the ride sharing problem is related to them.

History and Variations of Vehicle Routing Problems

Capacitated Vehicle Routing Problem (CVRP)

The classical Travel Salesman Problem (TSP) is the origin and the simplest form of vehicles routing problems: Given the locations of multiple cities, a salesman needs to cover each of them exactly once, with the shortest travel distance. Multiple Travel Salesman Problem (mTSP) is similar, but it allows more than one salesman to finish the task. Bektas offered a thorough overview of mTSP problem and its exact and heuristics solution methods [20, 21]. In 1959, Dantzig and Ram generalized TSP problem and applied it to Truck Dispatching Problem: one or more trucks are sent out to pick up goods from every station, which has a certain quantity of goods, and the trucks have limited capacities [22]. The goal is to find the best matching and route so the total service distance is minimized. Since then Truck Dispatching Problem has spurred decades of other studies of more complicated and practical configurations and formulation to support real life applications. A more general name "Vehicle Routing Problem (VRP)" has been used. Capacitated Vehicle Routing Problem (CVRP) is among the most frequently studied problems of this class. Clarke and Wright created "Savings" methods in 1964 [23]. It starts from generating short routes. Saving is defined as the decrease of travel distance when merging two shorter routes. Savings methods keeps merging the route pair, merging which will cause the largest saving, till no merging is feasible (all vehicles are filled up). Miller created the "Sweep" method, in which the customers are paired with vehicles based on their locations in a polar coordinate system, whose center is the vehicle's origin depot [24]. Instead of using polar shape, Foster and Ryan advanced Miller's method to petal like space, and named their method Petal Method [25, 26], which was reported to perform more accurate and faster than Sweep [27]. Christofides and Eilon designed 3-optimal method, which was claimed to perform much faster than Savings [28]. Besides Savings and Sweep, another class of heuristic algorithm for CVRP is two phase method: cluster first to partition the space and then find optimal local routing. The most famous two phase method is Fisher and Jaikumar's Generalized Assignment Algorithm [29], where the space is divided into cones and the nearest customer

inside each cone to the vehicles is chosen as a seed to initialize a route. Every passengers choose the most convenient route to insert, which causes the minimum distance increase and the vehicle is not filled up. Another well-known two phase algorithm is the cyclic transfer algorithm [30].

Vehicle Routing Problem with Pickup and Delivery (VRPPD)

CVRP applies to problems like logistics distributing, delivering goods to stores/customers' houses, etc. For taxi scheduling, ride sharing systems, this is not a proper model because a passenger has both a pick up location and delivery location, while in CVRP, every customer has only one service location. Therefore, there has been another type of vehicle routing model – Vehicle Routing Problem with Pickup and Delivery (VRPPD). Since pickup location and delivery location are not related spatially, they are not necessarily next to each other and can be far away, VRPPD has higher complexity than CVRP. The single location based methods overviewed above cannot be applied to VRPPD directly. Katoha and Yano studied the one-vehicle-multiple-passenger tree shaped network routing problem with pick and delivery demands [31]. Although their two-approximation method seems to work well on tree shaped network, it unfortunately does not apply general graph/network, which is what the real transportation network is. Tzoreff et al studied the same problem on other special shaped networks such as cycles, warehouse shapes, etc [32]. Gribkovskaia et al studied another restricted configuration where all delivery loads come from the vehicle depot and all loads picked up will be sent back to the same depot [33]. As the author pointed out in the original paper, this assumption does not describe many real applications and definitely does not fit the ride sharing case. Gribkovskaia et al developed a general solution mixed linear integer programming model for single-vehicle-multiple-customer VRPPD and used the Tabu search heuristics to find the approximated solution [34]. Nagy and Salhi formulated the most general multi-vehicle-multi-customer VRPPD model, where pickup and delivery locations, capacity constraints, pickup and delivery orders are included [35]. They also offered a thorough overview and classification of previous models on VRPPD.

Vehicle Routing Problem with Time Window (VRPTW)

CVRP and VRPPD are only focused on geographical locations, however, in practical situations, customers might request service to happen only within a certain time window, or can't not be later than some time point. For example, an customer can require a piece of furniture to be deliver between 5 p.m. to 7 p.m. To accommodate time window factor, there is a new type of routing problem called Vehicle Routing Problem with Time Window (VRPTW). Solomon studied VRPTW and came up with a two phase algorithm: First do a nearest neighbor search to

attach a customer to its nearest vehicle (although because of the constraint of capacity, a customer might not always gets assigned to the nearest vehicle), then do the one-vehicle-multiple-vehicle routing inside each cluster [36]. Cordeau etal formulated VRPTW as a network flow problem, and solved it using different optimization approaches, including branch and cutting, column generation and Lagrangian relaxation [37]. Braysy and Gendreau overviewed the approximated solution methods for VRPTW, including route construction methods (similar to Soloman's two phase method), solution improvement method (slightly and iteratively tune a given route), Tabu search, genetic algorithm, simulated annealing etc. [38, 39]. Braysy and Gendreau also benchmarked all the algorithms using Solomon's 56 test cases [40].

Vehicle Routing Problem with Pickup and Delivery with Time Window (VRPPDTW)

VRPPDTW is the pickup and delivery location enabled version of VRPTW. It is among the most complicated variations of VRPs. Four types of constraints are supported: capacities, time windows, pickup and delivery locations, and order (pick up happens before delivery). Because of the added complexity, the modelling and solution methods become more advanced. The most frequently cited literature on VRPPDTW is cordeau's mixed linear integer programming formulation of VRPPDTW and his branch and cut solution to it [41]. Spoke and Cordeau later came up with an enhanced branch-and- cut-and-pricing solution to further improve the solution [42]. The formulation of VRPPDTW is a three-index model and even increasing the number of vehicles and passenger just slightly could cause a dramatic increase in the dimension of solution space and so the computational time. Other researchers have tried other solution methods, such as the state-space-time scheme introduced by Yang [43] and Mahmoudi [44]. However, all solution methods for VRPPDTW so far is still computationally challenged. According to the most recent result reported in [44], to compute a 50-passenger-15-vehicle case, it takes almost two hours.

Timely Ridesharing Problem

Our timely ridesharing problem belongs to the last class – VRPPDTW. However, there is an essential difference in the configuration. Notice that all the problems above involving time windows have assumed that we can always find solution to meets all the time windows of the customers. All the optimization models have constraints responding to the "no delay is allowed" assumption. In a large taxi ride sharing system, this is not always the case as we see from Chapter one, that the conflictions of different passengers' preferred delivery time causes the taxis late. This study models a more practical situation that we allow conflictions of

passengers' time windows and situations where no feasible schedules to satisfy everybody is allowed. Our objective is that no matter how much confliction in people's requested time window, we are to minimize the total system service delay.

CHAPTER THREE

A MIXED-INTEGER LINEAR PROGRAMMING FORMULATION

Model Formulation

We describe the configuration of the centralized ride sharing system. In some region, at one moment, there are k in-service vehicles distributed in different locations, n ride requests awaiting service. Every request has a pickup location, destination, and a preferred latest arrival time. There are more riders than drivers and they are all willing to share a vehicle with others. The ideal situation is that every passenger can be delivered on time, but because one vehicle might have to serve more than one passenger, and there always exist conflicts between passengers' requested arrival time, the possibility of satisfying everybody is very slim. The objective here is to minimize the total delay of delivering every passenger compared to their preferred arrival time.

There is a classical but still actively pursued optimization model called Vehicle Routing Problem with Time Window (VRPTW) that is considered as the generalized prototype of many multi-vehicle multi-request pickup and delivery scheduling problems. Cordeau first formulated VRPTW in 2003 [41] and the model has been adpoted by many scholars when studying related problems. The model has been applied to applications such as truck delivery, patient transformation in hospital networks [45, 46], vehicle customer matching in taxi-sharing services [47, 48], facility location selection such as electric vehicle charge stations, theater, military supply bases, etc. [49, 50]. A few works have offered multi-facet overviews of previous studies on this class of problem [1, 51, 52].

In this study, our model development is also based on Cordeau's formulation. The differences from previous models and studies are: (1) The objective of most previous studies is about the "travel cost" of the vehicles and companies owning these vehicles, either the total travel distances, which represents how much fuel they consume, or adding more terms such as total travel time, which is related to labor cost. To the best of our knowledge, there has been no studies modeling the service delay issues. (2) The reason that service delay has not been looked into is that most studies focus on solvable cases where vehicles can always find a solution to reach customers' destinations on time without violating the requested time windows. This might be true in some situations where customer requests are not very intensive and schedules all have margins, but definitely not for the real-time ride sharing service, where requests pop up constantly and violations of preferred timeline is inevitable. In contrast to previous models, we focus on more practical situations in ride-sharing system where there is no way for the drivers to

completely satisfy everybody's request, and in this case we directly model and minimize the delivery delays.

Terminology

For an m - vehicle - n - passenger configuration, we define the vehicle set $V = \{1, 2, ...m\}$, passenger pickup location set $P = \{1, 2, ...n\}$, delivery location set $D = \{n+1, n+2, ...2n\}$, and set $N = P \cup D \cup \{0, 2n+1\}$, where 0 represents the origin depot and $2n+1$ the destination depot of a vehicle. q_i is the load associated with each location, with picking up being a positive value and dropping off negative, e.g., if two passengers need to be picked up at location 5, then $q_5 = 2$, and if one passenger will be dropped off at location 3, then $q_3 = -1$. Each passenger has its preferred latest arrival time to their destinations, denoted by l_i. Table 3.1 shows all the symbols that will be used in the objective and constraints, their data types and definitions.

Table 3.1. Symbols and their definitions.

Symbol	Type	Source	Definition
m	Integer	Given	Number of vehicles
n	Integer	Given	Number of Passengers
q_i	Integer	Given	Loads of passenger at each stop. Positive for picking up, negative for delivery.
l_i	Double	Given	Requested arrival time for each passenger.
x_{ij}^k	Binary	Variable	$x_{ij}^k = 1$ if vehicle k travels from location i to j, $x_{ij}^k = 0$ otherwise.
B_i^k	Double	Variable	The amount of minutes it takes vehicle k to arrive at location i
Q_i^k	Integer	Variable	Load of vehicle k after it leaves location i.
d_i^k	Double	Variable	Service time of vehicle k at location i
t_{ij}^k	Double	Variable	Travel time of vehicle k from location i to location j
C^k	Integer	Variable	Capacity of vehicle k
y_i^k	Double	Variable	Intermediate variables used to linearize the nonlinear term in the objective function.
z_i	Double	Variable	Intermediate variable used to linearize the nonlinear term in the objective.

Objective Function and Constraints

We first list all the objective and constraints before we explain them one by one. The objective is to minimize the delivery time delays for all serviced passengers:

$$Min \ \sum_{i \in P} max\left(0, \left(\sum_{k \in V}\left(B_{i+n}^{k} \sum_{j \in N} x_{i+n,j}^{k}\right)\right) - l_i\right) \quad \text{(o1)}$$

$$x_{ii}^{k} = 0, \ \forall i \in N \quad \text{(c1)}$$

$$x_{i+n,i}^{k} = 0, \ \forall k \in V, \forall i \in P \quad \text{(c2)}$$

$$x_{0,i+n}^{k} = 0, \ \forall k \in V, \forall i \in P \quad \text{(c3)}$$

$$\sum_{k \in V} \sum_{j \in N} x_{ij}^{k} = 1, \ \forall i \in P \quad \text{(c4)}$$

$$\sum_{j \in N} x_{ij}^{k} - \sum_{j \in N} x_{i+n,j}^{k} = 0, \ \forall k \in V, \forall i \in P \quad \text{(c5)}$$

$$\sum_{j \in N} x_{0j}^{k} = 1, \ \forall k \in V \quad \text{(c6)}$$

$$\sum_{i \in P} x_{i+n,2n+1}^{k} = 1, \ \forall k \in V \quad \text{(c7)}$$

$$\sum_{j \in N} x_{ji}^{k} - \sum_{j \in N} x_{ij}^{k} = 0, \ \forall k \in V, \forall i \in P \cup D \quad \text{(c8)}$$

$$B_j^k \geq (B_i^k + d_i^k + t_{ij}^k) x_{ij}^k, \ \forall k \in V, \forall i \in N, \forall j \in N \quad \text{(c9)}$$

$$B_{i+n}^k \geq B_i^k, \ \forall k \in V, \forall i \in P \quad \text{(c10)}$$

$$Q_i^k \geq 0, \ \forall k \in V, \forall i \in N \quad \text{(c11)}$$

$$Q_i^k \geq q_i, \ \forall k \in V, \forall i \in N \quad \text{(c12)}$$

$$Q_i^k \leq C^k, \ \forall k \in V, \forall i \in N \quad \text{(c13)}$$

$$Q_i^k \leq C^k + q_i, \ \forall k \in V, \forall i \in N \quad \text{(c14)}$$

$$Q_j^k \geq (Q_i^k + q_i) x_{ij}^k, \ \forall k \in V, \forall i \in N, \forall j \in N \quad \text{(c15)}$$

For the constraints, (c1)-(c3) set constraints on x_{ij}^k based on the service order requirements. A vehicle cannot travel back to itself, a vehicle cannot travels from a passenger's destination to the origin, and a vehicle cannot travel from the origin depot directly to a passenger's destination.

(c4) describes that exactly one vehicle picks up a passenger. (c5) together with (c4) describes the same vehicles picks up and delivers the passenger. (c6) enforces that a vehicles always starts from its origin depot. (c7) enforces a vehicles goes back to its destination depot after delivering the last passenger. (c8) is the

flow conservation at any pickup and delivery location. (c9) captures the arrival time relationship between two locations, if a vehicle travels from one location to another, then the arrival time at one is later than the other. (c10) states that a vehicle always arrives at a passenger's destination later than the pickup location, which enforces the service order that a vehicle always picks up a passenger before getting to his/her destination. This is a constraint that Cordeau's formulation did not use. Cordeau used some advanced order constraint techniques to enforce the orders. We found that it is much easier to just add a constraint on B_i^k and B_{i+n}^k. (c11) to (c14) restricts the lower and upper bounds of the load of a vehicle. At a pickup location, $q_i \le Q_i^k \le C^k$, while at a drop-off location, $0 \le Q_i^k \le C^k + q_i$. (c15) captures the load relationship for the vehicle at two locations, using the same logic as in (c9).

Let's explain the objective function now. It is formulated by adding up the delivery delay of every passenger. We only count if the arrival time is later than the requested, so the outside $\max\left(0, \left(\sum_{k\in V}\left(B_{i+n}^k \sum_{j\in N} x_{i+n,j}^k\right)\right) - l_i\right)$ filters out those that are on time. $\sum_{k\in K}\left(B_{i+n}^k \sum_{j\in N} x_{i+n,j}^k\right)$ is the actual arrival time of passenger i. From (c5) we know that $\sum_{j\in N} x_{i+n,j}^k = \sum_{j\in N} x_{i,j}^k$, and we know from (c4) that for any $i \in P$, there is only one pair of j and k to make $\sum_{k\in K}\sum_{j\in N} x_{ij}^k = 1$, which is because there is only one vehicle to pick up a passenger at his/her origin and departs to only one direction, so the dual summation will remain as B_{i+n}^k where vehicle k is the one that has picked up passenger i.

Linearization

The model is not completely linear yet. First, (c9) and (c15) are both nonlinear constraints. We use the big-M method to convert it to linear constraints. Here M and W are sufficiently large numbers.

$$B_j^k \ge (B_i^k + d_i^k + t_{ij}^k) - M(1 - x_{ij}^k), \forall k \in V, \forall i \in N, \forall j \in N \quad \text{(c16)}$$

$$B_j^k \ge 0 \quad , \ \forall k \in V, \forall j \in N \quad \text{(c17)}$$

$$Q_j^k \ge (Q_i^k + q_i) - W(1 - x_{ij}^k), \ \forall k \in V, \forall i \in N, \forall j \in N \quad \text{(c17)}$$

Second, the objective function is also nonlinear because of the product term and the *max()* operator. For the product term, we define new variables and apply big-M notation to linearize it. For the maximum operator, (c23) and (c24) define new a variable z_i and let it be bigger than both arguments. Here M' is a sufficiently large number.

$$y_i^k = B_{i+n}^k \sum_{j \in N} x_{j,i+n}^k, \ \forall k \in V, \forall i \in P \quad \text{(c18)}$$

$$y_i^k \le M' \sum_{j \in N} x_{j,i+n}^k, \ \forall k \in V, \forall i \in P \quad \text{(c19)}$$

$$y_i^k \le B_{i+n}^k, \ \forall k \in V, \forall i \in P \quad \text{(c20)}$$

$$y_i^k \ge B_{i+n}^k - M'(1 - \sum_{j \in N} x_{j,i+n}^k), \ \forall k \in V, \forall i \in P \quad \text{(c21)}$$

$$y_i^k \ge 0, \ \forall k \in V, \forall i \in P \quad \text{(c22)}$$

$$z_i \ge \sum_{k \in K} y_i^k - l_i, \ \forall i \in P \quad \text{(c23)}$$

$$z_i \ge 0, \ \forall i \in P \quad \text{(c24)}$$

From the discussion earlier, we know that $\sum_{j \in N} x_{j,i+n}^k$ is a 0-1 binary variable, so we can easily verify that (c19)-(c22) is equivalent to (c18).

Hence the objective becomes:

$$Min \ \sum_{i \in P} z_i \quad \text{(o2)}$$

with the constraints (c1)-(c8), (c10)-(c14), (c16)-(c24).

The model can be solved as a mixed integer programming (MIP) problem using the optimization solvers such as Gurobi [53], CPLEX [54], etc.

A Case Study

As an example, we study a 3-vehicle-6-passenger scenario. We randomly generate vehicles' depots and passengers' pickup and drop-off locations. Minimum travel time between a passenger's origin and destination is calculated, and the passenger's preferred arrival time is also randomly assigned, and the value is reasonably larger than the minimum travel time. We here make the capacity of each vehicle 3. Figure 3.1 visualizes the setup and optimal service plan for this system. (Notation on the paths of Figure 3.1(a) is "minimum travel time/passenger requested arrival time". Notation on the paths of Figure 3.1(b) is "passenger requested arrival time/actual arrival time/delay".)

Figure 3.1(a) visualizes the vehicles' origin depots and the six passengers' pickup and drop off locations. The notation on each path is the minimum travel time required followed by the passenger's preferred longest delivery time. For example "12m/18m" represents that it takes a taxi at least 12 minutes from the origin to the destination, and the passenger allows 6 more minutes.

Figure 3.1(b) presents the optimal service plan. The green vehicle serves only one passenger. It takes 27 minutes to finish this delivery with a 9-minute delay. The red vehicle picks up two passengers in sequence and drops them off in the same order as they are picked up. The delays for these two passengers are 4 min and 15 min, respectively. The blue vehicle takes care of the rest of the passengers, and it picks up all three passengers before delivering them one by one.

We observe that, because the objective is to minimize the overall delays, the vehicle does not give priority to any individual vehicle. Although a vehicle is close to one particular passenger, it could instead be dispatched to serve other passengers who need service equally but do not have a vehicle close by. The pickup and drop-off orders are all optimized for the global minimum. A passenger picked up earlier does not necessarily gets dropped off first, e.g. the blue vehicle first picks up the blue passenger but delivers the green one first.

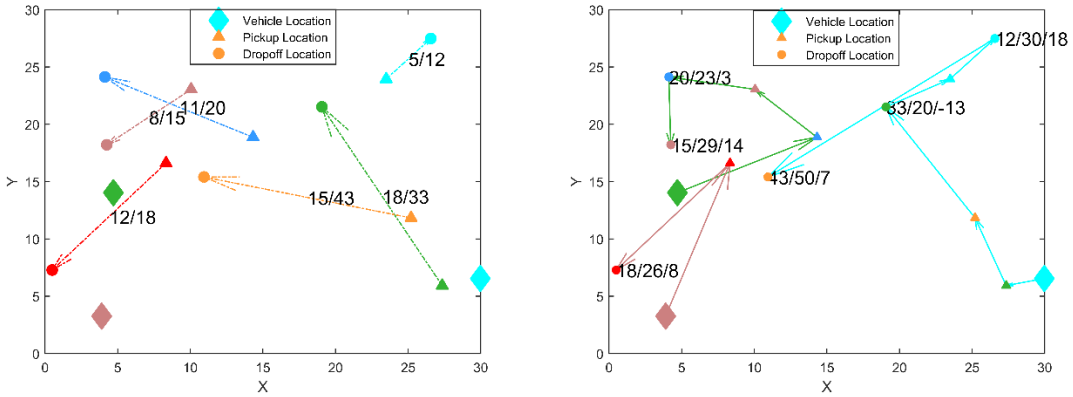


Figure 3.1. A 3-vehicle-6-passenger case study.

CHAPTER FOUR

GREEDY AGGLOMERATIVE CLUSTERING AND MONTE CARLO SIMULATION BASED METHOD

Although the mixed linear integer model is able to obtain the exact optimal solution for us, the case study in Chapter 3 tells us that solving the optimization model takes prohibitively long time to make it practical for real time deployment of medium or large scale ride sharing system.

In this chapter, we present an efficient algorithm to find the sufficiently optimal solution based on greedy agglomeration and Monte Carlo simulation. We will first present the algorithm, complexity analysis, case studies on systems of different scales, compare its performance to optimization approach, and then discuss an important detail - the selection of number of simulations.

Algorithm Development

The logic is quite straightforward. It is an agglomerative process, where in each iteration one passenger is assigned a vehicle. When choosing the "best" vehicle to accommodate this passenger, we use a greedy strategy - the combination of a passenger and a vehicles will cause the minimum increase of system-wide time delay.

The "best" has double meaning here.

On one hand, no matter which vehicle the passenger is eventually attached to, he/she will be placed at the optimal service order in this vehicle, which in other words, is that inserting this new passenger will cause the minimum time delay increase to this vehicle. Let's see an example, before adding passenger p_j to vehicle v_k, v_k has already been assigned m passengers $\{p_1, p_2, \ldots p_m\}$ with the most efficient (causes the minimum possible delay) service order: $R = \{+p_2, +p_1, -p_1, +p_3, -p_2, \ldots\}$. (Note: here we use $+p_2$ to represent picking up p_2 at his/her origin and $-p_2$ as deliver to his/her destination.) Adding p_j to v_k is a process of inserting $+p_j$ and $-p_j$ to existing route R, with the constraint that $+p_j$ must come earlier than $-p_j$ because delivering happens physically after picking up. The goal is to find out which order will carry the least delay increase. Because of the delay calculations, the only way to find the best is to re-enumerate all the

possibilities. This process can be implemented as a dual loop, where the outside loop is on the possible insertion spot s of $-p_j$, where $s = 0,1,2,...m+1$, and the inside loop is on the insertion spot t of $+p_j$, with $t = 0,1,2,...s$. The complexity of this step is $O(q^2)$, where q is the capacity of vehicle v_k. Procedure **Best_Insertion** in Figure 4.3. details this in-vehicle ordering process.

On the other hand, assigning a passenger to different vehicles will bring different delay increases, so we assign him/her to the vehicle that causes the minimum delay increase.

Since every assignment of an individual passenger brings a minimum possible delay increase, the total delay should also be the minimum. However, just like any other greedy methods that gets easily trapped in a local optimum [55], the greedy agglomeration step above is also a "short sighted" procedure that is insufficient to achieve global optimum. Although every passenger attaches to the best available vehicle and follows the best service order to achieve minimum system delay increase, the result is not necessarily a global minimum. This is because a later assigned passenger does not have as many vehicles to choose from as an earlier assigned passenger. A later assigned passenger might be assigned to a vehicle that carries the "best" but big delay, just because by the time he/she gets to choose, that is the only vehicle available. In other words, if an earlier assigned passenger does not attach to the best vehicle at that moment, but instead chooses a suboptimal one, and leaves the "best " vehicle for a later assigned passenger who will find it more convenient, the overall delay might be smaller than the other way round.

Our strategy is to use Monte Carlo simulation to address the "short sight" issue. Since the order of the passenger list matters, we can shuffle the passenger list and repeat the greedy agglomeration steps using the shuffled passenger list. Shuffling the passenger list is essentially to get a new random permutation. The randomness of the passenger order and sufficient number of simulations aim to give every passenger an equal chance to be combined with the best vehicle that can achieve minimum system delay.

Up to now, we have introduced the development idea behind our greedy agglomeration and Monte Carlo simulation algorithm for ride sharing scheduling towards the goal of on-time service. Figures 4.1. – 4.3. present the pseudo codes that details the system implementation.

```
Procedure Monte_Carlo: Shuffle passenger list and repeat greedy
agglomerative clustering steps
Input: M – The number of simulations
Output: min_delay – minimum delay achieved by the best service plan

min_delay = Monte_Carlo (M){
        min_delay = Inf
        for i = 1:M
                pList = shuffle(passenger_list)
                delay = Match(pList, vList)
                if delay < min_delay
                        min_delay = delay
                end if
        end for
}
```

Figure 4.1. Monte Carlo simulation and passenger list shuffling

```
Procedure Match: Greedy agglomerative clustering step to find the best
matching and service order to achieve minimum system-wide service delay
Inputs:  Passenger list pList, vehicle list vList
Output:  service delay

delay = Match(pList, vList){
        delay = 0;
        for each unassigned passenger p in pList
                min_delay_inc = Inf;
                for each vehicle v in vList
                        if v is not full
                                inc_delay = Best_Insertion(R, p)
                                if inc_delay < min_delay_inc
                                        min_delay_inc = inc_delay
                                        vehicle = v
                                end if
                        end if
                end for
                Assign p to vehicle
                Mark p as assigned
                delay = delay + min_delay_inc
        end for
}
```

Figure 4.2. Greedy Agglomeration step to find the best matching and service order to achieve minimum system-wide service delay

Procedure Best_Insertion: Find the best service order and delay increase after adding a new passenger to a vehicle.
Inputs: *R* – The best service route before inserting new passenger *p*.
Output: *inc_delay* – smallest possible delay increase of inserting *p*.
R^+ – best service route after *p* is added.

```
[inc_delay, R+] = Best_Insertion(R, p)
inc_delay = 0;
for i = 0:length(R)
    for j = 0:i
        R' = [R(1:j)  +p  R(j+1:i)  –p  R(i+1:end)]
        delta_delay = get_delay(R') – get_delay(R);
        if delta_delay < inc_delay
            inc_delay = delta_delay
            R+ = R'
        end if
    end for
end for
```

Figure 4.3. Find the best service order and delay increase after adding a new passenger to a vehicle.

Complexity Analysis

The **Monte_Carlo** procedure has M iterations. Inside **Monte_Carlo** is **Match**, which has at worst $p \times n$ iterations, where p is the number of passengers to be served, and n is the number of vehicles in service. Inside **Match** is **Best_Insertion** procedure, which has $O(q^2)$ complexity, where q is the capacity of a vehicle. So the overall time complexity of the algorithm is $O(\mathrm{M}\, pnq^2)$. Notice that q is usually a small constant number because a cab normally accommodates 3 to 5 people. Therefore, the time complexity for our greedy agglomerative clustering and Monte Carlo simulation based algorithm is $O(\mathrm{M}\, pn)$, in other words, it is decided by the number of simulations, number of passengers and number of vehicles. The latter two is given by the scale of the ride sharing system, but we can control the number of Monte Carlo simulations we use. More simulations might be helpful to obtain a finer solution but increases the computational cost. We will look into this factors in later sessions. But before that, let's first compare our heuristic algorithm to the pure optimization solution.

Case Studies

In this session, we conduct experiments to test the effectiveness and efficiency of the algorithm. We use cases of different scales. We first show that for small ride sharing systems, where there is a small number of vehicles and passengers, our algorithm can achieve the same optimal solutions as what the optimization models obtains, at a much lower computational cost. We also present the algorithm's capability of handling large ride sharing systems.

Let's first compare the two methods' performances on small systems where there is only a few vehicles and passengers. These cases are: 4-passenger-2-vehicle (p4v2), 5-passenger-3-vehicle (p5v3), 6-passenger-3-vehicle (p6v3), 6-passenger-4-vehicle (p6v4), 7-passenger-3-vehicle (p7v3), and 7-passenger-4-vehicle (p7v4). Appendix I gives the details of the data and results, including the origin and destination locations of the passengers, their requested arrival times, the origin depots of the taxis, the optimal service routes of each vehicle, and the associated system delays. In all six cases, our greedy agglomeration based algorithm can always hit the optimal solution within 100 simulations. In Figure 4.4, we can see that the achieved minimum system delay decreases as we put more simulations. Although we have used 200 simulations, all six systems have already achieved the optimal solution within the first 100 runs. Table 4.1 shows the runtime comparison of solving optimization model and our heuristic algorithm. For optimization model approach, as the number of passengers and cabs increases, even just by one, the runtime rises exponentially. This is not hard to explain, because as we can see from the formulation of the model, when the number of vehicles or passengers increases slightly, the number of variables, constraints will increase enormously, which adds much more dimensions to its solution search space. Instead, our methods is much steadier, and it runs over one thousand times faster than the optimization model for the p7v4 case. We can predict that for larger systems, this advantage will be immensely magnified.

Figure 4.5 (a) to (f) is the visualizations of the setups and the optimal scheduling of these six cases. Figure 4.6 presents the histograms of the system delays in the 200 Monte Carlo simulations.

We have also tested our algorithm on four larger ride sharing systems: 50-passenger-20-vehicle (p50v20), 100-passenger-50-vehicle (p100v50), 200-pasenger-100-vehicle (p200v100), and 300-passenger-150-vehicle (p300v150). The algorithm can handle much larger systems with the help of powerful parallel computing, which is the topic for next section. The setups and optimal scheduling are presented in Figure 4.6 – 4.10.

Table 4.1. Runtime Comparison of solving optimization model and greedy agglomeration algorithm on small cases.

Case	Runtime (s)	
	Optimization Model	Heuristics (200 runs)
p4v2	1.93	1.14
p5v3	15.19	1.63
p6v3	490.75	2.05
p6v4	100.24	2.67
p7v3	2910.36	3.06
p7v4	3779.94	3.30

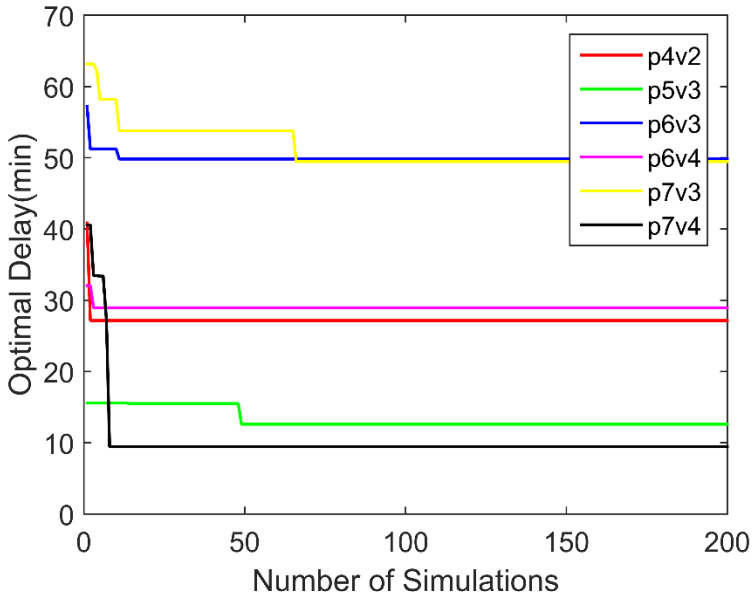
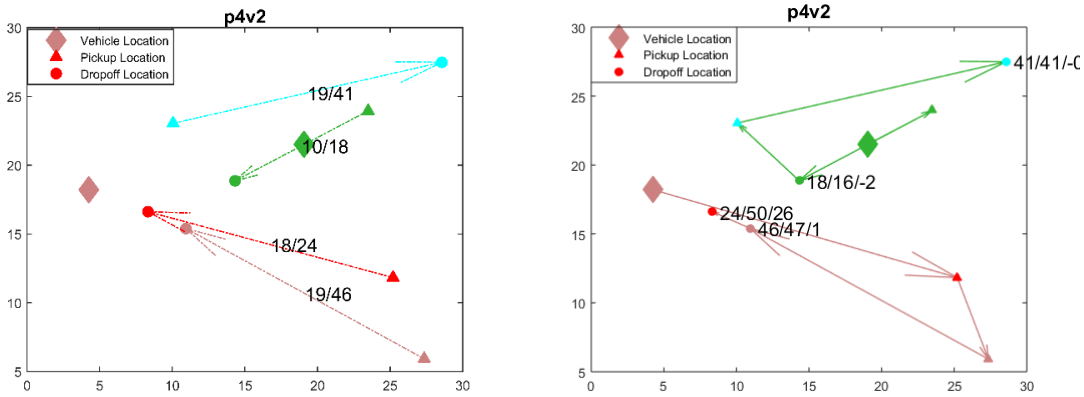
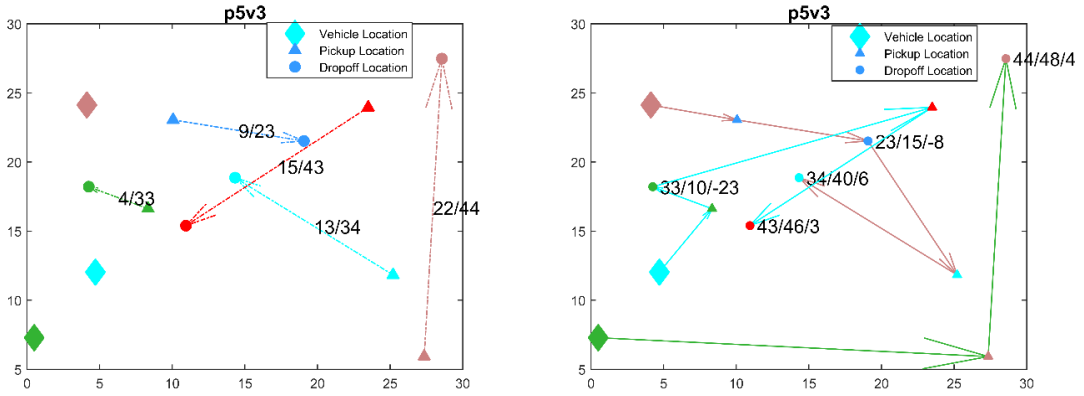


Figure 4.4. Relationship between achieved minimum system delay and the number of simulations.

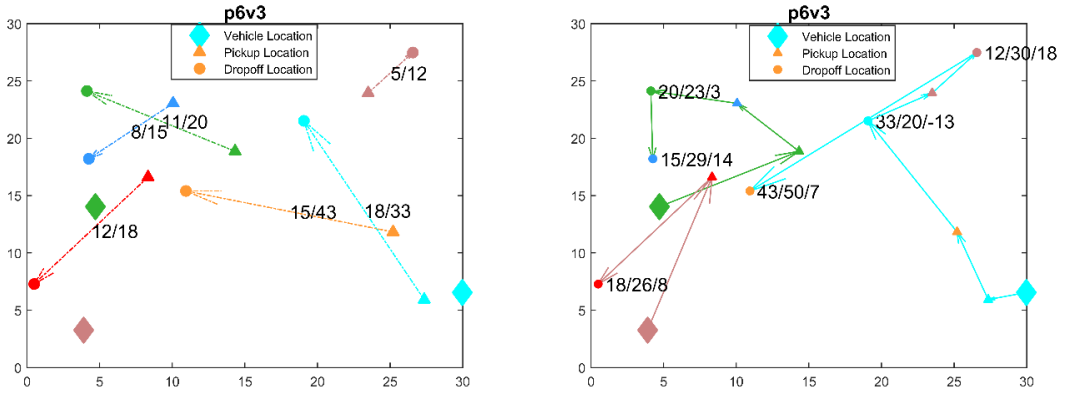
Figure 4.5. Setups and optimal scheduling of the 6 test cases. (Notation: left – setup (travel time/requested arrival time), right - optimal scheduling (requested arrival time/actual arrival time/delay))



(a) p4v2

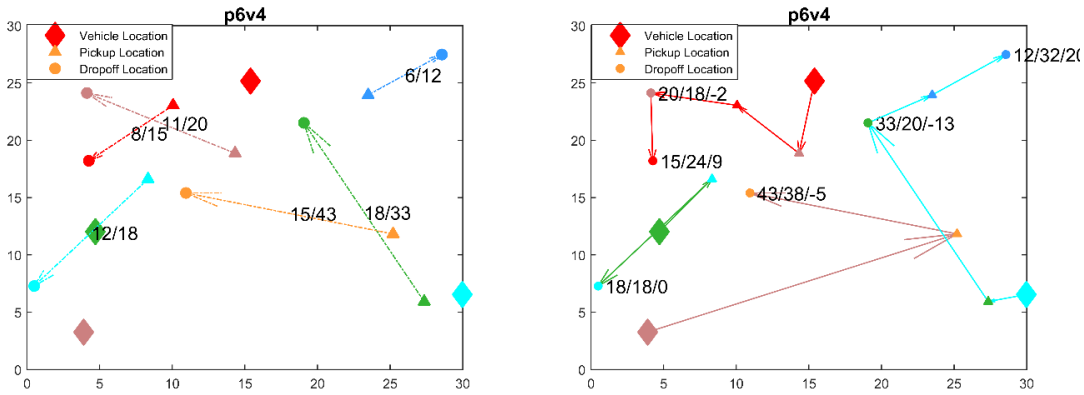


(b) p5v3

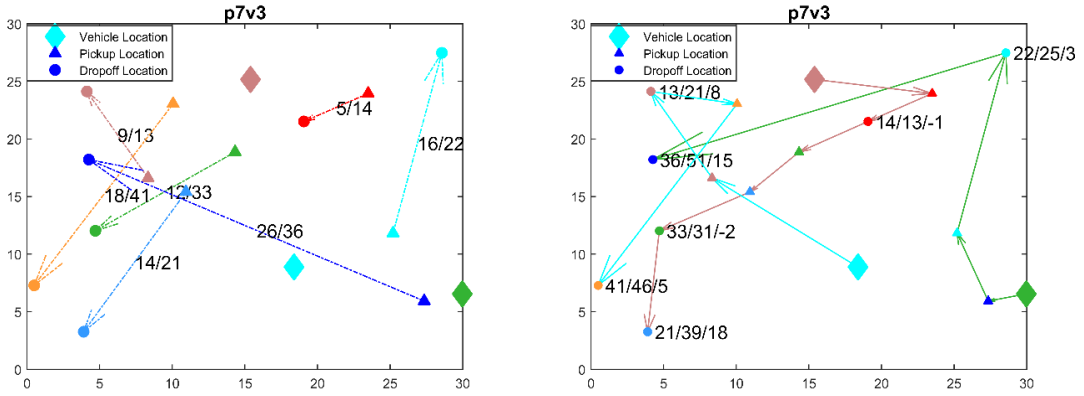


(c) p6v3

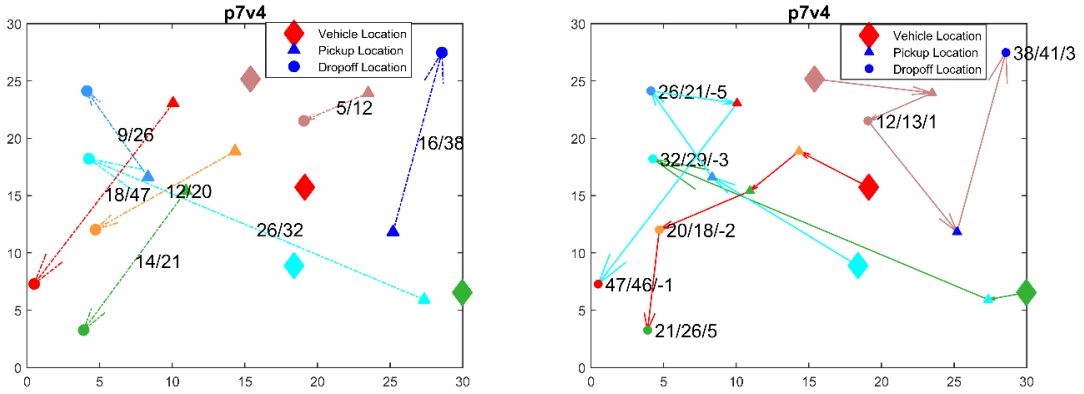
Figure 4.5. Continued



(d) p6v4



(e) p7v3



(f) p7v4

Figure 4.5. Continued

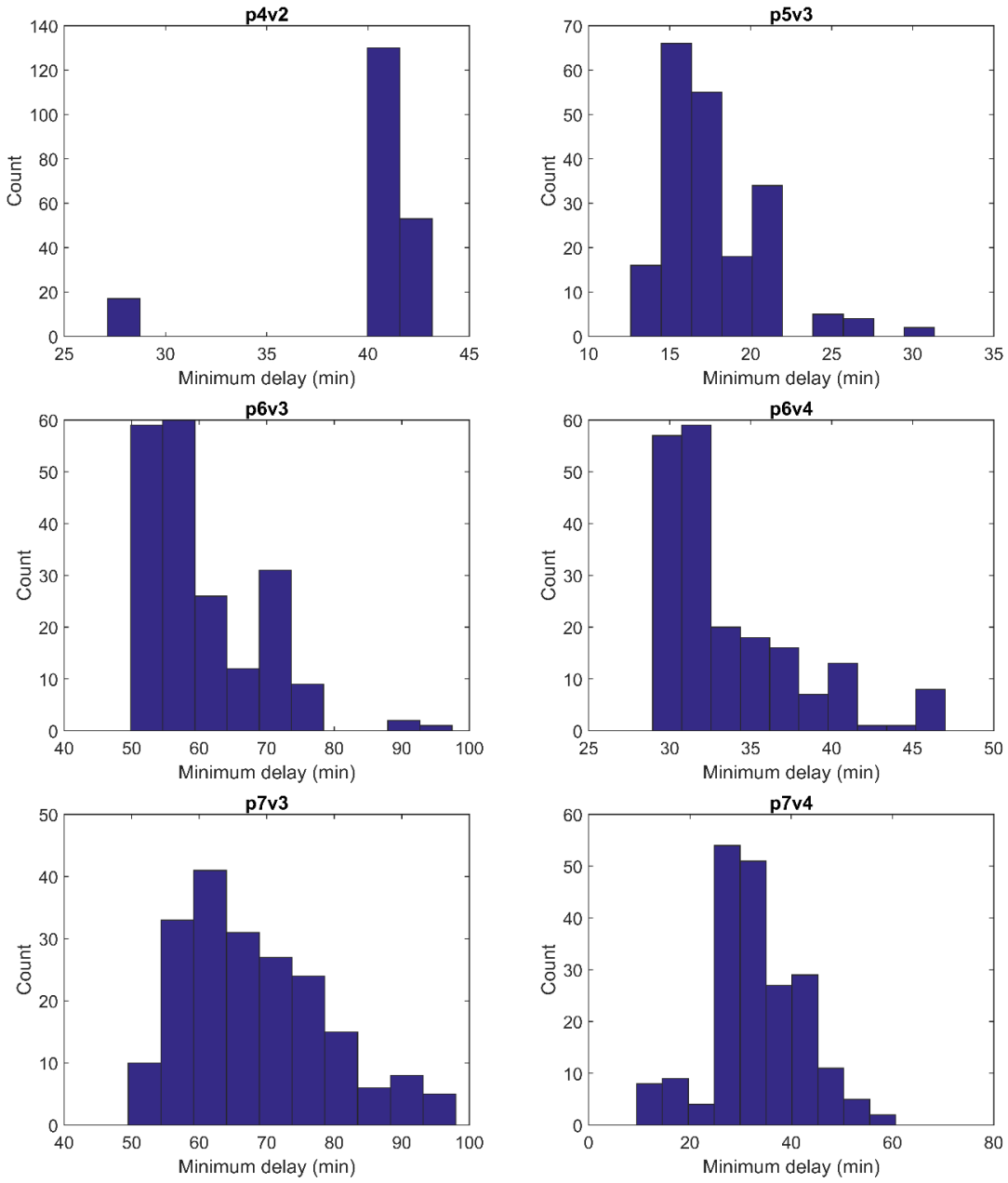
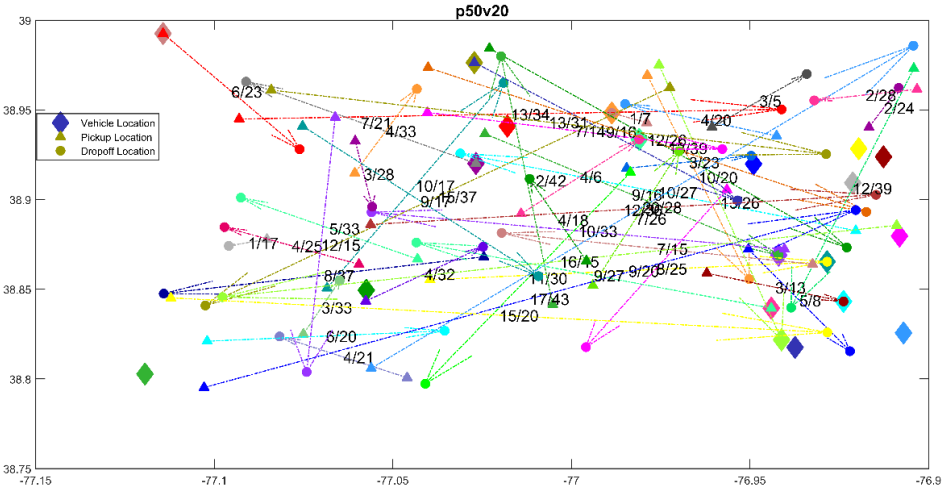
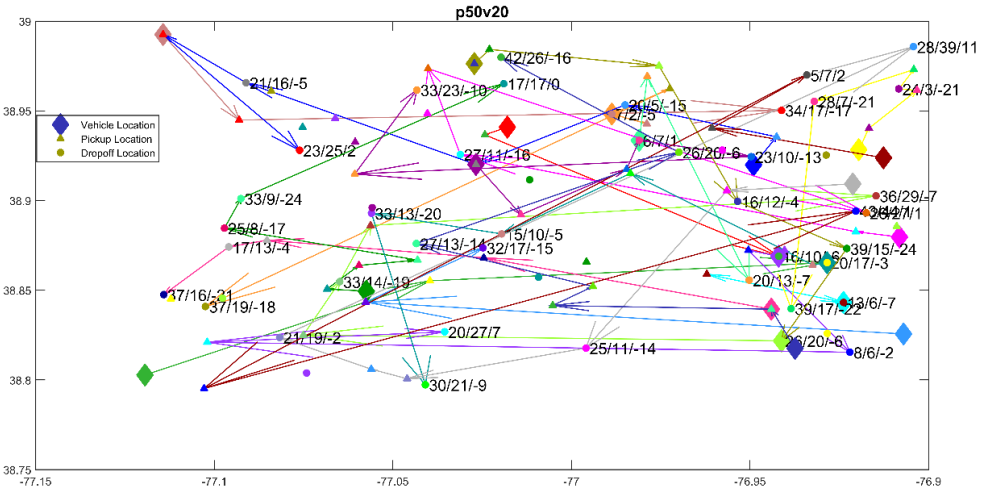


Figure 4.6. Histograms of the system delays in the 200 simulations.

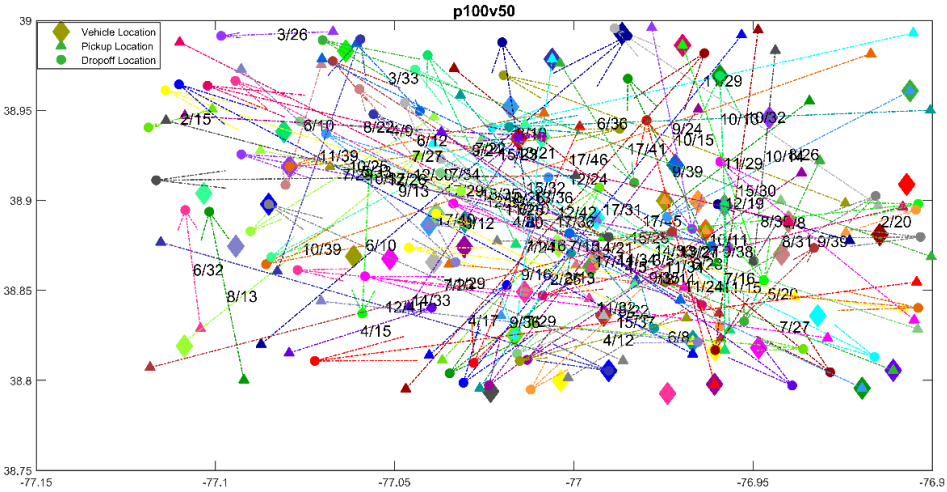


(a) Setup

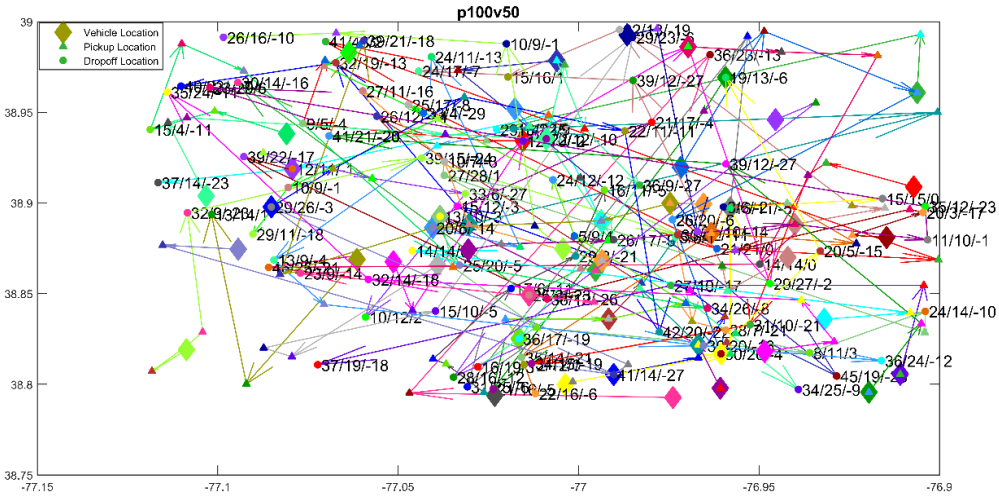


(b) Optimal Scheduling

Figure 4.7. A 50-passenger-20-vehicle case.

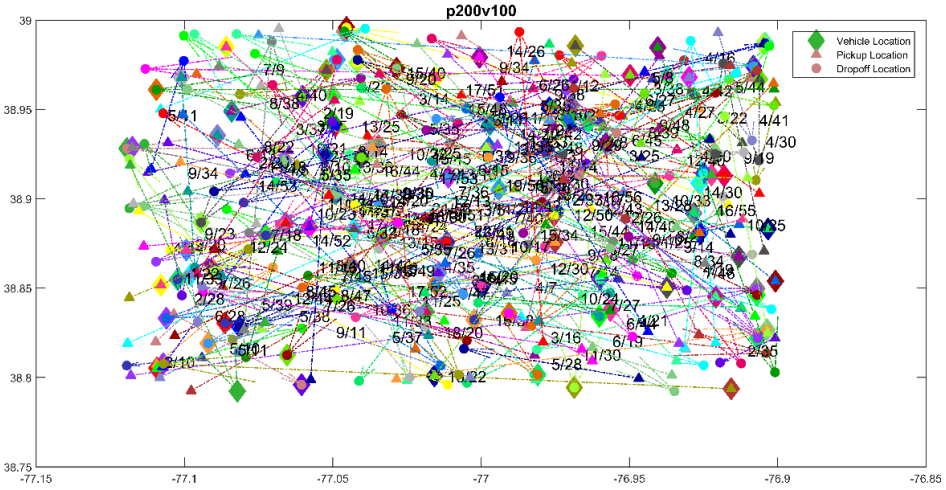


(a) Setup

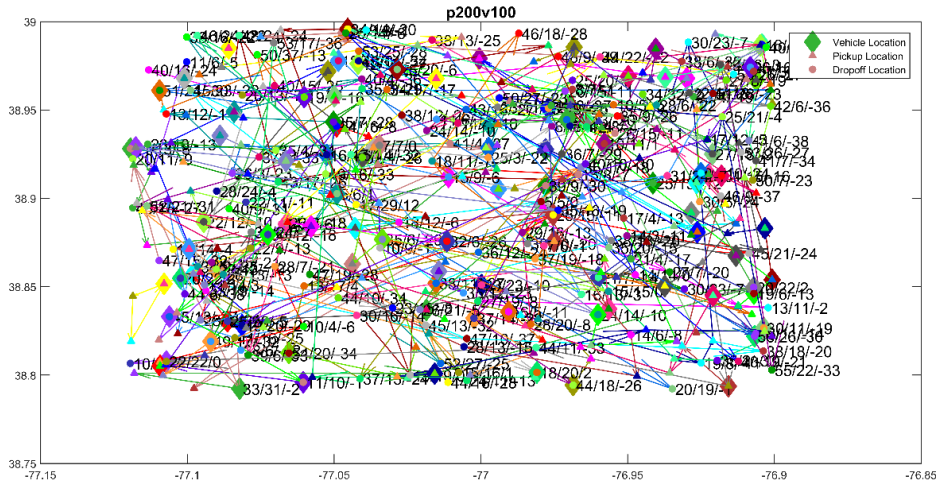


(b) Optimal Scheduling

Figure 4.8. A 100-passenger-50-vehicle case

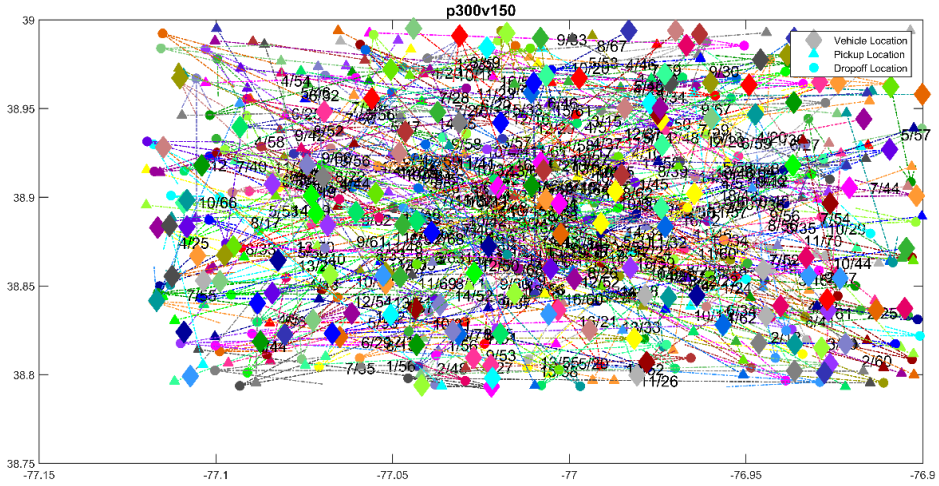


(a) Setup

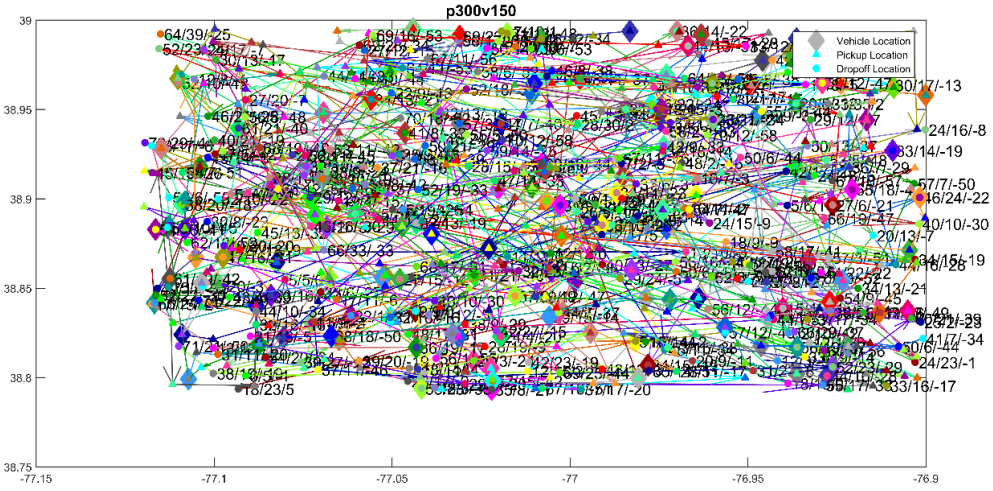


(b) Optimal Scheduling

Figure 4.9. A 200-passenger-100-vehicle case



(a) Setup



(b) Optimal Scheduling

Figure 4.10. A 300-passenger-150-vehicle case

Parallelization

The individual Monte Carlo simulations are completely independent from each other. They can be executed simultaneously and return their best solution to the master processor. No communication between the worker processors is needed. Therefore our algorithm can easily be parallelized. Below is an experiment designed to test the parallel computing effects on a 100-passenger-50-vehicle case. We conduct four sets of experiments, with 4, 8, 16, 32 processors respectively. Each experiment is repeated 20 times and the average runtime and standard deviations are reported, as plotted below. We can see that parallel computing can dramatically reduce the runtime as we invest more processors. Taxi service companies are supported by more powerful distributed computing infrastructures where there are thousands of processors available and we believe with the power of CPU clusters, the algorithm can handle large system scheduling problem in a real time fashion.

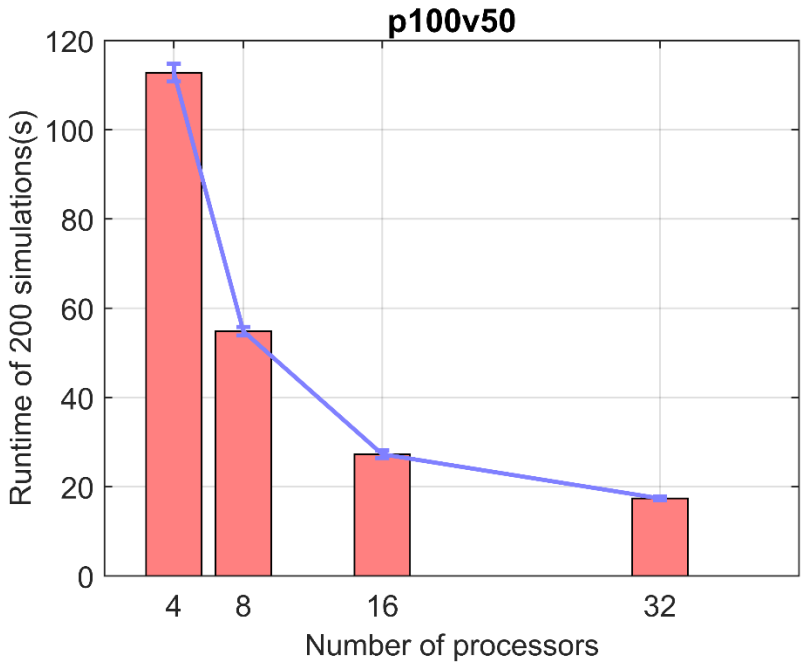


Figure 4.11 Runtime test of a 100-passenger-50-vehicle system using different numbers of processors.

Discussion

Selection of the number of simulations

We have known that because of the "short sight" issue that limits a single greedy agglomeration iteration to achieve a global optimal solution, and Monte Carlo simulation with randomly shuffled passenger list can improve the performance. We can imagine that with unlimited number of simulations, Monte Carlo simulations can always achieve the theoretical optimal solution. However, that would be too time consuming to be considered for real time deployment. On the other hand, as we have seen in Figure 4.4, where 200 simulations were executed and the minimum delay by each simulation is recorded, the best solution value drops dramatically at the beginning, then decreases slower and slower till it eventually reaches the theoretical optimal solution. For these six small test cases, the simulation lands on the optimal solution in less than 100 runs. For larger system, similarly, we can predict that, after a sufficient number of simulations, the achieved minimum delay value will be sufficiently close to the theoretical optimal value, although not strictly the optimal solution. From a practical/engineering perspective, in our ride sharing application, achieving the theoretical/mathematical absolute optimum at a huge computational cost is unnecessary. The theoretical optimum might be only a few seconds superior than a suboptimal solution, which really does not mean much for a hundred-vehicle-hundred-vehicle system, but the runtime cost might have to be doubled or even worse. Instead, achieving a "sufficiently optimal" solution within reasonable runtime is more practical, especially in an instantaneous dynamic scheduling system, where system response is critical. With this goal, we now discuss the selection of the number of simulations for a "sufficiently optimal" solution to systems of different number of passengers and vehicles.

We design five groups of experiments: 20-passenger-10-vehicle (p20v10), 50-passenger-20-vehicle, 100-passenger-50-vehicle (p100v50), 200-passenger-100-vehicle (p200v100), 300-passenger-150-vehicle (p300v150). For each group, we first conduct 1500 Monte Carlo simulations. The minimum delay curve all end up flat after 1000 simulations so it is out best belief that the optimal solutions have been reached within 1500 simulations. We also observe that the more passengers and vehicles we have, the more simulations it takes to reach the optimal solution. The optimal solution in the figure will be used to determine the number of simulations in the next session.

To determine a sufficient number of simulations for each case, we first have to define a "sufficient optimal solution". There is no precise definition of "sufficiently optimal solution" in the literature. We believe there are two reasonable way to define it:

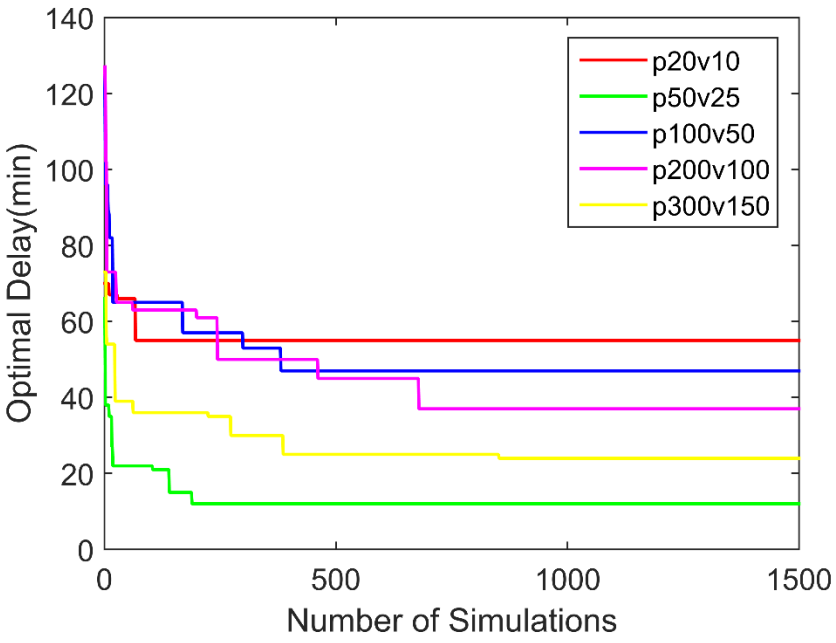


Figure 4.12. Improved solution values and increasing number of simulations

Definition 1: A solution that is within 10% of the optimal solution (For a minimization problem, such as out delay minimization problem, that is not exceeding 110% of the optimal solution).

Definition 2: A solution that is among the top 5 of all possible best solutions (smaller than or equal to the fifth minimum solution).

We now repeat each of the 5 experiments 200 times. In each experiment, instead of conducting the Monte Carlo simulation a fixed 1500 times, we terminate the simulation as long as it hits the "sufficiently optimal" solution we have just defined. We record the number of simulations when the algorithm is terminated. Figure 4.13 displays the histograms of the number of simulations of the 200 trials using Definition 1. Figure 4.14 are the histograms of the same experiments but evaluated by Definition 2.

We find the 95% quantiles of the sufficient number of simulations for all the groups, as shown in Table 4.2. Since the experiments are repeated 200 times, which is sufficient for us to use the 95% quantile to represent the sufficient number of simulations for that case. We further notice that the sufficient number of simulations is positively related to the product of number of passengers and number of vehicles. Figure 4.15 (a) and (b) plot the relationship between the number of simulations and the products of vehicle and passenger quantities, using

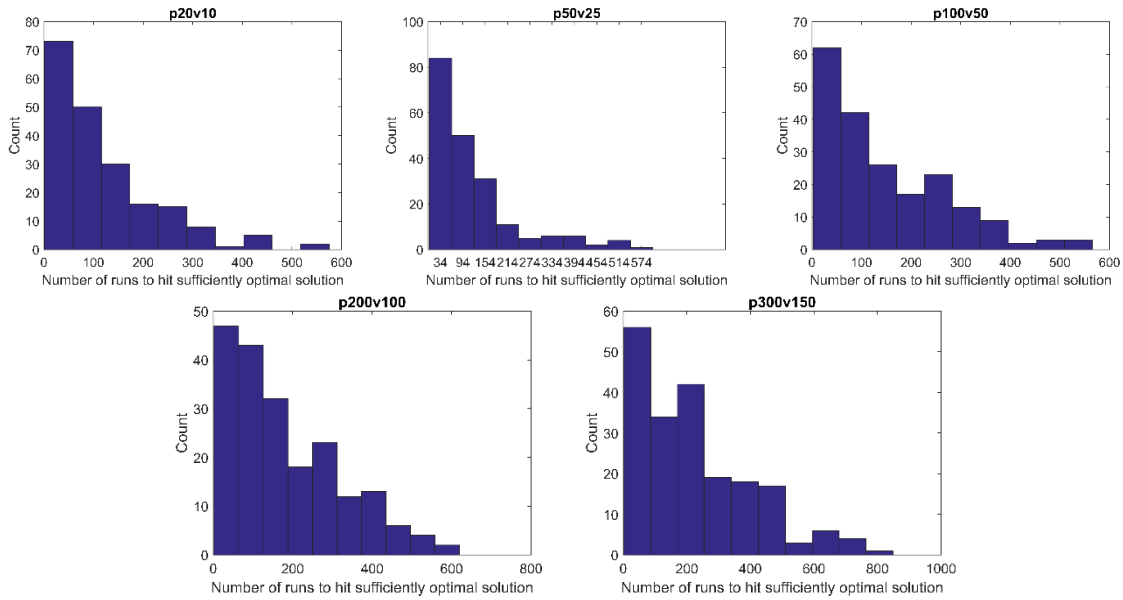


Figure 4.13. Histograms of the number of simulations to hit sufficiently optimal solution evaluated by Definition 1, 200 trials in total.

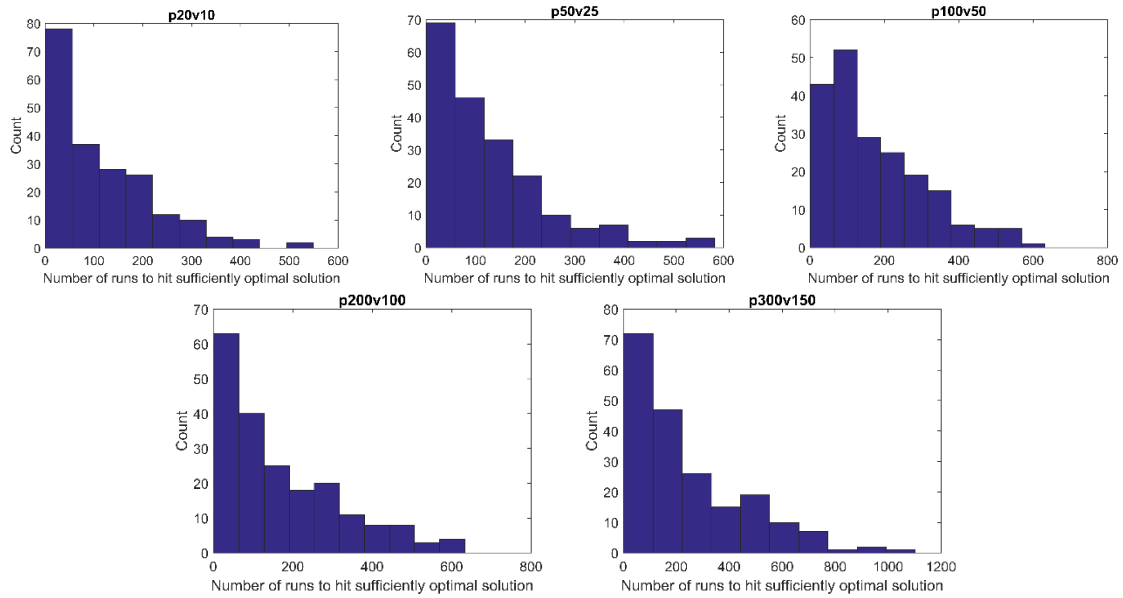
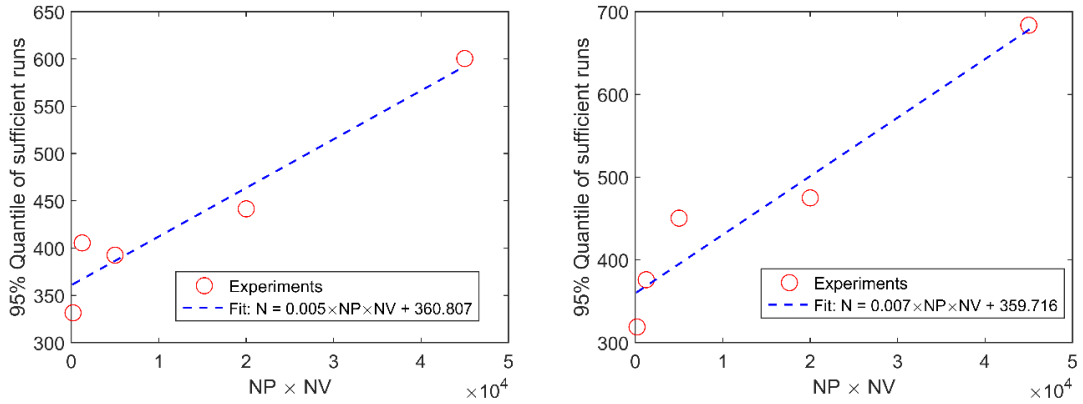


Figure 4.14. Histograms of the number of simulations to hit sufficiently optimal solution evaluated by Definition 2, 200 trials in total.

Table 4.2. 95% quantiles of sufficient number of simulations

Case	95% Quantile of Sufficient Number of Simulations	
	Definition 1	Definition 2
p20v10	327	319
p50v25	405	378
p100v50	380	451
p200v100	436	462
p300v150	587	689



(a) By Definition 1 (b) By Definition 2

Figure 4.15. Relationship between sufficient number of simulations and the product of number of passengers (NP) and number of vehicles (NV).

two definitions respectively. Although two definitions obtain two different regression fit, they can both be used to assess a reasonably sufficient number of simulations given the numbers of passengers and vehicles. Use Definition 1, the linear fit model is: $M = 0.005\ NP \times NV + 360.807$. Using Definition 2, the fit model is: $M = 0.007\ NP \times NV + 359.716$.

Proper calculation of delays

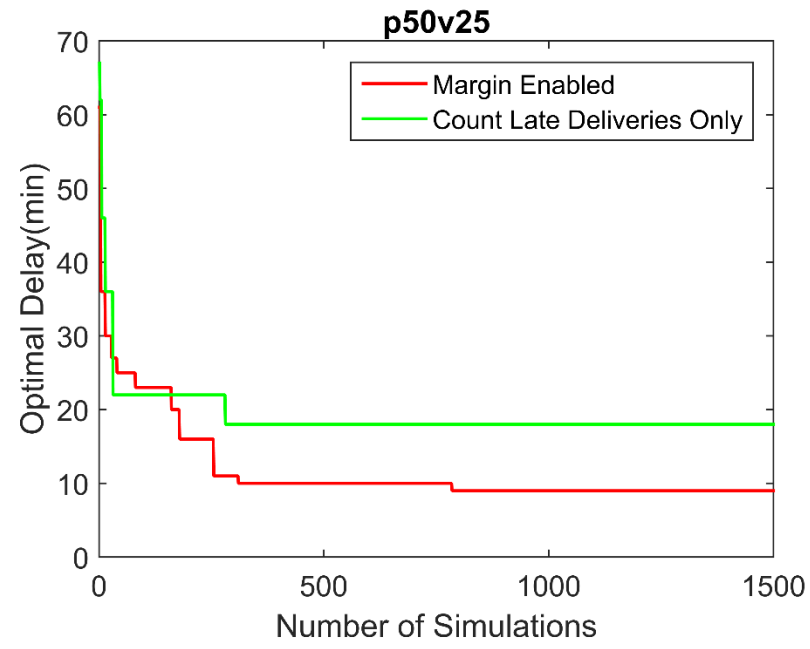
The **get_delay()** subroutine is frequently used in the **Best_Insertion** procedure. When trying to insert a new passenger into a vehicle, we loop through all possible routes and compare their system delays to find out the most efficient service order. Given a route, for example, $R = \{+p_{2,} + p_1, -p_1, +p_3, -p_{2,}...\}$, every passenger p_i has a requested arrival time l_i, where $i = 1, 2, ..., q$ and q is the number of

passengers served by the vehicle. Based on the origin-destination travel time matrix, we can also easily get the timestamps of arriving at every delivery location, say $T = \{t_1, t_2, ..., t_q\}$. There are two possible ways to calculate the delay: (1) Count only the late deliveries. No matter how early all other passengers are delivered, we count only the late delivered ones. $D = \sum_{i=1}^{q} (t_i - l_i) \mathrm{I}_i$, where $\mathrm{I}_i = \begin{cases} 0, & if\ t_i \le l_i \\ 1, & if\ t_i > l_i \end{cases}$. (2) Enable margins, count late deliveries as positive delays and early deliveries as negative delays, $D = \sum_{i=1}^{q} (t_i - l_i)$.

These two ways of calculating delays impact the quality of the algorithm differently. We conduct three experiments, using p50v25, p100v50, p200v100 cases. Two implementations of **get_delay()** are tested. Figure 4.16 shows their difference.

The three groups consistently show that counting late delivery only approach is inferior to margin enabled approach. The achieved minimum delay value by the former is obviously higher than that by the latter. It is not hard to explain the cause of this. If we count only the late deliveries for delay, at the beginning of the assigning process, all the vehicles have a zero delay, then the algorithm will assign passengers one after another to the first vehicle in the list until delay happens. Similarly the second vehicle will be first filled by other passengers. So all vehicles get filled up in a sequential order, and there is simply no comparison step involved. The overall system delay in this case is purely determined by the order of the vehicles. Although shuffling passenger list can improve its best solution, shuffling itself has limited power to make it close enough to the theoretical optimum. Instead, if the margins of the vehicle is considered, then even at the beginning, no two adjacent passengers will be assigned to the same vehicle , because once one passenger is assigned, that vehicle will have a decreased margin(increased delay), and so most likely is not the best option for the next passenger any more. Therefore, we should always use margin enabled way of calculating delays to count both positive and negative delays (margins).

Figure 4.16. Comparison of minimum delay value changes using two different ways of calculating delays



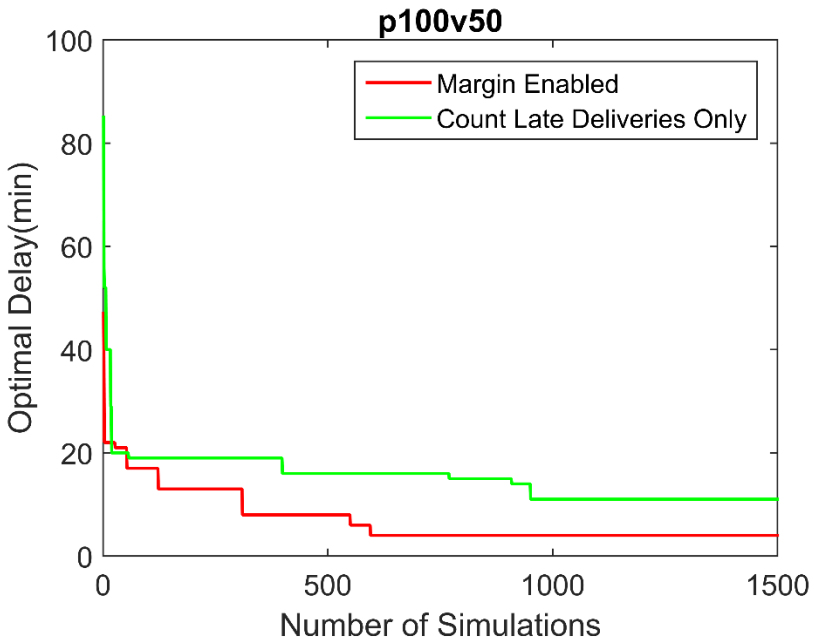


Figure 4.16. Continued

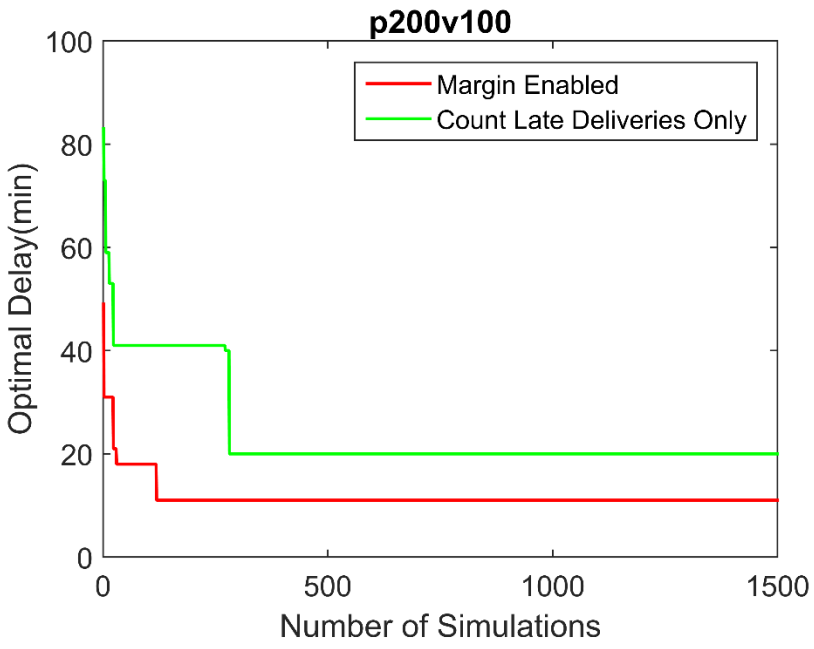


Figure 4.16. Continued

CHAPTER FIVE

TRAVEL TIME EXTRACTION FROM REAL ROAD NETWORK

The above optimization model relies on the travel time information between every pair of the locations involved, i.e. vehicles' depots, passengers' pickup locations and destinations. This is a multi-source multi-target shortest path (travel time) problem. Suppose we have a road network G of a city, $G=(E,V)$, where E is the set of the links, each link is associated with a weight, which is travel time on that link in our case, and V is the set of nodes. In multi-source multi-target shortest path problem, sources and targets are both a subset of node set V, we denote them as S, T, $S \subset V$, $T \subset V$. Here in our taxi – driver system, S and T are the same, and they are both the set of all the locations.

Floyd–Warshall algorithm is to get the all-pair shortest distances from a graph. It is not a good option here because Floyd – Warshall requires $O(n^2)$ space, where n is the number of nodes in the graph. Notice that here n is not the number of sources and targets, it is the number of nodes of the entire graph. Floyd – Warshall is not a good option for our application, because for a big road network that has thousands of nodes, it demands too much memory, and what is even worse is that if the number of sources and targets are only a small portion of the entire graph, it is just a waste of memory computation. What we need is a many-to-many, but not all-to-all shortest path algorithm. There is no direct method to solve a many-to-many routing problem, but we can decompose it to multiple single-source multi-target routing, i.e. repeatedly routing from each single source to reach all the targets.

There are two famous one-to-one shortest path algorithms: Dijkstra's and A*. For one-to-one Dijkstra's, the algorithm starts from the source node and visit and mark other nodes monotonically from the closest to the farthest. In the one-to-one case, the algorithm terminates when the target is visited. In the one-to-many scenario, we simply maintain a *target_list*, which is initially the target set T, and whenever a target is visited, remove it from the *target_list* until the list becomes empty. The details are shown in Figure 5.1.

Another approach is to use the one-to-one A* algorithm. A* algorithm is essentially breadth first search. A* has been used in single-source-single-target shortest path planning. For every node, three scores are defined; gScore – the distance/cost from Start to this node, hScore – the heuristic distance/cost from this node to Goal, fScore – the sum of gScore and hScore. To get gScore, we add the costs of the paths from Start to the node. To make A* algorithm work correctly, hScore has to

be an underestimate of the cost from the node to Goal. Sine straight line distance is always the underestimate of real shortest distance, we can simply calculate the gScore as the straight line distance based on their coordinates. A priority queue openList is used to store the nodes waiting to be evaluated, and the nodes are ordered based on their fScore. In every iteration, the closest node to the Goal is expanded. We calculate the three scores for the neighbors and push them to the openList for future visits. The algorithm terminates as soon as Goal is visited.

Algorithm: Multi-source Multi-target Shortest Path
Input: Graph *G*, source set *S*, target set *T*
Output: shortest distance(travel time) array *D*, where $D_{ij} = travel\_time(s_i,t_j)$

```
Many-to-many-Dijkstra(G,S,T){
        for each s in S:
                D[s,:] = One-to-many-Dijkstra(G,s,T)
        end for
}

One-to-Many-Dijkstra(G, s, target_list){
        for each node n of G:
                t[n] ← Infinity
        end for
        Q = {s}
        while Q not empty and target_list not empty:
                u ←Q[0]
                mark u as visited
                Q←Q -{u}
                if u is in target_list:
                        target_list.erase(u)
                end if

                for each neighbor v of u:
                     if v is not visited and t[u]+t(u,v) < t[v]:
                        t[v] = t[u]+t(u,v)
                        Q←Q ∪ {v}
                     end if
                end for
         end while
}
```

Figure 5.1. Multi-source multi-target shortest path algorithm based on Dijkstra's.

```
Algorithm: A* based Multi-source Multi-target Shortest Path
Input: Graph G, source set S, target set T
Output: shortest distance(travel time) array D, where D_ij = travel_time(s_i,t_j)
Many-to-many-Dijkstra(G,S,T){
        for each s in S:
                D[s,:] = One-to-many-A*(G,s,T)
        end for
}

One-to-many-A*(G, Start, goal_List){
//Push the goals into the 2-dimensional tree
for each goal in goal_List
2DTree.insert(goal)
end for

openList = {Start}   // A heap sorted by the fScore of its elements
gScore(Start) = 0;
hScore(Start) = get_nearest_distance(Start, 2DTree);
While openList is not empty && goal_List is not empty
        q = openList.top()   // the nodes with the least f score.
        openList.pop()
        for each neighbor of q in G:
                if neighbor is in goalList:
                        goalList.remove(neighbor)
                        2DTree.remove(neighbor)
                end if
                gScore(neighbor) = gScore(q) + d(q,neighbor) ;
                hScore(neighbor) = get_nearest_distance(neighbor, 2DTree);
                fScore(neighbor) = gScore(neighbor) + hScore(neighbor);
                if neighbor is not in openList:
                        openList. insert(neighbor)
                else if fScore(neighbor) < openList(neighbor).fScore
                        // update the neighbor's fScore in openList.
                        openList(neighbor).fScore = fScore(neighbor)
                end if
        end for
end while
}
```

Figure 5.2. Multi-source multi-target shortest path algorithm based on A*.

We can build the single-source-multi-target A* algorithm based on the single-source-single-target version. Now we have not only one Goal, but a goalList. The algorithm has two modifications: (1) When calculating hScore for a node, we use the straight line distance between this node and the nearest unvisited goal in goalList. This is correct because the distance to the nearest goal is always an underestimate to any other goal and so guarantee the result of A* is correct. (2) When one goal is visited, we have found the shortest path to this goal, and we should remove it from goalList. The algorithm terminates when goalList is empty. To accelerate the hScore calculation – the distance from a node to the nearest goal, we use K-dimensional tree as the support data structure. Initially, we push all the goal nodes into a 2-dimensional tree, indexed by their x-y (lat-lon) coordinates. Nearest neighbor search on a 2-dimensional tree has a complexity of only $O\log(\mathrm{N})$, where N is the number of goal nodes here. The pseudo-code of A* based multi-source-multi-target shortest path algorithm is presented in Figure 5.2.

We test the two algorithms on the Washington DC road network, which has 18532 links and 12006 nodes, with five cases (number of sources × number of targets): 10×10, 50×50, 100×100, 500×500 and 1000×1000. Figure 5.3 presents the runtime comparison. Although for small test cases (10×10, 50×50), A-star based algorithm seems to be faster than Dijkstra's, Dijkstra's algorithm outperforms A* when there are more sources and goals. Considering a real system should be able to support at least a few hundred of passengers and vehicles, we conclude that Dijkstra's is the winner and should be used to extra travel time data. Notice that here the algorithms are testes on a single processor environment, but repeated one-to-many shortest path algorithms can easily be distributed to multiple processors to speed up.

#Sources × #Goals	Runtime (s)	
	Dijkstra's	A*
10 × 10	0.98	0.42
50 × 50	5.20	5.05
100 × 100	9.82	10.98
500 × 500	49.05	229.82
1000 × 1000	98.10	817.73

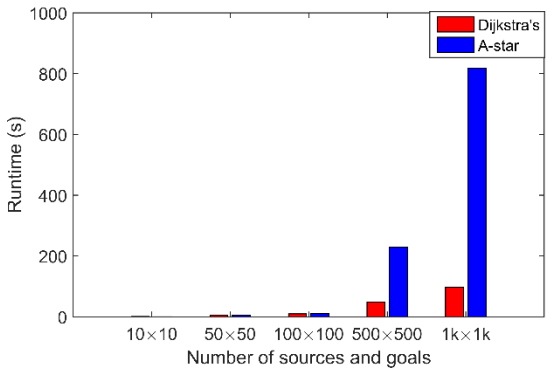


Figure 5.3. Runtime comparison of Dijkstra's and A* many-to-many shortest path algorithms.

CHAPTER SIX

CONCLUSION AND FUTURE WORK

Taxi sharing is a promising travel mode in big cities to address many traffic issues such as congestions, insufficiency of cabs, pollutions, etc. To tackle the service delay challenge in taxi-sharing services, we formulate a mixed linear integer programming model to match drivers and riders and offer the optimal service order to minimize overall delivery delays for all customers. The model provides the most efficient service schedules, and offers the time and distance information that is valuable for arrival time estimation at each stop, fare estimation and so is able to keep the riders informed of their cost and trip details in advance.

Solving the optimization model using standard solver packages takes forbiddingly long to make it practical for real-time deployment on large scale ride sharing systems. We have presented a greedy agglomeration and Monte Carlo simulation based method. Passengers are iteratively assigned to the "so-far-best" vehicle that causes the minimum increase to our objective – system-wide delay. As a greedy algorithm, it is short sighted and does not guarantee global minimum. To resolve this issue, we repeatedly shuffle the passenger list and repeat the greedy clustering process. We have conducted small scale test cases, where the comparisons with optimization model show that after sufficient number of simulations, our algorithm always achieve the theoretical optimal solution. Cases studies on large scale systems further validate its fast performance. The proposed algorithm is straightforward to be parallelized and utilize distributed computing resources to speed up and support larger scale real-time ride sharing services. We have also discussed two details: the selection of number of simulations and the proper calculation of delay given a route. These discussion provides useful advices on applying this algorithm in practical system design. The algorithm relies on a origin-destination travel time matrix. Considering travel time varies with time in a real road network, we then compare two many-to-many shortest path algorithms that can extract travel time information for a real road network. Results show that Dijkstra's one-to-many shortest path algorithm outperforms one-to-many A* in large scale network.

There are still many practical situations to consider in order to design a high quality high efficiency ride sharing system.

For one thing, the service delay should include two parts: waiting delay and delivery delay. Although our model have considered only the delivery delay, we can simply modify the Get_Delay() subroutine in the greedy algorithm to incorporate waiting delay, and maybe assign different weights to the two parts.

An interesting and valuable extension to the taxi sharing model itself is that, in practice, when a requester orders a ride, besides specifying a preferred latest delivery time, they can also choose an urgency level associated with that time point. For example, a user wants to be dropped at his destination at 9 am, but he also chooses “3” out of five importance levels. Choosing a lower level will be rewarded a lower fare rate because it allows the system to give higher priority to serve other passengers with higher urgency level. To show this in our optimization objective function, we basically give a higher penalty weight for the delay of higher urgency level, and vice versa. The feedbacks we acquired from Google store also inspire us to other potential extensions. For example, to ensure user comfortableness, the duration an individual passenger stays in the vehicle cannot exceed double (or other ratio) of the minimum time it needs, otherwise the passenger would get exhausted. For fairness considerations, the service order and individual delays could also be reflected in the differences of prices, which also makes the model possibly useful for price modeling.

From a system operation perspective, our current study focuses on a quite simple static operation mode. Every a certain amount of time, the system checks the vehicles that are off-duty (not serving anyone at the moment), and the passengers that have requested rides since last scheduling. Then these passengers and vehicles are to be scheduled. Vehicles must commit to the passengers assigned to them till all passengers are delivered. In fact, this type of operation is not a high-efficiency one. There might be new passengers coming up and their pickup and delivery locations are right on the one of the existing service route, and adding them to the service does not affect the earlier assigned passengers. Models enabling dynamic passenger insertion would be more complicated but more realistic toward real and efficient ride sharing system operation. Besides accommodating new ride requests, a system that allows and responds to ride cancellation by quickly re-scheduling would benefit system efficiency as well.

It is also open to discussion whether every passenger should be served regardless their distance to the vehicles, or some passengers can be rejected for service if they are too far away from available vehicles, or their time requests are too demanding. The underlying assumption in our model is that all passengers will be served. If it is the other situation, say we only have to serve a least portion of passengers, then the selection of to-serve and not-to-serve passengers becomes a new challenge.

Another discussion is on the objective function. Currently, our objective is the simple linear summation of all the delays. It is possible that in reality, small delay can be tolerated and neglected, while large delay can cause significant customer complaint, and so there could be a nonlinear mapping between service delay and our cost function.

Another important facet of a ride sharing service system is pricing. Different from regular taxi pricing that considers mainly distance and time, pricing for shared rides also should reflect individual customers' experience in the service, e.g. early delivered customers pay normal fare, while late delivered should be offered a discount. Our scheduling model offers all the travel time and distance information needed for fare estimates.

These extensions and modifications will be investigated in the next stage of our study.

LIST OF REFERENCES

1. Furuhata, M., et al., *Ridesharing: The state-of-the-art and future directions.* Transportation Research Part B: Methodological, 2013. **57**: p. 28-46.
2. Chan, N.D. and S.A. Shaheen, *Ridesharing in north america: Past, present, and future.* Transport Reviews, 2012. **32**(1): p. 93-112.
3. Clark, B., K. Chatterjee, and S. Melia, *Changes in level of household car ownership: The role of life events and spatial context.* Transportation, 2015: p. 1-35.
4. Teodorović, D. and M. Dell'Orco, *Mitigating traffic congestion: solving the ride-matching problem by bee colony optimization.* Transportation Planning and Technology, 2008. **31**(2): p. 135-152.
5. Downs, A., *Stuck in traffic: Coping with peak-hour traffic congestion*. 2000: Brookings Institution Press.
6. Schrank, D., B. Eisele, and T. Lomax, *TTI's 2012 urban mobility report.* Texas A&M Transportation Institute. The Texas A&M University System, 2012.
7. Knittel, C.R., D.L. Miller, and N.J. Sanders, *Caution, drivers! Children present: Traffic, pollution, and infant health.* Review of Economics and Statistics, 2016. **98**(2): p. 350-366.
8. Laumbach, R.J. and H.M. Kipen, *Respiratory health effects of air pollution: update on biomass smoke and traffic pollution.* Journal of allergy and clinical immunology, 2012. **129**(1): p. 3-11.
9. Jerrett, M., et al., *Traffic-related air pollution and obesity formation in children: a longitudinal, multilevel analysis.* Environmental Health, 2014. **13**(1): p. 1.
10. Heck, J.E., et al., *Childhood cancer and traffic-related air pollution exposure in pregnancy and early life.* Cancer Research, 2013. **73**(8 Supplement): p. 2531-2531.
11. Poulsen, A.H., et al., *Air pollution from traffic and risk for brain tumors: a nationwide study in Denmark.* Cancer Causes & Control, 2016. **27**(4): p. 473-480.
12. Deakin, E., K. Frick, and K. Shively, *Markets for dynamic ridesharing? Case of Berkeley, California.* Transportation Research Record: Journal of the Transportation Research Board, 2010(2187): p. 131-137.
13. Shirgaokar, M. and E. Deakin, *Study of park-and-ride facilities and their use in the San Francisco Bay Area of California.* Transportation Research Record: Journal of the Transportation Research Board, 2005(1927): p. 46-54.
14. Skok, W. and M. Tissut, *Managing change: the London taxi cabs case study.* Strategic Change, 2003. **12**(2): p. 95-108.
15. da Costa, D. and R. De Neufville, *Designing efficient taxi pickup operations at airports.* Transportation Research Record: Journal of the Transportation Research Board, 2012(2300): p. 91-99.

16. Fosgerau, M. and A. De Palma, *The dynamics of urban traffic congestion and the price of parking.* Journal of Public Economics, 2013. **105**: p. 106-115.
17. Duranton, G. and M.A. Turner, *Urban growth and transportation.* The Review of Economic Studies, 2012. **79**(4): p. 1407-1440.
18. Santi, P., et al., *Quantifying the benefits of vehicle pooling with shareability networks.* Proceedings of the National Academy of Sciences, 2014. **111**(37): p. 13290-13294.
19. Agatz, N., et al., *Optimization for dynamic ride-sharing: A review.* European Journal of Operational Research, 2012. **223**(2): p. 295-303.
20. Bektas, T., *The multiple traveling salesman problem: an overview of formulations and solution procedures.* Omega, 2006. **34**(3): p. 209-219.
21. Kara, I. and T. Bektas, *Integer linear programming formulations of multiple salesman problems and its variations.* European Journal of Operational Research, 2006. **174**(3): p. 1449-1458.
22. Dantzig, G.B. and J.H. Ramser, *The truck dispatching problem.* Management science, 1959. **6**(1): p. 80-91.
23. Clarke, G. and J.W. Wright, *Scheduling of vehicles from a central depot to a number of delivery points.* Operations research, 1964. **12**(4): p. 568-581.
24. Miller, L.R., *Heuristic algorithms for the generalized vehicle dispatch problem.* 1970.
25. Foster, B.A. and D.M. Ryan, *An integer programming approach to the vehicle scheduling problem.* Journal of the Operational Research Society, 1976. **27**(2): p. 367-384.
26. Ryan, D.M., C. Hjorring, and F. Glover, *Extensions of the petal method for vehicle routeing.* Journal of the Operational Research Society, 1993. **44**(3): p. 289-296.
27. Laporte, G., et al., *Classical and modern heuristics for the vehicle routing problem.* International transactions in operational research, 2000. **7**(4-5): p. 285-300.
28. Christofides, N. and S. Eilon, *An algorithm for the vehicle-dispatching problem.* Journal of the Operational Research Society, 1969. **20**(3): p. 309-318.
29. Acharyya, R. and G.K. Das, *Unit disk cover problem.* arXiv preprint arXiv:1209.2951, 2012.
30. Thompson, P.M. and H.N. Psaraftis, *Cyclic transfer algorithm for multivehicle routing and scheduling problems.* Operations research, 1993. **41**(5): p. 935-946.
31. Katoh, N. and T. Yano, *An approximation algorithm for the pickup and delivery vehicle routing problem on trees.* Discrete Applied Mathematics, 2006. **154**(16): p. 2335-2349.
32. Tzoreff, T.E., et al., *The vehicle routing problem with pickups and deliveries on some special graphs.* Discrete Applied Mathematics, 2002. **116**(3): p. 193-229.

33. Gribkovskaia, I., O. Halskau, and K.N.B. Myklebost, *Models for pick-up and deliveries from depots with lasso solutions.* NOFOMA2001, Collaboration in logistics: connecting islands using information technology, 2001: p. 279-293.
34. Gribkovskaia, I., et al., *General solutions to the single vehicle routing problem with pickups and deliveries.* European Journal of Operational Research, 2007. **180**(2): p. 568-584.
35. Nagy, G. and S.d. Salhi, *Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries.* European journal of operational research, 2005. **162**(1): p. 126-141.
36. Solomon, M.M., *Algorithms for the vehicle routing and scheduling problems with time window constraints.* Operations research, 1987. **35**(2): p. 254-265.
37. Cordeau, J.-F. and G.d.é.e.d.r.e.a.d. décisions, *The VRP with time windows*. 2000: Montréal: Groupe d'études et de recherche en analyse des décisions.
38. Bräysy, O. and M. Gendreau, *Vehicle routing problem with time windows, Part I: Route construction and local search algorithms.* Transportation science, 2005. **39**(1): p. 104-118.
39. Bräysy, O. and M. Gendreau, *Vehicle routing problem with time windows, Part II: Metaheuristics.* Transportation science, 2005. **39**(1): p. 119-139.
40. Solomon, M.M., *VRPTW Benchmark Problems* 2005.
41. Cordeau, J.-F., *A branch-and-cut algorithm for the dial-a-ride problem.* Operations Research, 2006. **54**(3): p. 573-586.
42. Ropke, S. and J.-F. Cordeau, *Branch and cut and price for the pickup and delivery problem with time windows.* Transportation Science, 2009. **43**(3): p. 267-286.
43. Yang, L. and X. Zhou, *Constraint reformulation and a Lagrangian relaxation-based solution algorithm for a least expected time path problem.* Transportation Research Part B: Methodological, 2014. **59**: p. 22-44.
44. Mahmoudi, M. and X. Zhou, *Finding optimal solutions for vehicle routing problem with pickup and delivery services with time windows: A dynamic programming approach based on state–space–time network representations.* Transportation Research Part B: Methodological, 2016. **89**: p. 19-42.
45. Hanne, T., T. Melo, and S. Nickel, *Bringing robustness to patient flow management through optimized patient transports in hospitals.* Interfaces, 2009. **39**(3): p. 241-255.
46. Beaudry, A., et al., *Dynamic transportation of patients in hospitals.* OR spectrum, 2010. **32**(1): p. 77-107.
47. Ma, S., Y. Zheng, and O. Wolfson. *T-share: A large-scale dynamic taxi ridesharing service*. in *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*. 2013. IEEE.

48. Huang, Y., et al., *Large scale real-time ridesharing with service guarantee on road networks.* Proceedings of the VLDB Endowment, 2014. **7**(14): p. 2017-2028.
49. Moccia, L., *New optimization models and algorithms for the management of maritime container terminals*. 2004, PhD thesis, Universita degli studi della Calabria.
50. Burks Jr, R.E., *An adaptive tabu search heuristic for the location routing pickup and delivery problem with time windows with a theater distribution application*. 2006, DTIC Document.
51. Parragh, S.N., K.F. Doerner, and R.F. Hartl, *A survey on pickup and delivery problems.* Journal für Betriebswirtschaft, 2008. **58**(1): p. 21-51.
52. Berbeglia, G., et al., *Static pickup and delivery problems: a classification scheme and survey.* Top, 2007. **15**(1): p. 1-31.
53. Optimization, G., *Gurobi optimizer reference manual.* URL: http://www. gurobi. com, 2012. **2**: p. 1.3-3.3.
54. CPLEX, I.I., *V12. 1: User's Manual for CPLEX.* International Business Machines Corporation, 2009. **46**(53): p. 157.
55. Tabatabaei, S.S., M. Coates, and M. Rabbat, *GANC: Greedy agglomerative normalized cut for graph clustering.* Pattern Recognition, 2012. **45**(2): p. 831-843.

APPENDIX

Data for the setups and results of the case study in Chapter 4.

A1. Case p4v2

Vehicle ID	X	Y	Passenger ID	Origin X	Origin Y	Dest X	Dest Y	Preferred Arrival Time
1	19.07	21.52	1	25.21	11.83	8.33	16.62	24
2	4.25	18.21	2	23.49	23.95	14.32	18.87	18
			3	27.35	5.93	10.94	15.40	46
			4	10.06	23.05	28.57	27.49	41

Optimal Scheduling:
v1: +p1 → -p1 → +p3 → -p3
v2: +p2 → +p4 → -p4 → -p2
Minimum system delay: 27.15 min.

A2. Case p5v3

Vehicle ID	X	Y	Passenger ID	Origin X	Origin Y	Dest X	Dest Y	Preferred Arrival Time
1	0.49	7.29	1	25.21	11.83	14.32	18.87	34
2	4.12	24.13	2	23.49	23.95	10.94	15.40	43
3	4.70	12.03	3	27.35	5.93	28.57	27.49	44
			4	10.06	23.05	19.07	21.52	23
			5	8.33	16.62	4.25	18.21	33

Optimal Scheduling:
v1: +p2 → -p2
v2: +p5 → -p5 → +p3 → -p3
v3: +p5 → +p6 → -p5 → +p4 → -p4 → -p6
Minimum system delay: 49.83 min.

A3. Case p6v3

Vehicle ID	X	Y	Passenger ID	Origin X	Origin Y	Dest X	Dest Y	Preferred Arrival Time
1	4.70	14.03	1	25.21	11.83	10.94	15.40	43
2	3.89	3.26	2	23.49	23.95	26.57	27.49	12
3	29.97	6.55	3	27.35	5.93	19.07	21.52	33
			4	10.06	23.05	4.25	18.21	15
			5	8.33	16.62	0.49	7.29	18
			6	14.32	18.87	4.12	24.13	20

Optimal Scheduling:
v1: +p1 → +p3 → -p1 → -p3
v2: +p2 → -p2
v3: +p1 → -p1 → +p4 → -p4
Minimum system delay: 12.60 min.

A4. Case p6v4

Vehicle ID	X	Y	Passenger ID	Origin X	Origin Y	Dest X	Dest Y	Preferred Arrival Time
1	4.70	14.03	1	25.21	11.83	10.94	15.40	43
2	3.89	3.26	2	23.49	23.95	26.57	27.49	12
3	29.97	6.55	3	27.35	5.93	19.07	21.52	33
4	15.39	25.17	4	10.06	23.05	4.25	18.21	15
			5	8.33	16.62	0.49	7.29	18
			6	14.32	18.87	4.12	24.13	20

Optimal Scheduling:
v1: +p2 → -p2
v2: +p6 → -p6
v3: +p5 → -p5 → +p3 → -p3
v3: +p1 → -p1 → +p4 → -p4
Minimum system delay: 28.94 min.

A5. Case p7v3

Vehicle ID	X	Y	Passenger ID	Origin X	Origin Y	Dest X	Dest Y	Preferred Arrival Time
1	29.97	6.55	1	25.21	11.83	28.57	27.49	22
2	15.39	25.17	2	23.49	23.95	19.07	21.52	14
3	18.38	8.88	3	27.35	5.93	4.25	18.21	36
			4	10.06	23.05	0.49	7.29	41
			5	8.33	16.62	4.12	24.13	13
			6	14.32	18.87	4.70	12.03	33
			7	10.94	15.40	3.89	3.26	21

Optimal Scheduling:
v1: +p5 → +p4 → -p4 → -p5
v2: +p1 → -p1 → +p6 → +p7 → -p6 → -p7
v3: +p3 → -p3 → +p2 → -p2
Minimum system delay: 49.47 min.

A6. Case p7v4

Vehicle ID	X	Y	Passenger ID	Origin X	Origin Y	Dest X	Dest Y	Preferred Arrival Time
1	29.97	6.55	1	25.21	11.83	28.57	27.49	22
2	15.39	25.17	2	23.49	23.95	19.07	21.52	14
3	18.38	8.88	3	27.35	5.93	4.25	18.21	36
4	19.13	15.73	4	10.06	23.05	0.49	7.29	41
			5	8.33	16.62	4.12	24.13	13
			6	14.32	18.87	4.70	12.03	33
			7	10.94	15.40	3.89	3.26	21

Optimal Scheduling:
v1: +p6 → -p6
v2: +p1 → -p1 → +p7 → -p7
v3: +p5 → -p5 → +p3 → -p3
v3: +p2 → +p4 → -p2 → -p4
Minimum system delay: 9.44 min.

VITA

Yang is a third year graduate student in the Ph.D. program of Transportation at The University of Tennessee, Knoxville. He is also getting a M.S. in Computer Engineering as well as a M.S. in Statistics. For his research, he develops new data mining and machine learning algorithms for geospatial data management and resource distributions over large transportation networks. He is interested in both building advanced mathematical models and designing efficient algorithms to solve the models.

Yang has served as an instructor of Engineering Fundamental 230 - Computer Solutions to Engineering Problems class. He has taught over 250 students and feels proud of teaching their first programming class. Outside of school and work, he works out and swims three times a week.

Yang obtained his bachelor in Electrical and Computer Engineering (Automatic Control System division) from Beijing Jiaotong University in Beijing, China. He graduated first place among 180 students and was a three-time winner of Chinese National Scholarship.