On Demand High Capacity Ride Sharing for Mobility on Demand (MoD) Systems

Vindula Muthushan Jayawardana

188003B

Thesis/Dissertation submitted in partial fulfillment of the requirements for the degree Master of Science in Computer Science and Engineering

Department of Computer Science & Engineering

University of Moratuwa

Sri Lanka

July 2019

DECLARATION

I declare that this is my own work and this dissertation does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: Date:

The above candidate has carried out research for the Masters thesis/Dissertation under my supervision.

Name of the Supervisor: Dr. Uthayasanker Thayasivam

Signature of the Supervisor: Date:

Name of the Supervisor: Dr. Amal Shehan Perera

Signature of the Supervisor: Date:

Name of the Supervisor: Prof. Samitha Samaranayake

Signature of the Supervisor: Date:

ACKNOWLEDGEMENTS

Firstly, I would like to express my sincere gratitude to my advisors, Prof.Samitha Samaranayake, Dr.Uthayasanker Thayasivam and Dr.Amal Shehan Perera for their continuous guidance and support throughout this project. I could not have imagined having a better set of advisors for my research. I am highly grateful for their insightful guidance and encouragement not just relating to the research work but also regarding important life decisions that I had to make.

I would also like to thank my thesis committee: Dr.Dilum Bandara, Prof.Sanath Jayasena, Dr.Srinath Perera and, Dr.Ruwan Udayanga, for their insightful comments and constructive criticism which helped immensely in re-shaping my thinking and to look in different perspectives towards my research work.

A special thank should goes to Digital Mobility Solutions Lanka Private Limited for their generous funding towards this research. I would like to thank Mr.Jiffy Zulfer, and Mr.Asanka Perera for their support and for the provided facilities throughout my research work.

I thank my fellow lab mates in Data Science and Engineering Research Hub of University of Moratuwa and also from Mobility, Algorithms and Society lab at Cornell University for the informative discussions and for the great time at the lab.

Also, I am grateful to Prof.Huy T. Vo and his team at New York University for the great support in the collaborative work we did. I recall the great company and support I received in my short stay at Cornell University and I am thankful for the people who made it happen.

I would like to thank the entire staff of the Department of Computer Science and Engineering for all their help during the course of this work and for providing me with the resources necessary to conduct my research.

Last but not the least, I would like to thank my family, my parents, my brother and my girlfriend for continuously supporting me in writing this thesis and my life in general.

ABSTRACT

On Demand High Capacity Ride Sharing for Mobility on Demand (MoD) Systems

Mobility-on-demand (MoD) systems are emerging as a novel mode for urban mobility which have enabled the users to have a reliable medium for mobility requirements. Ride sharing services provided by these mobility on demand systems provide not just a personalized experience in mobility but also present an immense potential for positive societal impacts with reference to pollution, energy consumption, congestion, and etc. Ride sharing services primarily concern with picking up spatiotemporally distributed mobility demand and delivering it within a pre-specified time window subjected to different other constraints. Large scale ride sharing in more sophisticated spatiotemporally distributed mobility demand distributions require mathematical models and algorithms in order to match riders and vehicle fleets in real time. As such common requirement of models which address the potential of ride sharing is meticulously planned routes for a fleet of taxis. Accordingly another such requirement is to incorporate meeting points in route planning. In this work, we address the use of aforementioned two requirements for ride sharing systems. Therefore our main contribution lies in the intersection of (1) determining optimal routes for fleet of taxis with arbitrary set of constraints and, (2) introducing meeting points to improve ride sharing.

We approach the optimal route planning problem by proposing two new types of queries called *Generalized Optimal Single Meeting Point Route (GOSMPR)* query and *Generalized Optimal Multi Meeting Point Route (GOMMPR)* query in road networks. Given a vehicle current location s, a set of origin and destinations pairs of passenger set P and, an arbitrary set of constraints C, *GOSMPR* aims to find the optimal route for the given taxi such that the selected route traverse all origins and destinations while adhering to the constraints set C and minimizing the total path cost. In the same way, *GOMMPR* address the same problem but by replacing each origin and destination with set of pickup points and drop off points respectively. We show that problem of *GOSMPR* and *GOMMPR* queries are NP-hard. Thus to address the *GOSMPR* problem efficiently, we propose a dynamic programming approach inspired by A^* algorithm. To reduce the search state space of the proposed algorithm, we propose a series of state elimination criteria and a novel variant of classical *Held Karp* lower bound of Travelling Salesman Problem (TSP) as a heuristic. Also to address the *GOMMPR* problem, we propose another dynamic programming based algorithm. We demonstrate that our proposed

algorithm for *GOSMPR* can efficiently solve 10 passenger requests (21 points)(more than the capacity of a typical ride sharing vehicle) optimally while the algorithm for *GOMMPR* can solve up to 4 passenger requests within acceptable time limits.

We also conduct extensive experiments to demonstrate the impact of meeting points in the context of ride sharing. We compare state of art ride sharing model *RS-ILP* and our new meeting points based model *RSMP-ILP* to demonstrate the impact of meeting points. We show that there is an improvement of 1.53% in service rates in *RSMP-ILP* compared to *RS-ILP* which corresponds to 5048 trips in total.

Keywords: ride-sharing; dynamic programming; travelling salesman problem; vehicle routing; meeting points; optimal route;

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

GOSMPR	Generalized Optimal Single Meeting Point Route
GOMMPR	Generalized Optimal Multi Meeting Point Route
ILP	Integer Linear Programming
RS-ILP	Ride Sharing with Integer Linear Programming
RSMP-ILP	Ride Sharing with Meeting Points and Integer Linear Programming
HK	Held Karp

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

Mobile-hailed dynamic ride sharing services are witnessing a growing popularity and exponential growth compared to traditional transportation mediums. As highlighted by Javier Alonso-Mora et al. [1], these services have the potential for a tremendous positive impact on personal mobility, pollution, congestion, energy consumption, and thereby quality of life. Being an intermediate communicator between potential riders and potential drivers, these ride hailing services have gained a prominence in the world economy.

However, designing this kind of complex systems is a not a trivial task. It involves careful planning focusing on efficiency and computational tractability. One of the most common requirement of these ride sharing services is the requirement of computing optimal routes for a fleet of taxis. Given a taxi with matched set of passengers, it requires to compute an optimal route under given set of arbitrary constraints such as capacity of the vehicle, precedence and time windows. Due to the hard nature of this problem, many recent studies have adopted exhaustive search method in finding the optimal routes [1] [2] for vehicles with lower capacity. For vehicles with larger capacity, heuristic methods such as Lin-Kernighan [3], Tabu Search [4] [5], Insertion method [1] or Simulated Annealing (SA) [5] have been used.

In the context of this work, we address this specific problem of optimal route planning by introducing two types of new queries. We introduce two new types of queries called *Generalized Optimal Single Meeting Point Route (GOSMPR)* query and *Generalized Optimal Multi Meeting Point Route (GOMMPR)* query in road networks for optimal route planning with arbitrary constraints. Given a vehicle current location s, a set of origin and destinations pairs of passenger set P and, an arbitrary set of constraints set C, *GOSMPR* aims to find the optimal route for the given taxi such that the selected route traverse all origins and des-

tinations while adhering to the constraints set C and minimizing the total path cost. In the same way, *GOMMPR* address the same problem but by replacing each origin and destination with set of n pickup points and m drop off points respectively. We show that boths of these problem, *GOSMPR* and *GOMMPR* queries are NP-hard. Thus, we propose a dynamic programming approach based on best first strategy to solve the *GOSMPR* query problem optimally. We also present a variant of Held Karp lower bound for TSP [6] and a series of state elimination criteria to reduce the exponentially large solution space in this dynamic programming approach. To optimally solve the *GOMMPR* problem, we propose another dynamic programming based algorithm. It should be noted that the specialization of *GOMMPR* to *GOSMPR* is a possibility. However,*GOMMPR* query makes the optimal route planning a problem known as the set TSP, group TSP or covering TSP. It is also equivalent to the group Steiner Tree problem, which is significantly harder than the TSP and is known to be at least as hard as Set Cover, and therefore hard to even approximate.

One of the major challenges of route planning in ride sharing is the limited flexibility in drivers' itineraries and passengers' satisfaction levels or commonly referred as constraints. A study done by Niels A.H.Agatz et al. [7] depicts that, depending on the different settings related to drivers' itineraries and passengers' satisfaction levels, approximately 15-40% of passengers and drivers remain unmatched. Recent studies that focus on addressing such challenges have identified the introduction of meeting points as a potential solution [8] [9] to mitigate such deficiencies. Introduction of meeting points will enable smaller detours for passengers while also maintaining satisfactory level of service. This also allows drivers to serve more passengers with a reasonable change in total travel miles driven and thereby improving their profit margins. With the meeting points, a passenger can be picked up *{w.l.o.g.dropped off}* from a selected meeting point instead of picking *{dropping}* it from its actual origin *{destination}*. This may allow the ride sharing services to match more passengers with the same fleet of taxis.

For an example, in Fig 1.1, we demonstrate the use of meeting points to increase the service rates. Assume that each passenger has a maximum waiting

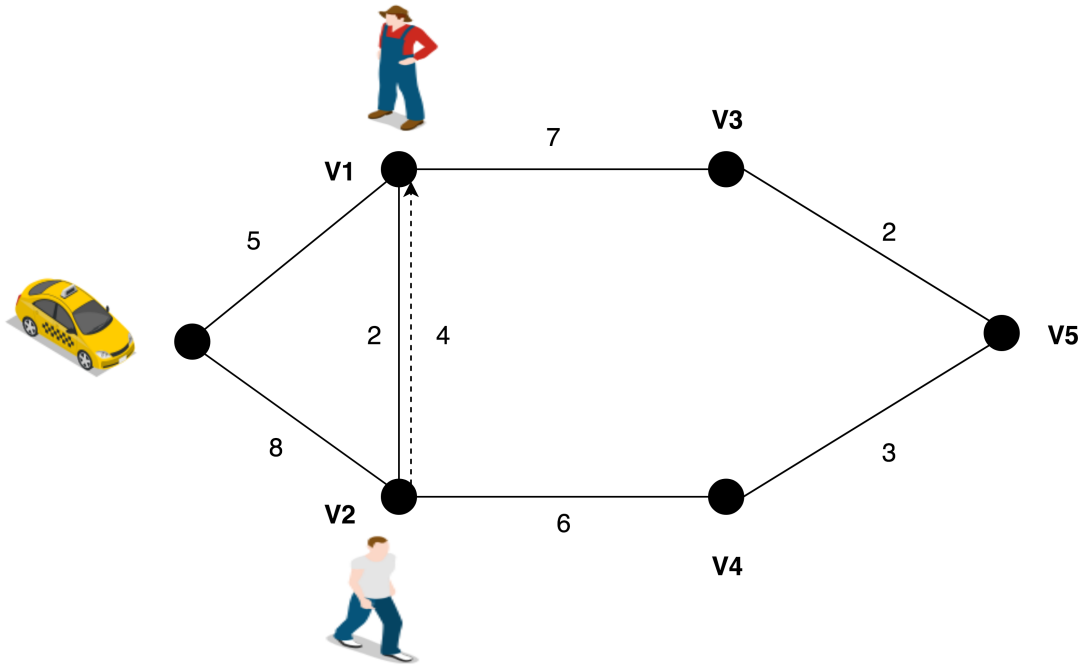

Figure 1.1: Example of the use of meeting points for better service rates

time of 5 minutes each. Each passenger's destination is $v5$. Therefore, there is only one option for the vehicle. That is to pick the passenger at $v1$ and drive it to its destination through the path $v1, v3, v5$. However, if the passenger at $v2$ is willing to walk to meeting point $v1$ by taking a *4 minutes* walk, the vehicle can serve both passengers without violating the waiting time constraints of both of them. Thereby the service rate of the system improves from 50% to 100%.

In the context of this work, we also look into the experimental impact of meeting points in ride sharing. We demonstrate that introduction of meeting points into existing ride sharing models can improve the service rate by 1.53%.

1.1 Problem Definition

Formally, we define both *GOSMPR* and *GOMMPR* on an undirected metric graph $G = (V, E, W)$ where the triple (V, E, W) are the node set, edge set and weight set respectively. Apart from that we use constraints set C to define the customer and driver related satisfaction levels, generally known as constraints.

Node Set : For a given initial location s of the taxi, let R be the set of on-

board passengers while P be the set of passengers who are assigned to it but yet to pick up. Let $|P| = n_1$ and $|R| = n_2$. We define the node set V as per the Equation 1.1.

$$V = \left(\bigcup_{i \in R} U_i^o \cup U_i^d \right) \cup \left(\bigcup_{j \in P} U_j^d \right) \cup s \tag{1.1}$$

Here, U_i^o and U_i^d denotes the super sets that contains $O(k)$ number of pre-defined pickup points and drop off points of the passenger i respectively. Note that passengers in set P only contains a destination as they have been already picked up by the vehicle. Thus, $|V| = n_1 + 2n_2 + 1$. Let U_i be a common definition for either U_i^o or U_i^d. Each meeting point $u \in U_i$ has a weight w_u which denotes the cost (time to walk, distance to walk) to {from} the meeting point from {to} the respective passenger's actual origin {destination}. In this context, we call any set U_i as a super vertex (a cluster of meeting points corresponding to the same origin or destination point) and our node set consists $n_1 + 2n_2$ of such super vertices.

Edge Set : Edge set contains edges which connect any v_i with any v_j where $v_i, v_j \in V$. Note that there is no need for edges between vehicle initial location s and drop off points corresponding to passengers' actual destinations as a vehicle cannot directly drop a passenger before picking it up. In the same way, we can omit the edges within the same super vertex U_i as visiting one node from each super vertex is sufficient.

Weight Set : The weight $w(v_i, v_j)$ of an edge (v_i, v_j) denotes pre-computed shortest travel time or distance from $v_i \in V$ to $v_j \in V$ or vice verse as the graph is undirected.

Constraint Set : We formulate our algorithms with the flexibility to add different constraints which are imposed by the physical setup, related to performance or customer satisfaction. In the following section, we describe the constraints used in this context.

- *Precedence* : For each request $r \in R$, its' origin org should be visited before its' destination des.

- *Capacity* : For the vehicle, at any given time, the maximum number of passengers on-board can not be higher than the maximum capacity $\mathcal{C}$ of the vehicle.

- *Maximum waiting time* : For each request $r \in R$, waiting time t_{wait} computed as the difference between pickup time and request time should be less than or equal to maximum waiting time T_{wait}.

- *Maximum travel delay* : For each request $r \in R$, the amount of travel delay t_{delay} (in vehicle delay due to detour) a passenger is willing to tolerate when sharing a vehicle with set of another passengers cannot exceed T_{delay}.

Let σ_s be a route starting at node s and visit each super vertex U_i exactly once. That means, σ_s consists of exactly one meeting point $v_i \in U_i$ from each super vertex. Thus, $\sigma_s = \{s, v_1, v_2, ..., v_t\}$ where $t = |V| - 1$ and σ_s is a $s \sim v_t$ path where v_t can be any $v \in V \backslash \{s\}$. We denote $C(\sigma_s)$ to be the cost of path σ_s and define as per the Equation 1.2.

$$c(\sigma_s) = \sum_{v_i, v_{i+1} \in \sigma_s} w(v_i, v_{i+1}) \tag{1.2}$$

Let $U^{'}$ be such set that contains all selected meeting points in all super vertices in path σ_s. We denote the sum of the cost of reaching to pickup points in $U^{'}$ from their corresponding origins and reaching to the actual destination points from the drop off points in $U^{'}$ as $c(\omega_s)$ and define as per the Equation 1.3.

$$c(\omega_s) = \sum_{u \in U^{'}} w_u \tag{1.3}$$

We define the weighted cost function $f(\sigma_s)$ for route σ_s with given G, s, C and α as,

$$f(\sigma_s) = \alpha \times c(\sigma_s) + (1 - \alpha) \times c(\omega_s) \tag{1.4}$$

Here, C denotes an arbitrary set of constraints where as $\alpha \in (0, 1)$ is a weight

parameter used to trade off between the $s \sim v_t$ path cost and the cost to reach to path σ_s through a selected set of meeting points for all passengers.

Given the graph G, constraints set C, parameter α, and the vehicle location s, we define Z as the optimization goal for both *GOSMPR* and *GOMMPR* queries such that $Z = min\{f(\sigma_s)\}$ for $\sigma_s \in \Sigma_s$ where Σ_s denotes the set of all $s \sim v_t$ routes which the constraints set is satisfied. It is obvious that in *GOSMPR*, $c(\omega_s) = 0$ as it does not include meeting points. However in general, both *GOMMPR* and *GOSMPR* aim to find a route ${\sigma_s}^* \in \Sigma_s$ such that Z is minimized and adheres to the imposed set of constraints. In below sections, we formally define both *GOSMPR* and *GOMMPR* query problems.

Definition 1.1.1. Given an undirecetd metric graph $G = (V, E, W)$, vehicle current location s, and constraint set C, the *Generalized Optimal Single Meeting Point Route* (*GOSMPR*) query $Q_{GOSMPR} = (G, C, s)$ aims at finding the ${\sigma_s}^*$, the optimal $s \sim v_t$ path, where v_t is any $v \in V \backslash \{s\}$ and that ${\sigma_s}^*$ adheres to all the constraints while minimizing the cost of the path as

$$c({\sigma_s}^*) = \min_{\sigma_s \in \Sigma_s} \{c(\sigma_s)\} \tag{1.5}$$

Definition 1.1.2. Given an undirecetd metric graph $G = (V, E, W)$, vehicle current location s, parameter α, meeting points set M, and constraint set C, the *GOMMPR* query $Q_{GOMMPR} = (G, C, M, s, \alpha)$ aims at finding the ${\sigma_s}^*$, the optimal $s \sim v_t$ path, where v_t is any $v \in V \backslash \{s\}$ and that ${\sigma_s}^*$ adheres to all the constraints while minimizing the weighted cost as

$$f(\sigma_s*) = \min_{\sigma_s \in \Sigma_s} \{f(\sigma_s)\} \tag{1.6}$$

1.2 Problem Analysis

Hardness of *GOSMPR* and *GOMMPR* : As stated before, we can easily specialize the *GOSMPR* query to *GOMMPR* query. Thus, in this section we analyze the hardness of the $GOMMPR$ query which is applicable to *GOSMPR* query as well.

Finding the optimal solution to *Generalized Optimal Multi Meeting Point Route* query problem with large number of passengers through the direct search is impractical due to exponentially large solution space induced. Here, we prove this by reducing the *GOMMPR* query to path variant of classical Traveling Salesman Problem (path TSP) [10].

Theorem 1.2.1. Computing the optimal solution for *Generalized Optimal Multi Meeting Point Route(GOSMPR)* query problem is NP-hard.

Proof. We show that it is possible to reduce *GOMMPR* query to path variant of Travelling Salesman Problem (TSP) which is known to be NP-hard [11], [12]. Rong-Hua Li et al. [8] have proved the hardness of *Optimal Multi-Meeting Point Route (OMMRP)* query through the use of the same reduction technique. Clearly, our weighted cost function has the same common representation as [8] and therefore can adopt the same proof. Accordingly, given any $s \sim v_t$ path TSP query $\bar{Q} = (G, s)$, it is equivalent to the *GOMMPR* query $Q = (G, C, M, s, \alpha = 1/3)$. We refer the reader to as [8] for the complete proof. □

However, it should be noted that the introduction of meeting points makes the optimal route planning (*GOSMPR*) a problem known as the generalized TSP [13], set TSP or covering TSP [14]. It is also equivalent to the group Steiner Tree problem [15], which is significantly harder than the TSP and is known to be at least as hard as Set Cover, and therefore hard to even approximate [16].

1.3 Proposed Solution

In this work, we propose two dynamic programming algorithms to solve *GOSMPR* query and *GOMMPR* query optimally. To solve the *GOSMPR* query problem we propose an algorithm which is inspired by A^* algorithm. Having the same nature as A^* algorithm, to reduce the search state space, we propose a series of state elimination criteria and a novel variant of classical *Held Karp* lower bound of Travelling Salesman Problem (TSP) as a heuristic. Also to address the *GOMMPR* query problem, we propose another dynamic programming based

algorithm. We demonstrate that our proposed algorithm for *GOSMPR* query and algorithm for *GOMMPR* query can solve the required number of passenger requests within acceptable time limits.

Later, we also demonstrate the impact of introducing meeting points into the existing ride sharing systems to improve their performances.

1.4 Contributions

We present three major contributions through this work.

- We introduce a novel dynamic programming algorithm for *Generalized Optimal Single Meeting Point Route* query based on best first strategy. The proposed algorithm is flexible in having arbitrary set of constraints correspond to customer and service provider's satisfaction perspectives.
- We introduce a novel dynamic programming algorithm for *Generalized Optimal Multi Meeting Point Route* query. The proposed algorithm can accommodate multiple pick up and drop off points in optimal route planning while being flexible in having arbitrary set of constraints correspond to customer and service provider's satisfaction perspectives.
- We demonstrate the use of meeting points in ride sharing systems to improve the current state of the art ride sharing models to obtain better performance.

1.5 Organization

The rest of this paper is organized as follows: In Section 2 review previous studies related to this work. The details of proposed algorithms for *Generalized Optimal Single Meeting Point Route* query and *Generalized Optimal Multi Meeting Point Route* query is introduced in Section 3. In Section 4, we present the results of our work. We conclude and discuss the future direction of the study in Section 5.

Chapter 2

LITERATURE SURVEY

In this section we intend to look into the previous related work of our work presented. We discuss the related work in three major categories.

2.1 Optimal Routing in Ride Sharing

Rong-Hua Li et al. [8] have presented *Optimal Multi-Meeting Point Route OMMPR* query which we identified as a specialization of our generic problem. Given an underlying road network G, a starting source node s, a ending destination node t, and a set of intermediate query nodes U, the *OMMPR* query is to find the best route such that it starts from the s and end at t while the weighted average cost between the cost of the route and the total cost of the shortest paths from every query node to the route is minimized. Rong-Hua Li et al. have presented a dynamic programming approach along with some lower bounding techniques to address the problem. However, *OMMPR* query only deals with selecting meeting points from the entire road network. Thus, they do not intend to introduce constraints in route planning. Also their work limits to route planning to pick up points only and does not accommodate drop off points making the precedence constraint out of the picture. Therefore, Rong-Hua Li et al. are addressing a comparatively smaller problem than *GOMMPR* query problem.

Optimal Sequenced Route (OSR) query in road networks, was proposed by M. Sharifzadeh et al. [17] and a variant of it called *Trip Planning Query (TPQ)* was proposed by F. Li et al. [18]. These studies were later generalized in [19] [20] [21]. In *OSR*, the best route is found such that it starts from the source and pass the typed nodes in a specified sequence based on types of the nodes and then ending at a target node such that the distance is minimized. They propose a light threshold- based iterative algorithm, that utilizes various thresholds to filter out the locations that cannot be in the optimal route. However, *OSR* query

differs with ours in the sense, in *OSR* query, the optimal route must pass through the specific typed nodes, whereas for the *GOSMPR* query and *GOMMPR*, the optimal route does not necessarily need to pass through set of specific nodes.

Apart from that, Jacques Desrosiers et al. [22] have introduced a dynamic programming approach to the large scale dial a ride problem with time windows for a single vehicle. This problem is a specialization of the *GOSMPR* such that accommodates given specific set of constraints. Their proposed forward dynamic programming approach and the development of state elimination criteria has resulted in solution times which increases linearly with the problem size. Developing on this, Steffan Ropkeet et al. [23] have introduced a Branch and Cut algorithms for pickup and delivery problems with time windows. While the pick up and delivery problem shares some common insights with *GOSMPR* query problem in general, in nature, the main difference comes with the fact that pick up and delivery problem concerns with a fleet of taxis starting and ending at the same deport while *GOSMPR* query problem deals with one vehicle which has a current location and no desired end point. However, their branch and cut algorithm is performing up to the expectations when considered the complexity of the problem they solve.

2.2 Held Karp Lower Bound for TSP

Given a set of cities and distance between every pair of cities, *Travelling Salesman Problem (TSP)* deals with finding the shortest possible route that visits every city exactly once and returns to the starting point [24]. This problem has been studied for decades now and is still in the peak of attention. TSP problem belongs to the category of *NP-hard* and thus researchers have identified lower bounds that can be used to extend to compute the exact TSP tour. Out of many lower bounds for TSP, the solution to the linear program (LP) of the integer programming formulation of TSP is known as *Held-Karp lower bound (HK)*. However evaluating HK for problems larger than a few hundred cities through the use of LP code, is either not readily available or easy to produce.

Also Linear Programming implementations do not scale well and rapidly become impractical for problems with many thousands of cities [25]. Thus *Held-Karp lower bound (HK)* is calculated based on graph modification algorithm which makes use of the concept of $1-tree$.

A $1-tree$ on an n city problem is defined as follows:

- A $1-tree$ is a connected graph with vertices $1, 2, ..., n$ consisting of a tree on the vertices $2, 3, ..., n$ together with two edges incident with city 1 [25].

The Held Karp lower bound is evaluated as a $1-tree$ relaxation where it involves computing sequence of *Minimum one-trees* where *Minimum one-trees* is defined as,

- A *Minimum 1-tree* is a *Minimum Spanning Tree (MST)* on the vertices $2, 3, ..., n$ together with the two lowest cost edges incident with city 1. [25].

In this study, we make use of the Held Karp lower bound proposed by [6] in arriving at a lower bound to reduce the exponentially large solution space of our algorithm for *GOSMPR* query problem.

2.3 Meeting Points for Ride Sharing

With the growing popularity of the ride sharing services, the focus is highly pointed on improving the existing models to achieve better performance. The introduction of meeting points is one of the leading research focuses which is leading this efforts. As pointed out by the Mitja Stiglic et al. [9] introduction of meeting points provides additional flexibility to match drivers with the passengers.

Alonso Mora et al. [1] have introduced an anytime optimal algorithm for ride sharing without meeting points. Their proposed method is explained in details in the Section 3 as we will be improving it to accommodate the meeting points. Also, Masayo Ota et al. [2] have introduced a greedy heuristic algorithm for real time ride sharing analysis. Their proposed algorithm varies to from the model of [1] such that it is light weight in execution as the heuristic nature. However,

the ride sharing model presented by [1] is producing an optimal assignment vs sub optimal solution in [2].

Xin Li et al. [26] address the problem of ride sharing with meeting points using a tabu based meta heuristic algorithm which they use to solve the proposed mixed integer linear program (MILP). They also validate the concept effectiveness of the meeting points by showing the that meeting points in small scale ride sharing systems can save the total travel time by 2.7%-3.8%. However, the impact of meeting points in large scale ride sharing environments still remain as an open ended question.

Ulrich Matchi Aivodji et al. [27] presents a privacy-preserving service to compute meeting points in ridesharing. Their proposed method ensures that each user remains in control of his location data. However, the decentralized architecture that provides strong security and privacy guarantees without sacrificing the usability, does not specifically address the impact of meeting points or the optimal techniques to accommodate meeting points in large scale ride sharing. Paul Czioska et al. [28] discuss about five simple optimization methods to identify reasonable meeting point locations on a real street network. They analyse and discuss the five methods, Geo-Radius (GR), Single Gradient Descent (SGD), Triple Gradient Descent (TGD), Catch-Up Zone (CUZ) and Intersection of Space-Time-Prisms (STP). However, all of the five methods described are to pick a suitable pick up or drop off point for a passenger given a vehicle and do not scale in to the global level optimums or contribute to optimal route selection.

Chapter 3

METHODOLOGY

We propose two Dynamic Programming (DP) algorithms to solve the *GOSMPR* query and *GOMMPR* query problems optimally. In below sections, we discuss the two algorithms and certain improvement criteria for them. Also in the later part of this section, we also discuss the technique we adopted in introducing meeting points to existing ride sharing systems to improve their performance.

3.1 The Algorithm for *GOSMPR* Query

As stated earlier, *GOSMPR* query is a specialization of the *GOMMPR* query. Thus, when each super vertex has one meeting point (origin or the destination itself) we arrive at the settings for *GOSMPR* query. In the below sections, we refer to such setting for *GOSMPR* query. Let (v, X) be a common representation of a state in the DP algorithm where v denotes the current super vertex being explored and X denotes the set of super vertices that have already been explored. Thus, $X^{'} = (V - s) - X$ is the set of unexplored super vertices. Accordingly, we define $f(v, X)$ as the weighted cost function of the path σ_{sv} which starts at s and ends at v traversing each $x \in X$ exactly once. As per the Equation 1.6, $f(v, X)$ can be defined as

$$f(v, X) = c(\sigma_{sv}) \tag{3.1}$$

We present the basic dynamic programming algorithm based on best first strategy as per the Algorithm 1. We do not employ any bounding techniques here thus it is called *Unbounded DP* algorithm. Algorithm 1 maintains a priority queue Q such that each element in the queue is a tuple in the form $((v, X), cost)$ where $cost = f(v, X)$ as defined previously. Also, the *cost* acts as the priority for best first strategy. Additionally, the algorithm maintains a set D to record the

visited states of the algorithm.

The algorithm starts by initializing Q and D to empty sets. The initial state of the Q is $((s,\ \emptyset), 0)$ where the vehicle is at its' current location s and no super vertices has been explored yet. Thus, $((s,\ \emptyset), 0)$ is pushed to the Q in the beginning (line 2). Then, an element is repeatedly popped from Q while Q is not empty. In line 5, the termination condition is checked. The algorithm should terminate, if the set X contains all the super vertices in G. Lines 8-11 deal with expanding the solution by exploring from the current best solution through the adjacent vertices to the currently visited super vertex. In line 9, *IsConstraintsSatisfied* function checks if the expanding solution violates any of the constraints in the constraints set C. Similarly, *IsStateEliminated* function check if we can eliminate the state as per the criteria presented in Section 3.1.2. If the constraints are satisfied and the state is not subjected to elimination, algorithm executes the *Update* procedure to record the legal expansion.

Note that Algorithm 1 can easily be expanded to optimal solution and can accommodate any constraint that need to be imposed in deciding the optimal route. *Unbounded DP* can easily be extended to return the exact path by maintaining an array of the visited meeting points without loosing the order.

Algorithm 1: Unbounded DP for *GOSMPR* Query

Data: $G = (V, E, W)$, source node s, constraint set C

Result: The optimal path cost

$Q \leftarrow \emptyset$; $D \leftarrow \emptyset$;

$Q.push\ ((s, \emptyset), 0)$;

while $Q \neq \emptyset$ **do**

    $((u, X), cost) \leftarrow Q.pop()$;

    if $X = V \backslash \{s\}$ **then**

        return $cost$;

    $D \leftarrow D \cup \{(u, X)\}$;

    foreach $(u, v) \in E$ **do**

        if $v \notin X$ **then**

            if $IsStateEliminated(v, X)$ *and* $IsConstraintsSatisfied\ (v, X, C)$ **then**

                $cost^{'} = cost + w(u, v)$;

                UPDATE$(Q, D, (v, X \cup v), cost^{'})$;

return ∞

Function *update* $(Q, D, (v, X), cost)$

    if $(v, X) \in D$ **then**

        return

    end

    if $(v, X) \notin Q$ **then**

        return $Q.push((v, X), cost)$

    end

    if $cost < Q.cost((v, X))$ **then**

        $Q.update((v, X), cost)$

    end

end

3.1.1 Solution Optimization

As mentioned previously, when the number of assigned passengers increase, solving *GOSMPR* query problem optimally becomes impractical due to exponentially large solution space. Thus to reduce the solution space, we define a heuristic to guide the search process towards the optimal solution and a series of state elimination criteria to eliminate redundant states. In this section, we first present the state elimination criteria we adopted and then the meticulously designed lower bounding technique. Later we introduce our new algorithm called *Bounded DP* which makes use of the heuristic.

3.1.2 State Elimination Criteria

In here, we introduce the state elimination criteria that we are incorporating in the proposed algorithm. Some of these criteria are inspired by the the work presented by Jacques Desrosiers et al. [22].

It should also be noted that each constraint in set C can also be considered as a state elimination criteria. Thus, in below, we present the criteria which were not covered by them.

- Let t_i be the time that vertex v_i is visited. Then we eliminate each adjacent state created by visiting the adjacent edges to v_i if the the time to visit each of those adjacent vertex v_j as denoted by $t_i + w(v_i, v_j)$ is violating the time window constraints for v_j.

- Let t_i be the time that vertex v_i is visited. Given t_i, it should be possible to visit each pair of destinations (d_j, d_k) where $d_j, d_k \in \mathrm{X}^{'}$ and whose origins are in set X such that order of visiting them is either v_i, d_j, d_k or v_i, d_k, d_j in the path σ_s and no time window constraints is violated for both of the destinations.

- Let t_i be the time that vertex v_i is visited. Given t_i, it should be possible to visit each pair of origins (o_j, o_k) where $o_j, o_k \in X^{'}$ such that order of

visiting them is either v_i, o_j, o_k or v_i, o_k, o_j in the path σ_s and no time window constraints is violated for both of the origins.

3.1.3 The Bounded DP Algorithm

The *Bounded DP* algorithm employs a variant of the weighted cost function of the path σ_{sv} as per the Equation 3.2.

$$f(v, X) = c(\sigma_{sv}) + h(v, X^{'}) \tag{3.2}$$

Here, $h(v, X^{'})$ is the cost of the lower bound for the route starting at v and traverse all the super vertices in $X^{'}$ and terminates at any $v_t \in X^{'}$. We derive a variant of Held Karp lower bound [25] of Travelling Salesman Problem (TSP) as the heuristic function $h(v, X^{'})$. We introduce a graph transformation which subsequently will be used in calculating $h(v, X^{'})$ according to the derived variant of Held Karp lower bound.

3.1.4 Graph Transformation

Given graph $G = (V, E, W)$ as explained in Section 1.1 which satisfy metric property, we create a new graph $G^{'} = (V^{'}, E^{'}, W^{'})$ by performing a graph transformation using following three steps. The specified graph transformation preserves the metric property of the new graph $G^{'}$.

1. Convert the vertex set V to $V^{'}$ in the new graph $G^{'}$ by introducing a hypothetical vertex d to G.

2. Introduce edges $(v, d) \backslash \{(s, d)\}$ in to $G^{'}$ where $v \in V \backslash \{s\}$ with
   $w(v, d) = \max_{u_i, u_j \in E} \{w(u_i, u_j)\}$.

3. Introduce edge (d, s) in to $G^{'}$ with
   $w(d, s) = \max_{u_i, u_j \in E} \{w(u_i, u_j)\} - \min_{u_i, u_j \in E} \{w(u_i, u_j)\}$.

Theorem 3.1.1. The optimal $s \sim v_t$ path σ_s^* in G is a sub path contained in the optimal tour T_s^* in $G^{'}$ such that edge (d, s) is included in T_s^*. Thus,

$c(\sigma_s^*) = c(T_s^*) - M$ where $M = 2 \times \max_{u_i,u_j \in E}\{w(u_i,u_j)\} - \min_{u_i,u_j \in E}\{w(u_i,u_j)\}$.

The proof for Theorem 4.1 is fairly straightforward and thus we omit the details.

Theorem 3.1.2. In $G^{'}$, the minimum 1-tree obtained by joining vertex d with the two edges (d,s) and any (v_i,d), into the minimum spanning tree of the rest of the vertices, is a lower bound for the optimal tour T_v^* of $G^{'}$ such that edge (d,v) is included in it.

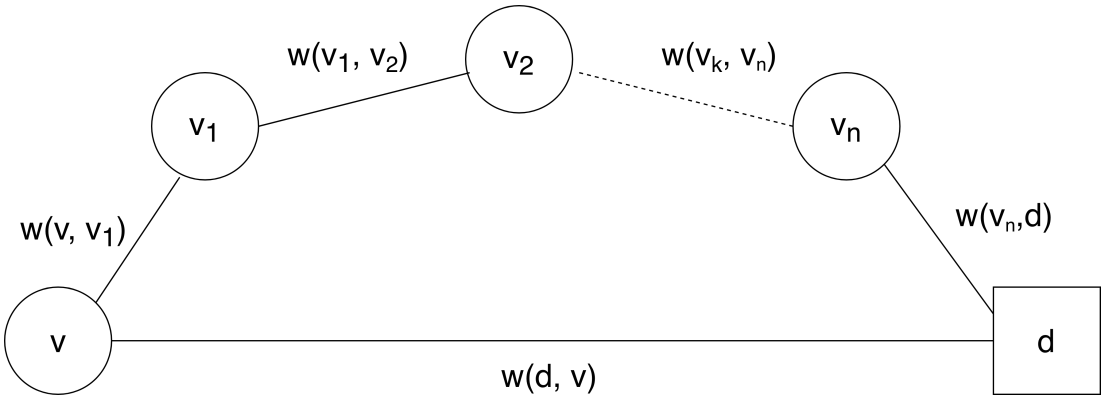

Figure 3.1: Optimal Path from v to v_n with two additional edges connecting to d

Proof. In Figure 3.1, we denote a specific instance, the optimal tour T_v^* in $G^{'}$ such that edge (d,v) is in T_v^*. Without the loss of generality, let $w(v,v_1) + \sum_{i=1}^{n-1} w(v_i,v_{i+1}) = C^*$ as per the Figure 3.1. As can be seen, C^* is the cost of the optimal path $v \sim v_n$. Similarly, let the cost of minimum spanning tree of the vertices $\{v,v_1,v_2,...v_n\}$ be C_{MST}. It is known that minimum spanning tree is a lower bound for the path TSP problem [29]. Therefore, $C_{MST} \leqslant C^*$. By adding the constant C to the both side of the inequality, we can derive Inequality 3.3.

□

$$C_{MST} + C \leqslant C^* + C \tag{3.3}$$

$$where\ C = 2 \times \max_{u_i,u_j \in E}\{w(u_i,u_j)\} - \min_{u_i,u_j \in E}\{w(u_i,u_j)\} \tag{3.4}$$

It is apparent that, from the left hand side of the Inequality 3.3, $C_{MST} + C$ is the cost of minimum 1-tree with vertex d is having exact degree of 2. On the other hand, we know, from the right hand side of the Inequality 3.3, $C^* + C$ is the cost of the optimal tour T_v^* such that edge (d, v) is in it. Therefore, it is clear that in $G^{'}$, the minimum 1-tree obtained by joining vertex d with the two edges (d, v) and any (v_i, d), into the minimum spanning tree of the rest of the vertices, is a lower bound for the optimal tour T_v^* of $G^{'}$ such that edge (d, v) is included in it.

Similarly, the cost of optimal tour T_v^* of $G^{'}$ such that edge (d, v) is included in it is C amount higher than the optimal $v \sim v_t$ path σ_v^* in G. Note that this constant C is a global level constant and will not be changed within states in the DP algorithm. Thus, in any given state (v, X), in finding the lower bound for the route starting at v and traverse all the super vertices in $X^{'}$ and terminates at any $v_t \in X^{'}$, we can use the stated theorems.

We re-define the goal state for the heuristic as, given any given state (v, X), we are interested in finding a lower bound for the route starting at v and traverse all the super vertices in $X^{'}$ and terminates at v such that edge (d, v) is included in its tour. That is, we add an additional cost C to our previous goal state. Since the constant C is a global level constant and will not be changed within states in the DP algorithm, this allows us to use Theorem 3.1.2 in the initial lower bound calculation and re-fine it over iterations.

In summary, we perform following steps in order to achieve the initial lower bound.

1. In state (v, X), create a graph $G_{sub} = (V_{sub}, E_{sub}, W_{sub})$ from the graph G such that, new $V_{sub} = X^{'}$, new E_{sub} and W_{sub} contain all the edges in E and the weights of them in W for each two vertices v_i and v_j, where $v_i, v_j \in V_{sub}$.

2. Convert the G_{sub} to $G_{sub}^{'}$ by using graph transformation illustrated in Section 3.1.4.

3. Find the minimum 1-tree by joining vertex d with the two edges (d, v) and

any (v_i, d), into the minimum spanning tree of the rest of the vertices in G'_{sub}

In next section, we present a variant of the 1-tree relaxation originally presented by Michael Held et al. [6] to refine the obtained initial lower bound.

3.1.5 Variant of Held Karp Lower Bound

We attach an arbitrary weight to every super vertex. Let π_{v_i} be the arbitrary weight of the the vertex v_i. Once, initial minimum 1-tree is calculated as previously explained, we now modify the edge costs of G'_{sub} according to the transformation presented in Equation 3.5. Subsequently with the new edges costs, we re-compute a second minimum 1-tree.

$$\beta_{v_i v_j} = \alpha_{v_i v_j} + \pi_{v_i} + \pi_{v_j} \tag{3.5}$$

Here, $\beta_{v_i v_j}$ and $\alpha_{v_i v_j}$ denote the new and old edge costs of the edge $(v_i, v_j) \in E'_{sub}$. In general, the second minimum 1-tree differs from the the initial minimum 1-tree but note that the order of tour costs according to this transformation remain unaffected. It's because the mentioned transformation of edge costs adds $2\sum \pi_{v_i}$ to every tour length upon transformation (in a tour, each vertex has a degree of exactly 2).

Let $\mathcal{C}(c_{v_i v_j}, \tau)$ be the cost of a tour or 1-tree τ with reference to the edge lengths set $c_{v_i v_j}$.

Let Q to be the set of 1-trees obtained by joining vertex d with the two edges (d, s) and any of (v_i, d), into a spanning tree of the rest of the vertices. Let T be the set of tours such that edge (d, s) is included in each of those tours. Since a tour can be defined as a 1-tree such that each vertex in it has a degree of 2, we know that $T \subseteq Q$.

$$\min_{\tau \in Q} \mathcal{C}(c_{v_i v_j}, \tau) \leqslant \min_{\tau \in T} \mathcal{C}(c_{v_i v_j}, \tau) \tag{3.6}$$

Let $d^{\tau}_{v_i}$ be the degree of vertex v_i in the 1-tree or tour τ. Also, let the set of

vertices in the tour or 1 -tree τ to be $\bar{V}$. If $\mathcal{C}(e_{v_iv_j}, \tau)$ is the cost of the original 1-tree or tour τ with reference to the edge costs set $e_{v_iv_j}$. As per the Equation 3.5, we can write,

$$\mathcal{C}(c_{v_iv_j}, \tau) = \mathcal{C}(e_{v_iv_j}, \tau) + \sum_{v_i \in \bar{V}} \pi_{v_i} d^{\tau}_{v_i} \tag{3.7}$$

If τ is a tour, then we know that each vertex in τ must have degree of 2. Therefore we can re-write Equation 3.7 as,

$$\mathcal{C}(c_{v_iv_j}, \tau) = \mathcal{C}(e_{v_iv_j}, \tau) + \sum_{v_i \in \bar{V}} \pi_{v_i} 2 \tag{3.8}$$

Suppose that the right hand side of Equation 3.6 is attained for a minimal cost tour τ^*,

$$\min_{\tau \in Q} \mathcal{C}(c_{v_iv_j}, \tau) \leqslant \mathcal{C}(e_{v_iv_j}, \tau^*) + \sum_{v_i \in \bar{V}} \pi_{v_i} 2 \tag{3.9}$$

Let $c^* = \mathcal{C}(e_{ij}, \tau^*)$ be the optimal tour length of the original problem, and $c_\tau = \mathcal{C}(e_{ij}, \tau)$ be the cost of a original 1-tree with respect to the edge cost set e_{ij}. Thus, for any π,

$$\min_{\tau \in Q} \left\{ c_\tau + \sum_{v_i \in \bar{V}} \pi_{v_i} d^{\tau}_{v_i} \right\} \leqslant c^* + \sum_{v_i \in \bar{V}} 2\pi_{v_i} \tag{3.10}$$

Therefore, for any π,

$$\min_{\tau \in Q} \left\{ c_\tau + \sum_{v_i \in \bar{V}} \pi_{v_i} (d^{\tau}_{v_i} - 2) \right\} \leqslant c^* \tag{3.11}$$

If we define $f(\pi)$ as,

$$f(\pi) = \min_{\tau \in Q} \left\{ c_\tau + \sum_{v_i \in \bar{V}} \pi_{v_i} (d^{\tau}_{v_i} - 2) \right\} \tag{3.12}$$

Then each $f(\pi)$ clearly serve as a lower bound for c^* and therefore we can define the variant of Held-Karp lower bound as,

$$L.B. = \max_{\pi} f(\pi) \tag{3.13}$$

Suppose the minimum in Equation 3.11 is attained at a 1-tree $\tau = \tau(\pi)$. Let $v_\tau = (d_1^\tau - 2, ..., d_n^\tau - 2)$ then,

$$f(\pi) = c_{\tau(\pi)} + \pi.v_{\tau(\pi)} \tag{3.14}$$

Michael Held et al. [30] have presented a *subgradient method* to address the problems of the form Equation 3.13 and Equation 3.14. Given initial π^0 and the sequence $\{t_j\}$ of positive scalars, called step sizes, they define the sequence $\{\pi^j\}$ by,

$$\pi^{j+1} = \pi^j + t_j.v(\pi^j) \; for \; j = 0, 1, 2.... \tag{3.15}$$

Also, they highlight two mere conditions for $f(\pi)$ to converge to its maximum $f(\pi)^*$ as,

$$when \quad t^j \to 0, \quad \sum_{j=0}^{\infty} t_j = \infty \tag{3.16}$$

using the choice,

$$t^j = \lambda_j \frac{\hat{f} - f(\pi^j)}{|v(\pi^j)|^2} \tag{3.17}$$

Here, $\epsilon < \lambda_j \leqslant 2$ for some fixed $\epsilon > 0$ and $\hat{f} < max(f(\pi))$.

Note that we have defined a variant of Held Karp lower bound by taking the minimum 1-trees obtained by joining vertex d with the two edges (d, v) and any of (v_i, d), into the minimum spanning tree of the rest of the vertices. This is a lower bound for the optimal tour such that edge (d, v) is included in it. Using the graph transformation explained in 3.1.4 and the derived variant of Held Karp lower bound, we now can find the lower bound for the route starting at v and traverse all the super vertices in $X^{'}$ and terminates at t which is any $x \in X^{'}$ from any given state $(v, X^{'})$.

3.2 The Algorithm for *GOMMPR* Query

In this section we introduce the dynamic programming approach for *GOMMPR* query problem. We define a *route* as a fixed ordered series of origins and destinations that a vehicle has to traverse over time. Any origin or a destination which is included in a route is one of the waypoints of that route. Let $\mathcal{R} = \{\sigma_1, ...\sigma_b\}$ be the set of all possible routes for the vehicle v with set of on board passenger P_v and set of assigned requests R_v. First, we obtain the valid route set $\mathcal{R}^{'} \subseteq \mathcal{R}$ by pruning the routes which violate the precedence and vehicle capacity constraints. This can be easily done by traversing through each of the route's way points. Let $v_{current}$ be the current location of the vehicle v and $W_\sigma = \{w_1, ..., w_h\}$ be the way points of the route σ. Each $w_i \in W_\sigma$ has a set of meeting points defined as $M_{w_i} = \{m_1, ...m_y\}$.

Next, for each valid route $\sigma_x \in \mathcal{R}^{'}$, we build a level graph $\bar{G} = (\bar{V}, \bar{E}, \bar{W})$ with the given fixed order of way points and the corresponding meeting points for each of the way point. Therefore, $\bar{V} = \bigcup_{i=1}^{h} M_{w_i} \cup v_{current} \cup l_{sink}$ where l_{sink} is a hypothetical location. Without loss of generality, we define the edges of the graph as $\bar{E}$, weights as $\bar{W}$ and discuss the derivation of them next.

In building the level graph, we first add the vehicle current location $v_{current}$ as the root (first layer) vertex (*root*). Subsequently, the second layer of the graph is created based on the set M_{w_1} of the way point $w_1 \in W_\sigma$. Each $m_j \in M_{w_1}$ is a vertex in the first layer of the graph. Next we do the same procedure to create each of the next $h-1$ layers. Finally we add a hypothetical location l_{sink} as the only vertex (*sink*) in the $(h+2)^{th}$ or the last layer. Accordingly, there are three types of edges in the level graph. An edge from *root* to each $m_j \in M_{w_1}$ is added and is denoted by $e(root, m_j)$. For any two adjacent layers from second layer to $(h+1)^{th}$ layer, we add $|M_{w_L}| \times |M_{w_{L+1}}|$ number of edges such that each $m_j \in M_{w_L}$ is connected to each $m_k \in M_{w_{L+1}}$ through an edge where $L \in \{1, .., h\}$. We define such edges as $e(m_j, m_k)$. Finally, we have edges from each $m_j \in M_{w_h}$ to *sink* and is denoted by $e(m_j, sink)$. All edge weights are corresponding to the shortest distance or travel time between the start and end of the edge except the

$e(m_j, sink)$ edges. For them, we add a weight of zero.

Next for each of the level graphs, we run Algorithm 2 to obtain the cost of the route. The Algorithm 2 returns the shortest distance or travel time to traverse all the layers of the level graph such that walking constraint, waiting time constraint or time window constraints are not violated for any of the new requests R_v and on board passengers P_v if such path exists and "invalid" otherwise. Let $D = \{d_{\sigma_1}, .., d_{\sigma_b}\}$ be the set of shortest distances {w.l.o.g. travel times} as returned by the Algorithm **??** for all routes in $\mathcal{R}^{'}$. Finally we take the minimum of D as the minimum distance.

In Algorithm 2, we present a Dynamic Programming algorithm to compute the optimal path from *root* to *sink* in $\bar{G}$. Algorithm 2 maintains a priority queue Q where each element in Q is a tuple of the form $(v, cost)$. Here, $v \in V^{'}$ and *cost* is the path cost from *root* to v and also act as the priority factor. Our queue has three main operations, *push, pop* and *update*. The *push* operation is used to enqueue elements to the queue. The *pop* operation dequeue the element which has the lowest *cost* value whereas *update* operation is used to update an element in the queue. Apart from that we also define two maps *dist* and *previous*. *previous* is used to store the predecessor of each of the $v \in \bar{V}$ in visiting v. Same way, we use *dist* to store the shortest distance to reach each $v \in \bar{V}$ without violating constraints. We define a function *weightOf* which takes an edge $e(u, v)$ as input and returns the weight of the edge. We also define the function *checkConstraints* which takes a vertex v and the cost of the path so far as inputs. It return two outputs, (1) status : whether the current path cost does not violate any of the imposed constraints and (2) the modified path cost to accommodate vehicle waiting time at the vertex v for the passenger if no other constraint is violated and the passenger has to spend more time to get to the meeting point. For an example, cost of the path so far to the vertex v is c and the vehicle has to wait t amount of time at v for the passenger to get to the point, *checkConstraints* returns $\bar{c}$ as the new modified path cost where $\bar{c} = c + t$. Note that if the vehicle does not have to wait for the passenger, then $t = 0$ and thus $\bar{c} = c$.

The algorithm first initialize the *Q, dist* and *previous* (line 1-6). Next, it

adds the *root* node to the Q with a path cost of zero length (line 7). Then while the Q is not empty, we *pop* elements from the Q and expand from each element $(u, cost)$ by visiting the adjacent vertices of it. For each of the new visit to an adjacent vertex v, we check the constraints (line 12) and if they are satisfied and the modified cost $\bar{c}$ is less than the so far known shortest path to v as given by $dist[v]$, we update the Q and the two maps (line 13-21). Finally once the Q is empty we check if the $dist[sink]$ has been set to value other than ∞. If it is set, then we return the value or otherwise "invalid"(line 24-28).

Algorithm 2: Route Finding algorithm for *GOMMPR* query

Data: $\bar{G} = (\bar{V}, \bar{E}, \bar{W})$, root, sink

Result: Optimal path cost if exists or "Invalid" otherwise

```
1  foreach v ∈ V̄ do
2      dist[v] := ∞
3      previous[v] := undefined
4  end
5  dist[root] ← 0
6  Q ← ∅
7  Q.push (root, 0)
8  while Q ≠ ∅ do
9      (u, cost) ← Q.pop()
10     foreach e(u, v) ∈ Ē do
11         d_uv = cost + weightOf(e(u,v))
12         d_uv‾, status ← checkConstraints(v, d_uv) ;
13         if status is valid and d_uv‾ < dist[v] then
14             if Q contains (v, dist[v]) then
15                 Q.update((v, d_uv‾))
16             else
17                 Q.push(v, d_uv‾)
18             end
19             dist[v] ← d_uv‾
20             previous[v] ← u
21         end
22     end
23 end
24 if dist[sink] is not ∞ then
25     return dist[sink]
26 else
27     return invalid
28 end
29
```

3.3 Impact of Meeting Points in Ride Sharing

In this section we present model for ride sharing with meeting points. The framework used here is a further improvement of the more generic mathematical model presented by Alonso Mora et al. [1] for real-time high capacity ride sharing. Given a set of passenger requests $R = \{r_1, ...r_n\}$ and set of vehicles $V = \{v_1, ...v_m\}$ at their current state, the algorithm involves computing incrementally optimal solution by solving a constrained optimization problem to obtain the final batch assignment of request set R to vehicle set V. The algorithm mainly consists of four steps as depicted in Figure 3.2, which are: computing pairwise request-vehicle shareability graph, computing the graph of feasible trips and the vehicles that can serve them, solving an Integer Linear Programming problem to obtain the optimal assignment and finally the rebalancing step to re-distribute the remaining idle vehicles. We define the model presented by Alonso Mora et al. [1] as *RS-ILP* and the modified model which can accommodate meeting points as *RSMP-ILP*. In below sections we explain each of the aforementioned four steps in details and then the improved method for ride sharing with meeting points.

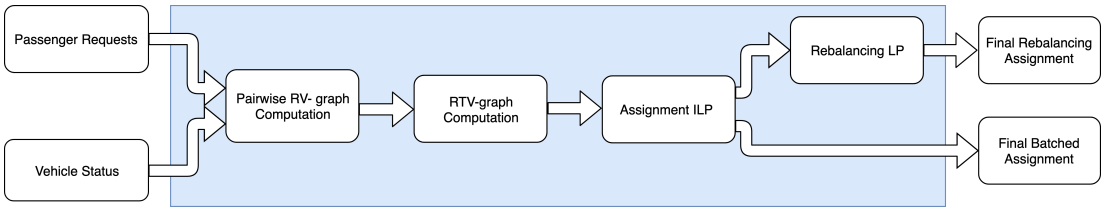

Figure 3.2: Schematic overview of the methodology for batch assignment of multiple requests to multiple vehicles.

3.3.1 Optimal Route of a Vehicle

Given a vehicle v with a set of on-board passengers P_v and a set of assigned requests R_v, we use $travel(v, R_v)$ function to determine the optimal route σ_v for the vehicle to pick up and drop off all the requests $P_v \cup R_v$ while minimizing the distance traveled by the vehicle and satisfying the constraint set Z (waiting time, travel delay, and capacity). As this vehicle routing problem is a computationally

hard problem, we use exhaustive search for vehicles with low capacity. The *travel* function returns the distance d_{σ_v} of the optimal route to serve the requests $P_v \cup R_v$ by the vehicle v if a valid path exists and "invalid" otherwise.

3.3.2 Pairwise Request-Vehicle Shareability Graph (RV Graph)

The purpose of computing the *RV Graph* is to determine, (1) which requests can be pairwise combined and (2) which vehicles can be used to serve which requests. Two requests r_1 and r_2 are connected by an edge $e(r_1, r_2)$ if and only if a hypothetical vehicle starting at either of the two origins of the two requests can pick up and drop off both requests while satisfying the constraints set Z. In the same way, a vehicle v and request r is connected by an edge $e(v, r)$ if and only if the vehicle can serve the request while satisfying the constraints set Z. We use the *travel* function as explained in Section 3.3.1 for the purpose of computing this. Thus, each edge $e(r_1, r_2)$ and $e(v, r)$ is associated with a weight of d_{σ_v} as returned by the *travel* function.

3.3.3 Request-Trip-Vehicle Graph (RTV Graph)

Based on the computed *RV Graph*, we then compute the feasible trips. A trip $T = \{r_1, .., r_t\}$ is a collection of t number of requests which can be combined together to be served by a single vehicle while satisfying the constraints Z for all requests in T. A trip is said to be feasible if it can be served by a vehicle while adhering to the constraints. To compute the feasible trips, we explore the cliques (regions for which its induced sub graph is complete) of the *RV Graph*. A given request r may belongs to more than one trip and each trip T may be serviceable by more than one vehicle.

RTV Graph contains two type of edges. A trip T and a request r is connected by an edge $e(r, T)$ if and only if $r \in T$. A trip T and a vehicle v is connected by an edge $e(T, v)$ if and only if T can be served by vehicle v without violating the individual constraints for all $r \in T$. A edge weight of d_{σ_v} as returned by the *travel* function is associated with each edge $e(T, v)$.

3.3.4 Optimal Assignment (ILP)

Next, based on the *RTV Graph*, we compute the optimal assignment of trips to vehicles. We formulate an ILP with two binary variables to solve this optimization problem.

For each edge $e(T_i, v_j)$ in *RTV Graph*, a binary variable $\epsilon_{i,j} \in \{0, 1\}$ is introduced. We let $\epsilon_{i,j} = 1$ if vehicle v_j is assigned to trip T_i. In the same way, another binary variable $x_k \in \{0, 1\}$ is introduced for each request $r_k \in R$. We let $x_k = 1$ if request r_k is not assigned to any of the vehicles in V. Let ξ_{TV} be the set of $\{i, j\}$ indexes for which an edge $e(T_i, v_j)$ exists in the *RTV Graph* where as Γ_j^V be the indexes i for which an edge $e(T_i, v_j)$ exists in the *RTV Graph*. We define the cost function C as per the Equation 3.18.

$$C = \sum_{i,j \in \xi_{TV}} c_{i,j}\epsilon_{i,j} \quad + \quad \sum_{k \in \{0,..,n\}} c_{ko}x_k \tag{3.18}$$

where $c_{i,j}$ denotes the d_{σ_v} as returned by the $travel(v_j, T_i)$ and c_{ko} is a large enough constant to penalize the ignored requests.

For the aforementioned cost function, we define two constraints based on the physical set up of the problem.

1. Each vehicle in the fleet is assigned to at most one trip.

2. Each request is either assigned to a vehicle in the fleet or is ignored.

Below, we formulate the above two constraints formally as per the the Equation 3.19 and Equation 3.20 respectively.

$$\sum_{i \in \Gamma_j^V} \epsilon_{i,j} \leq 1 \quad \forall v_j \in V \tag{3.19}$$

$$\sum_{i \in \Gamma_k^R} \sum_{j \in \Gamma_i^T} \epsilon_{i,j} + x_k = 1 \quad \forall r_k \in R \tag{3.20}$$

where as Γ_k^R and Γ_i^T denote the indexes i for which an edge $e(r_k, T_i)$ exists in the *RTV Graph* and the indexes j for which an edge $e(T_i, v_j)$ exists in the *RTV*

Graph respectively.

3.3.5 Rebalancing

Once a batch assignment is completed, we use a re-balancing strategy to re-distribute the remaining idle (empty) vehicles towards unserved requests. We rebalance the remaining idle (empty) vehicles such that total travel distance or time to pick up the unserved requests is minimized for the system. We formulate a linear program to solve this problem efficiently. In re-balancing, we make sure either all unserved requests or all idling vehicles are assigned. We refer the reader to [1] for the formulation of this problem.

3.3.6 Ride Sharing with Meeting Points

As explained previously, we extend the framework presented by Alonso-Mora et al. [1] to include the meeting points in the route planning for the fleet of taxis. In doing this, we only need to alter a single component from the framework. That is, we only need to alter the computation of vehicle routing through the *travel* function. We use the the proposed dynamic programming algorithm for *GOMMPR* query problem as presented in Algorithm 2 to replace the existing functionality of the *travel* function. However, it should be noted that even the extension is applied to a single component, the introduction of meeting points in to route planning adds an significant computational complexity to the system.

Chapter 4

RESULTS

4.1 *GOSMPR* query and *GOMMPR* query evaluation

In this section we present the results of running the the two queries, *GOSMPR* and *GOMMPR* with varying number of passenger requests. We report the execution time for each of such query execution.

Number of Passengers	Execution Time (s)
1 (3 nodes)	0.003
2 (5 nodes)	0.004
3 (7 nodes)	0.009
4 (9 nodes)	0.020
5 (11 nodes)	0.024
6 (13 nodes)	0.055
7 (15 nodes)	0.130
8 (17 nodes)	0.229
9 (19 nodes)	0.353
10 (21 nodes)	0.472

Table 4.1: Excution time results of *GOSMPR* query problem. (All values are measured in seconds)

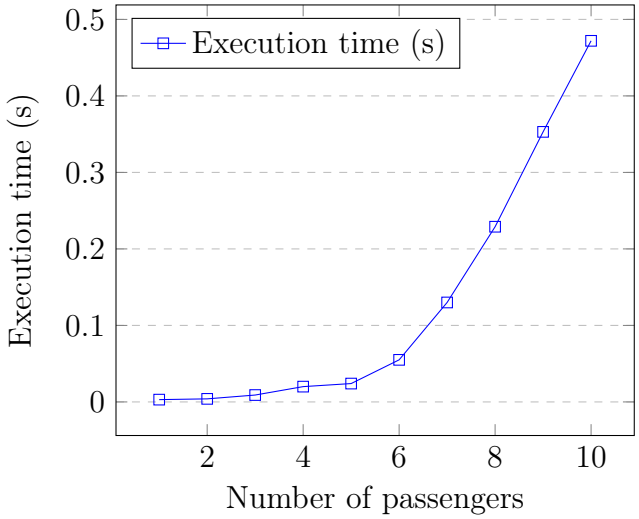

Figure 4.1: Execution time of *GOSMPR* query problem

We tested the *GOSMPR* query problem on real world data using NYC taxi data set based on the city network of the Manhattan island as obtained by Open Street Maps. As can be seen in Table 4.1, the execution time of *GOSMPR* query problem grows exponentially with the number of requests. This incident is clearly visible in the Figure 4.1. Since the problem is sufficiently constrained, the algorithm is efficient enough to perform the complex query to suit for real time systems. Also, in the ride sharing systems, we primarily concerned with vehicles which have a capacity of 4 or less. Thus, the results obtained are pragmatic to be used in real time ride sharing systems.

Number of meeting points ——	Number of Passengers			
	1	2	3	4
5	0.0004	0.002	0.050	1.93
10	0.0005	0.005	0.107	4.13
20	0.001	0.011	0.253	10.08
30	0.001	0.014	0.355	14.467
40	0.002	0.016	0.403	16.306
50	0.002	0.017	0.412	17.225

Table 4.2: Excution time results of *GOMMPR* query problem. (All values are measured in seconds)

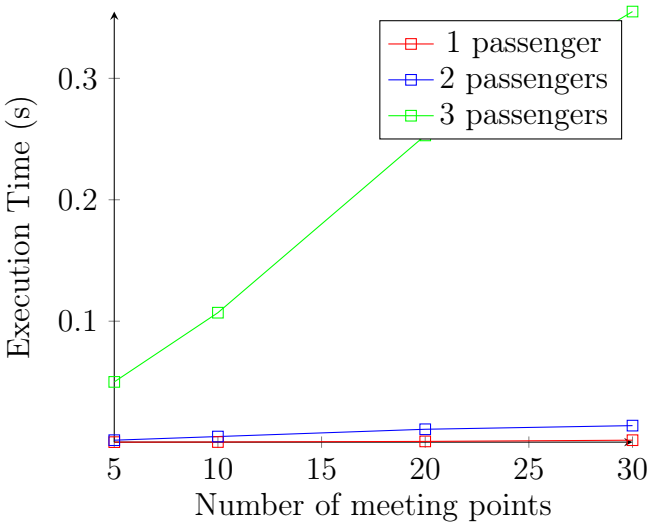

Figure 4.2: Execution time of *GOMMPR* query problem

We report the execution time of the *GOMMPR* query problem in Table 4.2 and visualize it in Figure 4.1. As the data set for the experiments, we use a NYC taxi data set and city network of the Manhattan island as obtained by Open Street Maps. As expected the execution time varies rapidly with the number of passengers and number of meeting points being considered. As can be observed, when the taxi is being considered for a trip of 4 passengers, the execution time of the algorithm is increasing rapidly with the number of meeting points being considered. However, as stated previously, in ride sharing systems, a taxi will carry 4 or less number of passengers and most of the time it will not carry 4 passengers due to natural spatial-temporal distribution of the demand. Thus, even though our algorithm is not highly efficient with increasing number of passengers, it is efficient enough for the purpose of real world ride sharing with meeting points.

4.2 *RS-ILP* and *RSMP-ILP* result comparison

Below we presents the results for two models *RS-ILP* and *RSMP-ILP*. We use the publicly available NYC taxi data set [31] for the experiments purpose. As the underlying road network, we use the data received from Open Street Maps corresponding to Manhattan Island, New York. A default speed of 17.5 mph, which is 70% of the NYC streets speed limit is used as the vehicle speed.

- Fleet size: 1000 Vehicles
- Total requests: 391926
- Simulation date: 5th September 2013
- Vehicle capacity: 4
- Default speed: 28 kmh/17.5 mph
- Preferred walking speed: 1.4 m/s
- Preferred walking distance: 400m
- Maximum waiting time: 5 minutes
- Maximum total travel Delay: 10 minutes

Notation	Description
SR	Service rate : Percentage of requests that were served
$(\overline{t_{\text{wait}}})$	Average waiting time per request (seconds)
$(\overline{t_{\text{wait}}^{\text{served}}})$	Average waiting time of served requests (seconds)
$\overline{t_{\text{delay}}}$	Average in car delay per request (seconds)
$\overline{t_{\text{total}}}$	Average total travel delay per request (seconds)
$\overline{VMT}$	Average Vehicle Miles Traveled by a vehicle (mile)
$\overline{occ}$	Average occupancy per vehicle
$\overline{w_{\text{pick}}})$	Average pick up walk (meters)
$\overline{w_{\text{drop}}})$	Average drop off walk (meters)
$\overline{w_{\text{total}}}$	Average total walk (meters)
$\overline{t_{\text{iteration}}}$	Average computation time per iteration (seconds)
t_{exe}	Total execution time (seconds)

Table 4.3: Description of resultant metrics

- Batch interval: 30 seconds

Table 4.3 denotes the performance metrics we used in comparing the two models. In the Table 4.4 we record the results we obtained for the two models. Clearly the results indicate the positive impact of meeting points. It is evident that additional 5056 trips have been able to server through the introduction of meeting points. By relaxing the constraints used such as maximum walking time, more improvements can be expected. The introduction of meeting points has reduced the average travel delay per passenger by 27.66 seconds. This is because the passengers are now being asked to walk to their destination from a pick up point without the vehicle taking a detour to serve them. This particular incident is validated by the high average drop off walk we can see in the results. The higher service rates in the *RSMP-ILP* compared to *RS-ILP* model is validated by the the difference in the occupancy rates. Clealy, a vehicle in *RSMP-ILP* model carry more passengers than a vehicle in *RS-ILP* model. However, as stated before, the introduction of meeting points in to existing ride sharing models is generally hard due to the nature of induced overheads. This can be observed from the difference in the computation times of the two models.

In the following sections we demonstrate the change in different matrices measured in the two simulations over the course of an entire day.

Model	SR	$\overline{t_{wait}}$	$\overline{t_{wait}^{served}}$	$\overline{t_{total}}$	$\overline{t_{extra}}$	$\overline{w_{pick}}$	$\overline{w_{drop}}$	$\overline{w_{total}}$	$\overline{occ}$	$\overline{VMT}$	$\overline{t_{iteration}}$	t_{exe}
RS-ILP	84.72	195.77	184.54	369.09	184.59	-	-	-	2.57	333.37	25.62	73861.07
RSMP-ILP	86.01	197.05	180.41	336.67	156.93	109.5	355.5	465.0	2.79	336.44	39.59	114009.70

Table 4.4: Comparison off results for *RS-ILP* and *RSMP-ILP*

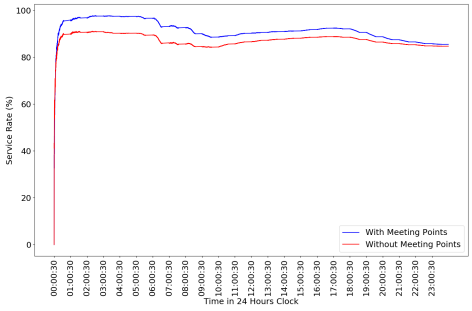

Figure 4.3: Service rate

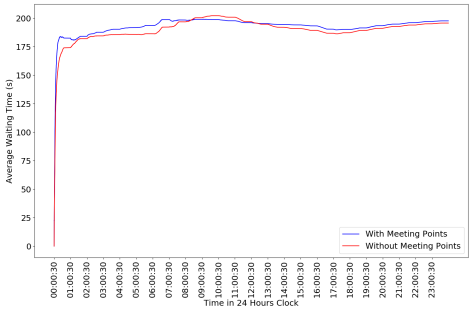

Figure 4.4: Average waiting time

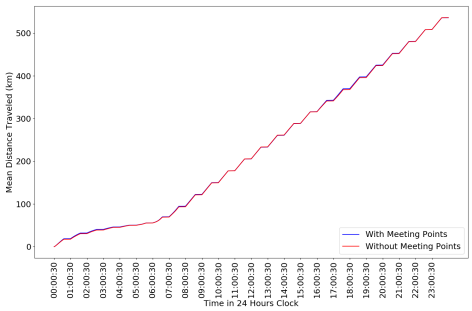

Figure 4.5: Mean distance travelled

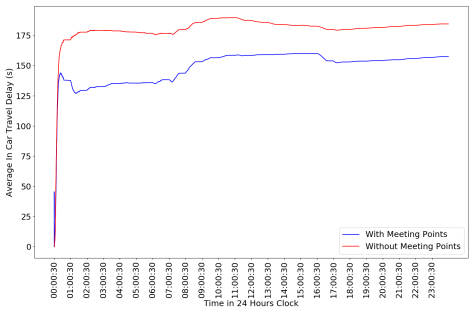

Figure 4.6: Average in car travel delay

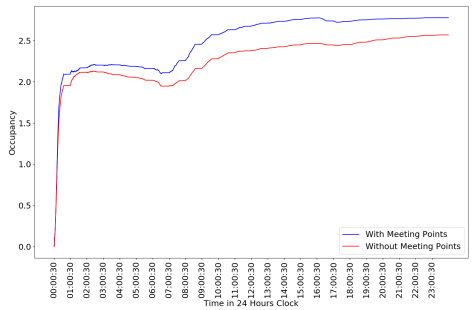

Figure 4.7: Occupancy

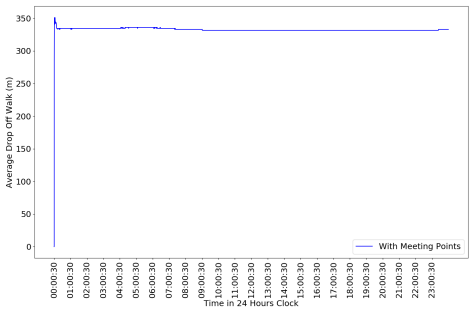

Figure 4.8: Drop off walk

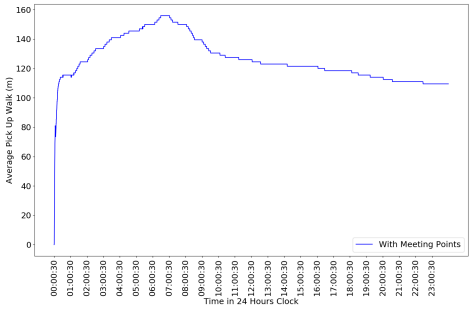

Figure 4.9: Average pick up walk

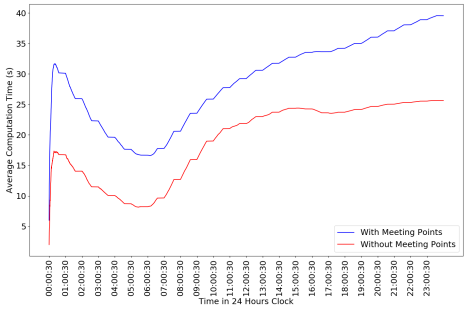

Figure 4.10: Average computation time

Chapter 5

CONCLUSIONS AND RECOMMENDATIONS

With rapidly growing complexity of urban mobility, ride sharing services are becoming a major mobility mode for commuters. However, designing such large scale systems is not a trivial task and involves systematic approaches looking at computational tractability and efficiency. One of the major focus of modern ride sharing systems is the requirement of computing optimal routes for a fleet of taxis. This problem is identified as a *NP-hard* problem and thus requires efficient algorithms to solve it in real time systems. In this work, we focused on *Generalized Optimal Single Meeting Point Route(GOSMPR)* query and *Generalized Optimal Multi Meeting Point Route(GOMMPR)* query problems, which we identified as a common requirement in most of the ride sharing models. In *GOSMPR* query problem, we are interested in finding an optimal route for a given taxi with a set of matched passengers while adhering to given set of constraints. Similarly, in *GOMMPR* query problem, we are solving the same problem but with meeting points included. With meeting points, a passenger can be picked up and dropped off from its origin or destination or from a meeting point which satisfy some given set of constraints. We showed that solving both *GOSMPR* query and *GOMMPR* query are NP-hard. Thus, we introduced a dynamic programming approach to solve the *GOSMPR* query optimally while using a variant of Held Karp lower bound as a heuristic to reduce the exponentially large solution space and a series of state elimination criteria to eliminate redundant states. We also presented another dynamic programming algorithm to solve the *GOMMPR* query problem. Our results showed both of the proposed algorithms are efficient enough to be used in real time ride sharing systems.

Apart from that, we also did extensive experiments to demonstrate the use of meeting points in ride sharing. We use real data from NYC taxi data set and conducted two types of experiments. We showed our ride sharing model with

meeting points, *RSMP-ILP* is performing better than the state of are ride sharing model without meeting points, *RS-ILP*. It was evident from the results that *RSMP-ILP* model can serve additional 5048 trips than the *RS-ILP* which corresponds to 1.3% enhancement to the existing service rate. We also presented the experimental results of the change of other set of performance matrices.

In this study, we limited the vehicle capacity of the vehicles to a maximum of four passenger at a time. It is expected with higher vehicle capacities, the performance of the route planning algorithm to degrade as of the exponential time complexity. Therefore, we keep the efficient relaxation of this limitation as a future work. Also, in the problem of *GOSMPR* query, we used a lower bound to guide the solution towards the optimal value. Our lower bound was based on a novel variant of classical Held-Karp lower bound for TSP and does not take into account any of the constraints we introduced in the algorithm. Thus our lower bound can be improved further. We keep the improvements to the lower bound as future work of this study.

In summary, our proposed two algorithms for *GOMMPR* query problem and *GOMMPR* query problem are performing efficiently to be used in real time ride sharing systems. Also, it was evident that the use of meeting points in ride sharing can improve the performance matrices of the existing ride sharing models to achieve better service for passengers with the same fleet of taxis.

References

[1] Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 114(3):462–467, 2017.

[2] M. Ota, H. Vo, C. Silva, and J. Freire. Stars: Simulating taxi ride sharing at scale. *IEEE Transactions on Big Data*, 3(3):349–361, Sep. 2017.

[3] Keld Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106 – 130, 2000.

[4] Michel Gendreau and Jean-Yves Potvin. *Tabu Search*, pages 165–186. Springer US, Boston, MA, 2005.

[5] Jon Rigelsford. Intelligent optimisation techniques: Genetic algorithms, tabu search, simulated annealing and neural networks. *Industrial Robot: An International Journal*, 27(5):null, 2000.

[6] Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees: Part ii. *Math. Program.*, 1(1):6–25, December 1971.

[7] Niels A.H. Agatz, Alan L. Erera, Martin W.P. Savelsbergh, and Xing Wang. Dynamic ride-sharing: A simulation study in metro atlanta. *Transportation Research Part B: Methodological*, 45(9):1450 – 1464, 2011. Select Papers from the 19th ISTTT.

[8] R. Li, L. Qin, J. X. Yu, and R. Mao. Optimal multi-meeting-point route search. *IEEE Transactions on Knowledge and Data Engineering*, 28(3):770–784, March 2016.

[9] Mitja Stiglic, Niels Agatz, Martin Savelsbergh, and Mirko Gradisar. The benefits of meeting points in ride-sharing systems. *Transportation Research Part B: Methodological*, 82:36 – 53, 2015.

[10] J.A. Hoogeveen. Analysis of christofides' heuristic: Some paths are more difficult than cycles. *Operations Research Letters*, 10(5):291 – 295, 1991.

[11] Hyung-Chan An, Robert Kleinberg, and David B. Shmoys. Improving christofides' algorithm for the s-t path TSP. *CoRR*, abs/1110.4604, 2011.

[12] Fumei Lam and Alantha Newman. Traveling salesman path problems. *Mathematical Programming*, 113(1):39–59, May 2008.

[13] Camelia-M. Pintea, Petrică C. Pop, and Camelia Chira. The generalized traveling salesman problem solved with ant algorithms. *Complex Adaptive Systems Modeling*, 5(1):8, Aug 2017.

[14] John R. Current and David A. Schilling. The covering salesman problem. *Transportation Science*, 23(3):208–213, 1989.

[15] Carlos E. Ferreira and Fernando M. de Oliveira Filho. Some formulations for the group steiner tree problem. *Discrete Applied Mathematics*, 154(13):1877 – 1884, 2006. Traces of the Latin American Conference on Combinatorics, Graphs and Applications.

[16] Naveen Garg, Goran Konjevod, and R Ravi. A polylogarithmic approximation algorithm for the group steiner tree problem. *Journal of Algorithms*, 37(1):66–84, 2000.

[17] Mehdi Sharifzadeh, Mohammad Kolahdouzan, and Cyrus Shahabi. The optimal sequenced route query. *The VLDB Journal*, 17(4):765–787, Jul 2008.

[18] Feifei Li, Dihan Cheng, Marios Hadjieleftheriou, George Kollios, and Shang-Hua Teng. On trip planning queries in spatial databases. In Claudia Bauzer Medeiros, Max J. Egenhofer, and Elisa Bertino, editors, *Advances*

in Spatial and Temporal Databases, pages 273–290, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[19] Yaron Kanza, Roy Levin, Eliyahu Safra, and Yehoshua Sagiv. Interactive route search in the presence of order constraints. *Proc. VLDB Endow.*, 3(1-2):117–128, September 2010.

[20] Roy Levin and Yaron Kanza. Tars: traffic-aware route search. *GeoInformatica*, 18(3):461–500, Jul 2014.

[21] Haiquan Chen, Wei-Shinn Ku, Min-Te Sun, and Roger Zimmermann. The partial sequenced route query with traveling rules in road networks. *Geoinformatica*, 15(3):541–569, July 2011.

[22] Jacques Desrosiers, Yvan Dumas, and François Soumis. A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows. *American Journal of Mathematical and Management Sciences*, 6(3-4):301–325, 1986.

[23] Stefan Ropke, Jean-François Cordeau, and Gilbert Laporte. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks*, 49(4):258–272.

[24] J. K. Lenstra and A. H. G. Rinnooy Kan. Some simple applications of the travelling salesman problem. *Journal of the Operational Research Society*, 26(4):717–733, 1975.

[25] Christine L. Valenzuela and Antonia J. Jones. Estimating the held-karp lower bound for the geometric tsp. *European Journal of Operational Research*, 102(1):157 – 175, 1997.

[26] Xin Li, Sangen Hu, Wenbo Fan, and Kai Deng. Modeling an enhanced ridesharing system with meet points and time windows. *PLOS ONE*, 13(5):1–19, 05 2018.

[27] Ulrich Matchi Aïvodji, Sébastien Gambs, Marie-José Huguet, and Marc-Olivier Killijian. Meeting points in ridesharing: A privacy-preserving approach. *Transportation Research Part C: Emerging Technologies*, 72:239 – 253, 2016.

[28] Paul Czioska and Monika Sester. Determination of efficient meeting points in ride-sharing scenarios. *AGILE International Conference on Geographic Information Science*, 2015.

[29] Radosław Grymin and Szymon Jagiełło. Fast branch and bound algorithm for the travelling salesman problem. In Khalid Saeed and Władysław Homenda, editors, *Computer Information Systems and Industrial Management*, pages 206–217, Cham, 2016. Springer International Publishing.

[30] Michael Held, Philip Wolfe, and Harlan P. Crowder. Validation of subgradient optimization. *Mathematical Programming*, 6(1):62–88, Dec 1974.

[31] Tlc trip record data. https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page. Accessed: 2019-07-01.