12-2021

Intelligent Traffic Management: From Practical Stochastic Path Planning to Reinforcement Learning Based City-Wide Traffic Optimization

Kamilia Ahmadi
Utah State University

INTELLIGENT TRAFFIC MANAGEMENT: FROM PRACTICAL STOCHASTIC PATH PLANNING TO REINFORCEMENT LEARNING BASED CITY-WIDE TRAFFIC OPTIMIZATION

by

Kamilia Ahmadi

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

DOCTOR OF PHILOSOPHY

in

Computer Science

Approved:

Vicki H. Allan, Ph.D. Major Professor	Curtis Dyreson, Ph.D. Committee Member
David Paper, Ph.D. Committee Member	Mario Harper, Ph.D. Committee Member
Chad Mano, Ph.D. Committee Member	D. Richard Cutler, Ph.D. Interim Vice Provost of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2021

ABSTRACT

Intelligent Traffic Management: From Practical Stochastic Path Planning to Reinforcement learning Based City-Wide Traffic Optimization

by

Kamilia Ahmadi, Doctor of Philosophy

Utah State University, 2021

Major Professor: Vicki H. Allan, Ph.D.
Department: Computer Science

This research focuses on two main areas of intelligent traffic management: a) stochastic path planning and b) city-wide traffic optimization. Stochastic path planning copes with the uncertainty of road traffic conditions by stochastic modeling of travel delay on road networks and helps individuals pursue specific goals and avoid congested areas. Next, we expand the model to make it applicable to city scale by utilizing pre-computation and approximation. The city graph is partitioned to smaller groups of nodes and each group is represented by its exemplar. For path planning queries, source and destination pair are connected to the respective exemplars corresponding to the travel direction and the path between those exemplars is found. Approximation provides paths with mean and variance which are not exact but clearly close to that exact paths, while the solution is space and time efficient.

City-wide traffic management focuses on optimizing traffic through structural changes of the city graph such as modifying lane direction, ramp metering, speed limits, and signal timings on road segments. Under these assumptions, we propose a multi agent reinforcement learning (RL) framework which the goal of RL agents is to interact with the environment to learn the optimal modification for each road segment with the goal of maximizing the

cumulative reward over the set of possible actions in the state space. Our proposed method has two level learning. In the first level, a single agent is the only modifier of the traffic system so it directly learns an initial policy. In the next level, we have multiple agents changing the environment at the same time, each based on the initial policy learned in the previous step while updating their policy based on the interaction with the dynamic environment and in agreement with other agents.

(126 pages)

PUBLIC ABSTRACT

Intelligent Traffic Management: From Practical Stochastic Path Planning to Reinforcement learning Based City-Wide Traffic Optimization

Kamilia Ahmadi

This research focuses on intelligent traffic management including stochastic path planning and city scale traffic optimization. Stochastic path planning focuses on finding paths when edge weights are not fixed and change depending on the time of day/week. Then we focus on minimizing the running time of the overall procedure at query time utilizing precomputation and approximation. The city graph is partitioned into smaller groups of nodes and represented by its exemplar. In query time, source and destination pairs are connected to their respective exemplars and the path between those exemplars is found. After this, we move toward minimizing the city wide traffic congestion by making structural changes include changing the number of lanes, using ramp metering, varying speed limit, and modifying signal timing is possible. We propose a multi agent reinforcement learning (RL) framework for improving traffic flow in city networks. Our framework utilizes two level learning: a) each single agent learns the initial policy and b) multiple agents (changing the environment at the same time) update their policy based on the interaction with the dynamic environment and in agreement with other agents. The goal of RL agents is to interact with the environment to learn the optimal modification for each road segment through maximizing the cumulative reward over the set of possible actions in state space.

To my better half, **Farzin**
and my wonderful angels **Eva** and **Navah**.

ACKNOWLEDGMENTS

There are many people whom I want to mention their names here. Those who have helped me in numerous ways not only on my path through the completion of my Ph.D, but also in the other aspects of my life. First and foremost, I want to thank my major professor, Dr. Vicki Allan, for granting me the freedom to develop and pursue my research ideas, for constructive feedback about my ideas, and for scholarly guidance and support throughout my PhD candidature. Her deep insights and positive manner have always been helpful and encouraging. Next, special thanks go to my committee members, Dr. Dyreson, Dr. Harper, Dr. Mano, and Dr. Paper for their support, help, valuable comments, and particularly for their patience during my Ph.D journey. Using this space, I should specially thank Dr. Paper for his consistent support, encouragement and guidance specifically in the deep learning domain. I used the state of the art concepts explained in his deep learning books in developing my DQN architecture.

I also would like to express my gratitude to the Utah State University, as I enjoyed studying over there and I learned a lot during my graduate studies.

Finally and most importantly, I am grateful for my parents, Batoul and Golmurad, and my siblings Pouria and Pedram for enriching my life beyond my scientific endeavours. I am so grateful for my two sweet angels, Eva and Navah, which are the source of inspiration for me from the time they came to this world. Words fail me to express my gratitude to my wonderful loving husband Farzin for believing in me, his love, constant support, a great deal of patience, and comforting me by his own subtle blend of wit and charm in this stressful period. Farzin, Eva and Navah your love and encouragement are what carried me through this work and made it attainable.

Kamilia Ahmadi

CONTENTS

Page

ABSTRACT iii

PUBLIC ABSTRACT v

ACKNOWLEDGMENTS vii

LIST OF TABLES x

LIST OF FIGURES xi

1 Introduction 1

2 Congestion-Aware Stochastic Path Planning and Its Applications in Real World Navigation 6
- 2.1 Abstract 6
- 2.2 Introduction 6
- 2.3 Previous Work 8
  - 2.3.1 Contribution of This Work 10
- 2.4 Model Description 11
  - 2.4.1 City 11
  - 2.4.2 Open Street Map Data 13
  - 2.4.3 Agents 14
  - 2.4.4 Pruning Heuristic in Path Finding 14
  - 2.4.5 Cost Function 18
  - 2.4.6 Modelling Agents' goals 21
- 2.5 Experiments and Results 23
  - 2.5.1 Path Finding based on users' goals 23
  - 2.5.2 Compare paths with shortest-length path 27
- 2.6 Conclusion 31

3 Practical City Scale Stochastic Path Planning with Pre-Computation 33
- 3.1 Abstract 33
- 3.2 Introduction 34
- 3.3 Previous Work 35
- 3.4 Framework 38
  - 3.4.1 City and Edge Weights 38
  - 3.4.2 Traffic Data 40
  - 3.4.3 Open Street Map 40
  - 3.4.4 Agents 41
  - 3.4.5 City Graph Partitioning 41
  - 3.4.6 Exemplar Assignment 50
  - 3.4.7 Base Path planning framework 52

3.4.8 Pre-processing: Building distance oracles 55
3.4.9 Scalable Algorithm 55
3.5 Experiments and Results 56
3.5.1 How Many Partitions Are Needed to Represent The City Graph? 57
3.5.2 Which Partitioning Method We Picked? 58
3.5.3 Which Exemplar Assignment Approach is The Best? 59
3.5.4 How is The Quality of Approximate Paths? 60
3.5.5 What is the Time and Space Complexity of Scalable Algorithm? 64
3.6 Conclusion and future work 65

4 Dynamic Reinforcement Learning Based Traffic Optimization in Smart City Paradigm 67
4.1 Abstract 67
4.2 Introduction 68
4.3 Previous Work 71
4.4 Model Framework 74
4.4.1 Time discretization 75
4.4.2 Space discretization 75
4.4.3 Learning Agents 76
4.4.4 State 77
4.4.5 Action 78
4.4.6 Reward 80
4.4.7 Learning Algorithm 81
4.4.8 DQN Architecture 83
4.4.9 Parameters 85
4.4.10 Training 87
4.5 Experiments and Results 88
4.5.1 Single vs Multi-agent Impact 88
4.5.2 Agent's Decisions 90
4.5.3 Patterns of traffic signals 92
4.5.4 Effect of lane change penalty 94
4.5.5 Importance of two-stage learning 95
4.6 Conclusion and Future works 95

5 Conclusion 97

REFERENCES 100

CURRICULUM VITAE 109

LIST OF TABLES

Table Page

4.1 Summary of the parameters used in this work. 86

4.2 Summary of the impact lane change penalty in non-smooth lane changes. . 95

4.3 Comparison of two stage learning pre-train and main train with only main train without pre-train step 96

LIST OF FIGURES

Figure Page

2.1 P_{SN} is the part of the path from source to node N and it is retrieved from the history of previous expansion steps. P_{ND} is the approximate shortest-length path from node N to destination. If the summation of mean of P_{SN} and P_{ND} is greater than the provided deadline, node N is not getting expanded. 15

2.2 Left: Distribution of main nodes in each partition. We have the total of 150 partitions. Right: Visualization of partitions on Salt Lake City. Each color represents one partition. 16

2.3 Finding an approximate shortest-length path from N to destination using $A*$ algorithm through centroids. 17

2.4 Paths from a specific origin (O) to a target (T) are presented as nodes (m_p,v_p) in the mean-variance plane. 18

2.5 Selected paths for different query times of Friday for Source node=83590367 and Destination node=352876209 based on each cost function. Query times from left to right the times are: a) 8:00, b) 15:00, and c) 18:10 PM. Deadline is set as 1200 seconds after start time 25

2.6 Selected paths for different query times of Tuesday for Source node=358207657 and Destination node=384734324 based on each cost function. Query times from left to right the times are: a) 7:30, b) 11:40, and d) 17:45. Deadline is set as 1400 seconds 26

2.7 Selected paths for different query times of Monday for Source node=2053542172 and Destination node=352883524. Query times from left to right are: a) 6:40, b) 8:10, and d) 18:00. Desired arrival time is within the 1600 seconds after query time. Best time for start the trip to have the smallest travel time is as follow: a) 6:40, b) 8:23, and c) 18:21. 28

2.8 Selected paths for different query times of Wednesday for Source node=1218569178 and Destination node=2421320748. Query times from left to right are: a) 7:40, b) 11:10, and d) 17:30. Desired arrival time is within the 2000 seconds after query time. Best time for start the trip to have the smallest travel time is as follow: a) 8:01, b) 11:10, and c) 18:09. 29

2.9 Comparison of mean and variance of highest probability path, smallest travel time path and shortest-length path in 17:20 PM of Tuesday for 10 different source and destinations (A to J). 30

2.10 Comparison of mean and variance of highest probability path, smallest travel time path and shortest-length path in 17:20 PM of Wednesday for 10 different source and destinations (A to J). 30

2.11 Comparison of mean and variance of highest probability path, smallest travel time path and shortest-length path in 17:20 PM of Friday for 10 different source and destinations (A to J). 30

2.12 Comparison of average mean and variance of highest probability path, smallest travel time path and shortest-length path in 8:00 AM of Weekdays for 100 different source and destinations. 31

3.1 Paths from a specific origin (O) to a target (T) are presented as nodes (m_p,v_p) in the mean-variance plane. 39

3.2 Distribution of main nodes in each cluster and visualization of them on Salt Lake City for *K-means* clustering. Each color represents one cluster. 45

3.3 Left: Distribution of cluster sizes in *Mean Shift*. Right: Visualization of clusters on Salt Lake City. Each color represents one cluster. 46

3.4 Distribution of communities on Salt Lake City in *Leading Eigenvector* approach. 48

3.5 Distribution of communities in *Walktrap* approach for the clusters with node size less than 300 nodes along with representation of all of the communities. Each red dot represent a community. The edge between communities shows the relationship between one community to another. 49

3.6 Left: Distribution of communities in Label Propagation approach. Right: Representation of communities. Each red dot represent a community. The edge between communities shows the relationship between one community to another. 50

3.7 Left: Distribution of communities in *Multilevel* approach and representation of them. Each red dot represent a community 51

3.8 **(a)**: For finding a path from n_1 to n_2, successor nodes of n_1 (orange circles) are explored. Among the successor, the marked ones meet the deadline and rest are discarded. Dotted paths are heuristics paths from the successor nodes to n_2 and they are used as a pruning criteria for the successor nodes of n_1. **(b)**: Finding the heuristic path from m (a middle node in expansion shown as green circles) to n_2 uses $A*$ algorithm through exemplars of graph (green triangles). **(c)**: From current exemplar to the neighboring exemplar the one with with the least $g(n) + h(n)$ is selected (green triangle). Figure
shows the selected exemplar in each step of A^*on exemplars for heuristic path. 53

3.9 Red rectangles are exemplars of each region. Green circles are the typical source and destination. 56

3.10 Number of partitions vs the mean difference of travel time of exact and approximate path. 58

3.11 Left: Distribution of nodes in each partition. Right: Visualization of partitions on Salt Lake City. Each color represents one partition. 58

3.12 Comparison of four methods of exemplar selection a) highest traffic, b) highest reach, c) closeness centrality and d)random walk centrality 60

3.13 Y axis is the relative difference percentage of mean and variance of travel time of paths for exact and approximate approach for peak and non-peak hours for the agent's goal of highest probability path 61

3.14 Ratio of paths with the closest mean-variance to the exact path in peak and non-peak hour for the agent's goal of highest probability path 62

3.15 Relative difference of travel time of mean and variance of paths for exact and approximate approach for rush and non-rush hours for the agent's goal of shortest en-route time. 63

3.16 Ratio of paths with the closest mean-variance to the exact path for the agent's goal of shortest en-route time. 64

4.1 Allocating one lane from right side to left side may decrease the congestion. 69

4.2 Delaying cars from entering the freeway using a ramp metering. 70

4.3 Implementation of variable speed limit. Variable speed limit allows speed limits to be changed based on current road conditions and the level of congestion (original image from PennDOT [1]) 71

4.4 Sample of state discretization. Road segments are shown with different colors (original image cropped from Google Maps ™). 76

4.5 The 8 possible phases of a signal at each junction. 77

4.6 Main segment and extra segments that have been considered for a) lane, b) signal, c) ramp and d) speed agents. 79

4.7 Architecture of Deep Q-Network (DQN) using experience replay and target network 85

4.8 The graphs compares the improvement we got over the baseline traffic with different groups of agents modifying the city during rush hour and non-rush hour time. 89

4.9 Summary of modifications that our agents made during the three traffic profiles: a) low traffic, b) medium traffic and c) high traffic 91

4.10 Distribution of speed limit changes in low traffic and lane changes in high traffic 93

4.11 Possible signal phases for a north-south direction. 93

4.12 Signal Pattern for base, none rush hour and rush hour. 94

CHAPTER 1

Introduction

Intelligent agents can be used in simulation of real-world domain applications. A simulation is an imitation of the operation of a real-world process or system. Simulation has the following advantages: a) one can study the behavior of a system without building it, b) results are accurate in comparison to analytical model, c) can find unexpected phenomenon and the behavior of the system, d) easy to perform ”What-If” analysis and e) experiments on a real system may be impossible or impractical, often because of cost or time. Hence, multi-agent systems with the power of mimicking the intelligence of interacting humans are a suitable paradigm to facilitate the experimentation in a real world domain. In this research, we utilize the capability of multi-agent systems in order to propose models for intelligent traffic management. Also we want to assess how effective our proposed algorithms work in a real world domain setting [2].

A recent survey [3] estimates that the annual nationwide cost of traffic congestion is 78 billion, including 4.2 billion hours in lost time and 2.9 billion gallons in wasted fuel. In modeling a city scale graph, congestion changes throughout the day which results in having uncertain costs on the road segments [4–8]. Congestion is affected both by the total traffic and by the path selection of drivers in the network. In addition, there are many factors that affect the congestion pattern such as road conditions, drivers' path choice, time of the day, weather conditions, and events throughout the city [4, 6, 9–11].

One aspect of an intelligent traffic management system is to enable users to make more coordinated, efficient, and smarter decisions through a path planning paradigm. Our path planning framework defines edge weights based on the mean and variance of travel time on them and edge weights change depending on the time of day/week. Such path planning system is a useful addition to on-board navigation systems to provide the paths that meet desired travel goals. It is also a worthwhile addition to Web-based mapping services. The

other aspect of traffic management is to improve the traffic flow and reduce the traffic load on highly congested areas by minimizing overall city congestion. Recently, more attention has been paid to leveraging operational techniques in traffic management like widening roads, variable speed limits, and modifying signal timing [12–20]. Our traffic optimization framework focuses on making structural changes to the city graph such as dynamically changing the direction of lanes, ramp metering, modifying the speed limits, and modifying the signal timing in order to manage the traffic in congested areas. Reducing congestion throughout the city has the benefits of decreased pollution, fewer accidents, less wasted time, and less fuel costs ([6, 8, 21–23].

The main questions we want to address in this research are as follow:

- In the stochastic domain, where edge weights are not fixed and they stochastically changing during the day, can we propose a path planning algorithm that models different agents' goals and satisfies their desired characteristics? How is the proposed path planning algorithm impacted by agents' goals?
- In the domain that the primary goal of agents is to pick the path with minimum cost, how do we realistically model paths' costs in order to mimic the real world domain path planning?
- How can we make the proposed path planning algorithm applicable to real world domains where there are thousands of path planning requests at the same time and the characteristics of the domain is changing over time?
- In the case of a futuristic smart city that is making structural changes to the city graph, what are the set of changes that can efficiently reduce the overall traffic?
- Is it possible to learn the structural changes dynamically and have trained agents that can make decision dynamically to reduce overall traffic?

A scalable path planning framework finds a path from a specific origin to a destination over a network of road segments. Path planning algorithms use the road segment costs in

order to come up with the best path. If the road segments' costs are fixed, planning the best path through the network is a well understood task via algorithms like Dijkstra and A* algorithms [24, 25]. However, in real world navigation problems, depending on the level of congestion on the road segments, the cost associated with the legs of the trip changes over time. The challenge is to find the best paths under uncertainty and the constraints of a real-world domain. The definition of best path differs based on the goal of the agents. For finding the best path, queries have an origin, a destination and the desired arrival time (deadline) along with the agents' goals. Inspired by time-dependent traffic situation, we parameterize travel time distributions by time, which allows us to speak of time-dependent path costs and study the problems of reaching a goal by a deadline and delaying departure to minimize traversal-to-goal time. The best path is the path with lowest cost, and the cost is mainly based on travel time which depends on the level of congestion at different times of the day/week. For modelling cost functions, we consider three possible cost functions (linear, exponential, and step cost functions) in order to model the main classes of realistic goals, but any cost function can be incorporated in the model. Since the graph is large, optimizing for lowest cost on all possible paths between any source destination pairs is not feasible. Therefore, we propose a pruning technique to shrink the search domain. For finding the candidate paths between nodes pair of nodes, we start from first node and explore the successor nodes (*expanding*) until we reach the second node. When exploring each node, the heuristic estimates a path from that node to destination and if the expected estimated arrival time when using the heuristic path is greater than the provided deadline, the node is not expanded (pruned). We continue this process until we find the candidate paths between a source and a destination. Then candidate paths are passed to the path selection part which picks the path with minimum cost which satisfies agents' goals.

We build on the top of proposed path planning framework and make it practical to be used in large scale path planning applications. For expediting the path planning process, the city is partitioned, and each part is represented with an exemplar location. The exemplar of each partition is decided based on random walk centrality concept which tries to find the

center of a partition based on the expected length of a random walk. Two approaches have been used for partitioning the city graph: 1) community detection, and 2) graph clustering. In response to a path planning request, source and destination nodes are connected to their nearest exemplars (with respect to the path direction) and the path between exemplars is retrieved. The paths between exemplars are stored in distance oracles based on the preceding year data at the time of update. The oracles are updated every week to reflect the recent changes in the network. Approximation provides paths with mean and variance which are not exact but clearly close to the exact paths, while the solution is space and time efficient.

City-wide traffic optimization focuses on optimizing the traffic congestion on the whole city network through a smart city paradigm. A smart city provides the capability of modifying the structure of the city graph and gathering information from the sensors to learn the best possible modification for each condition of the traffic. These changes include modifying lane direction, ramp metering, speed limits, and signal timings on road segments. For this reason, we propose a multi agent reinforcement learning system (RL Agents) that finds the best structural changes based on multiple dynamic factors such as current traffic condition, dependent road segment structures, and recent structural modifications. These structural changes not only impact the flow in both directions of the road segment but also the flow at surrounding road segments. Therefore, we need to consider the impact of structural changes not only on each road segment, but also on dependent road segments and the whole network. RL agents interact with the environment in the training phase and learn the optimal modification for each road segment considering the current road segment and the impact on the dependent segments. We proposed the process of learning the optimal policy as a two step process. In the first step, a single agent is the only modifier of the traffic system so it directly learns the initial policy. In the second step, there are multiple agents changing the environment, each based on the initial policy learned in the previous step while still updating their policy. The goal of RL agents is to maximize the cumulative reward over the set of possible actions in state space. First step is critical to reduce the

noise in the reward system and be able to converge to a stable policy. Then, in the second step, the agents update their policy to take into account the indirect interaction with the other agents.

The following chapters are published or submitted articles.

CHAPTER 2

Congestion-Aware Stochastic Path Planning and Its Applications in Real World Navigation

2.1 Abstract

One[1] of the main applications of path finding is in real world vehicle navigation. Most of the algorithms use edge weights in order to select the best path for navigation from an origin point to a specific target. This research focuses on the case where the edge weights are not fixed. Depending on the time of day/week, edge weights may change due to the congestion through the network. The best path is the path with minimum expected cost. The interpretation of best path depends on the point of view of car drivers. We model two different goals: 1) drivers who look for the path with the highest probability of reaching the destination before the deadline and 2) the drivers who look for the best time slot to leave in order to have a smallest travel time while they meet the deadline. Both of the goals are modelled based on the cost of the path which is highly dependent on the level of congestion in the network. Minimizing the paths' cost helps in reducing traffic in the city, alleviates air pollution, and reduces fuel consumption. Findings show that using our proposed intelligent path planning algorithm which satisfies users' goals and picks the least congested path is more cost efficient than picking the shortest-length path. Also, we show how agents' goals and selection of cost function impacts paths' choice.

2.2 Introduction

Path planning finds a path from a specific origin to a destination over a network of road segments. Path planning algorithms use the road segment costs (travel time, distance, and

[1]The first version of this work is presented and published in proceedings of IEEE International Conference on Computational Science and Computational Intelligence (CSCI 2017). DOI: 10.1109/CSCI.2017.22 The current version (extended) is presented and published in proceeding of 13th International Conference of Agents and Artificial Intelligence (ICAART 2021) DOI:10.5220/0010267009470956

congestion) in order to come up with the best path. If the road segments' costs are fixed, finding the best path through the network is a well understood task. Many algorithms have been proposed either to find a shortest length path between nodes of a graph or to find the optimized path by considering fixed edge weights like Dijkstra and A* algorithms [21,24,25]. However, in real world navigation problems, it is more complex to find a best path for routing. Depending on the level of congestion on the road segments, the cost associated with the legs of the trip changes over time. Also, it is not feasible to use an adaptive algorithm due to the urgency in having a quick response to the queries and hesitancy of drivers to change their route frequently.

In modeling a city scale graph, congestion changes throughout the day which results in having uncertain costs on the road segments [6, 8, 26, 27]. Congestion is highly affected by the path selection of drivers in the network. In addition, patterns of the congestion on the road segments of a city graph are not always predictable. There are many factors that affect the congestion pattern such as road conditions, drivers' path choice, time of the day, weather conditions, and events throughout the city [4, 6, 10, 11, 23, 27].

We consider expected travel time on the road segments as the cost of that segment. The variability of congestion level on road segments implies that the real world navigation is a problem of finding a path through a stochastic network. One prominent step toward having a minimum cost path through a stochastic network is to understand traffic conditions and use them for planning drivers' paths. Being able to minimize the paths costs, ultimately results in reducing the city scale congestion by picking less congested paths. Reducing congestion throughout the city has the benefits of decreased pollution, fewer accidents, less wasted time, and less fuel costs [6, 8, 21–23, 28].

This paper focuses on path planning over a stochastic network which is a graph of a city. The challenge is to find the best paths under uncertainty and the constraints of real world domain. Agents are car drivers which can pursue different goals: First, the ones who are not willing to take risk and look for the path with highest probability of reaching destination before a desired arrival time, even if it may take them longer. Secondly, the

agents who are open to take a riskier decision if it helps them in having the smallest en-route time. These agents are flexible in leaving anytime while they still need to make the trip.

To make it clearer, one good example of these kind of agents' goals is in the context of a package delivery system. For example, suppose that we guarantee the delivery of a package by 4 PM, otherwise the customer doesn't accept the delivery and we lose the shipping costs. In that case, we are interested in picking a path that has the highest chance of reaching destination before the deadline to avoid losing the shipping cost. The other possible case is delivering perishable products. For example, if we promised the delivery of perishable products before 6 PM to the customers, we are interested to pick a path that has the smallest en-route time due to the nature of our package. In this case, we are flexible in leaving anytime, but we do need to have the smallest en-route path while still making the destination before 6 PM.

As mentioned earlier, the definition of best path differs based on the goal of the agents. For finding the best path, queries have an origin, a destination and the desired arrival time (deadline) along with the agents' goals. Then, intelligent path planner finds the least cost path corresponding to the path finding queries and send it to the agents.

2.3 Previous Work

Miller-Hooks and Mahmassani [29] consider travel costs as edge weights of a navigation graph in their model. Costs depend on travel times, and their goal is to find the least expected travel time path by setting each arc's weight to its expected value for peak and non-peak time of the day. Then they solve an equivalent deterministic problem. This model considers some extent of uncertainty in path finding. However, there has been little work on decision theoretic models which directly consider uncertainty, congestion awarenessand time dependency of edge weights and which find the optimal path on the basis of the distributional information of the stochastic edge weights.

Fan, Kalaba and Moore [22] consider a special monotone increasing cost based on the probability of arriving late and suggests that the Gamma distribution is natural for modelling stochastic edge travel times. The probability calculation requires computing a

continuous-time convolution product. Therefore, it makes the path finding a computationally expensive and time consuming task.

Niknami et al [4], present an efficient technique for computing the route that maximizes the probability of on-time arrival in stochastic networks. Their method uses a heuristic for the optimal path that chooses the direction at every intersection based on the current state. The solution for this problem can be obtained by evaluating zero-delay convolution on the path probability and expected travel time. They made three major assumptions that travel time distributions are 1) time invariant, (2) exogenous (not impacted by individuals routing choices), and (3) independent. These assumptions make it not desirable for us as we look for a model that considers changing the edge weights through the day and the routes' choices are affected by other driver's decision as it is the major source of congestion on stochastic networks.

Zhiguang [30], proposed the Probability Tail model based on stochastic shortest path problem to find the most reliable path. They formulated the problem as a cardinality minimization problem by directly utilizing travel time data on each road link. Then, the minimization problem is approximately solved via relaxing the cardinality by $L1$-norm and its variants, and formulating it as a mixed integer linear programming (MILP) problem. Their model uses a data driven approach for path finding and assumes that the traffic pattern on the edge links are invariant. For extracting the edge weights, it uses travel time samples on each arc as input and adopts some random distributions to generate the weights. As the result, this model doesn't consider traffic patterns for different times of a day.

Some stochastic path planning methods utilize pruning techniques in order to shrink the path finding search region. Some of the famous ones are 'reach' and 'arc-flags'. Lots of other pruning techniques are based on either reach or arc-flag. In reach-based pruning [31, 32], a node is expanded if it lie on a shortest path that extends a long distance in both directions from the vertex. The idea of pruning in arc-flag [33] is to divide the graph into a set of regions. Then, each edge has an associated vector of Booleans with one value for each region where each Boolean is true if the edge is used by at least one path ending in the

corresponding region. Then any edge without the Boolean corresponding to the region that the destination belongs to is pruned from the graph. One of the major limitations of both mentioned methods is the long time takes to reflect any possible changes of the network due to the vast amount of computation.

Lim [6] proposes stochastic path finding where edge weights are represented as linear combination of mean and variance of travel time (mean+ λ * variance) controlled by a λ parameter. They used GPS traces of roughly 500 million anonymized GPS points to model the city graph. Then, extracted GPS traces from individual vehicles and, using map matching techniques, they build the city graph. In their model, the key property is the fact that the optimal path occurs among the extreme points of the convex hull containing all the path points. The λ is used to prune the search regions and selects only a small number of λ values. Then best path is found by Dijkstra [24] based on minimizing the cost function of two modelled goals: 1) maximizing the probability of reaching a destination before a deadline and 2) identifying the latest departure guaranteeing arrival before the deadline.

2.3.1 Contribution of This Work

Firstly, in this research, we used Open Street Map (OSM) data for modelling the city graph [34]. OSM data is an open source collaborative map and is widely accessible to model any city of interest.

Secondly, in our model, we consider travel costs in intervals of 10 minutes for each day of a week in order to extract the typical mean and variance on that edge for the specific time slots. Means and variance of each edge, does not need to be combined and they present the variation in any given point of day/time. Time steps are as short as 10 minutes to reflect the changes throughout the day accurately. Considering different days of a week is also important because the traffic pattern of Monday 8 : 00 AM is different from Saturday 8 : 00 AM. In our model, mean and variance of edge e at 3 : 25 PM of Monday is the mean and variance of travel time on edge e for all Mondays of a year in the time segment of [3 : 20, 3 : 30]. In this way, we model the behavior of traffic flow more realistically.

Also, we study three options of cost functions to have an understanding of main classes

of cost modelling: 1) linear cost function in which the cost of the path linearly increases by travel time, 2) exponential cost function which represents the case where the cost rapidly increases by travel time and 3) step cost where there is no cost if the user arrives before the deadline while the paths that arrive after the deadline are penalized.

In addition, our path finding has two main steps: a) pruning search region to select few paths among all possible paths, and b) finding optimal path from the selected ones in step *a*. In pruning phase, a node is expanded if expected mean of the travel time of the approximate path through the node is less then the user's deadline. (The process explained in 3.4.7). This pruning technique which is easily adaptable to the changes of network, helps in reducing the search region and hence makes the path finding process applicable to real world domain. As in city graphs, because of interconnected nature of the graph, there are large number of paths to be considered between any source and destination.

The last contribution of this paper focuses on agents' goals. First group of agents are looking for the path that maximizes the probability of reaching a destination before the deadline. Second group, look for the best departure time slot in order to have the least travel time and arrive at the destination before deadline, these agents are interested to take riskier decision if it provides them shorter en-route time.

2.4 Model Description

2.4.1 City

The city is modelled as a directed graph consisting a set of vertices, V , which represent road intersections and edges, E, that represent road segments between vertices as shown in Equation 3.1.

$$E \subset V^2 \tag{2.1}$$

We consider the city graph to be planar (i.e., edges intersect only at their end points). If we consider the number of nodes in the planar graph as n and the number of edges as m, the relationship between them is $m << n^2$ [6, 26, 35]. Associated with each edge of

the graph is a travel cost used as the edge weight in our directed graph. We use expected travel time as the cost of an edge. Edge costs are not fixed, and they are represented by an expected travel time random variable. The travel time random variable, W, is represented as a tuple of mean and variance of the delay on that edge at the specific travel time interval shown in Equation 2.2.

$$W_{edge}(t) = (m_{edge}(t), v_{edge}(t)) \tag{2.2}$$

We compute time segments in the intervals of 10 minutes for each day of a week. For finding the mean and variance of each edge in time segments of a week, we summarized yearlong traffic data based on 10 minutes time segments for each day of a week. The target city in this model is Salt Lake City, Utah, and we use monitored traffic data from Utah Department of Transportation (UDOT) [36] to extract edge weights of the city graph.

Travel time of each edge is an independent Gaussian random variable [6, 10, 11, 22, 26, 28,37]. Since the sum of independent Gaussian random variables is also a Gaussian random variable, the travel time for the whole path is also Gaussian (shown in Equation 2.3).

$$t_{path} \sim Normal(m_{path}, v_{path}) \tag{2.3}$$

Stochastic dependency between adjacent edges can be considered by transforming the graph in a way to add a new edge between two dependent edges with mean equals to 0 and variance equals to covariance of the weights of two dependent edges. Note that correlation of two consecutive edges is always positive as they have similar traffic directions. This new edge captures the correlation between the two correlated distributions [6, 22, 26, 37]. We consider edge weights to be independent from each other as the time dependent variance on edges represents the dependency of the congestion on adjacent edges [4, 6, 10, 11, 26, 38, 39]. For example, suppose that edge *e* takes 30 percent longer than when congestion free in a specific time slot, an adjoining edge is likely to take 30 percent longer than when congestion free in the same time slot. Then for the specific edge and its adjoining edge, the variance reflects all of these changes throughout different time slots of the day.

The mean of a path is the sum of the means of all edges included in the path (Equation 2.4).

$$m_{path}(t) = \sum_{e \in path} m_e(t + \delta) \tag{2.4}$$

Variance of the path is the sum of variance values of all edges included in the path from an origin O to destination D (Equation 3.7) [6, 21, 26, 38, 39]. If we consider each edge as an independent random variable, then the sum of variances is derived from (Equation 2.5). Since we assume edge weights are independent from each other, then $cov(X_i, X_j)=0$ for $i \neq j$ and Equation 2.6 is the result. Based on Equation 2.6, the variance of a path is the sum of variance of all edges included in the path shown in Equation 2.7.

$$var(\sum_{i=}^{n} X_i) = E([\sum_{i=}^{n} X_i]^2) - [E(\sum_{i=}^{n} X_i)]^2 \tag{2.5}$$

$$var(\sum_{i=}^{n} X_i) = \sum_{i=1}^{n}\sum_{j=1}^{n} cov(X_i, X_j) = \sum_{i=}^{n} cov(X_i, X_j) = \sum_{i=}^{n} var(X_i) \tag{2.6}$$

$$v_{path}(t) = \sum_{v \in path} v_e(t + \delta) \tag{2.7}$$

While we could use the mean and variance of a path, using the sum of means and variances of its constituent parts yields the same results. Computing mean and variance of a path based on its including edges helps in storing less data for the whole city, because each edge might be used in multiple paths. For finding the mean and variance of a path, a sliding time window has been considered. A sliding time window implies that the cost of each edge in the path depends on the amount of time that took to reach it, not just the initial departure time. For example each $m_e(t)$ is actually $m_e(t + \delta)$ in which delta is the estimated arrival time from source node to edge e.

2.4.2 Open Street Map Data

For building the city graph, we used Open Street Map data [34]. Open Street Map is a collaborative open source project which creates a free editable map that can be used widely. Open Street Map represents physical entities on the ground like buildings, roads,

intersections, bridges and so on. It uses the basic data structure of entities and tags for describing the characteristics of that entity. The data structure includes nodes, ways, and relations. A *node* is a single point in space defined by its latitude, longitude, and node id. A *way* is a list of nodes used to represent linear features such as a series of roads. A *relation* is a multi-purpose data structure that relates two or more data elements like a route, turn restriction, traffic signal or an area. Entities in Open Street Map have tags which specify the characteristics of the entity. Each tag describes a geographic attribute of the feature of the specific node, way or relation. Most features can be described using only a small number of tags, such as a path with a classification tag such as highway or foot way.

2.4.3 Agents

We consider drivers as agents. Agents get suggested directions from a central path planner by entering source, destination, deadline and their goal. Definition of best path may be different from the point of view of one agent to another. Having the origin (O), target (T), and deadline (D), here are the two main questions that clarifies agents' goals in this model.

- What is the path with the maximum probability of reaching destination before the deadline? (the most secure path, hence might be longer)
- What is the best time to leave in order to have the smallest travel time and reach the target before the deadline? (riskier decision, while getting smallest travel time path)

2.4.4 Pruning Heuristic in Path Finding

In a city scale graph with interconnected nodes, there are many possible paths between a source node (S) to a destination node (D). Considering all of those paths is computationally intractable and lots of them are not aligned with the query's deadline and goal. Thus, we need to prune the search region in order to consider the paths with the closest characteristics to the desired path. For finding the candidate paths between a source (S) to a destination (D), we start from S and explore the successor nodes (*expanding*) until we reach D. When

exploring each node, the heuristic estimates a path from that node to destination and if the expected estimated arrival time when using the heuristic path is greater than the provided deadline, the node is not expanded (pruned).

The path from source to destination through node N is the combination of the path from source to the node (P_{SN}) and the approximate shortest-length path from the node to destination (P_{ND}). For each node in expansion phase, P_{SN} is known from the history of previous expansion steps. For finding P_{ND}, we consider an approximate shortest-length path from that node to destination as finding the actual shortest-length path from N to destination is also computationally intractable due to the large branching factor in each step of the city scale graph.

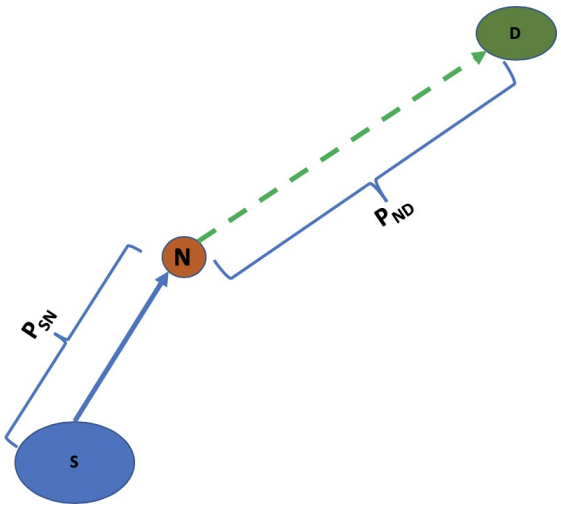


Fig. 2.1: P_{SN} is the part of the path from source to node N and it is retrieved from the history of previous expansion steps. P_{ND} is the approximate shortest-length path from node N to destination. If the summation of mean of P_{SN} and P_{ND} is greater than the provided deadline, node N is not getting expanded.

For approximating the P_{ND}, we use grid-based city partitioning. We partition the city based on a simple gird of $10 * 15$ to have 150 partitions. 150 partitions has been selected based on the shape of Salt Lake City. Each partition includes a set of nodes and it is represented by its centroid. Centorid of each partition is one of the main nodes in

the center of the partition. Figure 2.2 shows the grid of Salt Lake City along with the distribution of nodes in each partition.

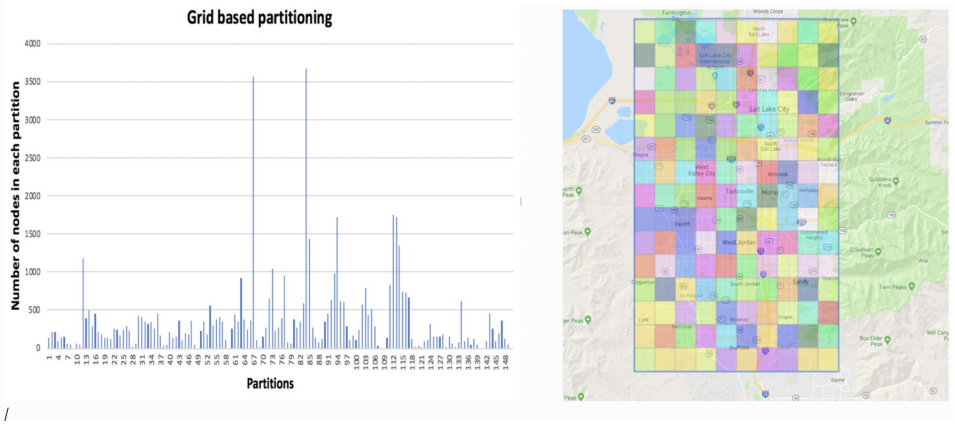


Fig. 2.2: Left: Distribution of main nodes in each partition. We have the total of 150 partitions. Right: Visualization of partitions on Salt Lake City. Each color represents one partition.

Approximate shortest-length path (P_{ND}) is found by using A* algorithm on centroids, i.e. instead of considering all the nodes from N to D, only centroids are considered. In each step of A*, the next centroid is picked based on the smallest $g(n) + h(n)$ value, where $g(n)$ is the shortest-length path from current centroid to the neighboring centroid and $h(n)$ is the direct path from the neighboring centroid to the destination. Shortest-length paths between adjacent centroids are pre-computed and they are retrieved to build the approximate shortest-length path.

As it can be seen from Figure 2.2, Salt Lake City has 150 partitions, therefore, pre-computing and storing the shortest-length paths between the adjacent centroids is not a complex task. Also pre-computation of shortest-length path between centroids is a one time task as the shortest-length path between centroids doesn't change over time.

After finding the approximate shortest-length path, expected mean of travel time for both P_{SN}) and P_{ND} are found considering the query time and if the summation of their means is greater than the provided deadline, the node is not expanded. This heuristic helps

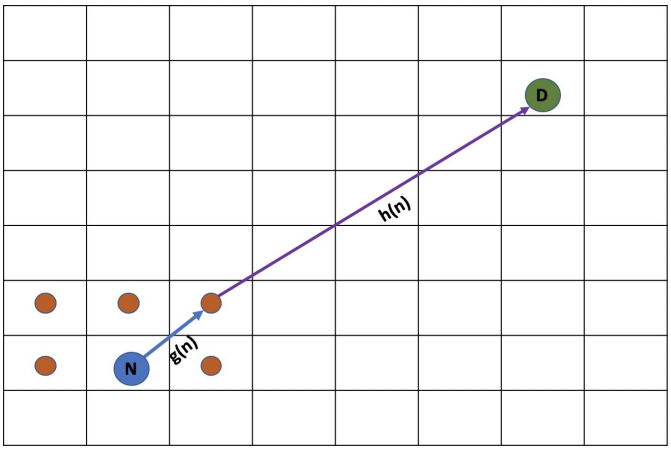


Fig. 2.3: Finding an approximate shortest-length path from *N* to destination using *A** algorithm through centroids.

us to prune the path finding search region. Using this heuristic, we find the potential paths which have the mean of the path in the reasonable range aligned with provided deadline.

As mentioned earlier, edge weights are represented based on mean and variance of the traffic flow on that edge at the query time. Also, each path is the finite sequence of edges. Therefore, paths from a specific source (*C*) to a destination (*D*) are presented as nodes (m_p,v_p) in the mean-variance plane (Figure 2.4).

In the mean-variance plane, the horizontal axis represents the mean and the vertical axis represents the variance. Each small rectangle represents one of the candidate paths for a specific source, destination pair.

Paths may vary from the one with highest variance and lowest mean (marked as b) to a path with highest mean and lowest variance (marked as a)in the mean variance domain shown in Figure 2.4. Paths are in a convex hull and the best path is somewhere in the convex hull between the extreme points. Convexity certifies that in the search region, there can be only one optimal solution which is globally optimal [4,6,26]. Then based on the cost function and agent's goal, one of these paths is selected as the best path which we explain in further sections.

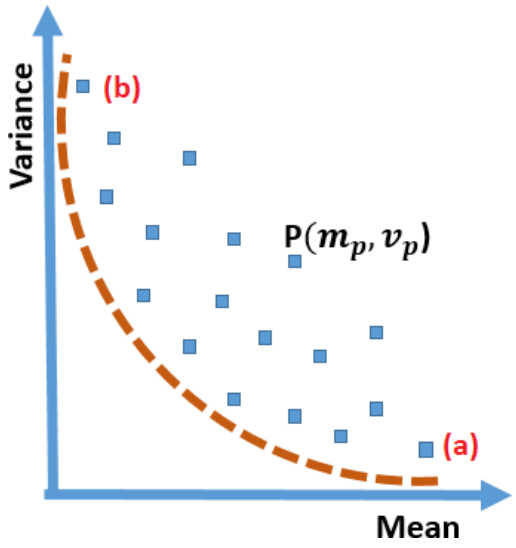


Fig. 2.4: Paths from a specific origin (O) to a target (T) are presented as nodes (m_p,v_p) in the mean-variance plane.

2.4.5 Cost Function

As explained before, there may be more than one path between two nodes and each path has specific characteristics (paths are found based on 3.4.7). The main objective is to find a path with minimum expected cost when agents have more than one option. In order to minimize the paths' expected cost, we need to have a function which models each path's cost $Cost(t)$ along with the probability of arriving by that time $f_{path}(t)$. Expected cost of a path is found using Equation 2.8.

$$ExpectedCost(t) = cost(t) * f_{path}(t) \tag{2.8}$$

For modelling paths' cost $Cost(t)$, we studied three main classes of cost functions: a) linear, b) exponential, and c) step cost function and we discuss the characteristics of each one in the subsequent sections. Obviously, modelling paths' cost is not limited to the cost functions we discuss here and any cost function can be applied either by combining linear, exponential, and step function or by directly putting $Cost(t)$ in Equation 2.8.

For finding the probability of each path $f_{path}(t)$, we considered travel time on edges as a continuous random variable which follows a normal distribution [6, 11, 22, 40]. The Probability Density Function (PDF) is used [41] to define the probability of travel time random variable at each specific time (described in Equation 2.9). The Probability Density Function is normally used to specify the probability of the continuous random variable falling within a particular range of values [41,42].

$$f_{path}(t|m_{path}, v_{path}) = \frac{1}{\sqrt{2\pi v_{path}}} e^{-\frac{(t-m_{path})^2}{2v_{path}}} \tag{2.9}$$

Linear Cost Function

In the linear cost function model, the cost of the path increases linearly by travel time (Equation 2.10). The longer the travel time (t), the more expensive the path is. In this model, agents provide the origin and destination of their trip, and the goal is finding a path with minimum cost. The expected cost is calculated using Equation 2.11. The amount for f_{path} (t_{path}) comes from Equation 2.9 which is PDF of travel time random variable (t).

$$cost(t) = t \tag{2.10}$$

$$ExpectedCost(t) = cost(t) * f_{path}(t_{path}|m_{path}, v_{path}) = \int_{-\infty}^{+\infty} t f_{path}(t)dt = m_{path}$$

Therefore, if we model cost as linear, expected cost of the paths are equal to average travel time of those paths. In that case, neither deadline and nor agent's goal plays a role here. All matter is mean of the path. It even removes the effect of variance of travel time of paths.

Exponential Cost Function

Exponential cost function refers to the case where the cost of a path increases rapidly by travel time. Equation 2.12 shows the exponential cost model based on travel time (t), and Equation 2.13 is used for calculating the expected cost. In Equation 2.12, k is the steepness of the exponential cost increase.

$$cost(t) = e^{k*t} \tag{2.12}$$

$$ExpectedCost(t) = cost(t) * f_{path}(t_{path}|t_{path}|m_{path}, v_{path}) = \int_{-\infty}^{+\infty} e^{k*t} f_{path}(t)dt = e^{k(m_{path} + \frac{v_{path}}{2})} \tag{2.13}$$

Based on the result of Equation 2.13, minimizing the expected cost depends on minimizing the linear combination of mean and variance in accordance with cost steepness parameter k.

Even though modelling cost as exponential considers the effect of variance in path planning, hence it always picks the path with minimum m_{path}, v_{path} at query time and other parameters such as deadline and agents' goals are not in the picture of decision making.

Step Cost Function

Another way of modeling the cost function is to penalize the paths which reach the destination after the deadline. In this case, a step function is used to model the cost (Equation 2.14). In Equation 2.14, u represents a step function [41], d stands for the desired arrival time, and t is travel time random variable which is shown in Equation 2.15. Then expected cost of each path is calculated using a step function and the probability of reaching deadline using Equation 2.16.

$$cost(t, d) = u(t - d) \tag{2.14}$$

$$u(t - d) = \begin{cases} 1 & if \quad t > d \\ 0 & if \quad t < d \end{cases} \tag{2.15}$$

$$ExpectedCost(t) = cost(t, d) * f_{path}(t_{path}|m_{path}, v_{path}) = \int_{-\infty}^{+\infty} u(t - d) f_{path}(t)dt \tag{2.16}$$

Since the step function does not consider any penalty if the agent reaches the destination before deadline, the cost in the interval of $[-\infty, d]$ is zero and Equation 2.16 is re-written as Equation 2.17. Equation 2.17 is equal to Cumulative Density Function (CDF) of Standard

Normal Distribution [41, 42]. Based on Equation 2.17, when there is a set of paths from a specific origin to a destination the path with minimum expected cost is the path that maximizes Equation 2.18.

$$ExpectedCost(t) = \int_{d}^{+\infty} f_{path}(t)dt = 1 - \Phi\left(\frac{d - m_p}{\sqrt{v_p}}\right) \tag{2.17}$$

$$\Phi\left(\frac{deadline - m_{path}}{\sqrt{v_{path}}}\right) = \Phi(path) \tag{2.18}$$

In the step cost model, m_{path} and v_{path} are not linearly related to each other. In order to select the best path, we need to consider deadline, users' goals and query time in the objective function as shown in Equation 2.18.

2.4.6 Modelling Agents' goals

As mentioned in 2.4.3, two agents' goals have been considered in this work and per the discussion in 2.4.5, if we model agents' cost as linear and exponential, agents' goals are not considered in expected cost minimization. Therefore, we focus on step cost function as one of the possible cost functions to study the agents' goals.

Highest Probability Path

If we model the cost as step function and expected cost as Equation 2.17, in order to minimize the expected cost we need to maximize Equation 2.18 which is equal to the Cumulative Distribution Function (*CDF*) of Standard Normal Distribution [42]. *CDF* generates a probability of the random variable (travel time in this case) when distribution is normal to be less than a specific value which is *d* (deadline) here. Then the user's goal is to select a path that maximizes Equation 2.18 which is the path with highest probability of reaching the destination before deadline [4, 6, 30]. For finding the best path, we need to consider the set of candidate paths from origin (*O*) to destination (*D*) in the mean-variance domain based on the approach explained in 2.4.4 in order to select the path which maximizes Equation 2.18.

Smallest Travel Time

In this model, cost function is modelled as the step cost. Then having the desired arrival time $\tau 2$, the possible departure time $\tau 1$, and the probability of making the trip before $\tau 2$, we are looking for the specific time t_G for departure which results in the smallest travel time. Therefore, the departure time is not fixed and it is a specific time t_G bounded in the interval of $[\tau 1, \tau 2]$. For simplicity of referral, we call $[\tau 1, \tau 2]$ interval as the trip interval.

For this model, we modify the deadline variable in Equation 2.18 as the difference of desired arrival time and departure time which is equal to travel duration and rewrite it in the objective function for this model as shown in Equation 2.19. The goal is to minimize the travel duration if departure time is in $[\tau 1, \tau 2]$. In Equation 2.19, ϕ is the argument of the Cumulative Distribution Function (CDF) that makes the CDF equal to the given probability of making the trip before $\tau 2$ and it is fixed here.

$$desired\ arrival\ time - departure\ time = m_{path} + \Phi(path) * \sqrt{v_{path}}$$

$$if \quad departure\ time \subset [\tau 1, \tau 2] \quad (2.19)$$

For finding the best departure time, first we find the latest possible time (τ_L) that the agent can reach destination before deadline. Then we divide the interval of $[\tau 1, \tau_L]$ to sub-intervals in accordance with weekly 10 minute time segments. As we mentioned earlier, mean and variance of the edges vary from one time segment to another. For each time segment in trip interval $[\tau 1, \tau_L]$, we select the path that minimizes the Equation 2.19 for that time segment. Afterward, we pick the time segment which has the minimum cost path in comparison to other time segments. This ultimately results in having the path in a specific time segment which has the smallest travel time. Here is the summary of the work need to be done for this scenario.

- Find the latest time (τ_L) that agent can arrives destination before deadline

- Divide trip interval $[\tau 1, \tau_L]$ to sub-intervals of $[t_1, t_2, t_3, t_4, t_5, \ldots, t_n]$ in accordance with time segment definition.
- For each of the sub-intervals k which is $[t_{(k-1)}, t_k]$:
  - Find the paths from an origin (O) to destination (D) in a case that if they starttheir trip in $[t_{(k-1)}, t_k]$, they can make the trip before deadline.
  - From the set of paths found in last step, select the one which minimizes the expectedcost of the objective function described in Equation 2.19.
- Now for each time segment k we have one path which is the best for that time segment. Then, select the interval which has the path with minimum travel time.

2.5 Experiments and Results

2.5.1 Path Finding based on users' goals

In this experiment, we study how agents' goals in path finding affect the paths selection in different times of the day. For this reason, we pick some source, destination pairs to show the path finding effect. As mentioned in 2.4.2, nodes have latitude and longitude associated to them and the distance between source and destination of sets are in the range of seven to ten miles. Then we find possible candidate paths between each source and destination based on 2.4.4. Two main user's goals are modelled in picking the best path which are the path with highest probability of reaching destination before deadline and the path with smallest travel time using step cost function. Best path based on linear cost function and exponential cost function are also shown for comparing different path finding options. As mentioned in 2.4.5 (Linear) and 2.4.5 (Exponential), best path based on linear cost function is the path with minimum mean and best path based on exponential cost function is the path that minimizes the linear combination of mean of variance of the path.

Highest Probability Path

Given the deadline (set as the expected amount of time after starting the trip) and cost function to model the paths' costs, in this experiment we aim to see the path selection for each cost function (linear, exponential and step cost function).

Figure 2.5 and Figure 2.6 show the results of this experiment for three different time slots of Friday and Tuesday for two sets of randomly picked nodes. Each circle represents one of the possible paths between the source and destination. Considering the set of paths between a specific source and destination, we want to find the highest probability path based on the linear cost function, the exponential cost function, and step cost function. A red triangle identifies the path with the least mean to satisfy the linear cost function criteria. A green trapezoid is the best path based on exponential cost model which considers the both mean and variance of the path. A black rectangle identifies the path with highest probability and considers mean, variance and deadline.

The results of Figure 2.5 and Figure 2.6 show that the characteristics of paths for the same source and destination nodes changes in different times of the day. It indicates that how traffic on the paths changes the characteristics of those paths during different times of the day.

An interesting pattern in both Figure 2.5 and Figure 2.6 shows us that when we query for best paths in rush hours, the difference between linear, exponential and highest probability path is large. While in non-rush hour times, these paths are closer to each other and there is not a significant difference between them. This means that having a realistic cost function model along with considering the deadline helps in finding better paths in rush hour. In non-peak times, since traffic is low, paths are similar to each other and navigation might not be that crucial. Having a good cost function modeling and a wise criteria of picking the best path is crucial when paths are congested.

Another interesting point is the way different models pick the best path. Linear model focuses on picking the path with minimum mean, while in some cases like Fig2.(a) the path might have a high variance. Exponential model considers mean and variance but it does

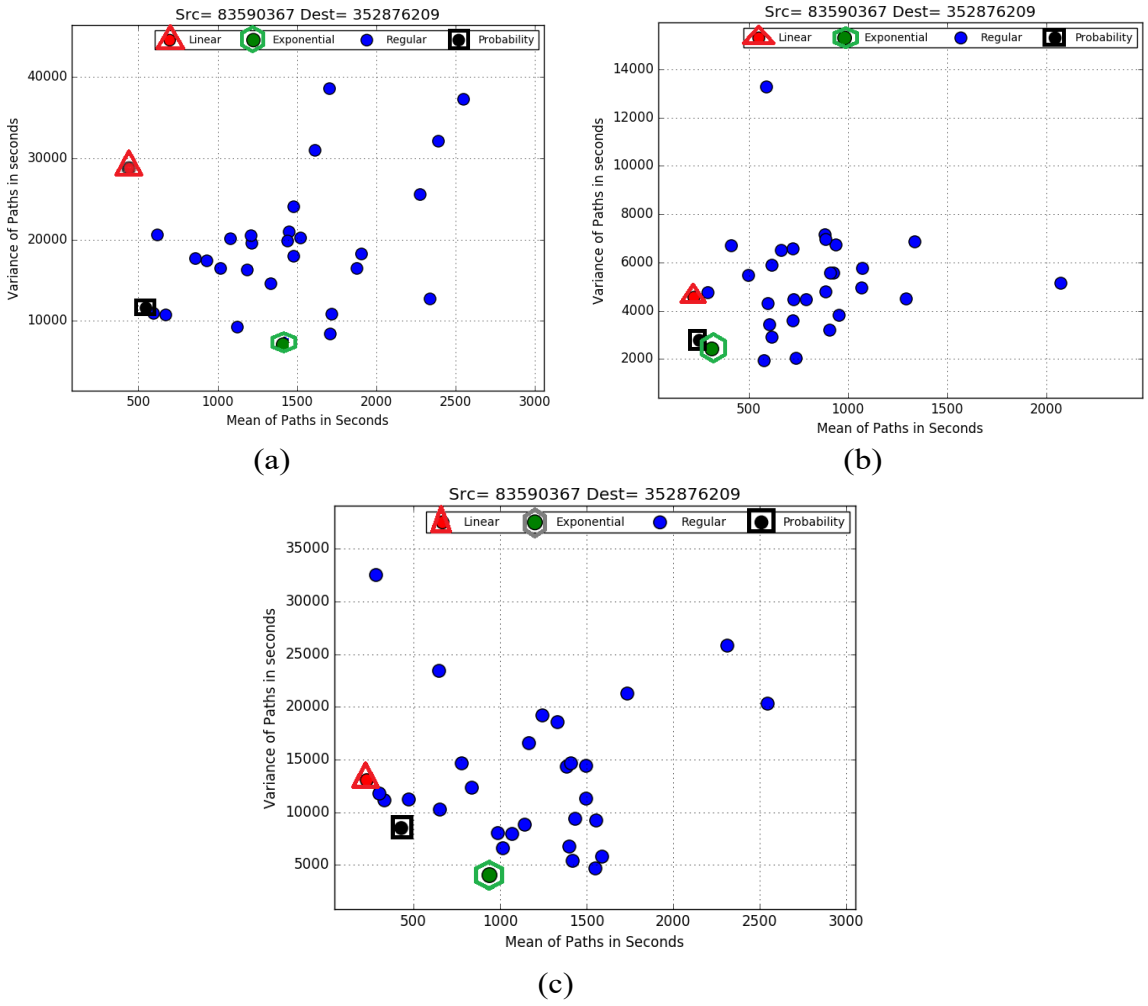


Fig. 2.5: Selected paths for different query times of Friday for Source node=83590367 and Destination node=352876209 based on each cost function. Query times from left to right the times are: a) 8:00, b) 15:00, and c) 18:10 PM. Deadline is set as 1200 seconds after start time.

not consider the deadline. Therefore, in Figure 2.5.(a), Figure 2.6.(a), and Figure 2.6.(c) the paths selected by exponential model all violate the deadline. Highest probability path, pick the least risky path which makes the deadline without a high variance which is what we are expecting.

Smallest Travel Time

In this experiment, we consider an agent who wants to have the smallest travel time

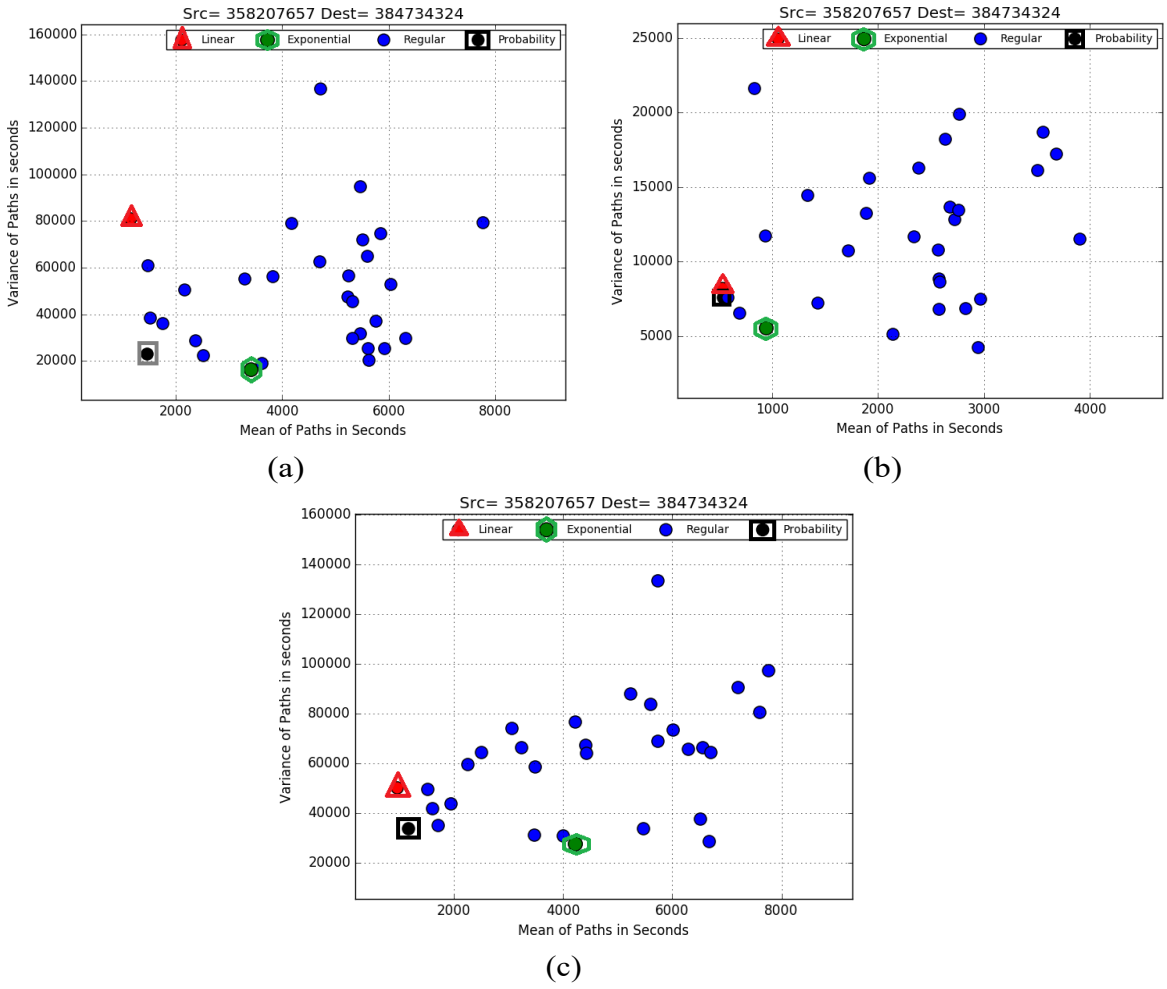


Fig. 2.6: Selected paths for different query times of Tuesday for Source node=358207657 and Destination node=384734324 based on each cost function. Query times from left to right the times are: a) 7:30, b) 11:40, and d) 17:45. Deadline is set as 1400 seconds

within a desired arrival interval. We determine the best time to start the trip and which path yields the smallest travel time. Figure 2.7 and Figure 2.8, show the results of this experiment for three different time slots of Monday and Wednesday for two different sets of nodes with all three mentioned cost functions. Each circle represents one of the paths between the source and destination. In each figure, pink rectangle represents the path with smallest travel time, green trapezoid is the best paths if cost function is exponential, and red triangle is the best path based on linear cost function model.

Similar to the findings in the previous section, Figure 2.7 and Figure 2.8 shows that

in rush hours selected paths for different models differ from each other significantly, while in non-peak hours they are almost the same. It emphasizes the effect of congestion in busy hours and how it can change the weights on edges of the graphs. As in the previous experiment, the linear model picks the path with smallest mean, while that path might have a high variance like Figure 2.7.(b). Exponential path does not consider deadline and it may pick a path which does not make the trip within the desired arrival. Smallest travel time path considers mean, variance, desired arrival time and probability which user likes to consider in order to make the best decision in path finding. Desired probability for this experiment is considered as 85 percent. This means that we are interested to find the paths that have the smallest travel time and within the chance of 85 percent can make the trip before deadline (85 percent is a number we picked to keep the experiments consistent here, it can be any probability).

Another finding, indicates that smallest travel time path sometimes is a risky decision as it has a high variance of reaching destination before deadline. For example, in Figure 2.7, Figure 2.8 the smallest travel time path has the higher variance in comparison with the path exponential path.

2.5.2 Compare paths with shortest-length path

In the realm of path planning, shortest-length path is always a practical option and for most of the users, it is the primary choice for path finding between two points. Hence in our context, it can be used as a baseline to see how our paths are different from the shortest-length path. In this experiment, we compare means and variances of shortest-length path with highest probability path and smallest travel time path. The experiment has been done for departure times of 17 : 20 on Tuesday, Wednesday and Friday for 10 different sources and destinations in the range of [7, 10] mile (A to J) to compare the changes across different days. The time is considered to be in the peak time, therefore it is a good case for comparison as the traffic is high. For highest probability path deadline is considered as 1400 seconds (based on the average time takes to get from a source to destination with the distance for 7 to 10 miles in rush hour) and probability for smallest travel time path is 85

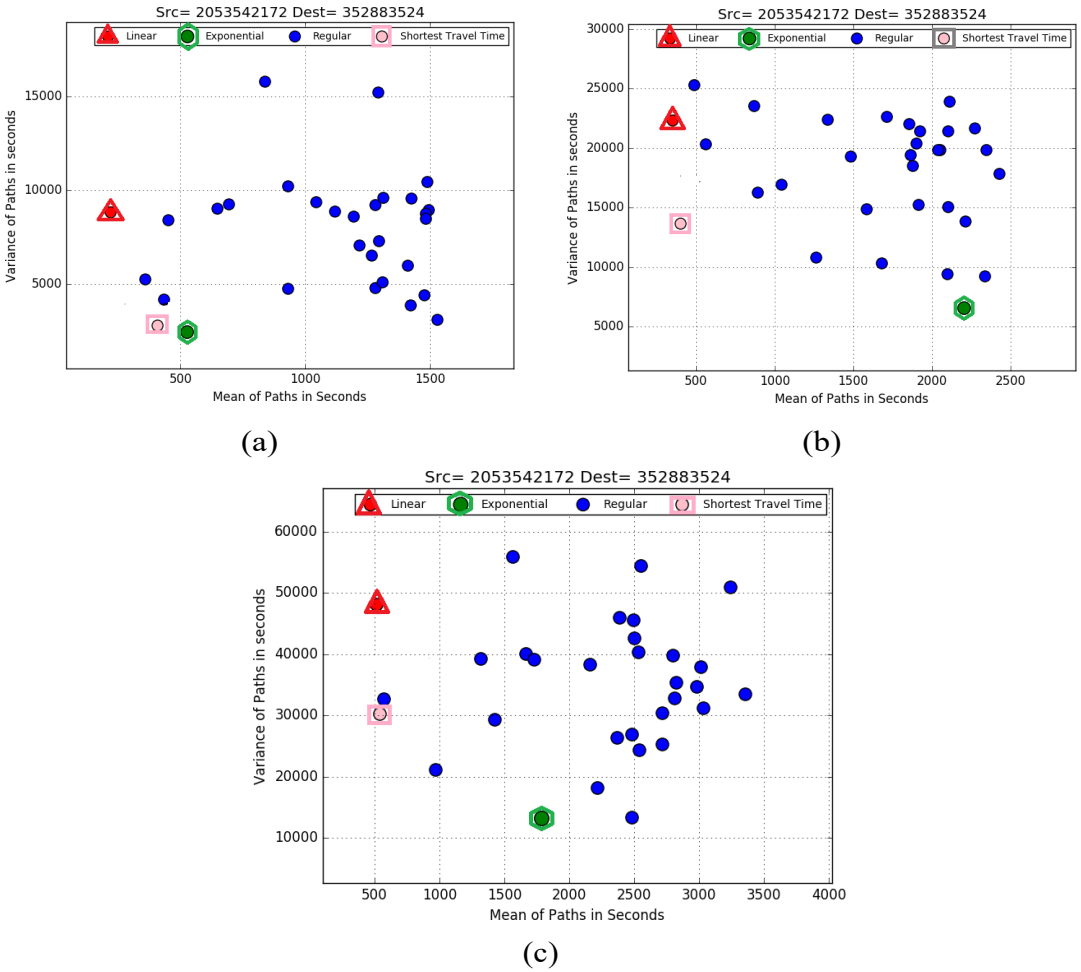


Fig. 2.7: Selected paths for different query times of Monday for Source node=2053542172 and Destination node=352883524. Query times from left to right are: a) 6:40, b) 8:10, and d) 18:00. Desired arrival time is within the 1600 seconds after query time. Best time for start the trip to have the smallest travel time is as follow: a) 6:40, b) 8:23, and c) 18:21.

percent.

As it can be seen from Figure 2.9, Figure 2.10 and Figure 2.11, Figure 2.12 shortest-length paths have the higher mean in comparison with highest probability paths and smallest travel time paths. This is obvious as shortest-length path mostly is the first choice of the majority of drivers for reaching their destination regardless of how congested the path might be. Therefore, it is more congested and has higher mean in comparison with other paths. Another interesting finding is that the results show that in all of the cases smallest travel

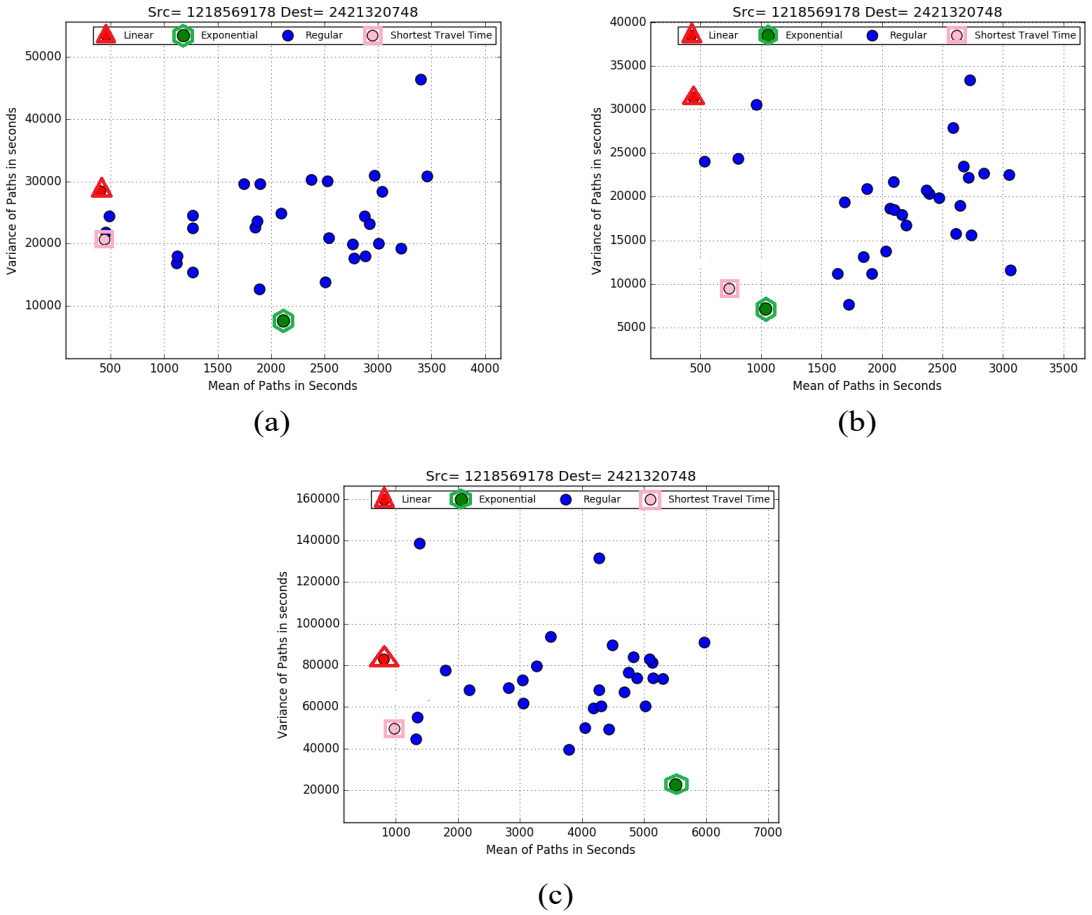
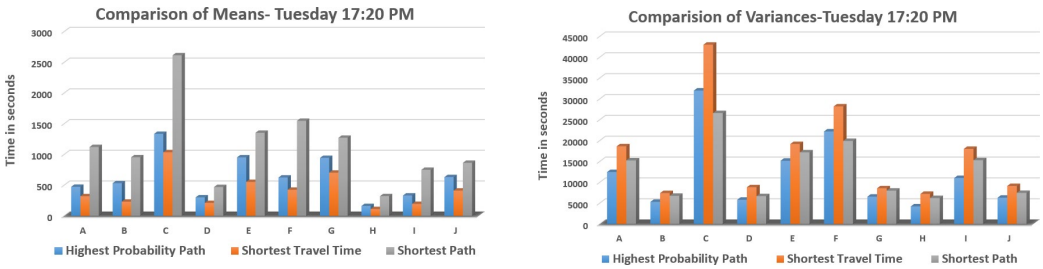


Fig. 2.8: Selected paths for different query times of Wednesday for Source node=1218569178 and Destination node=2421320748. Query times from left to right are: a) 7:40, b) 11:10, and d) 17:30. Desired arrival time is within the 2000 seconds after query time. Best time for start the trip to have the smallest travel time is as follow: a) 8:01, b) 11:10, and c) 18:09.

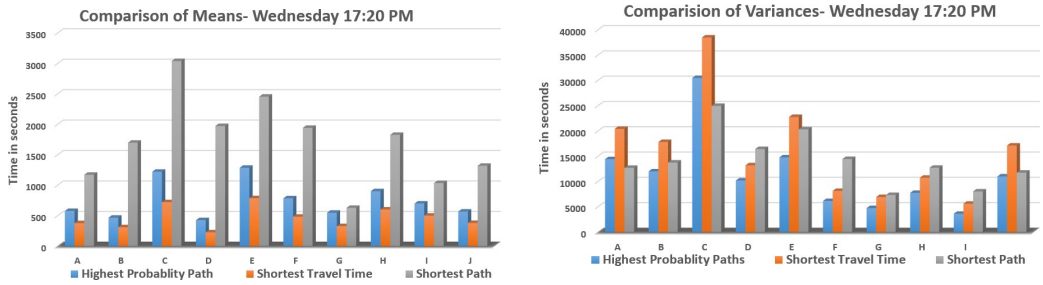
time paths have shorter mean and larger variance in comparison with highest probability paths. It is clear that highest probability path is the least risky path while smallest travel time optimizes for the path with smallest travel time. We note that the smallest travel time path might have high variance and ultimately have a high risk (sample use case of this explained in 2.2).

After the above experiment, we repeat the experiment for the bigger sample size in a different time slot and compare the highest probability, smallest travel time with shortest-



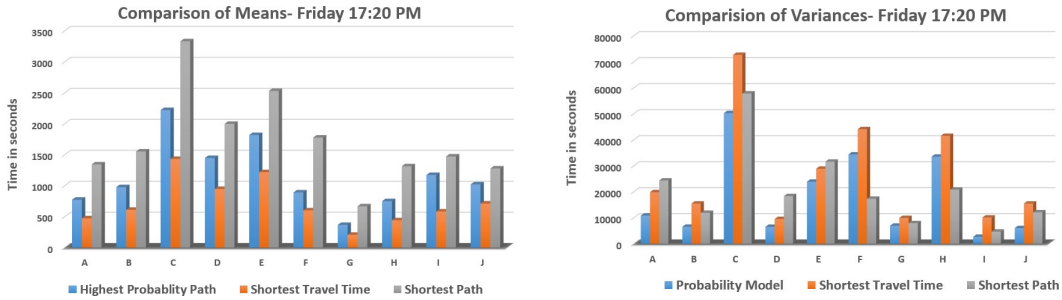
(a) (b)

Fig. 2.9: Comparison of mean and variance of highest probability path, smallest travel time path and shortest-length path in 17:20 PM of Tuesday for 10 different source and destinations (A to J).



(a) (b)

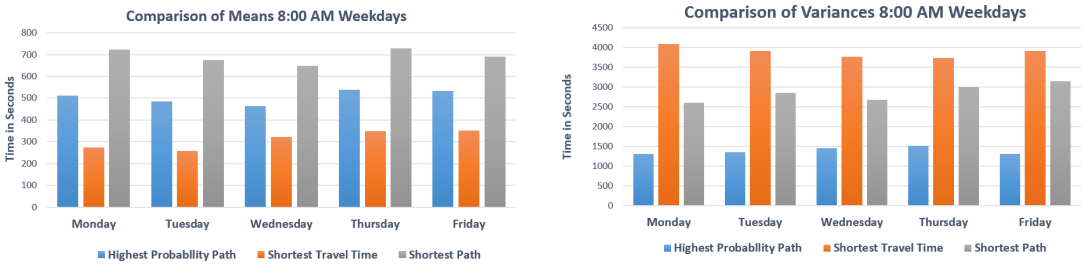
Fig. 2.10: Comparison of mean and variance of highest probability path, smallest travel time path and shortest-length path in 17:20 PM of Wednesday for 10 different source and destinations (A to J).



(a) (b)

Fig. 2.11: Comparison of mean and variance of highest probability path, smallest travel time path and shortest-length path in 17:20 PM of Friday for 10 different source and destinations (A to J).

length path. We considered 100 random pairs of source and destinations that are in the distance of 10 to 12 miles in different areas of Salt Lake City. As we learned earlier, the two main rush hours (morning and evening) have the similar pattern, thus we just pick the morning rush hour. The result for each of the goals is averaged over all 100 pairs in weekdays (Monday through Friday). Desired travel time for the pairs is considered as 2200 seconds (again based on the average time takes to get from a source to destination with the distance for 10 to 12 miles in rush hour) and desired probability for this experiment is considered as 85 percent. The following is the demonstration of means and variances for each day of the week.



(a) (b)

Fig. 2.12: Comparison of average mean and variance of highest probability path, smallest travel time path and shortest-length path in 8:00 AM of Weekdays for 100 different source and destinations.

As it can be seen from the graphs, we have the same pattern as the previous test. Shortest-length path is not performing well as it just tries to pick the minimized length path even if it is congested. Smallest travel time paths have less mean in comparison to highest probability paths while they have higher variance which makes highest probability paths more secure option but longer. Even though shortest-length paths have reasonable variance, their high mean value makes them not a good option to pick.

2.6 Conclusion

This research represents path finding in the real world domain in which edge weights

are not fixed but are stochastically affected by the time of the day/week. Stochastic edge costs are considered as independent Gaussian random variables, whose distributions are extracted from data monitored by the Utah Department of Transportation. Inspired by time-dependent traffic situation, we parameterize these distributions by time, which allows us to speak of time-dependent path costs and study the problems of reaching a goal by a deadline, and delaying departure to minimise traversal-to-goal time. The best path is the path with lowest cost. The cost is based on travel time which highly depends on the level of congestion in the network and congestion highly related to the time of the day/week. Three different cost functions (linear, exponential, and step cost function) are investigated. The best path is the path with lowest cost with respect to agents goals. Since the graph is interconnected, optimizing for lowest cost on all possible paths is not feasible, therefore, we prune the search region to do cost minimization on the subsets of paths. In pruning phase, for any node *N* to be expanded we consider an approximate path from source to destination through node *N* and if the mean of that path is more than the provided deadline, we don't expand that node. Users can pursue two main goals: 1) picking the least risky path and 2) picking the smallest travel time along with awareness of when to start the trip. Results show that in rush hour time when the congestion is high, using a smart/realistic method of path finding is crucial while in non-peak hours paths are almost the same. In addition, we demonstrate that a suitable path finding approach must consider different aspects such as path's mean, path's variance and the deadline to provide optimal options. Removing any of these factors may result in having a path which either violates the deadline or has a very high variance. We compare the mean and variance of highest probability paths and smallest travel time paths with shortest-length paths at the same query time. This experiment proves that the shortest-length path is not always the best path due to the fact that it is highly congested in rush hour times as it is traditionally the first choice of most drivers. Highest probability path has less variance because it takes the most secure path, while the smallest travel time has the lowest mean and might have a high variance.

CHAPTER 3

Practical City Scale Stochastic Path Planning with Pre-Computation

3.1 Abstract

This work [1] prescribes a practical city scale path planning in the presence of traffic delays. Edge weights are not fixed and are stochastically defined based on the mean and variance of travel time on them. One main objective is to minimize the running time of the overall procedure at query time, and hence the response time to the shortest-path queries are crucial. Agents are car drivers who are moving from one point to another point at different times of the day/night. Agents pursue two types of goal. The first group desires the path with the highest probability of reaching their destination before their desired arrival time. They look for the most secure route. The second group are the agents who are open to take a riskier decision if it helps them in having the shortest en-route time. Pre-computation and approximation has been used in order to scale the path planning process and make it practical in city scale route planning. The city graph is partitioned to smaller groups of nodes using community detection and clustering methods, and each partition is represented by its exemplar. In query time, source and destination pairs are connected to their respective exemplars and the path between those exemplars is found. Paths are stored in distance oracles for different time slots of day/week in order to expedite the query time. Distance oracles are updated weekly in order to capture the recent changes in traffic. The proposed framework handles queries in real time while the approximate paths are 3 to 5 percent longer than exact paths.

[1]The first version of this work is presented and published in proceedings of 13th International Conference of Agents and Artificial Intelligence (ICAART 2021) as a full paper. DOI: 10.5220/0010394104540463
The extended version has been submitted to the journal of Lecture Notes in Artificial Intelligence (LNAI), Springer

3.2 Introduction

When the volume of traffic is greater than the capacity of the streets congestion is likely to happen. In the era of accelerated urbanization around the world, practical path planning approaches considering the level of congestion are more critical than ever before [43, 44]. In modeling city scale traffic, congestion changes throughout the day which results in having uncertain travel time on the road segments [6, 7]. One of the main approaches is stochastic path planning framework which city is modelled as a graph and the graph's edge weights are the mean and variance of the travel time on each edge during the given time interval. Travel time is defined as a random variable as its value depends on the time of the day and day of the week. Travel time variability is one of the most useful indicators to measure the performance and reliability of a transportation systems [45].

In this work we adopted stochastic path planning framework and modelled agents as car drivers which pursue different goals. Two types of agents have been modelled for a given origin, destination pair: a) the agents that look for a path that maximizes the probability of reaching the destination within a given deadline and b) the agents who look for the shortest en-route time while the probability of making the deadline is at least a given threshold. A path planner satisfies the agents' goals by minimizing the path costs over the travel time random variable toward the agents' goals. Edge weights are defined as mean and variances of travel time for each time slot. While the mean shows the average traffic on the edge, variance reflects how far the values are spread out from their average value with respect to all of the changes and uncertainties in network congestion. The data for extracting edge weights is the historical traffic logged data of the preceding 12 months.

This paper focuses on a scalable algorithm for stochastic path planning under congestion. The main objective of this work is to minimize the query time in order to handle the large number of requests in the real world domain. The approach uses a stochastic path planning framework and improves the query time utilizing pruning, graph partitioning, preprocessing and approximation techniques. A key part is to find region-based partitions in the city graph and represent each region with an exemplar. Instead of planning a path

from a source node to the destination node, we connect each node to the closest exemplar considering the direction to the destination and find a path between exemplars. This enables responding to path finding queries in real time with an approximate path instead of an exact path. In the pre-processing phase, all of the paths from every pair of exemplars for every time slot of each day of the week is stored by the distance oracles. An approximate distance oracle is a data structure that efficiently answers path planning queries in graphs. Therefore, in the query time, a source and destination are connected to their corresponding exemplar, and the path from two exemplars is retrieved. Distance oracles are updated every week with respect to the data of the preceding 12 months in order to reflect the recent traffic pattern changes such as seasonality, events, and weather conditions on the congestion of each edge.

3.3 Previous Work

Stochastic path planning in scale is challenging in real time due to the high volume of queries and dynamic nature of traffic. Fan et al. [46] determine the optimal route by selecting the best next direction at each junction using stochastic dynamic programming problem. Their approach uses a standard successive approximation algorithm. The problem is their algorithm has no finite bound on the maximum number of iterations to converge on cyclic road networks.

Nie and Wu [47] proposed a framework which calculates the optimal a-priori path in query time using a multi-criteria label-correcting algorithm by generating all non-dominated paths based on the first-order stochastic dominance condition (FSD). The proposed algorithm provides an approximate solution in pseudo-polynomial time in the best case, but since the number of FSD non-dominated paths grows exponentially with network size; the run time of the solution is exponential in the worst-case.

Nikolova et. al. [26] presented a framework for reliable stochastic combinatorial optimization that includes mean-risk minimization and probability tail model. Their algorithm is independent of the feasible set structure and uses solutions for the underlying linear (deterministic) problems as oracles for solving the corresponding stochastic models. They

showed the problem can be solved in $n^{log(n)}$ time if we assume distribution of the travel time random variable is Gaussian. The solution utilizes pre-computation in path planning but still the time complexity of their provided solution are $n^{log(n)}$ which is not practical in real world domain.

Samaranayake et al. [48] presented a label-setting approach to speed up the computation of stochastic path finding based on zero-delay convolution, and localization techniques for determining an optimal order of policy computation. Their proposed approach is still too slow to be implemented in scalable navigation systems.

Gutman, et. al. [32] used pruning as a technique to speed up the stochastic planning process. In their model, a node is expanded if its reach value is larger than some threshold. A node with higher reach is a node that appears the most in the shortest paths between pairs of nodes. Reach values are obtained in a pre-processing step. Arc-flag acceleration method [49] also uses pruning to tackle the stochastic path planning at scale. They divide the the graph into a set of regions and a Boolean vector representing roads. For each region, the corresponding road value is true if the edge is used by at least one path ending in the corresponding region. Then, any edge without the Boolean corresponding to the region is pruned. One of the major limitations of both mentioned methods is it takes a long time to respond to changes in the network due to the vast amount of computation, even in pre-processing phase.

Contraction hierarchies [7] and arc-flags [49] use bidirectional search in pre-processing. However, speedup techniques that rely on bidirectional search are not applicable to the stochastic path planning problem, because the final and intermediate solutions are a function of the remaining time budget and remaining time budget is not deterministic. When performing a bidirectional search, the reverse search needs to stochastically estimate the time budget at each step, hence there are cases where bi-directional search might not converge.

PACE [50] is a path centric stochastic path planning which estimates the cost of paths instead of edges. Their path finding builds upon the ’path + another edge’ pattern to find

paths for each source, destination pairs. They store the paths between possible pairs from trajectory data and retrieve in query time. Their approach has the following shortcomings for use in the real world domain: a) estimating the costs of paths highly depends on trajectory data which may suffer from sparsity [51], b) best path is picked after finding candidate paths and estimating the joint cost distribution of the paths which is not real time, and c) their only model finds high probability path.

Lim et. al. [6] showed how to solve the scalable stochastic path finding in $\Theta(nlog_n)$ time where n is number of nodes in the network. They assume edge weights in the city graph are independent and distribution of travel time is Gaussian. Their approach is quasi-polynomial with a rate of growth between polynomial and exponential and use a data structure that occupies space roughly proportional to the size of the network for storing distance oracles.

Ahmadi et. al. [43] proposed a framework that can answer large scale stochastic path planning queries in real time using graph clustering, pruning, pre-computation, and approximation with two agent goals of highest probability path and shortest en-route time. They used historical traffic data and consider the changes of traffic at different time slots of a day in each day of the week. They reduce the city graph to partitions and pre-compute paths for representative of partitions. Pre-computed paths are updated every week in order to reflect the recent changes. Current work extends Ahmadi et. al.'s work and enriches the framework in the following ways:

- Expanding graph partitioning methods by adding *meanshift* clustering and *Walktrap* community detection methods to cover variations of graph partitioning methods on the Salt Lake City graph.
- Adding direct (elbow) and statistical (gap statistics) methods for deciding the optimal number of clusters on the city graph which is crucial for clustering algorithms.
- Studying the effect of four exemplar selection methods on approximate paths: a) highest traffic, b) highest reach, c) closeness centrality and d) random walk centrality.

- studying the impact of the following additions on the overall performance of the framework.

3.4 Framework

The main idea behind scaling of the path planning process is to partition the city to smaller parts and get an exemplar for each cluster that can represent the nodes of the cluster. Then instead of planning a path from each source node to a destination node, we connect each node to one of the neighboring exemplars and find a path between the exemplars. The paths between exemplars are pre-computed for faster response in query time.

3.4.1 City and Edge Weights

The city is modelled as a directed graph consisting of a set of vertices, V , which represent road intersections and edges, E, that represent road segments between vertices. We consider the city graph to be planar (i.e., edges intersect only at their end points). If we consider the number of nodes in the planar graph as $|V|$ and the number of edges as
$|E|$, the relationship between them is $|E| << |V|^2$ [6, 26]. Associated with each edge of the graph is a travel cost used as the edge weight in our directed graph.

We use expected travel time as the cost of an edge. Edge costs are not fixed and are represented as mean and variance of expected travel time random variable.

$$E \subset V^2 \tag{3.1}$$

$$W_e(t) = (m_e(t), v_e(t)) \tag{3.2}$$

Edge weights are represented as a probability distribution of travel time rather than a fixed value. Travel time random variable is a tuple of mean and variance of the expected travel time on each edge. We assume travel time on edges are independent and follow Gaussian random variable shown in Equation 3.3 [6, 52]. The mean of a path is the sum of the means of all edges included in the path (Equation 3.4) in which t is query time and

δ is the time takes to reach to any edge from the query time. Equation 3.5 shows how to calculate the variance of the path. Since we assume edge weights are independent from each other, then $cov(X_i, X_j)=0$ for $i \neq j$ and Equation 3.6 is the result. Based on Equation 3.6, the variance of a path is the sum of variance of all edges included in the path as shown in Equation 3.7 [6,26].

$$t_e \sim N(m_e, v_e) \tag{3.3}$$

$$m_{path}(t) = \sum_{e \in path} m_e(t+\delta) \tag{3.4}$$

$$var\left(\sum_{i=}^{n} X_i\right) = E\left(\left[\sum_{i=}^{n} X_i\right]^2\right) - \left[E\left(\sum_{i=}^{n} X_i\right)\right]^2 \tag{3.5}$$

$$var\left(\sum_{i=}^{n} X_i\right) = \sum_{i=1}^{n}\sum_{j=1}^{n} cov(X_i, X_j) = \sum_{i=}^{n} cov(X_i, X_j) = \sum_{i=}^{n} var(X_i) \tag{3.6}$$

$$v_{path}(t) = \sum_{e \in path} v_e(t+\delta) \tag{3.7}$$

Once mean and variance of a path is known, we can plot possible paths between any source destination in a mean-variance plane (shown in Figure 3.1).

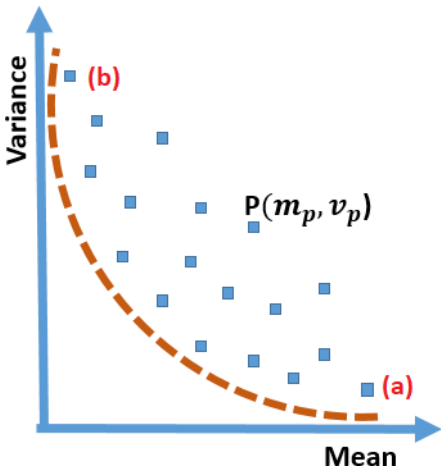


Fig. 3.1: Paths from a specific origin (O) to a target (T) are presented as nodes (m_p,v_p) in the mean-variance plane.

Stochastic dependency between adjacent edges can be modelled by adding extra edges between dependent edges. This new edge captures the correlation between the two correlated distributions, maintaining the property that the variance of a path is the sum of the variances of all the edges in the path. Suppose that adjacent edges X_i and X_j are dependent. Based on Equation 3.8, if we want to sum the variance of X_i and X_j, $Cov(X_i, X_j$ is non-zero as edges are dependent. Therefore, we add one edge with mean 0 and variance of $2 * Cov(X_i, X_j$. The number of nodes and edges grows by one for each pair of correlation.

In our model, we do not transform the graph, and the assumption is that the dependence between edges affects the variance of the consecutive edges. For example, if edge A, has a strong dependency with edge B and congestion on edge A causes congestion on edge B, then the variance on edge B is high enough to represent this dependence [4, 6, 26, 53].

$$var(X_i + X_j) = var(X_i) + var(X_j) + 2 * Cov(X_i, X_j) \tag{3.8}$$

For finding the mean and variance of a path, a sliding time window is used to imply the cost of each edge in the path depends on the amount of time that took to reach it, not just the initial departure time. For example, if we look at the path at time a and take δ to reach the 4^{th} edge, the cost of the 4^{th} edge is considered at the time of $a + \delta$.

3.4.2 Traffic Data

Edge weights are based on mean and variance of the expected travel time on them. In order to extract edge weights, yearlong traffic data from Utah Department of Transportation (UDOT) [36] has been used which is logged in 10 minute intervals for each day of a week on Salt Lake City, Utah.

3.4.3 Open Street Map

For building the city graph, we used Open Street Map data [54]. Open Street Map is a free editable geographic map of the city. Open street map data data structure has three main components: a) *nodes* which is a single point defined by latitude and longitude, b)

ways which is a list of nodes, and c) *relations* which relates two or more data elements like a route, turn restriction, traffic signal or an area. Open Street Map represents physical entities on the ground like buildings, roads, intersections, and bridges. The data structure it uses is based on entities and for each entity there are multiple tags describing the characteristics of that entity.

3.4.4 Agents

In this framework, agents are car drivers, capable of pursuing different goals. We can technically model any type of agents' goals and incorporate it in the path finding framework. We modelled two following goals as they have interesting characteristics in the path planning domain.

- Risk seeking agents: the agents who are open to take a riskier route if has the shortest en-route time. These agents are flexible in leaving time.
- Risk averse agents: agents who are not willing to take risk and look for the path with highest probability of reaching destination before a desired arrival time, even if travel time is increased.

To make these two goals clearer, one example is in the context of a package delivery system. Suppose that we guarantee the delivery of a package by 4 PM, otherwise the customer doesn't accept the delivery, and we pay the shipping costs. In that case, we are interested in picking a path that has the highest chance of reaching destination before the deadline to avoid losing the shipping cost. The other possible case is delivering perishable products. If the product needs to be kept hot or cold, we desire a path that has the shortest en-route time. We are flexible in leaving anytime, but need to have the shortest en-route path while still making the target delivery time.

3.4.5 City Graph Partitioning

In dealing with large scale graphs, one of the possible approaches is to reduce the node set. The reduction process happens through partitioning the graph to a group of nodes

in a way that the group can be represented by one node. For this work, we investigated 1) unsupervised learning (clustering methods), and 2) community detection methods for partitioning the city. After partitioning phase, an exemplar for each partition is selected. One can argue that community detection is similar to clustering. Clustering is a machine learning unsupervised technique in which similar data points are grouped into the same cluster based on their attributes. So in networks, clustering is merely based on the position of the nodes. On the other side, community detection methods are focused on partitioning the graph based on edges as communities are a group of well-connected nodes that are more strongly connected among themselves than the others.

Unsupervised Learning

A cluster refers to a collection of data points aggregated together due to the certain similarities using unsupervised methods. Given a set of data points, clustering puts each data point into a specific group. In theory, data points that are in the same group should have similar properties, while data points in different groups should have highly dissimilar properties. There are various clustering methods to be used on graphs and in this work, we used *k-means* [55] and *meanshift* clustering methods [56]. In our clustering a node has the following attributes: a) latitude, b)longitude, and c) traffic profile which is historical traffic level on the node (low, medium and high).

Optimal number of clusters Most of the common clustering methods including *k-means* and *meanshift* require the number of clusters (k) to be defined ahead of time. There are various methods for deciding the optimal number of clusters in data including direct and statistical methods. Direct methods like elbow method [55] usually optimize a criterion such as the within cluster sums of square distance. Statistical methods such as gap statistics [57] compare evidence against expectation under random sample of data under uniform distribution.

Elbow method looks at the total within-cluster sum of square distances (WSS) as a function of the number of clusters. WSS is the sum of the squared deviations from each

observation and the cluster centroid (Equation 3.9).

$$\min_{i=1}^{n}(\|x_i - \mu_i\|) \tag{3.9}$$

In general, a cluster that has a small sum of squares is more compact than a cluster that has a large sum of squares. The number of clusters is increased until adding another cluster does not improve the total WSS significantly. Steps of elbow method are as follows:

- Run clustering algorithm on varying values of k.
- For each k, calculate WSS.
- Plot the curve of WSS against number of clusters k.
- The location of a bend (knee) in the plot is considered an indicator of the appropriate number of clusters.

Gap statistic compares the total within intra-cluster variation (W_k) for different values of k and compare it with their expected values under null reference distribution of the data. Null reference distribution of data is the samples of data under uniform distribution which we consider to reject the null hypothesis. Null hypothesis here states that our clusters of the data is same as the clusters of a uniform distribution of the data. W_k is the within-cluster sum of squared distances from the cluster means and can be found using Equation 3.10. D_r is the some of pairwise distances for all of the points in the cluster and $d_{ii'}$ represents the distance between every i and i' pairs.

$$W_k = \sum_{r=1}^{k} \frac{1}{2n_r} D_r \tag{3.10}$$

$$D_r = \sum_{i,i' \in C_r} d_{ii'} \tag{3.11}$$

The optimal number of clusters is the value of k that yields the largest gap statistic. The largest gap means that the clustering structure is far away from the random uniform distribution of points. For gap statistics, we first cluster the data by varying the number of k

and compute the corresponding W_k. Then, we generate B reference data sets with uniform random distribution and cluster each of them with varying number of clusters. Compute the estimated gap statistic as the deviation of the observed W_k value from its expected value $W_k{}^*{}_b$ shown in Equation 3.12. Afterward, we choose the number of clusters as the smallest value of k such that the gap statistic is within one standard deviation of the gap at k+1 using Equation 3.13. In Equation 3.13 where s_k is the simulation error calculated from standard deviation of B replicas and found using Equation 3.14.

$$Gap(k) = \frac{1}{B} \sum_{1}^{B} log(W_k^*) - log(W_k) \tag{3.12}$$

$$Gap(k) \geq Gap(k) - s_{k+1} \tag{3.13}$$

$$s_k = sd(k) + \sqrt{1 + \frac{1}{B}} \tag{3.14}$$

K-means Clustering *K-means* is a very popular clustering algorithm because it easily scales to large data sets, guarantees the convergence and easily adapts to the new data points. The *k-means* clustering aims to partition *n* observations into *k* clusters. Clusters are formed to minimize within cluster variance. The centroid (used as the exemplar) is the arithmetic mean position of all the points in the cluster. The *k-means* algorithm starts with a first group of randomly selected centroids, which are used as the beginning points for every cluster, and then performs iterative calculations to optimize the positions of the centroids. It ends when there is no change in the value of centroids or the defined number of iterations has been achieved. As *k* needs to be defined ahead of time, we use both elbow and gap statistics methods to get optimal number of clusters on the graph of Salt Lake City before using *k-means*. The optimal number of clusters using the elbow method is 157 and for gap statistics is 172. Therefore, we set the value of *k* as the average of these two and set the number of k as 162. Figure 3.2 (top) shows the distribution of 162 clusters and the visualization of the clusters on Salt Lake City. As distribution shows, the majority of clusters have 70 to 400 nodes and few large clusters with node size larger than 1000 nodes.

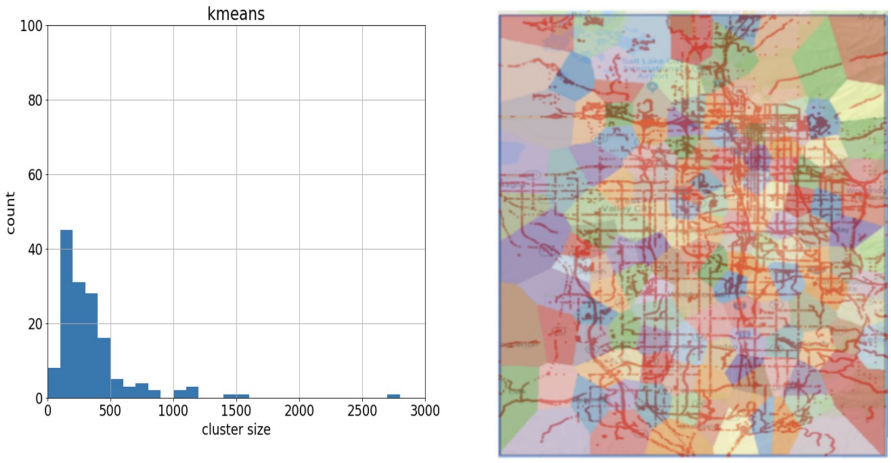


Fig. 3.2: Distribution of main nodes in each cluster and visualization of them on Salt Lake City for *K-means* clustering. Each color represents one cluster.

MeanShift *Meanshift* [56] clustering is a non-parametric centroid-based algorithm, which works by updating candidates for centroids to be the mean of the points within a given region. Unlike other clustering algorithms which assigns the data points to the clusters iteratively, *meanshift* tends groups points towards the mode. Hence, it is also called a mode seeking algorithm. In the context of *meanshift*, mode is the highest density of data points in the region. *Meanshift* uses the concept of kernel density estimation (KDE) [58] which is a method to estimate the probability density function of the data. It works by applying a Gaussian kernel on each point in the data set. Adding up all of the individual kernels generates a probability surface example density function. *Meanshift* is a model free approach which doesn't assume any distribution of the data. It is robust to outliers as it uses kernel density functions. Similar to *k-means*, the number of clusters needs to be defined ahead of time. We run elbow method and gap statistics method on the graph of Salt Lake City with *meanshift* clustering. Elbow method provided 211 and gap statistics's optimal number of clusters were 247. Then we set the average as 229. Figure 3.2 (bottom) shows the distribution and the visualization of the clusters on Salt Lake City. It provided 229 clusters with majority of clusters had the size in the range of 50 to 700 and few large clusters.

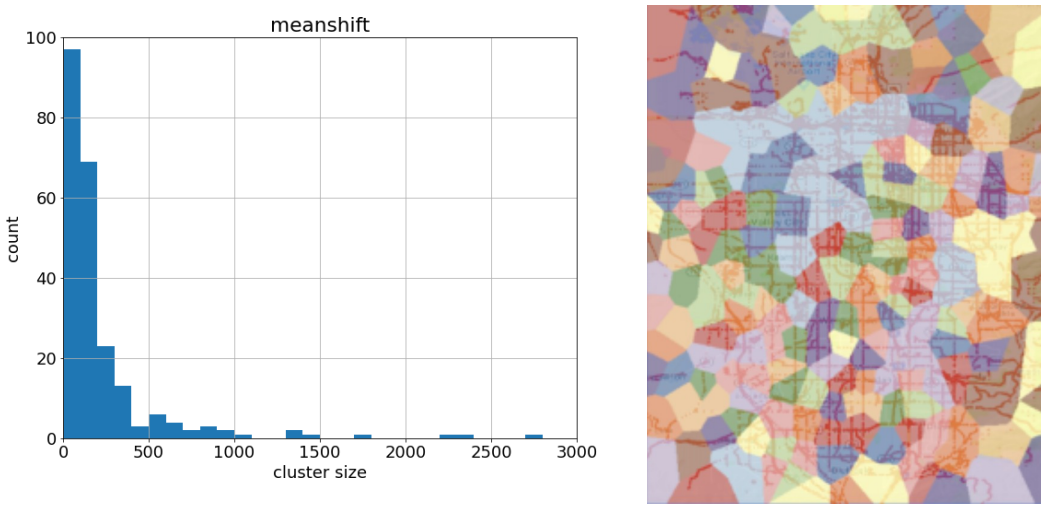


Fig. 3.3: Left: Distribution of cluster sizes in *Mean Shift*. Right: Visualization of clusters on Salt Lake City. Each color represents one cluster.

Community detection methods

A community, with respect to graphs, is defined as a subset of nodes that are densely connected to each other and loosely connected to the nodes in the other communities in the same graph. Depending on the type of the community detection methods, the city graph can be partitioned differently. Major community detection methods are divided into three main categories: a) divisive methods, b) agglomerative methods and c) optimization based methods [59]. Divisive algorithms begin with a complete network and iteratively divide the network into smaller communities. An example of divisive methods is *Leading Eigenvector* [60]. Agglomerative based methods begin by considering each node as its own community and then iteratively combine nodes into larger communities. *Walktrap* [61] and label propagation [62] are the examples of agglomerative methods. Optimization based methods find the optimal set of communities based on an objective function. *Multilevel* [63] is an example of optimization based method. There are a few important definitions which is common in community detection algorithms:

- Modularity measures the strength of division of a network into communities and reflects the concentration of edges within modules compared with random distribution

of links between all nodes regardless of modules. If the number of edges within groups exceeds the number expected on the basis of chance, then modularity is positive. If modularity value is zero, then edges are randomly distributed and negative value of modularity indicates the absence of a community in the graph.

- A random walk is a path between two nodes where each step is randomly chosen.
- Path lengths for walks between two nodes are the number of edges one would have to use to walk from one node to another.

Leading Eigenvector *Leading Eigenvector* [60] is a divisive community detection approach which is built on maximizing modularity over possible divisions of the graph utilizing the properties of adjacency matrix of the city graph. Based on the adjacency matrix, the modularity matrix is defined. Modularity matrix is defined using the Equation 3.15 where A_{ij} is adjacency matrix, k_i, k_j are degrees of the vertices and m is $\frac{1}{2}\sum_i^n k_i$.

$$B_{ij} = A_{ij} - \frac{k_i k_j}{2m} \tag{3.15}$$

Then, the eigenvector corresponding to the *leading eignevector* of the modularity matrix is considered. Utilizing the signs of elements in this vector we decide the group. The elements of the *leading eigenvector* measure how firmly each vertex belongs to its assigned community. In the maximization process, the division of the graph with maximum modularity value is the best distribution of communities. We run the same algorithm over the newly formed communities and continue unless all the communities obtained are indivisible. Running *Leading Eigenvector* on the graph of Salt Lake City produces 21 communities with 48398 nodes in one community. The big community includes the main downtown area of the Salt Lake City.

Walktrap *Walktrap* [61] method is an agglomerative method for community detection based on random walks in which distance between vertices are measured through random walks in the network. The basic intuition of the algorithm is that random walks on a graph

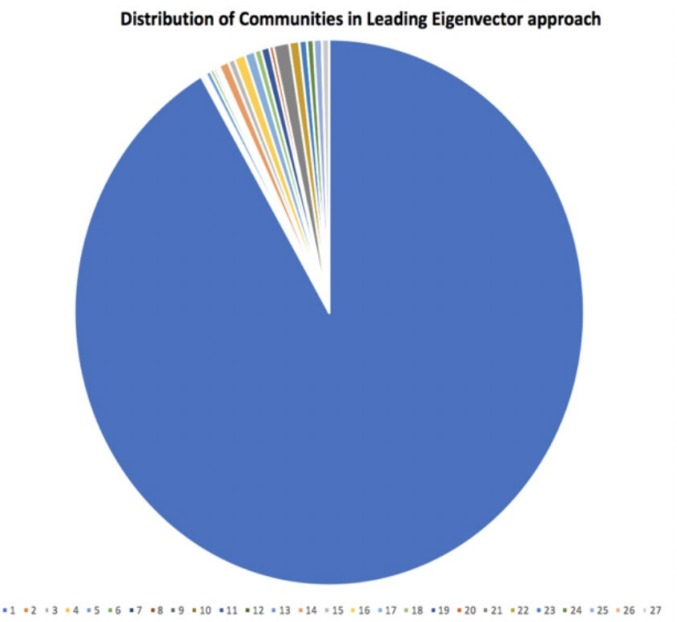


Fig. 3.4: Distribution of communities on Salt Lake City in *Leading Eigenvector* approach.

gets trapped into densely connected parts corresponding to communities. *Walktrap* uses the result of random walks to iteratively merge separate communities in a bottom-up manner by minimizing the overall random walk distance between nodes and communities defined by random walks. The algorithm continues until no more merge is possible. Figure 3.5 (top) shows the results of running Running *Walktrap* on Salt Lake City. *Walktrap* provided 2751 communities with majority in the range of 20 to 40 nodes and few large communities in the range of 3000 to 4000 nodes.

Label propagation The *Label Propagation* algorithm [62] is a another agglomerative algorithm which detects communities using network structure without a pre-defined objective function. The intuition behind the algorithm is that a single label quickly becomes dominant in a densely connected group of nodes as it get trapped inside a densely connected group of nodes. In the beginning, every node is initialized with a unique community label. Then, the labels propagate through the network and at every iteration of propagation, each node updates its label based on the maximum numbers of its neighbours' labels and the communities are formed this way. *Label Propagation* stops if he user-defined maximum

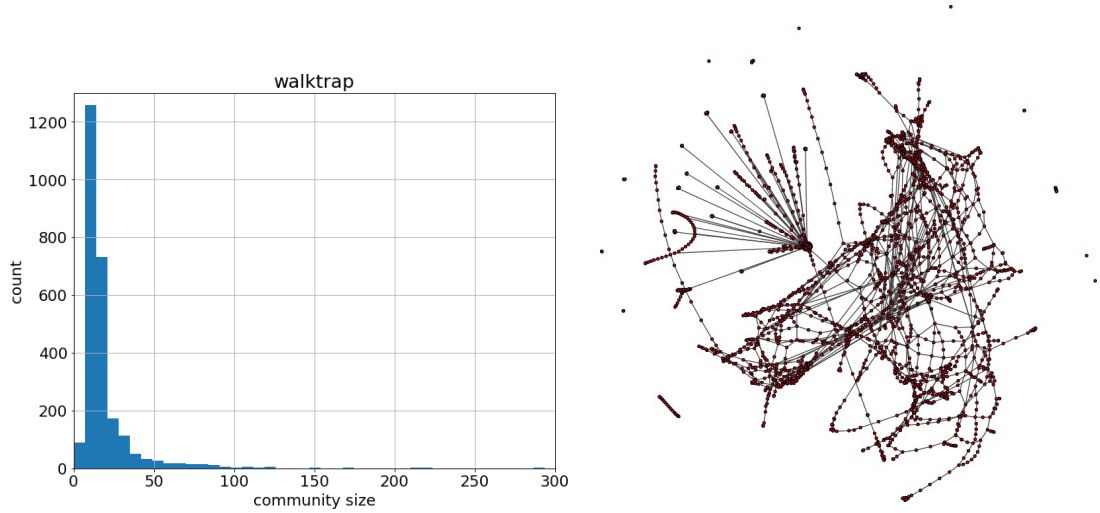


Fig. 3.5: Distribution of communities in *Walktrap* approach for the clusters with node size less than 300 nodes along with representation of all of the communities. Each red dot represent a community. The edge between communities shows the relationship between one community to another.

number of iterations is achieved or algorithm converges. Convergence occurs when each node has the majority label of its neighbours or no merge happens in further iterations. Running *Label Propagation* on the graph of Salt Lake City provides 2007 communities with the distribution similar to truncated normal distribution depicted in Figure 3.5 (bottom). In the distribution most of the community sizes are in the range of 10 to 40.

Multilevel *Multilevel* [63] method is built on modularity optimization and creates communities in a way that the edges inside of the community are significantly denser than between communities. Multi-level is a two step iterative algorithm which stops when no improvement is gained. In step 1, every node is assigned to a random community, then in step 2 each node is removed from its own community and assigned to its neighboring community if the gain of modularity is positive. Applying *multilevel* community detection method on the city of Salt Lake City provides 157 communities on the graph of Salt Lake City which is shown in Figure 3.7.

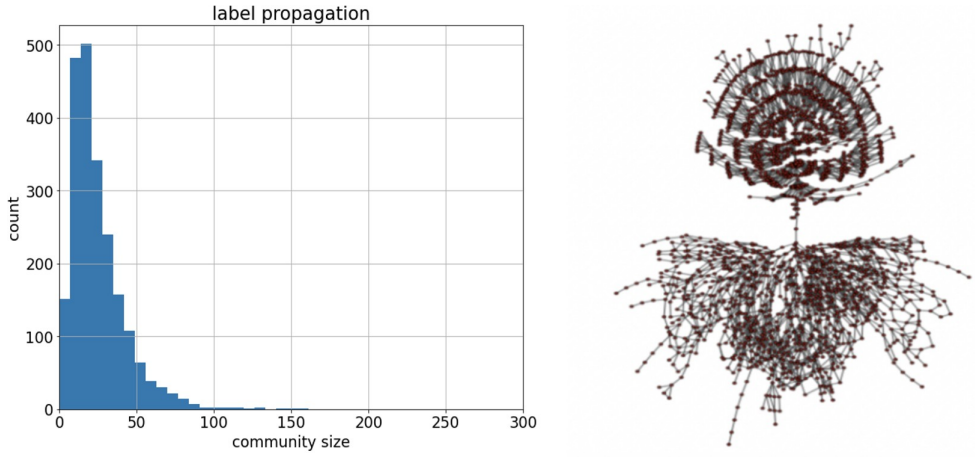


Fig. 3.6: Left: Distribution of communities in Label Propagation approach. Right: Representation of communities. Each red dot represent a community. The edge between communities shows the relationship between one community to another.

3.4.6 Exemplar Assignment

After partitioning the city, we need to find a node (termed the exemplar) that represents each partition of the graph. There are few possible ways of finding exemplars.

- The node with highest historical traffic. The idea is the node which historically has the highest traffic is the node that should represent the partition as historically lots of paths went through it.

- Node with highest reach. Reach is a concept introduced by Gutman et, al. [32] and basically measures the use of a node. A node with higher reach is a node that appears the most in the shortest paths between pairs of the partition. For finding the node with highest reach, we run Floyd-Warshall [64] algorithm on all of the nodes of the partition which gives us shortest path for all pairs of vertices in the partition. The node that appears in the maximum number of paths is the exemplar of the partition.

- Center of the partition based on closeness centrality. Closeness centrality [65] of a node is calculated as the reciprocal of the sum of the lengths of the shortest paths between the node and all other nodes in the partition. Thus, the more central a node is, the closer it is to all other nodes in that partition. Equation 3.16 shows the normalized

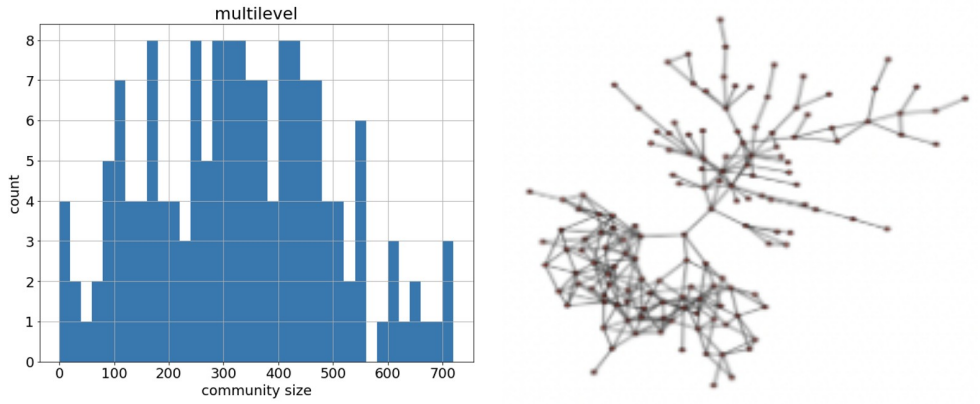


Fig. 3.7: Left: Distribution of communities in *Multilevel* approach and representation of them. Each red dot represent a community.

value of closeness centrality as normalization allows comparisons of centrality value between nodes. In Equation 3.16 $d(i, j)$ represents the distance between node i and j. Distance is defined based on shortest path between the pair of nodes.

$$C(i) = \frac{n - 1}{\sum_{j \in nodes} d(j, i)} \tag{3.16}$$

- Center of the partition based on Random walk closeness centrality. Random walk closeness centrality also called Markov centrality is very similar to the closeness centrality but here closeness is measured by the expected length of a random walk rather than by the shortest path. A node is considered to be close to other nodes if the random walk process initiated from any node of the network arrives to this particular node in relatively few steps on average. The random walk closeness centrality of each node is the inverse of the average mean first passage time to that node. In order to calculate the centrality of each node, the inverse of the mean first passage time between every pair of nodes is taken. Nodes with higher centrality scores indicating that they occupy a more central position within the partition. The mean first-passage time (MFPT) defines an average timescale for a stochastic event to first occur. The mean first passage time from node i to node j is the expected number of steps it takes to reach node j from node i for the first time. Equation 3.17 shows how to

calculate mean first passage time for each node. In Equation 3.17, $P(i, j, r)$ denotes the probability that it takes exactly r steps to reach j from i for the first time.

$$MFP(i, j) = \sum_{r=1}^{n} rP(i, j, r) \tag{3.17}$$

3.4.7 Base Path planning framework

The goal of base path planning framework is to find paths align with agents' goals and deadline between every pair of nodes. The base path planning framework is extended from [43]. The first step is to find the candidate paths that can possibly satisfy the agents' goals. As it is computationally intractable to consider all of the paths between any pair of nodes. Then, among those candidates we select the one which has a minimum cost and matches the agents' goals.

Finding Candidate Paths

Considering all of the paths between any source, destination pair is computationally intractable; hence a primary step is needed to reduce the number of paths to the ones that have the highest similarity to the query. The goal of the pruning is to reduce the number of candidate paths to only explore the paths that meet the agent's goals and query deadline. For finding the candidate paths between nodes (n_1 and n_2), we start from n_1 and explore the successor nodes (*expanding*) until we reach n_2. When exploring each node, the heuristic estimates a path from that node to destination and if the expected estimated arrival time when using the heuristic path is greater than the provided deadline, the node is not expanded (pruned). Figure 3.8 (left) shows the pruning step.

The heuristic path from a node m (a middle node in expansion) to node n_2 is obtained by running A^* [66] on the exemplars instead of all the nodes of the graph (Figure 3.8 (middle)). In each step of A^*, the next exemplar is picked based on the smallest $g(n) + h(n)$ value, where $g(n)$ is the shortest-length path from current node to the connecting exemplar and $h(n)$ is the direct distance heuristic from the connecting exemplar to n_2. Shortest-length

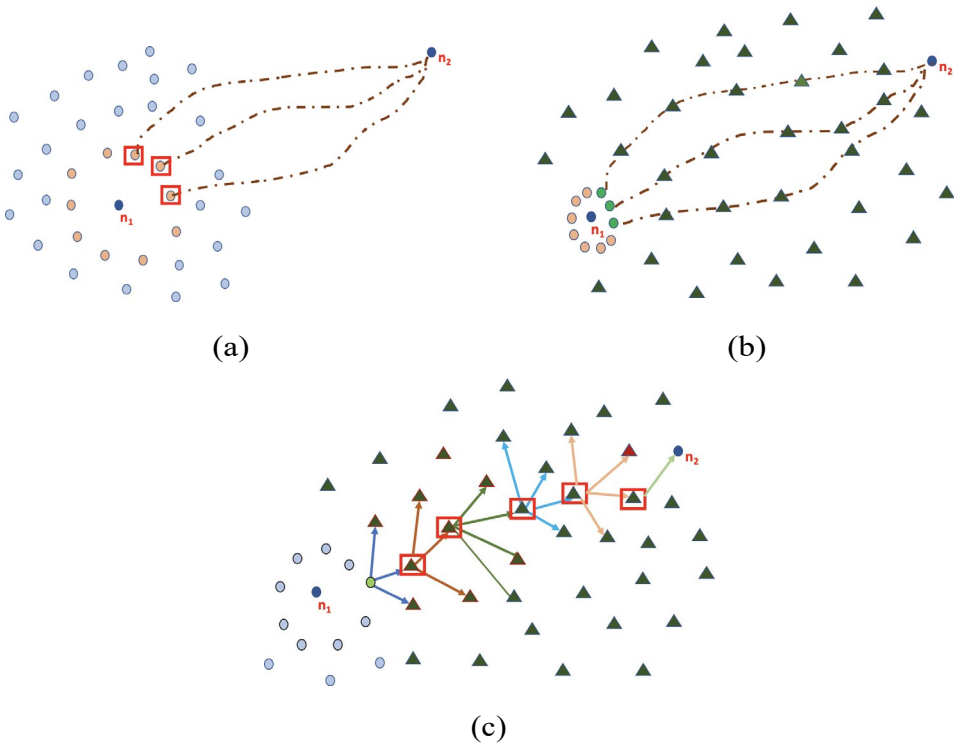


Fig. 3.8: **(a)**: For finding a path from n_1 to n_2, successor nodes of n_1 (orange circles) are explored. Among the successor, the marked ones meet the deadline and rest are discarded. Dotted paths are heuristics paths from the successor nodes to n_2 and they are used as a pruning criteria for the successor nodes of n_1. **(b)**: Finding the heuristic path from m (a middle node in expansion shown as green circles) to n_2 uses $A*$ algorithm through exemplars of graph (green triangles). **(c)**: From current exemplar to the neighboring exemplar the one with with the least $g(n) + h(n)$ is selected (green triangle). Figure shows the selected exemplar in each step of A^*on exemplars for heuristic path.

paths between nodes and their exemplars and also the adjacent exemplars are pre-computed and they are retrieved to build the heuristic path. Figure 3.8 (right) shows the approach.

Paths Cost Definition and Selecting Best Path

The goal of the path planning is to pick a path with minimum cost which matches the agents' goals among the candidate paths. Expected cost of a path can be found using Equation 3.18. In 3.18, t_{path} is the expected arrival time of the path and d is the deadline the agent has to make.

$$expected\ cost = Cost(t_{path}, d) * f_{path}(t_{path}|m, \delta^2_{pat}) = \int_{-\infty}^{+\infty} u(t-d) f_{path}(t_{path}) dt_{path} = \int_{d}^{+\infty} f_{path}(t_{path}) dt_{path} \quad (3.18)$$

Path cost is modelled using a step cost function, but generally any type of cost function can be used in Equation 3.18. The step cost function only penalizes the agent if it reaches the destination after the deadline. Based on Equation 3.18, the whole cost is equal to the Cumulative Density Function (CDF) of Standard Normal Distribution. *CDF* generates a probability of the random variable (travel time in this case) when distribution is normal to be less than a specific value which is *d* (deadline) here. Then, maximizing the Θ value (Equation 3.19), ultimately results in having a path that maximizes the probability of reaching the destination before deadline which matches the first agent goal.

$$\underline{\Theta}(path) = \frac{deadline - m_{path}}{\sqrt{v_{path}}} \quad (3.19)$$

The second type of agent goal seeks the path with shortest en-route time that can make the deadline while they are flexible on departure time. We can modify Equation 3.19 to Equation 3.20 by replacing deadline as the difference of desired arrival time and departure time. Deadline is the amount of time the agent needs to reach the destination from query time. In this case, desired arrival time is fixed but departure time is flexible. In Equation 3.20, ϕ is the argument of Gaussian *CDF* that makes the CDF equal to the probability of making the trip before deadline which in our case is 90 percent.

$$desired\ arrival\ time - departure\ time = m_{path} + \Phi(path) * \sqrt{v_{path}} \quad if \quad departure\_time \subset [\tau 1, \tau 2] \quad (3.20)$$

To find the best departure time, the first step is to find what is the latest possible departure

time (τ_2) to make the trip before the deadline. Then, considering the query time as the earliest possible departure time as τ_1, the interval of $[\tau_1, \tau_2]$ is the time frame that includes the best departure time. Divide the interval into 10-minute segments. For each segment, the path that minimizes the trip duration (Eequation 3.20) is selected. Afterward, we pick the ”time segment” which has the minimum cost path (based on Equation 3.20) in comparison to other time segments. The found minimum cost path with this approach, is the path that has the least en-route time.

3.4.8 Pre-processing: Building distance oracles

In city scale path planning, the volume of path finding requests is high and, hence, calculating a path which satisfies the agents' goals and abides the deadline at query time is not a practical approach. Techniques such as pre-processing and approximation help in expediting the path finding process. For a graph of n vertices, one way is to simply store an n × n-distance matrix for a n-vertex graph. In that case, each query can be answered in constant time, but the space requirement is large and updating the n × n-distance matrix is very time consuming. Approximation is a way of making distance oracles more compact. Their aim is to find solutions which are not exact but clearly close. For example, in our case we don't need to store an n × n-distance matrix, but we can store the paths between exemplars in our city graph which helps in reducing the path finding time. Distance oracles store the best path between every two exemplars for each agent goal for different time slots of each day of week. Time slots are every 10 minutes of every day of the week. Building distance oracles is an offline task and distance oracles are updated weekly to reflect recent traffic patterns on the edges of the city. In each update, the traffic data for the preceding year is used. Stored paths between exemplars help us to quickly answering the path finding requests by connecting the source-destination pair to their respective exemplars and provide the path. Details of the approximate path finding is explained in section 3.4.9). The solution is space and time efficient.

3.4.9 Scalable Algorithm

Each path finding query contains source, destination, agent's goal and deadline. The first step is to connect the source and destination to their respective exemplar. Each node may have up to nine exemplars around it, one candidate is the exemplar of the region it is located and the others are the exemplars of neighboring regions. For selecting the right exemplars, a hypothetical direct path between source and destination is considered and the exemplars with the most similarity to the direction of the hypothetical path are selected. The connecting paths that connects the source, destination to the exemplars are calculated based on shortest length path. Then, the path between exemplars that matches the agent's goals in the query is retrieved from distance oracles. Afterward, connecting paths are added to the retrieved path from distance oracle and the final path is constructed and sent as the result of the query. The path between exemplars does not necessarily need to go through other exemplars. Figure 3.9 illustrates a path between source and destination.

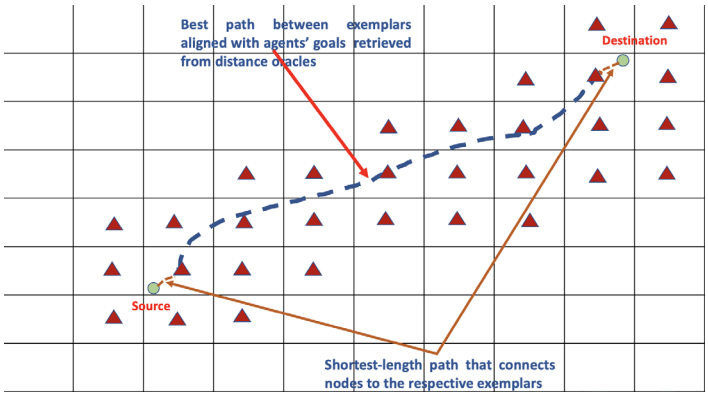


Fig. 3.9: Red rectangles are exemplars of each region. Green circles are the typical source and destination.

3.5 Experiments and Results

We experimented multiple methods of city graph partitioning. Each partitioning method provided various partitions. The key question is, "How many partitions are needed to represent Salt Lake City with reasonable approximation?" This question is answered in section 3.5.1. We discuss which community and clustering approach is picked for parti-

tioning the city in section 3.5.2. In section 3.5.3 we show which node in each partition should be used as exemplar to represent all of the nodes in the partition. We run exact and approximate path finding algorithms to compare the approximate paths to exact paths in section 3.5.4. Lastly, we analyze the space and time complexity of our proposed framework in section 3.5.5.

3.5.1 How Many Partitions Are Needed to Represent The City Graph?

City partitioning is used to reduce the large scale city graph to the set of exemplars. We ran multiple clustering and community detection approaches on the city graph and discussed the provided distribution of each in section 3.4.5. It is obvious that, the more the partitions the more approximate paths get closer to the exact paths as we increase the number of partitions. However, our goal is to reduce the number of nodes of city graph as it impacts the storage required by distance oracles. Also, we want to keep the accuracy of approximate paths in the acceptable range. Accuracy is measured based on the deviation of travel time of generated approximate paths from exact path for each source and destination for the 5000 source destination samples in various time slots of different days of a week. Travel time basically tells us how much longer the paths will be due to approximation. For picking the right number of partitions, we divide the city based on a variety of partition numbers and look at the percentage of travel time deviation of approximate paths from exact paths for both of the agents' goals. If the point of inflection on the curve is seen, then it is a good indication that the underlying number of partitions fits best at that point. For measuring the deviation, grid-based city partitioning is used as baseline of the city partitioning in approximate path planning. Grid based partitioning is a simple method to partition the city in a grid.

Figure 3.10 shows the percentage deviation of travel time of exact and approximate paths for the both agents' goals for variation of partitions. As it shows, the more the partitions the more accurate the paths are. However, having more partitions increases the node size which leads to more storage and time to update distance oracles. Based on figure 3.10, having 150 to 170 partitions looks reasonable with the mean difference of travel time of

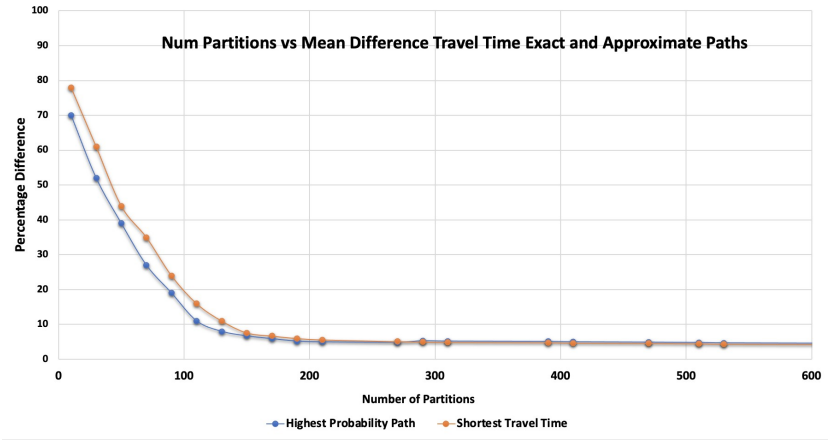


Fig. 3.10: Number of partitions vs the mean difference of travel time of exact and approximate path.

exact and approximate paths around 7 percent. Inspired by the findings of Figure 3.10, we defined a grid based partitioning which partitions Salt Lake City to 150 partitions to be used as the baseline of the other experiments. Figure 3.11 demonstrates the grid based partitions.

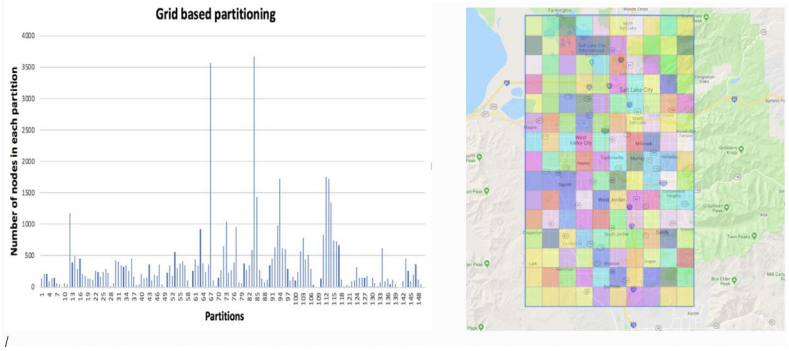


Fig. 3.11: Left: Distribution of nodes in each partition. Right: Visualization of partitions on Salt Lake City. Each color represents one partition.

3.5.2 Which Partitioning Method We Picked?

Clustering For the clustering approach, we tried *k-means* and *meanshift* methods. *K-means* provided 162 clusters and *meanshift* produced 229 clusters. Both of the methods

provides clusters in the accepted range of number of partitions. Hence, we need to look at the quality of clusters in order to pick best choice. One widely used metric for measuring the quality of clustering algorithms is Silhouette Index [67]. This metric uses concepts of cohesion and separation to evaluate clusters, using the distance between nodes to measure their similarity found using Equation 3.21.

$$S(C_i) = \frac{\sum_{v \in C_i} S_v}{|C_i|} \quad where \quad S_v = \frac{b_v - a_v}{max(a_v, b_v)} \tag{3.21}$$

where a_v is the average distance between vertex v and all the other vertices in the same cluster and b_v is the average distance between v and all the vertices in the nearest cluster. The silhouette index for a given cluster is the average value of silhouette for all its member vertices. Comparing silhouette index of *k-means* and *meanshift* shows that *k-means* has a better index, therefore we picked *k-means* as our clustering algorithm.

Community detection Among the community detection methods that we used, *Leading Eigenvector* provided 21 partitions with majority of the area in one partition. Hence, this is not a good method for us. Among the *Label Propagation* 3.4.5, *Multilevel* 3.4.5 and *Walktrap* 3.5, *Multilevel* divides the city to 157 communities which is aligned with our findings in section 3.5.1, hence we select this approach for community-based graph partitioning.

3.5.3 Which Exemplar Assignment Approach is The Best?

After selecting the partitioning method, the next step is to select the exemplar of each partition. In section 3.4.6 we discussed four possible exemplar selection methods: a) highest traffic, b) highest reach, c) closeness centrality and d) random walk centrality. To determine which is best, we picked 5000 source destination pairs in various time slots of different days of a week. We use grid based city partitioning as a baseline. Then we find approximate paths for the 5000 source destination pairs each time with one exemplar selection method for each agents' goal. Then, we look at the percent of times the approximate path planning

approach has the closest (mean, variance) of travel time to the exact path. Then the method with highest number of similarity is selected as the exemplar selection method.

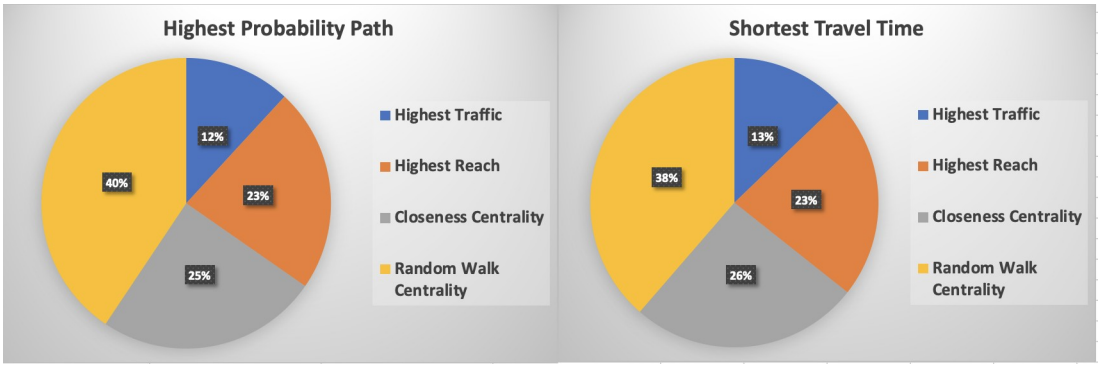


Fig. 3.12: Comparison of four methods of exemplar selection a) highest traffic, b) highest reach, c) closeness centrality and d)random walk centrality.

As Figure 3.12 shows, random walk centrality provides a better representation of exemplars in comparison to the other three methods for the both agent goals as the paths using random walk centrality are more similar to exact paths.

3.5.4 How is The Quality of Approximate Paths?

For the purpose of experiments, we choose 5000 source, destination pairs randomly among all of the possible source-destination pairs to represent the path planning universe. Path planning queries are distributed across the traffic profiles at different time slots of weekdays. Each path planning query has the following inputs: a) source, b) destination, c) time of query, d) deadline and e) agent's goal. Then, for all of the queries we compare the approximate path generated from our proposed algorithm and the exact path.

Highest Probability Path

In order to compare the quality of our proposed approximate paths, we considered the relative difference of mean and variance of travel time of paths between exact and approximate path. For partitioning part of the approximate paths we used community detection (multi-level method), clustering (*k-means*) and grid based. Grid based is used as

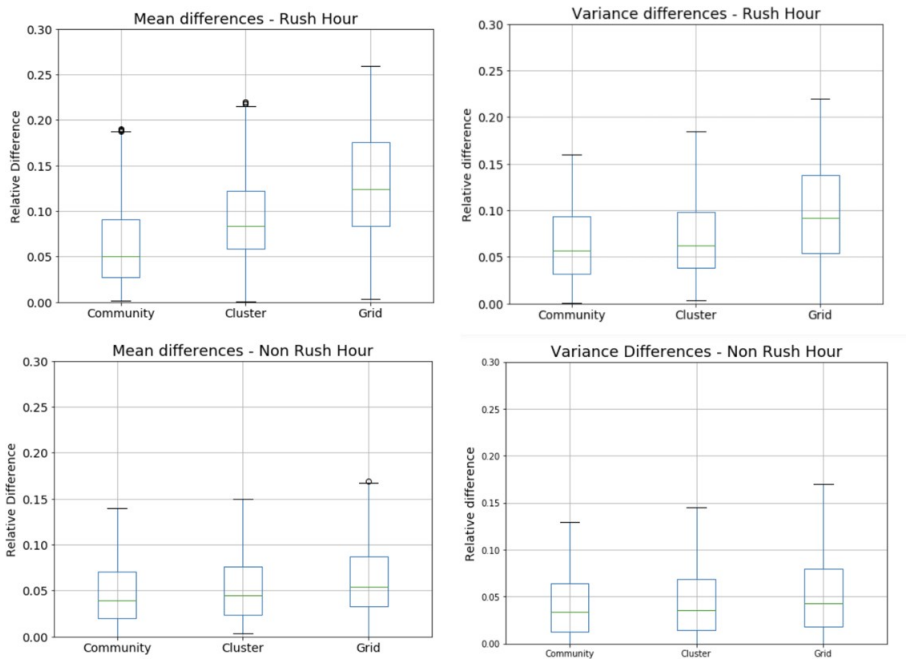


Fig. 3.13: Y axis is the relative difference percentage of mean and variance of travel time of paths for exact and approximate approach for peak and non-peak hours for the agent's goal of highest probability path.

a baseline as it doesn't have an intelligent way of partitioning the city graph and it is only based on a grid. Gird-based method help us to compare the effectiveness of community and clustering method. Figure 3.13 shows the relative difference of mean and variance of travel time of paths between exact and approximate path planning approaches for rush and non-rush hours.

Here are the finding from Figure 3.13:

- Multi-level approach has the least relative difference to exact paths in comparison to clustering and grid-based method partitioning.
- The relative difference of travel time of all of the approximate methods is more significant in rush hour in comparison to non-rush hour. As in non-rush hour, the traffic is not high, hence both approximate and exact approach are almost the same.
- The mean of travel time of community approach is 5 percent longer than the exact path in rush hour and this percentage is 3 percent longer in non-rush hour.

- The variance of travel time of community approach is 8 percent longer than the exact path in rush hour and this percentage is 4 percent longer in non-rush hour.
- in both rush and non-rush hour community and clustering method outperform grid-based method and this shows the impact of an intelligent graph partitioning on the quality of approximate paths.

After looking at the relative difference of travel time of community, clustering and grid based approximate paths, now we want to see among the 5000 source, destination samples of the experiment, what is the ratio of each method in terms of having the closest mean, variance of travel time to exact paths. Figure 3.14 shows this ratio and based on it in rush hour, 54 percent of the closest paths to the exact were from the community approach with 37 percent clustering and a small fraction of grid approach (8%). In non-rush hour traffic the differences are less but still the pattern is the same. This emphasizes the fact that having an accurate graph clustering approach is more important during high traffic.

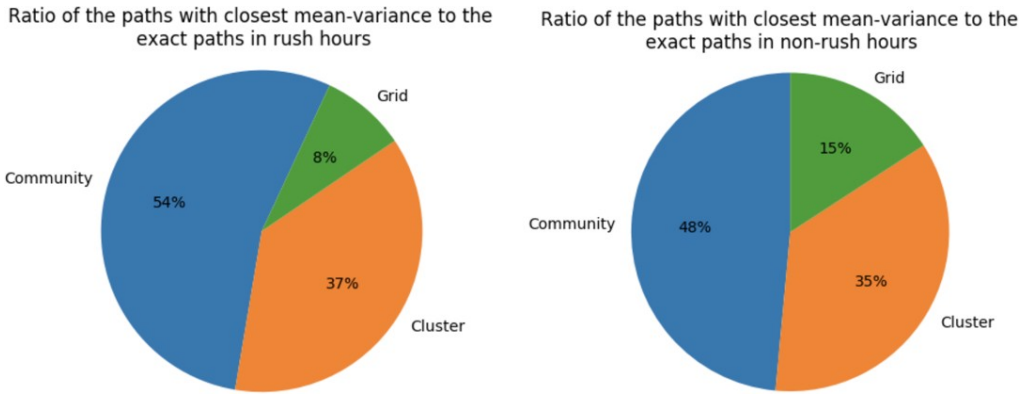


Fig. 3.14: Ratio of paths with the closest mean-variance to the exact path in peak and non-peak hour for the agent's goal of highest probability path.

Shortest Travel Time

In this section we repeat the experiments in section 3.5.4 for the agent goal of shortest en-route time. Figure 3.15 and Figure 3.16 illustrates the results.

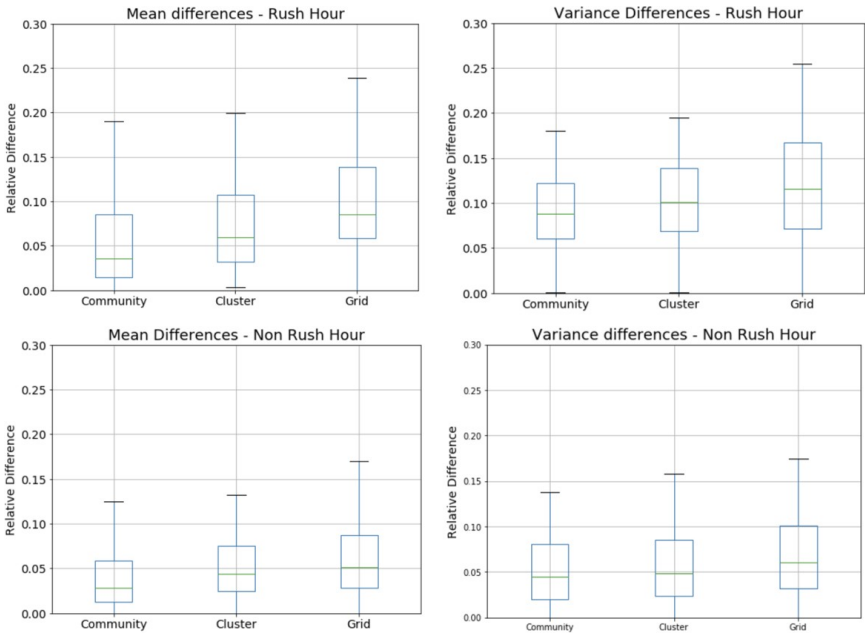


Fig. 3.15: Relative difference of travel time of mean and variance of paths for exact and approximate approach for rush and non-rush hours for the agent's goal of shortest en-route time.

Here are the findings from Figure 3.15 and Figure 3.16 and also the comparison of the results with section 3.5.4.

- Similar to the previous section, community method outperforms other approximate approaches in terms of closeness of relative difference from approximate and exact paths.
- Approximate path planning methods in rush hour have larger travel time difference than non-rush time and in rush hour the mean of community method is 4 percent and its variance is 9 percent higher than the exact path.
- In highest probability paths, paths have higher mean and lower variance in comparison with shortest travel time paths which is aligned with their goals' definitions. The agents with highest probability path look for the most secure path hence their variance is lower. Agents with the goal of shortest travel time are willing to risk if the risk

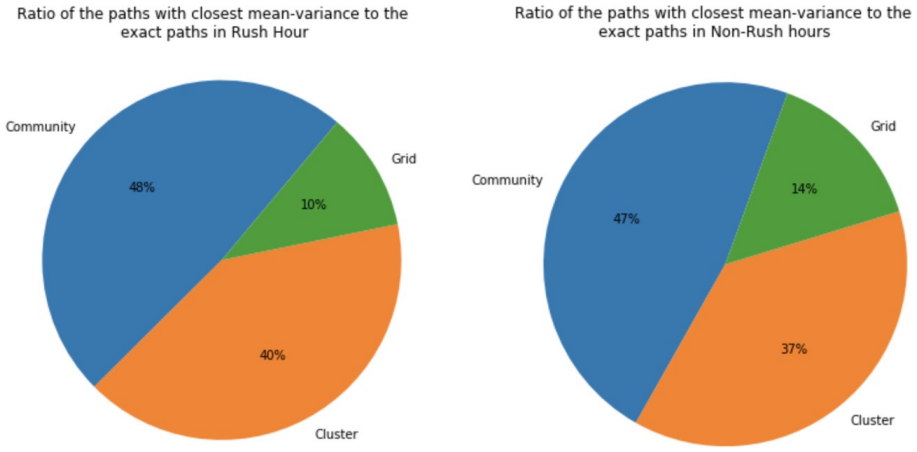


Fig. 3.16: Ratio of paths with the closest mean-variance to the exact path for the agent's goal of shortest en-route time.

provides them the paths with shorter travel time, hence their variance is higher and the mean of their found paths are shorter.

- In rush hour, community and clustering partitioning outperform the grid based method in both agents' goals. In non-rush hour, the difference is less significant.

3.5.5 What is the Time and Space Complexity of Scalable Algorithm?

The previous experiments show that our proposed algorithm is at least 95 percent as good as the exact paths while it responds to the queries in real time. When a path finding query comes, source and destination nodes are connected to their respective exemplars. Then, the stored path between exemplars for that time of the day, day of the week and type of agents' goal is retrieved. The retrieved path along with the sub-paths that connect source, destination pairs to the respective exemplars construct the path and send it to the agent. The real time response to the path finding queries are possible utilizing the distance oracles. Now one of the concerns here is the size of the distance oracles and the effort to update them. If we consider nodes of the city as N , and the number of exemplars as M , utilizing the approximate path finding instead of $N * N$ paths, we are storing $N * M + M * M$ paths which in our case M is 157. We store one distance oracles for each time slot of a day, for 7 days of a week for each of the agent's goals. Updating the distance oracles is an offline

process, and it happens weekly. Every week, the traffic data of the preceding 12 months is used for updating the distance oracles.

3.6 Conclusion and future work

In this paper, we propose a scalable algorithm that is practical in large scale path planning applications for the use cases where agents have goals, and the planner aims to satisfies agents' goals rather than just providing a path which can move agents from a source node to a destination node. The city is modelled as a large scale graph. Agents have two types of goals: 1) those who look for the path with highest probability of reaching destination before deadline, and 2) the agents who are interested to have the shortest travel duration while they are flexible on the time they can leave. Associated with each path is a defined cost and the goal of the path planner is to find a path that satisfies the agents' goals. For expediting the path planning process, the city is partitioned and each partition is represented with an exemplar. The exemplar of each partition is decided based on random walk centrality. For partitioning the city graph, we used community detection methods and clustering methods. When a path planning request comes, source and destination nodes are connected to their corresponding exemplars with respect to the path direction and the path between exemplars is retrieved. The paths between exemplars are stored in distance oracles based on the preceding year data at the time of update and the oracles are updated every week to reflect the recent changes on the network. Results show that among all of the graph clustering approaches, community-based approaches produce closer results to exact path planning approach. Approximation provides paths with mean and variance which are not exact but close to that exact paths, while the solution is space and time efficient.

The main contribution of current work is providing a paradigm to handle large scale path planning requests utilizing pre-computation and approximation while satisfying agents' goals. Graph partitioning reduces the graph size; pre-computation helps in answering the queries in real time and approximation helps in reducing the space needed for storing the paths. Even though the approximate paths are not as accurate as exact paths, but they have acceptable accuracy in comparison to the actual paths given the fact that they saved time

and space in the whole process. Possible future work of this research includes: a) adding new agents goals to the domain and b) considering traffic data prediction to enhance the decision making process which is currently based on historical data.

CHAPTER 4

Dynamic Reinforcement Learning Based Traffic Optimization in Smart City Paradigm

4.1 Abstract

[1] Traffic congestion on urban road networks has increased substantially during the last decade, characterized by slower speeds, longer travel times, increased vehicular queuing, and increased pollution. The main pain point in traffic management is the static nature of our city structures that cannot adapt to the traffic dynamics changing throughout the day. This work focuses on a futuristic smart city design where making structural changes to the city graph is possible. These changes include modifying lane direction, ramp metering, speed limits, and signal timings on road segments. We also assume local observability of the system where sensors can provide all the data needed for decision making. Under these assumptions, we propose a multi agent reinforcement learning (RL) framework for improving traffic flow in city networks. Our learning agents observe their assigned environment and find the best structural changes based on a set of features that represent the recent traffic conditions, dependent road segment characteristics, and recent structural modifications. The goal of RL agents is to interact with the environment to learn what is the optimal modification for each road segment with the goal of maximizing the cumulative reward over the set of possible actions in state space. Then, once the RL agents are fully trained, they can easily adapt to the dynamic changes of the traffic. Our proposed method has two level learning. In the first level, a single agent is the only modifier of the traffic system so it directly learns the initial policy. In the next level, we have multiple agents changing the environment at the same time, each based on the initial policy learned in the previous step while still updating their policy based on the interaction with the dynamic environment and

[1]The first version of this work is presented and published in proceedings of IEEE International Conference on Computational Science and Computational Intelligence (CSCI 2017). DOI: 10.1109/ISC253183.2021.9562951
The current version (extended) is submitted to the journal of IEEE Intelligent Systems.

in agreement with other agents. Our results show that the proposed framework improves the total travel time (TTT) of the city by 36.2% during rush hours in the Salt Lake City area.

4.2 Introduction

Traffic congestion occurs when the volume of traffic is greater than the available street capacity. In an era of accelerated urbanization around the world, the ability to travel freely is more critical than ever before [3, 4, 18, 43, 44, 68]. The cost of traffic congestion is large and solutions to reduce the congestion save time and money and reduce environmental pollution [3, 18, 44]. The main goal of traffic management is to improve the traffic flow and reduce the traffic load on highly congested areas. To solve the traffic congestion problem, numerous methods and approaches have been implemented. Most of the traditional traffic management approaches assume the city structure is fixed and concentrate on re-routing the traffic to alleviate the traffic on highly congested areas [3, 52, 69–73]. While this is a practical approach in traffic management, re-routing might not be favorable for most of the car drivers, and hence it might create a situation in which drivers decline the routing directions. Several types of dynamic control methods based on traffic flow theory have been developed and deployed in real-world applications. They prove that traffic control is a way to prevent, or at least relieve traffic congestion, hence improving traffic conditions. Recently, more attention has been paid to leveraging operational techniques in traffic management like widening roads, variable speed limits, and modifying signal timing [12–19].

In this research, we focus on making structural changes to the city graph such as dynamically changing the direction of lanes, ramp metering, modifying the speed limits, and modifying the signal timing in order to manage the traffic in congested areas. For example, if a road has three southbound and three northbound lanes and the southbound lanes are overly congested, one of the possible modifications is to allocate one of the northbound lanes to the southbound direction for a specific period of time (Figure 4.1 illustrates the process). Thus, the road capacity for the southbound traffic is increased as we leverage the underused lanes from the northbound side.

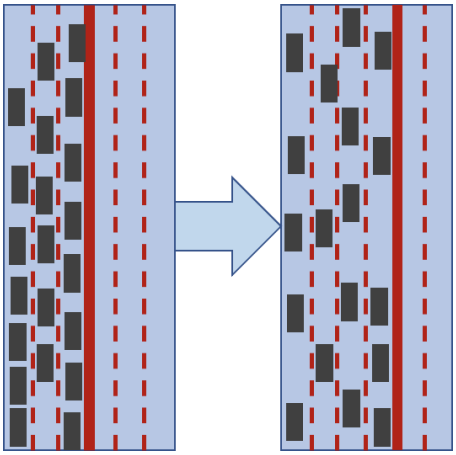


Fig. 4.1: Allocating one lane from right side to left side may decrease the congestion.

Modifying signal timing by dynamically changing the phase duration of the signals can also improve congestion. If one segment of the road is highly congested, then we can increase the signal timing in order to manage the congestion. Ramp meters are stop-and-go traffic signals that control the frequency with which vehicles enter the flow of traffic on the freeway. Ramp meters are used to improve traffic flow. Most ramp meters create a delay between cars entering the highway. This delay helps reduce disruptions to freeway traffic and reduces accidents that occur when vehicles merge onto the highway. We can leverage the capability of modifying the delay time of ramps in order to reduce congestion. Figure 4.2 shows the process.

Modifying speed limits are also used when traffic volumes are building and congestion is likely. When traffic volumes exceed a predetermined threshold, one strategy is to handle more traffic volume at a slower, but not stop-and-go speed. Variable speed limit allows speed limits to be changed based on current road conditions and the level of congestion. The system's goal is to slow traffic uniformly in a way that allows smooth traffic flow. In both cases, the speed limit decrease is intended to alert drivers of conditions downstream. Also, when congestion is low, we might be able to move the traffic faster, in this case the speed limit of road segments will increase to move more vehicles. Ideally, the speed limit

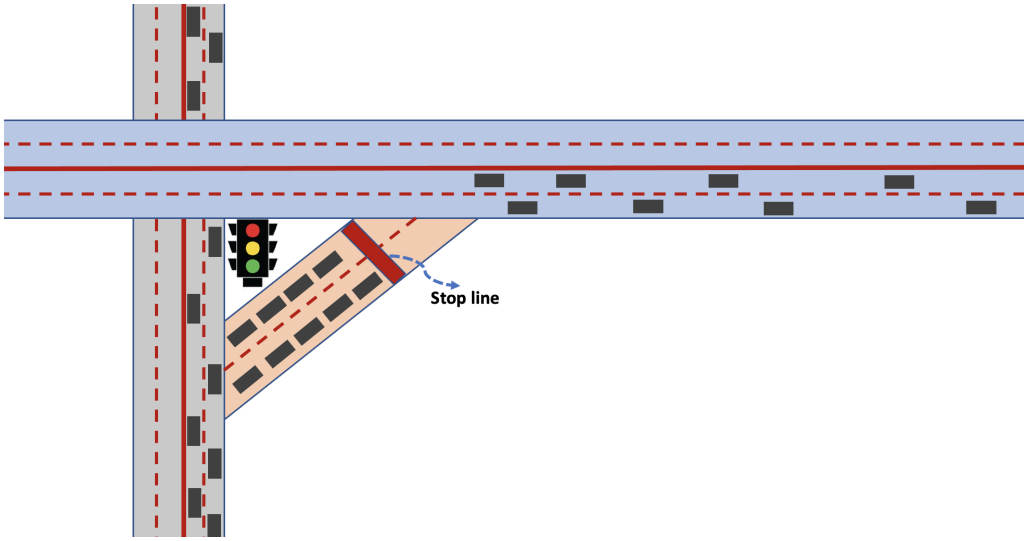


Fig. 4.2: Delaying cars from entering the freeway using a ramp metering.

and message alerts are automated and do not require intervention from any operator [74]. Figure 4.3 shows the implementation of variable speed limit.

The focus of this work is on optimizing the traffic congestion on the whole city network through a smart city paradigm. A smart city provides the capability of modifying the structure of the city graph and gathering information from the sensors to learn the best possible modification for each condition of the traffic. For this reason, we propose a multi agent reinforcement learning system (RL Agents) that finds the best structural changes based on multiple dynamic factors such as current traffic condition, dependent road segment structures, and recent structural modifications. These structural changes not only impact the flow in both directions of the road segment but also the flow at surrounding road segments. Therefore, we need to consider the impact of structural changes not only on each road segment, but also on dependent road segments and the whole network. RL agents interact with the environment in the training phase and learn the optimal modification for each road segment considering the current road segment and the impact on the dependent segments. We proposed the process of learning the optimal policy as a two step process. In the first step, a single agent is the only modifier of the traffic system so it directly learns the initial policy. In the second step, there are multiple agents changing the environment,

Fig. 4.3: Implementation of variable speed limit. Variable speed limit allows speed limits to be changed based on current road conditions and the level of congestion (original image from PennDOT [1])

each based on the initial policy learned in the previous step while still updating their policy. The goal of RL agents is to maximize the cumulative reward over the set of possible actions in state space. First step is critical to reduce the noise in the reward system and be able to converge to a stable policy. Then, in the second step, the agents update their policy to take into account the indirect interaction with the other agents.

4.3 Previous Work

In the domain of city traffic optimization, there are two main approaches. One group of models use linear programming to optimize the city congestion [13–15]. This set of approaches define the overall city congestion as affected by decision variables such as direction

of lanes, signal timing, and the speed limit of the roads. The main objective is to minimize the overall congestion via finding the best values of the decision variables. Since, the domain of traffic optimization is a highly dynamic domain, traditional linear programming approaches are not suitable in this situation. Linear programming approaches assume the impact of structural changes on further road segments are known at the beginning of the calculation which is an unrealistic assumption in practical applications. Also, in linear programming approaches, the problem is solved from scratch every single time. Hence the computational complexity is high, and these models are not adaptive to the traffic changes [3, 13–16, 75].

The second group of approaches use reinforcement learning for traffic optimization [16, 17, 76–79]. In this set of approaches, the RL agents interact with the environment and through maximizing the cumulative reward, they learn the best possible actions for each specific state of the world.

Aslani et al. [17] proposed an RL based framework for traffic signal optimization. They model the behavior of car drivers and their reaction toward system disturbance such as jaywalking and sensor noise. Since they study a microscopic view of traffic (at drivers level), the state space of their model becomes large. Therefore, they leverage linear function approximation and state space reduction to reduce the size of the problem which may lead to undesired results if the reduction process is ineffective. Our goal is to focus on optimizing the whole city traffic optimization and having a realistic view of traffic; hence microscopic traffic models, linear function approximation, and state space reduction is not applicable for our use case.

Paul et al. [80] proposed a framework where single deep reinforcement learning agent manages the traffic signal of multiple intersections using policy gradient algorithm. In particular, the agent is trained with spatio-temporal data of the environment that allows it to perform action in different deep neural network models. Their proposed model outperforms the baseline results which is fixed signal duration systems for few intersections. While this approach focuses on optimizing traffic through modifying signal timings, since our focus is

on city wide traffic optimization, single agent modelling is not practical for our use case.

Walraven at, al. [77] proposed a model to find a method that defines the speed limits for the road segments in order to minimize the global delay of car drivers in the city network. The model learns the best speed limit for each road segment among the possible speed limits of 60, 80, 100, and 120. Their RL model's reward function is only built based on penalty rather than actual reward in order to discourage unnecessary modification of speed. For our proposed framework, our goal is to have a model that interactively learns the best speed limit to proactively move the system toward minimizing the traffic flow by motivating agents with reward. Also, our speed agents have a larger set of actions and a higher flexibility of modifying speed limits rather than only limited to few options.

Zhou, et al. [81] proposed a model which develops a coordinated dynamic traffic control system that integrates variable speed limit information using a reinforcement learning technique. Their ultimate goal is to build a model that enables traffic scenario analysis, such as time-of-day, freeway trajectory, future demand assessment, and special event traffic conditions. They used q-learning to decide optimized variable speed limit on one freeway with a limited state space. Therefore, it is not applicable to our use case which is a high dimensional state space simulating the whole city traffic.

Gunarathna et al. [18] presented a dynamic lane configuration solution for improving traffic flow using a two-layer, multi-agent architecture. At the bottom-layer, a set of RL agents find the best number of lanes for each side of the road. Then the lane-direction changes proposed by the reinforcement learning agents are coordinated by another layer of agents in order to evaluate the global impact of the proposed lane-direction changes. The coordinating agents try to predict the impact of the decision proposed by RL agents on the nearby road segments and accept/decline the change based on some rules. Their framework only focuses on dynamically changing number of lanes for traffic management and also the rule based coordination process can be replaced by a learning agents which can easily adopt to the dynamic changes of the traffic environment.

In our proposed model, we focus on a futuristic smart city design where making structural changes to the city graph such as modifying number of lanes, opening or closing the ramps, and changing signal timings on road segments are possible, sensors can provide all the data we need, and traffic data is easily measurable. All of the mentioned RL frameworks only focus on one aspect of active traffic management such as signal timing, lane modification, or variable speed limit, while our proposed framework focuses on all possible modifications. The main advantage of reinforcement learning methods is, with proper training, they can easily adapt to the changes of the domain without the need of the recomputation. However, there are two challenges with reinforcement learning agents. First, an RL agent finds the rules to achieve an objective by repeatedly interacting with an environment. In order to be able to have a proper action at prediction time, the agent needs to have seen a similar situation in the training phase through an action/reaction paradigm to be able to estimate the reward of the situation [79, 82–84]. The second challenge with reinforcement learning agents is that the state space can grow exponentially when the dimension of the state grows linearly. This problem is known in literature as the curse of dimensionality [83, 85, 86]. Any model that uses RL needs to consider the two mentioned challenges. In order to tackle the first challenge of RL based models, we train our agents on a myriad of changes through a traffic simulator which is calibrated by real traffic data. To tackle the curse of dimensionality, we considered a set of distributed agents at each road segment. Each agent makes decisions independently for a specific road segment using the features of the current road segment and the dependent segments. This independent decision making prevents the state space from getting so large.

4.4 Model Framework

In the model framework, we first discuss time and space discretization (4.4.1 and 4.4.2). Then, we define learning agents (4.4.3), state (4.4.4), action (4.4.5), and reward (4.4.6). Then we explain the learning algorithm (4.4.7). All of the parameters used in this framework are listed in section 4.4.9.

4.4.1 Time discretization

In order to represent the state space, we need to discretize the time. We divide the time into the intervals of length m minutes and use each interval as a time step. In each of these time steps, agents make structural changes to their assigned road segment. Each agent can take just one action at each time step. It is important that m is chosen carefully as a small m may increase the system noise and make both training and simulation slow, while a large m will decrease the adaptivity of the system to the changes [79, 82].

4.4.2 Space discretization

In a reinforcement learning approach, states are a representation of the current world or environment. Defining states for complex environments is not a straightforward task due to the limitation of fixed-size representation of learning systems [79, 82, 87]. In learning systems, we are bounded by the set of features and have to fit our problem in the feature space. Also, in RL, we need to be concerned with the Markov Property, where each state should provide enough information to accurately predict the expected reward and the next state given an action, without the need for any additional information [79, 82, 86].

Agents in our model make decisions considering the impact of the decisions on the current road segment and the dependent road segments as well. Therefore, they need to have information about the environment around them. For example, an agent may need to know the time value of all the signals in its proximity with radius r. For one location, there might be 5 signals but for another location there might be no signals around the agent. To work around this problem, we discretized the space based on the fixed segment lengths. Figure 4.4 shows an example of segments. Each road segment might have different properties. Those properties can be number of signals, number of lanes, speed limit, ramp, type of road segment, and existence of landmarks. When we define road segments based on equal lengths, we can compare the properties of road segments. Also, it helps in consistent tabular representation of state space. Length of the segments (l) is a parameter that can be tuned. Note that traffic signals can be located anywhere in a road segment and not necessarily at the endpoints of the road segment.

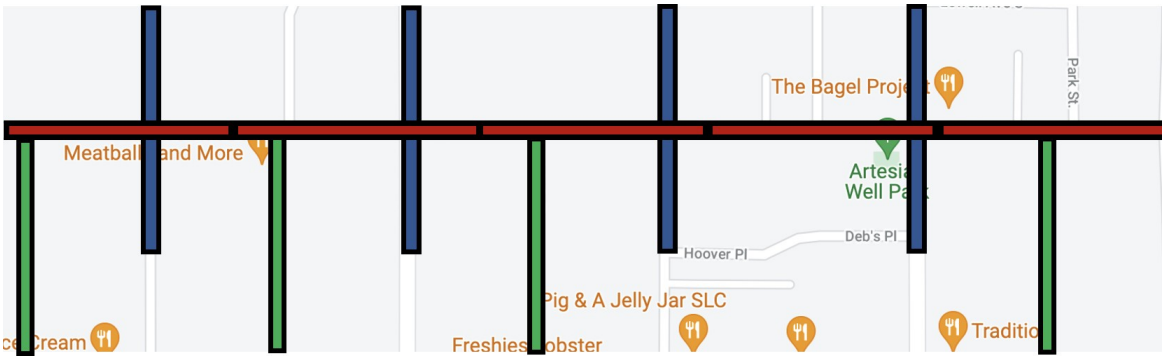


Fig. 4.4: Sample of state discretization. Road segments are shown with different colors (original image cropped from Google Maps ™).

4.4.3 Learning Agents

We define four types of agents, namely, lane agent, signal agent, ramp agent, and speed agent.

- The lane agent controls the number of the lanes for each side of a road segment containing multiple lanes. The lane agent decides the optimal number of lanes for each direction of the road segment. Note that the total number of lanes over both directions of a road segment is a fixed number.

- The ramp agent uses traffic signals (ramp meter) to control the rate at which vehicles enter a freeway by modifying the entrance delay to the freeway.

- The signal agent is responsible for controlling a traffic signal at an intersection. It is known that a phase scheduling decision at one intersection largely impacts the traffic conditions in its neighbouring intersections. The traffic signals considered in this work have 8 phases and the signal agent controls timing for each phase by learning the appropriate phase timing based on traffic patterns. Figure 4.5 shows the 8 phases on the traffic signal.

- The speed agent is responsible for modifying the speed limits on each direction of a road segment in order to reduce the congestion.

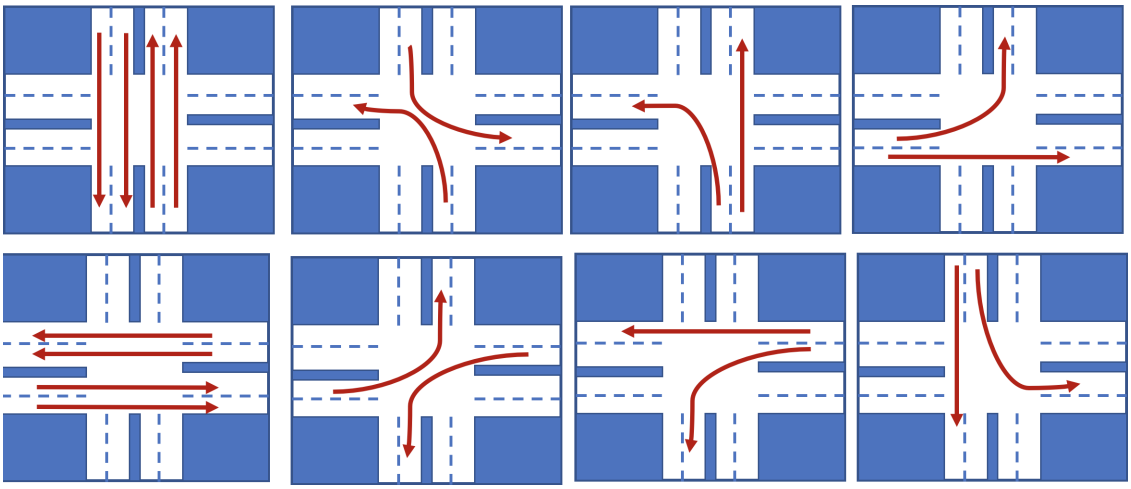


Fig. 4.5: The 8 possible phases of a signal at each junction.

4.4.4 State

There are a set of features that we can measure for each direction of a road segment during the time step t: number of lanes, number of vehicles, posted speed limit, average speed of vehicles, number of traffic signals in the road segment, time of traffic signals for each phase, and number of connecting roads (intersections without a traffic light measured by number of lanes). Using these features, we represent the state for each of our learning agent types as follows. Figure 4.6 illustrates the features for each type of road segments.

- Lane agent: the road segment features for the last T_l time steps for a) the assigned segment, b) n_l^b segments before the assigned segment, and c) n_l^a segments after the assigned segment.

- Speed agent: the road segment features for the last T_v time steps for each direction of a) the assigned segment, b) n_v^b segments before the assigned segment, c) n_v^a segments after the assigned segment.

- Signal agent: a) distance of the signal to the beginning and end of the road segments in which it is located, in both north-south and east-west directions; b) features for the road segment where the traffic signal is present for the last T_s time steps in both

north-south and east-west directions; c) the timing for each signal phase for the last T_s time steps; d) road segment features for the n^b_s and n^a_s road segments before and after the road segment in which the signal is located for the last T_s time steps in both north-south and east-west directions.

- Ramp agent: the road segment features for the last T_r time steps for: a) the road segment in which the ramp is initiated from (S_{r_i}), b) the road segment in which the ramp enters to (S_{r_e}); c) $n^b_{r_i}$ road segments before S_{r_i}, d) $n^a_{r_i}$ road segments after S_{r_i}, e) $n^b_{r_e}$ road segments before S_{r_e}, f) $n^a_{r_e}$ road segments after S_{r_e}, and g) timing of the meter.

4.4.5 Action

In a RL paradigm, the action set is the set of all possible moves the agent can make. Agents usually choose from a list of possible actions. Here we define the set of possible actions for each agent.

- Lane agent: The actions for the lane agent consist of increase-by-2, decrease-by-2, increase-by-1, decrease-by-1, or no-change at each time step. Since the total number of lanes is fixed for a road segment, increase-by-1 action in one direction means decrease-by-1 in another direction. We can also consider other actions such as increase-by-3 and decrease-by-3, but for our use case (Salt Lake City), those were not applicable. However, the model is easily adaptable to those actions.

- Signal agent: Since there are eight time values that the signal agent can change (see Figure 4.5), and for each value the agent can take one of the actions (increase-s-seconds, don't-change, decrease-s-seconds), we will have 3^8 possible actions. We have set a maximum $t_{S_{max}}$ and a minimum value $t_{S_{min}}$ for each phase of the signal to prevent extreme changes.

- Ramp agent: The action for ramp agent is similar to signal agent. The goal of the ramp agent is to modify the rate of cars entering the freeway. Therefore, ramp agent

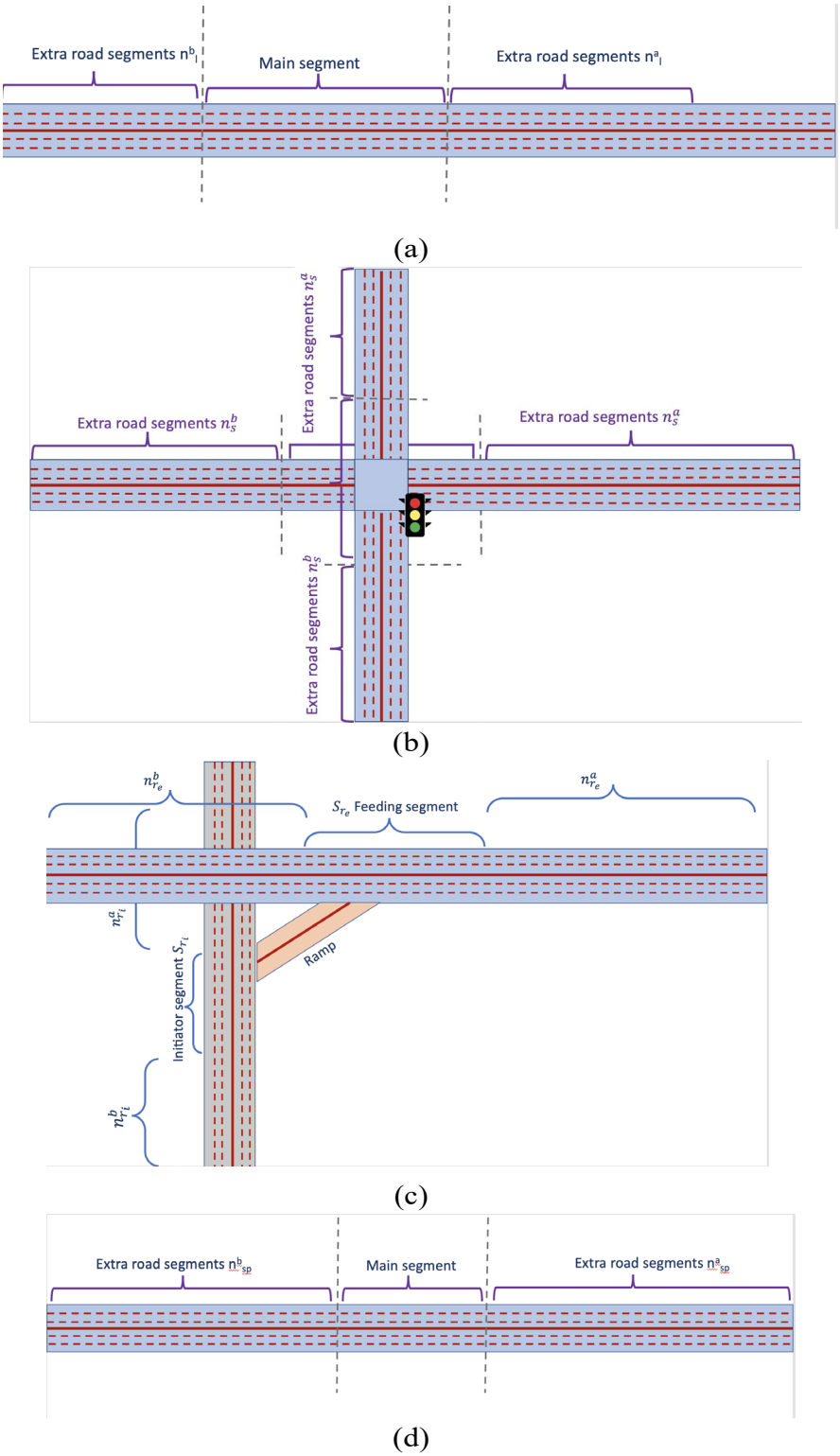


Fig. 4.6: Main segment and extra segments that have been considered for a) lane, b) signal, c) ramp and d) speed agents.

can take any of the following actions: a) increase-r-seconds, b) don't-change, and c) decrease-r-seconds. We have set a maximum $t_{r_{max}}$ and a minimum value $t_{r_{min}}$ on the ramp meter to prevent the problem of infinite wait for the vehicles in a ramp.

- Speed agent: For this type of agent, possible actions are a)increase-m-mph, b) decrease-m-mph and c) no-change. Since we have 2 directions, the total number of actions is 3^2 actions. There is a maximum $V_{mph\ \ max}$ and minimum V_{mph} defined here to abide the traffic rules.

4.4.6 Reward

We use the same reward metric for all our 4 agent types as their same goal is to minimize traffic. There are multiple metrics that we can use as a reward for an action in traffic systems. Total travel time, the amount of CO2 emission, total vehicle delay time, and vehicle throughput are examples of these metrics [3, 44, 88]. Out of all the possible metrics, the ones that directly measure the throughput of traffic are expected to work slightly better as they provide a more clear reward signal tied to the taken action [16,18,77,79,88]. Traffic throughput is the number of cars that pass a road segment during the time interval *tau*. In this work, we use total vehicle hour (TVH) to measure the throughput in our road segments as the reward. TVH measures the number of vehicles that pass the road segment in an hour [88].

- The lane agent considers the TVH for the segment it controls and one additional road segment before and after the target road segment.
- The signal agent adds the TVH for both segments on north-south and east-west directions and one additional dependent road segment in north-south and east-west directions.
- Reward for the ramp agent is calculated by adding the TVH for the segment that it initiates from to the segment it feeds to and one additional dependent road segment from each side.

- Similar to the lane agent, reward for speed agent is the TVH for the road segment and one additional road segment before and after the target road segment.

In addition to the throughput feedback, we need to discourage agents from making unnecessary lane changes as too frequent changes can cause traffic issues. We introduce a penalty for lane change as p_l which is in terms of TVH for the assigned and dependent road segments. The impact of penalty of the agents' decision making is studied in Section 4.5.4.

4.4.7 Learning Algorithm

As mentioned earlier, this work employs reinforcement learning (RL) to interactively learn how to minimize the traffic. In RL problems, an agent interacts with the environment by taking an action and receives feedback through a reward mechanism. An RL problem is generally formalized in the form of a Markov Decision Process (MDP) using the notation of $< S, A, P_a(s, s'), R_a(s, s'), \gamma >$ where S represents the environment and agent states, A is the set of actions that the agent can take, $P_a(s, s')$ is the transition probability between state s and s' under action a, $R_a(s, s')$ is the reward of transitioning from state s to s' under action a, and γ is the reward discount factor to denote that rewards for the events in the immediate future are weighted more than events in the distant future [79, 82, 86, 87].

In this setting, the agent's goal is to find the policy $\pi = A \times S \rightarrow [0, 1]$, that will maximize the discounted cumulative reward (R_t) from each state s_t:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{4.1}$$

The expected value of this cumulative reward at state s when taking action a under policy π is called action value function and is defined as:

$$Q^{\pi}(s, a) = E(R_t | s_t = s, a_t = a, \pi) \tag{4.2}$$

Consequently, we can define the optimal action value $Q^*(s, a)$ as the action value under the optimal policy π^*:

$$Q^*(s, a) = \max_{\pi^*} Q^{\pi^*}(s, a) \tag{4.3}$$

The ultimate goal here is to find the optimal policy π^* for the agents via Q-learning. In Q-learning, if we can accurately estimate the action value function (Q function), then a greedy policy can be used to perform the best action. However, if the state space is huge, then it is difficult to infer the Q-value of new states from already explored states because, the amount of memory required to save and update that Q-table increases as the number of states increases. Also, the amount of time required to explore each state to create the required Q-table is huge. Since in our case we are dealing with a large state space, it is difficult to directly estimate the Q function; therefore we need to approximate it using a deep Q-learning method [79, 82, 86, 89]. For most problems, it is impractical to represent the Q-function as a table containing values for each combination of s and a. Instead, we train a function approximator, such as a neural network with parameters θ, to estimate the Q-values, i.e. $Q(s, a; \theta) \approx Q^*(s, a)$. The state is given as input and next action is determined by the maximum output of the Q-network based on the objective value in 4.4 where the tuple (s, a, r, s') representing (state, action, reward, new state) in a training example [79, 82, 86, 89, 90]. Architecture of the DQN is explained in 4.4.8.

$$\min_{\theta} \quad \left(r + \gamma \max_{a'} Q(s', a'; \theta) - Q(s, a; \theta)\right) + \lambda \|\theta\| \tag{4.4}$$

To overcome the challenge of convergence due to the noisy reward system in a multi-agent system where each agent is independently changing the environment [79, 82, 89], we propose a 2-step learning process, pre-train and main train. In pre-train, for each episode, a single agent is the only modifier of the traffic system so it directly learns the initial policy π_0, hence the noise is minimal. In main train, the episodes have multiple agents changing the environment at the same time, each based on the initial policy learned in the previous step while still updating their policy based on the interaction with the dynamic environment.

For example, in the case of the lane agent, in the pre-train step, the single lane agent learns to change the number of lanes on a road segment on a series of episodes while no other agent is making any change on the city structure. Then, in the main train step, there are all the other agents (lane, signal, speed, and ramp agents) that are also making changes to the environment at the same time. In main train, the agents learn what is the best structural modification knowing that other agents are making changes to the city structure at the same time.

Pre-train step is critical to help agents not get lost by the noise in the reward system and be able to converge to a stable policy. We used an ϵ-greedy approach with a rather high exploration rate (ϵ_1), and an (ϵ_2)-decreasing approach for the main train step to balance the exploration and exploitation when choosing actions. Exploration allows an agent to improve its knowledge about the domain in order to maximize the reward in long term and take more informed decisions while exploitation chooses the greedy action to get the most reward by exploiting the agent's current action value estimates. If an agent always takes greedy actions, it may not actually get the most reward and end up with a sub-optimal solution. The ϵ-greedy approach is a simple method to balance exploration and exploitation by choosing between exploration and exploitation randomly by having ϵ refers to the probability of choosing to explore and exploits the rest of the time. The ϵ-decreasing strategy is an extension of the ϵ-greedy strategy. Instead of selecting a fixed value of ϵ like in ϵ-greedy, the ϵ-decreasing strategy chooses an initial ϵ value that gradually decreases over time typically with an exponential decay rate. Using (ϵ)-decreasing approach ensures that exploration occurs in the early stages, but as time progresses fewer iterations are spent on exploration.

4.4.8 DQN Architecture

Deep Q-Learning replaces the regular Q-table with a neural network (NN). Rather than mapping a state-action pair to a q-value, a neural network maps input states to (action, Q-value) pairs. Figure 4.7 shows the illustration of the DQN. Here are the main components of the DQN we used:

- Experience Replay: Deep Q-Learning uses experience replay to learn in small batches in order to avoid skewing the data set distribution of different states, actions, rewards, and next states. [82]. Importantly, the agent doesn't need to train after each step. In our implementation, we use Experience Replay to train on small batches once every b steps rather than every single step. We found this approach really helps speed up our Deep Q-Learning implementation.

- Target Network: To make training more stable, there is an approach, called target network, by which we keep a copy of our main neural network and use it in the Bellman equation. These networks have the same architecture but different weights. Every n_t steps, the weights from the main network are copied to the target network. Using two networks leads to more stability in the learning process and helps the algorithm to learn more effectively.

- Huber loss function: Using the Huber loss function rather than the Mean Squared Error loss function also helps the agent to learn more efficiently. Huber loss function is the combination of Mean Square Error (MSE) and Mean Absolute Error (MAE) means it is quadratic (MSE) when the error is small else MAE. The Huber loss function weighs outliers less than the Mean Squared Error loss function [91, 92].

- Architecture: We use a multilayered NN to estimate the Q-function. The neurons of the input layer are equal to the number of features we consider to represent a state and the output layer size is equal to the number of actions we have. The input layer is followed by three hidden fully-connected layers. The nonlinear approximation is exploited by Rectified Linear Unit (ReLU) for activation in our NN setup. The NN takes data mini-batches of size m_b. We use Adam optimization with learning rate γ. Adam optimization maintains a per-parameter learning rate that improves performance on problems with sparse gradients as opposed to stochastic gradient descent which maintains a single learning rate for all weight updates and the learning rate does not change during training. Adam optimization also maintains per-parameter

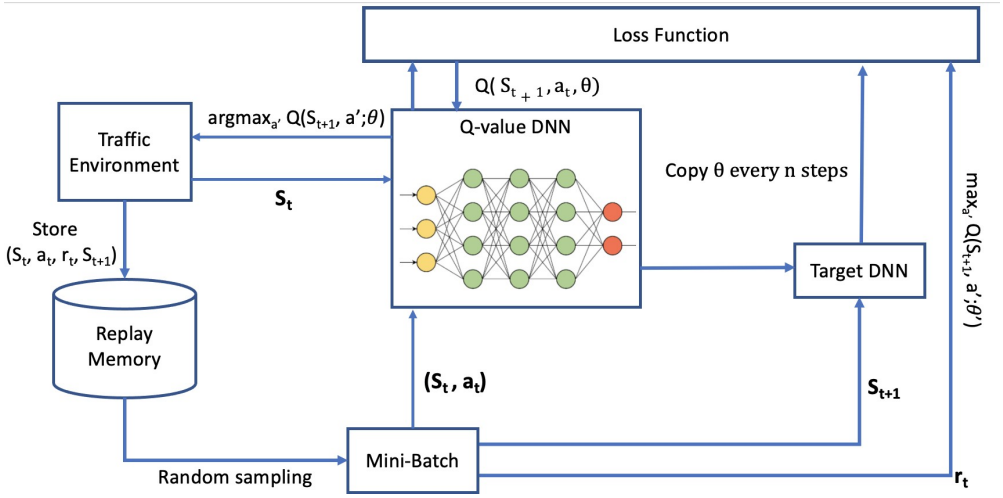


Fig. 4.7: Architecture of Deep Q-Network (DQN) using experience replay and target network

learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight [93].

4.4.9 Parameters

Table 4.1 shows the parameters and the values that have been used in this work. The values are obtained through hyper-parameter tuning. Hyper-parameter tuning objectively searches different values for model hyper-parameters and chooses a subset that results in a model which achieves the best performance. The result of a hyper-parameter optimization is a single set of well-performing hyper-parameters that can be used to configure the model. In this work, we use random search for hyper-parameter tuning [82, 84, 86, 94]. Random search does not check all different hyper-parameter combinations when finding an optimal combination. Instead, it checks a randomly selected fixed number of combinations in multiple iterations. Random search has a very high probability of finding the optimal hyper-parameter combination within the randomly selected combinations. This method is useful to find the optimal hyper-parameter combination quickly and efficiently when the search space is higher dimensional and contains many combinations of values.

Table 4.1: Summary of the parameters used in this work.

Parameters	Definition	Value
m	Duration of time steps	6 min
s	Number of seconds that the signal agent can change at any time step	15
$t_{S_{max}}$	Max duration of signal for a phase	3 min
$t_{S_{min}}$	Min duration of signal for a phase	0 sec
r	Number of seconds that the ramp agent can change at any time step	10
γ	Discount factor	0.98
δ	Learning rate	0.001
b	Number of steps the main network select the batches	4
n_t	Number of steps in which we copy the weights from main network to the target network	8
$t_{r_{max}}$	Maximum value of ramp meter	30
$t_{r_{min}}$	Minimum value of ramp meter	0
l	Length of each road segment. For setting this we followed the convention in [81]	0.5 mile
n_l^b, n_l^a	Number of road segments before and after of a lane agent that considered the feature set	3
$n_{r_i}^b$, $n_{r_i}^a$	Number of road segments before and after of the segment in which the ramp initiates	3
$n_{r_e}^b$, $n_{r_e}^a$	Number of road segments before and after of the segment in which the ramp ends into	3
n_s^b, n_s^a	road segments before and after the road segment in which the signal is located	4
n_v^b,n_v^a	Number of road segments before and after of the segment in which the speed limit is changing for	5
T_r, T_s, T_l,T_v	Number of the time steps we keep track of the road segment features for the ramp agent, signal agent, and lane agent	5
ϵ_1	Exploration rate for pre-train	0.25
ϵ_2	Exploration rate for main-train	0.5-0.0
m	Number of miles the agent can change in each time step	5
$V_{mph_{max}}$	Maximum of speed limit on the road segments	+20%
$V_{mph_{min}}$	Minimum of speed limit on the road segments	-20%
m_b	Size of mini-batches	64 data points
p_l	Penalty for lane changes	4% of the road TVH

4.4.10 Training

We needed a traffic simulator to generate data for training our agents and also to measure the impact of the proposed method on the traffic. We used SMARTS (Scalable Microscopic Adaptive Road Traffic Simulator) [95] traffic simulator. For training, we created 10 synthetic city areas and made random perturbations to each to make 5000 different areas. Perturbations consists of changing the timing of the ramp, changing the lanes of a segment, changing the speed limit and changes made to the signal timings. For each area we generated 15 different traffic profiles where 3 are low traffic, 4 are normal traffic, and the rest are high traffic profiles. High, medium and low traffic are defined based on the capacity of the roads. In that sense, we considered low traffic as the road capacity of 40 percent or less; medium is the traffic between 40 to 80 percent and high traffic is the traffic above the 80 percent of the road capacity. These traffic profiles are calibrated based on real logged data from Utah Department of Transportation (UDOT) [36] on the selected area of Salt Lake City for the preceding 12 months. The traffic profiles are used to create different possibilities and provide the opportunity for the agents to see various city/traffic configuration in training phase.

After the scenario generation step, we have 5000 * 15 scenarios. For pre-train, we assigned one agent to a random road segment in each of these scenarios. Then each of these scenarios is passed to the simulator and simulator runs for 5000 * 15 scenarios where each run of the simulation is 15 step and the duration of each step takes m minute. In each time step the agent can take an action that changes the configuration of the area. When each scenario finished, we use the data of the last 10 steps to make sure that our historical road segment features have stable values as at the first steps, we might not have stable values.

In the main train, we randomly sub-sampled the number of city area configurations to 500, and we consider the 15 traffic profiles. For each scenario, We assign all types of agents to all road segments and let them make modifications at the same time. Similar to the pre-train step, the simulator runs 500 * 15 times where each run is 15 steps and each step takes m minutes where we only use the data of the last 10 step of each scenario.

Random sub-sampling of city area configurations is done because of the high computation time needed for running each configuration with multiple assigned agents in the main-train step.

4.5 Experiments and Results

To evaluate the effectiveness of the proposed algorithm, we designed multiple experiments that we are going to discuss here. For these experiments, we used the map of Salt Lake City area using Open Street Map [34] and imported that to SMARTS traffic simulator [95]. Experiments are designed to study a) single versus multi-agent impact on traffic 4.5.1, b) agents' decisions in different traffic situations 4.5.2, c) patterns of traffic signals 4.5.3, d) effect of lane change penalty 4.5.4 and e) impact of two stage learning 4.3. Each of the experiments is repeated 20 times simulating 20 random days of a year to make sure the results that we are getting is reproducible. Our training step was based on synthetic data (snapshot of the city with random perturbations) in order to provide variation of opportunities for agents to learn. Hence, testing on Salt Lake City is an unseen case for the agents. The outcome of each experiment is compared against the baseline, defined as the evaluation metric for the city without any assigned agent (normal traffic flow) in terms of Total Traveling Time (TTT) of the all generated traffic. The metric TTT is defined on the whole city and it is used as a measure of overall congestion improvement in the city while TVH measures the throughput of the road segment and it is used as a reward mechanism for the agents that are assigned to each road segment to pick the best action.

4.5.1 Single vs Multi-agent Impact

This experiment studies the impact of multi-agent versus single agent in improving TTT in rush and non-rush hour time.

Single agent type

This experiment aims to understand the effect of having a single agent type without the confounding effect of the other types. To do this, for each of the simulations, we just

keep one of the agent types enabled. The results show that the ramp agent alone does not have any significant effect in non-rush hour, the lane agent improves TTT by 2.2% in non-rush hour, speed agent improves TTT by 8.8% and the signal agent has 4.7% improvement over TTT when compared to the baseline. As one can expect, the true power of the agents become visible during the rush hours where lane agents alone can make 23.0% improvement of the TTT metric. During this time, signal agents have 14.4% better TTT while ramp agents didn't show any statistically significant improvement. Speed agent is able to improve the TTT by 4.4% (the effect of speed agent is higher in non-rush hour). These numbers are important because they allow city planners to do a proper cost-benefit analysis when choosing what smart city feature to include in their design. Figure 4.8 summarizes these results.

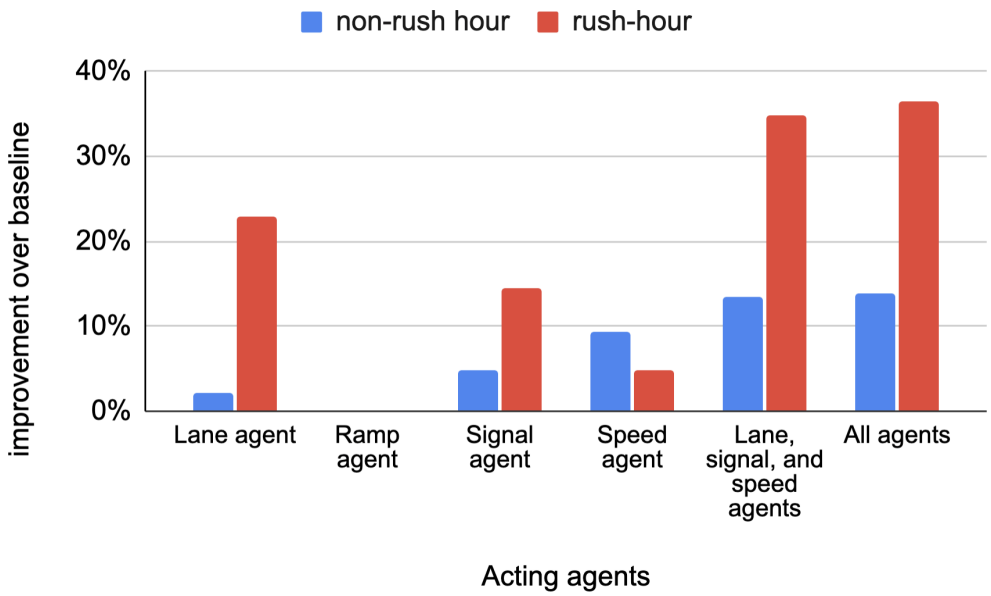


Fig. 4.8: The graphs compares the improvement we got over the baseline traffic with different groups of agents modifying the city during rush hour and non-rush hour time.

Multi-agent with multiple agent types

This is the full-scale experiment in which all the agents are enabled and are changing the environment dynamically. The results show that we got about 13.2% improvement of TTT during non-rush hours implying that the current city designs seem to be good enough for handling the regular traffic. However, during the rush hours, our agents could collaboratively improve the traffic by 36.2%. The interesting point here is that the aggregated effect of all the agents is smaller than the summation of individual agent type's effects in the previous experiment (see Figure 4.8). This can be explained by the fact that the effect of the agents is not independent: changes one agent makes on the traffic pattern can overlap with another agent's effect so the aggregated effect will be smaller than the summation of the two.

In the first experiment, the ramp agent did not show any significant improvement over the traffic flow when it was the only modifier of the system. However, this agent type could potentially help the traffic flow when working with the other agent types. To better understand this, we run the same simulation as above but with ramp agents disabled. As Figure 4.8 shows, disabling the ramp agents reduced the improvement of the traffic compared to when all the agent types are enabled for about 1.5% percentage points in rush hour. This well demonstrates how the ramp agents can help in managing the traffic flow when interacting with the other agent types. However, the ramp impact is not that large as we have the total of 41 ramps in the city area. The number of signals are 1178 and we have 13967 road segments.

4.5.2 Agent's Decisions

In our work, improvement of TTT is the main focus. However, we are also interested in understanding the decisions that agents made in different traffic situations. For this, we look at the distribution of actions per agent type in three different traffic profiles (low, medium and high). The percentage values in Figure 4.9 are defined for each traffic profile and each decision type. The percentages are obtained by dividing the number of the road segments (or directions in case of speed agent) experienced that type of the decision (like

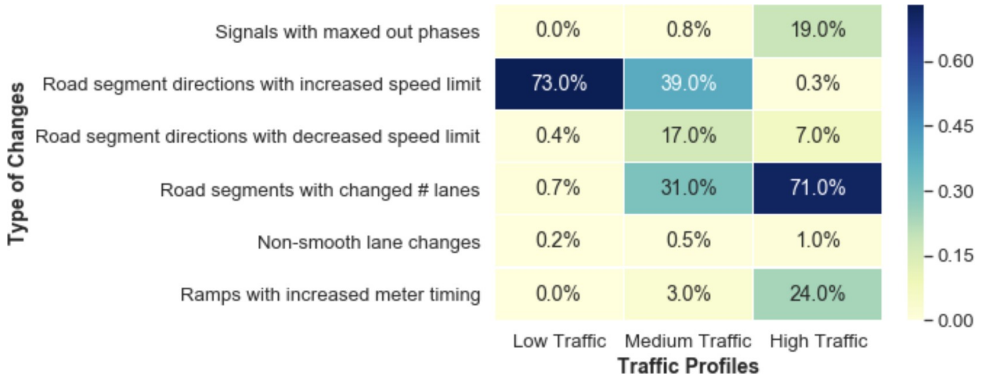


Fig. 4.9: Summary of modifications that our agents made during the three traffic profiles: a) low traffic, b) medium traffic and c) high traffic

lane change or speed change) by all of the road segments/directions in the same traffic profile. For example, the value of "Road segment directions with increased speed limit" in low traffic is 73% which means 73% of the road segment directions with low traffic went through increase speed limit. Here are multiple interesting observations from Figure 4.9.

- In low traffic we didn't have signals with maxed out phase and a very low number of signals maxed out in medium traffic. Even in high traffic, only 19% experienced maxed out. This tells that even in high traffic, increasing the time of a phase to maxed out (180 seconds) is not the best solution. Section 4.5.3 provides a detailed analysis of signal timing patterns.

- In low traffic, there is a high number of road directions with increased speed limit. The percentage is smaller in medium traffic and it is much smaller in high traffic. As expected, the results show that the increase in speed limit is an intuitive way of moving the traffic faster. One takeaway from this observation is, time-dependent variable speed limit can be a good leverage in managing traffic more effectively.

- We see that there are a low percentage of road directions with decreased speed limit in all traffic profiles. The percentage is higher in medium traffic in order to handle more traffic volume at a slower speed while in high traffic the flow is already slowed

down. These results confirm that speed limit reduction might not be the best leverage for improving the traffic flow.

- The results of lane changes show that in high traffic, we had 71% of lane changes which shows this is a very useful action in managing traffic as it increases the capacity of roads which independently improved the overall TTT by 23% in rush hour based on Figure 4.8.

- Non-smooth lane changes defined as road segments that their previous and next segments have the same number of the lanes which is different from their own number of lanes. The high value for the changed lanes combined with the small value of the non-smooth lane changes show the smooth adaptiveness of lane agents to different traffic situations.

- In Figure 4.8, we saw that ramp agent didn't have a significant impact independently and in the multi-agent phase it was able to improve the traffic by 1.5 percentage point in rush hour. Figure 4.9 shows that we had meter timing increase in 24% of the ramps in high traffic while this percentage is very low in medium and low traffic.

The two major changes that we see in Figure 4.9 are increasing the speed limit in low traffic by 73% and changing number of lanes in high traffic by 71%. Figure 4.10 shows the distribution of the magnitude of the changes in each case. As it can be seen, the majority of speed limit increases is 5 mph and the majority of lane changes are changing by 1 lane.

4.5.3 Patterns of traffic signals

In order to have a better understanding of changes of signals, we look at the pattern of signal phase timings in rush and non-rush hour. Figure 4.11 shows possible states of a signal in a north-south direction. The duration of each signal phase can vary from 0 seconds to 180 seconds which is the maxed out state [44]. In this experiment, we look at the phase timing pattern to see how signal agent modifies the base phase timing in rush and non-rush hour.

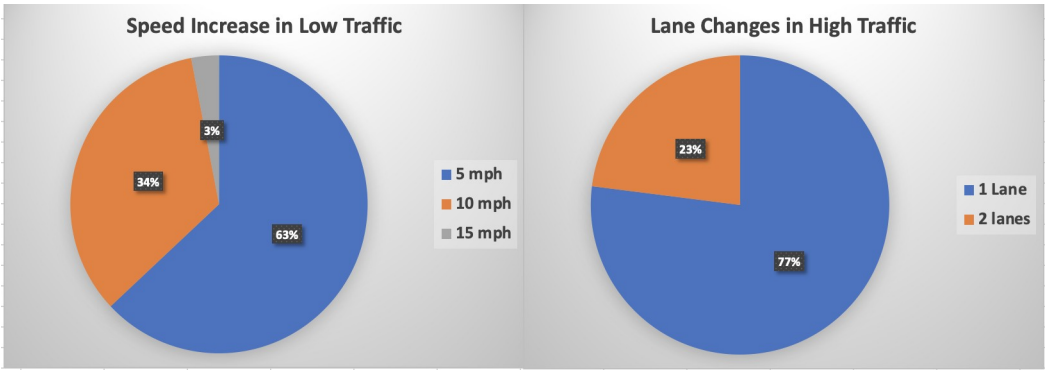


Fig. 4.10: Distribution of speed limit changes in low traffic and lane changes in high traffic

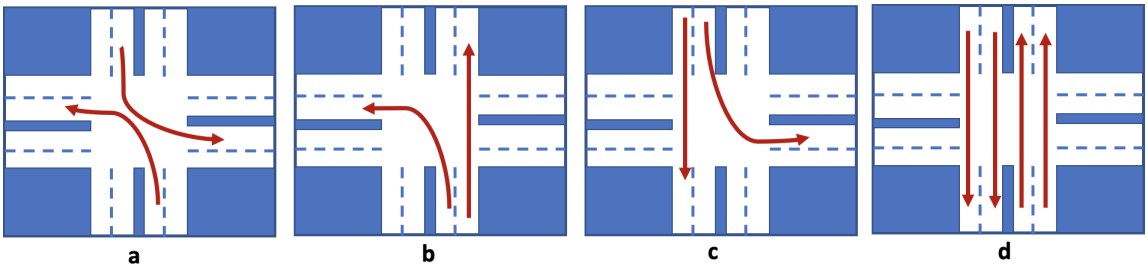


Fig. 4.11: Possible signal phases for a north-south direction.

Figure 4.12 shows the pattern of phase timing in baseline, rush hour and non rush hour for all of the signals used in the simulation. Followings are the findings from the patterns:

- Comparison of the patterns in non-rush and rush hour shows how dynamically our signal agents responds to the traffic situation and modifies the phase timings to adjust to the traffic situation.
- In non-rush hour, number of zeros are expanded in comparison to baseline which due to the elimination of lots of state (b) and (c) because of the low congestion.
- Phase timing in non-rush hour is distributed to the range of 10 to 80 seconds. In this distribution, the timings between 10 to 40 are majorly state (a),(b) and (c) with few state (d) and the timings after 40 are majorly state (d).

- In rush hour, we have a long tail along with an increased number of maxed outs which indicates that our algorithm modifies signal timings to optimize the traffic. Hence, some phases need to be longer and some need to get maxed out to manage the traffic.
- In rush hour, phase timings greater than 80 second is all state (d) which shows that the signal agent increased the timing of state (d) to manage the congestion.

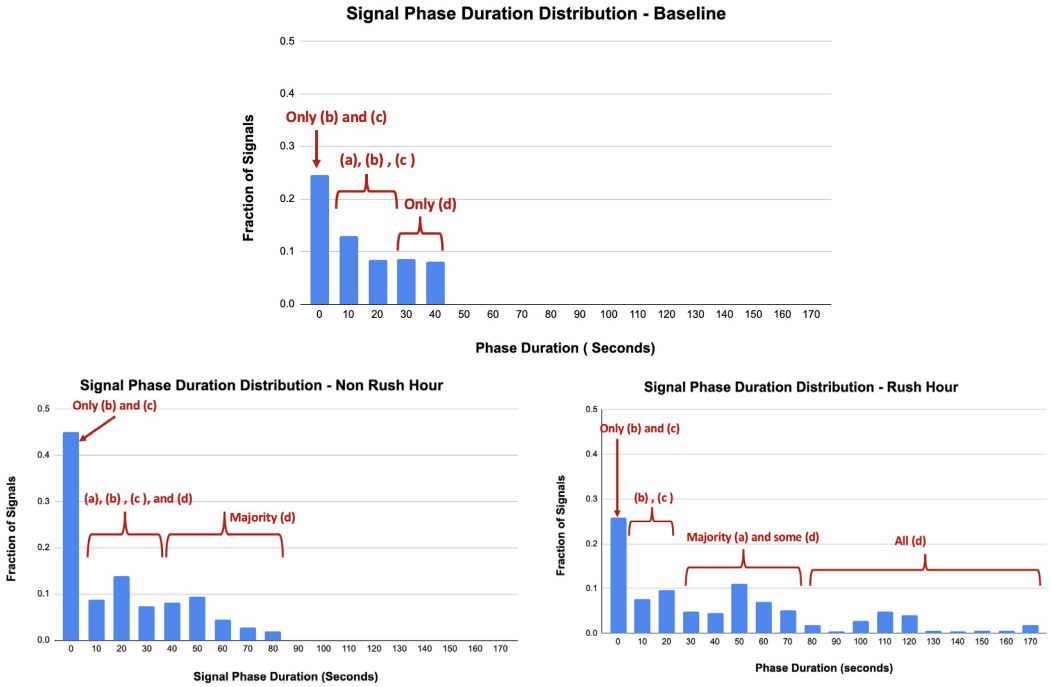


Fig. 4.12: Signal Pattern for base, none rush hour and rush hour.

4.5.4 Effect of lane change penalty

One of our concerns during the design phase of this project was the possibility of unnecessary actions by the lane agents; for example, the periodic actions of the lane agent, i.e. to increase number of the lanes at time t, and decrease it at $t + 1$, and then increase it again at $t + 2$. For changing the number of lanes in SMART simulator [95], the target lanes are blocked from one side and get allocated to the other side of the road segment. In this

process the simulator allocates the traffic of the target lane(s) to the other active lanes of the same direction. The simulator also re-routs part of the traffic which their travel time will be impacted significantly from the precedent road segments. To prevent the unnecessary lane changes, we introduced a penalty for the changes by lane agents (p_l) in the reward function. Non-smooth lane changes is one of the indicators of unnecessary lane changes which causes inconsistency of lanes in consequent road segments. Table 4.2 summarizes the impact of lane change penalty on the percentage of non-smooth lane changes.

Table 4.2: Summary of the impact lane change penalty in non-smooth lane changes.

	Non-rush hour	**Rush hour**
Non-smooth lane change - No Penalty	14.4%	3.6%
Non-smooth lane change - Penalty	0.5%	0.9%

4.5.5 Importance of two-stage learning

As mentioned previously, we have proposed a two-stage training process to prevent the problem of non-convergence often seen in the RL systems with many different actors. Our system is an extreme case of uncertainty in the reward outcome as the observed reward might be due to the direct and indirect actions of many agents. Table 4.3 summarized the comparison of two stage learning (pre-train and main train) with only main train without pre-train. Convergence is defined as the number of simulations that all agents converged to a stable policy. Improvement of TTT is defined based on the improvement the converged simulations made in comparison to baseline. Baseline is defined as the city without any assigned agent. The model with converged solutions in main train only significantly under-performed the model trained with pre-train and main train (11.3% improvement over the baseline compared to 36.2% in 2-stage).

4.6 Conclusion and Future works

In this research, we proposed a dynamic traffic management framework in a smart

Table 4.3: Comparison of two stage learning pre-train and main train with only main train without pre-train step

Learning type	Convergence	Improvement of TTT
Main train only	3%	11.3%
Pre-train and Main train	100%	36.2%

city paradigm. In the proposed model, we utilize the structural modifications of the city network such as changing the number of lanes, ramp metering, changing the speed limit of road segments, and modifying signal timing to alleviate the traffic congestion in the overall city network. The proposed framework uses a multi-agent reinforcement learning with the goal of finding the best structural changes based on multiple dynamic factors through maximizing the cumulative reward over the set of possible actions in state space. Once the RL agents are trained on all possible features, they can easily adopt to the dynamic changes of the traffic. Our results shows that the proposed framework improves the total travel time (TTT) by 36.2% during rush hours in Salt Lake City area. There are few directions that we can improve upon the current work. First, in this work, we did not deal with the problem of agents' connectivity. We assume that agents can easily get the information from any dependent road segment. This is a separate line of work that can be further expanded with the consideration of all the uncertainties and sensitivity of agents to the lost or invalid inputs. Also, we assumed that the road segments have equal lengths. One can potentially consider variable length road segments based on specific features and study the impact of different lengths for road segments.

CHAPTER 5

Conclusion

Traffic congestion occurs when the volume of traffic is greater than the available street capacity. In an era of accelerated urbanization around the world, the ability to travel freely is more critical than ever before. The cost of traffic congestion is large, and solutions to reduce the congestion save time, money, and environmental pollution. The traditional traffic management system was designed when the number of vehicles were lower, and they were able to efficiently manage the traffic. Now, due to the massive increase in transportation needs in many cities, there is a need for smarter traffic management solution. The main goals of intelligent traffic management systems are to improve the traffic flow and reduce the traffic load in highly congested areas. In this research, we look at the applications of multi agent systems in intelligent traffic management. Since multi-agent systems are built on the concept of cooperation between intelligent agents, it is a powerful paradigm to test and analyze the real world domain scenarios. Our intelligent traffic management framework has two main parts: a) scalable stochastic path planning and b) intelligent city-wide traffic optimization.

Stochastic path planning focuses on path finding in the real-world domain in which edge weights are not fixed but are stochastically affected by the time of the day/week. Stochastic edge costs are considered as independent Gaussian random variables, whose distributions are extracted from data monitored by the Utah Department of Transportation. Inspired by time-dependent traffic situation, we parameterize these distributions by time, which allows us to speak of time-dependent path costs and study the problems of reaching a goal by a deadline and delaying departure to minimize traversal-to-goal time. The best path is the path with lowest cost. The cost is based on travel time which depends on the level of congestion in the network and congestion highly related to the time of the day/week. Three different cost functions (linear, exponential, and step cost function) are investigated.

Users can pursue two main goals: 1) picking the least risky path and 2) picking the shortest travel time while still meeting the deadline. In the path planning domain, we first focus on finding a path that satisfies the domain constraints and agents 'goals. Results show that in rush hour time when the congestion is high, using a smart/realistic method of path finding is crucial while in non-peak hours paths are almost the same. In addition, the findings show that a suitable path finding approach must consider different aspects such as path's mean, path's variance and the deadline to provide optimal options. Removing any of these factors may result in having a path which either violates the deadline or has a very high travel time variance. Also, the comparison of the two agents' goals with shortest length path proves that the shortest-length path is not always the best path due to the fact that it is highly congested in rush hour times as it is traditionally the first choice of most drivers. Highest probability paths have less variance because it takes the most secure path, while the smallest travel time paths have the lowest mean and might have a high variance.

Then, we built on top of the stochastic path planning and propose a scalable algorithm that is practical in large scale path planning applications for the use cases where agents have goals, and the planner aims to satisfies agents' goals rather than just providing a path which can move agents from a source node to a destination node. The city is modelled as a large scale graph. Agents have two types of goals: 1) seekingr the path with highest probability of reaching destination before deadline, and 2) seeking the shortest travel duration while they are flexible on the time they can leave. Associated with each path is a defined cost. The goal of the path planner is to find a path that satisfies the agents' goals. For expediting the path planning process, the city is partitioned and each partition is represented with an exemplar. The exemplar of each partition is decided based on random walk centrality. For partitioning the city graph, we used community detection and clustering approaches. When a path planning request comes, source and destination nodes are connected to their corresponding exemplars with respect to the path direction and the path between exemplars is retrieved. The paths between exemplars are stored in distance oracles based on the preceding year data at the time of update and the oracles are updated every week to reflect the recent changes on

the network. Results show that among all of the graph clustering approaches, community-based approaches produce closer results to exact path planning approach. Approximation provides paths with mean and variance which are not exact but close to that exact paths, while the solution is space and time efficient. The proposed framework handles queries in real time while the approximate paths are 3 to 5 percent longer than exact paths

Our intelligent city wide traffic optimization framework is a dynamic model based on a smart city paradigm. In the proposed model, we utilize the structural modifications of the city network such as changing the number of lanes, ramp metering, changing the speed limit of road segments, and modifying signal timing to alleviate the traffic congestion in the overall city network. These structural changes not only impact the flow in both directions of the road segment but also the flow at surrounding road segments. Therefore, we need to consider the impact of structural changes not only on each road segment, but also on dependent road segments and the whole network. The proposed framework uses a multi-agent reinforcement learning with the goal of finding the best structural changes based on multiple dynamic factors through maximizing the cumulative reward over the set of possible actions in state space. Once the RL agents are trained on all possible features, they can easily adopt to the dynamic changes of the traffic. The learning process has two steps: a) single agent is the only modifier and learns the initial policy, b) agents update their policy by interacting by other agents. First step is critical to reduce the noise in the reward system and be able to converge to a stable policy. Then, in the second step, the agents update their policy to take into account the indirect interaction with the other agents. The main advantage of reinforcement learning methods is, with proper training, they can easily adapt to the changes of the domain without the need of the re-computation. Our results shows that the proposed framework improves the total travel time (TTT) by 36.2% during rush hours in Salt Lake City area. Also, the biggest changes were either in modifying speed limit or changing number of lanes. In addition, we showed that using two-stage learning we had 100 percent coverage with 36.2 TTT improvement in oppose to 3 percent coverage in one stage learning.

REFERENCES

[1] PennDOT, "Pennsylvania Department of Transportation," Harrisburg, PA, 2021. [Online]. Available: https://www.penndot.gov/Pages/default.aspx

[2] K. Ahmadi, "Decision making using trust and risk in self-adaptive agent organization," *MS. Thesis, All Graduate Theses and Dissertations, Utah State University*, 2014. [Online]. Available: digitalcommons.usu.edu/etd/2159

[3] M. Shahgholian and D. Gharavian, "Advanced traffic management systems: An overview and a development strategy," *arXiv: Signal Processing*, 2018.

[4] M. Niknami and S. Samaranayake, "Tractable Pathfinding for the Stochastic On-Time Arrival Problem." Springer, Cham, June 2016, pp. 231–245. [Online]. Available: https://arxiv.org/abs/1408.4490

[5] E. Nikolova, J. A. Kelner, M. Brand, and M. Mitzenmacher, "Stochastic shortest paths via quasi-convex maximization," ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pp. 552–563. [Online]. Available: https://link.springer.com/chapter/10.1007/11841036_50

[6] S. Lim, H. Balakrishnan, D. Kenneth, G. Samuel, R. Madden, and D. Rus, "Method and apparatus for traffic-aware stochastic routing and navigation," 2020. [Online]. Available: https://patents.google.com/patent/US10535256B1/en

[7] R. Geisberger, P. Sanders, D. Schultes, and C. Vetter, "Exact Routing in Large Road Networks Using Contraction Hierarchies," *Transportation Science*, vol. 46, no. 3, pp. 388–404, Aug. 2012. [Online]. Available: http://dx.doi.org/10.1287/trsc.1110.0401

[8] Yaoxin Wu, Wei Chen, Xuexi Zhang, and Guangjun Liao, "Improving the performance of arrival on time in stochastic shortest path problem." 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), nov 2016, pp. 2346–2353. [Online]. Available: http://ieeexplore.ieee.org/document/7795934/

[9] S. Lim and D. Rus, "Congestion-Aware Multi-Agent Path Planning: Distributed Algorithm and Applications," *The Computer Journal*, vol. 57, no. 6, pp. 825–839, Jun. 2014. [Online]. Available: https://academic.oup.com/comjnl/article/57/6/825/378655

[10] D. Wilkie, J. v. d. Berg, M. Lin, and D. Manocha, "Self-aware traffic route planning," in *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, ser. AAAI'11. AAAI Press, 2011, p. 1521–1527.

[11] C. E. Sigal, A. Pritsker, and J. Solberg, "The stochastic shortest route problem," *Operation Research*, vol. 28, pp. 1122–1129, 1980.

[12] D. Pérez-Méndez, C. Gershenson, M. E. Lárraga, and J. L. Mateos, "Modeling adaptive reversible lanes: A cellular automata approach." *PlOS ONE*, vol. 16, pp. 1–16, January 2021. [Online]. Available: https://doi.org/10.1371/journal.pone.0244326

[13] J. Wu, H. Sun, Z. Gao, and H. Zhang, "Reversible lane-based traffic network optimization with an advanced traveller information system," *Engineering Optimization*, vol. 41, no. 1, pp. 87–97, 2009.

[14] M. Hausknecht, T.-C. Au, P. Stone, D. Fajardo, and T. Waller, "Dynamic lane reversal in traffic management," in *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, 2011, pp. 1929–1934.

[15] K. F. Chu, A. Y. Lam, and V. O. Li, "Dynamic lane reversal routing and scheduling for connected autonomous vehicles," in *2017 International Smart Cities Conference (ISC2)*, 2017, pp. 1–6.

[16] S. El-Tantawy, B. Abdulhai, and H. Abdelgawad, "Multiagent reinforcement learning for integrated network of adaptive traffic signal controllers (marlin-atsc): Methodology and large-scale application on downtown toronto," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 3, pp. 1140–1150, 2013.

[17] M. Aslani, S. Seipel, M. S. Mesgari, and M. Wiering, "Traffic signal optimization through discrete and continuous reinforcement learning with robustness analysis in downtown tehran," *Advanced Engineering Informatics*, vol. 38, pp. 639–655, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474034617302598

[18] U. Gunarathna, H. Xie, E. Tanin, S. Karunasekara, and R. Borovica-Gajic, "Real-time lane configuration with coordinated reinforcement learning," in *Machine Learning and Knowledge Discovery in Databases: Applied Data Science Track*, Y. Dong, D. Mladenić, and C. Saunders, Eds. Cham: Springer International Publishing, 2021, pp. 291–307.

[19] E. R. Müller, R. C. Carlson, W. Kraus, and M. Papageorgiou, "Microsimulation analysis of practical aspects of traffic control with variable speed limits," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 1, pp. 512–523, 2015.

[20] K. Ahmadi and V. H. Allan, "Trust-based decision making in a self-adaptive agent organization," *ACM Trans. Auton. Adapt. Syst.*, vol. 11, no. 2, jun 2016. [Online]. Available: https://doi.org/10.1145/2839302

[21] N. Chiabaut, C. Buisson, and L. Leclercq, "Fundamental Diagram Estimation Through Passing Rate Measurements in Congestion," *IEEE Transactions on Intelligent Transportation Systems*, vol. 10, no. 2, pp. 355–359, jun 2009. [Online]. Available: http://ieeexplore.ieee.org/document/4914846/

[22] Y. Y. Fan, R. E. Kalaba, and J. E. Moore, "Arriving on Time," *Journal of Optimization Theory and Applications*, vol. 127, no. 3, pp. 497–513, dec 2005. [Online]. Available: http://link.springer.com/10.1007/s10957-005-7498-5

[23] X. Pi and Z. S. Qian, "A stochastic optimal control approach for real-time traffic routing considering demand uncertainties and travelers' choice heterogeneity," *Transportation Research Part B: Methodological*, vol. 104, no. Supplement C, pp. 710 – 732, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0191261516309924

[24] E. W. Dijkstra and E. W., "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, December 1959. [Online]. Available: http://link.springer.com/10.1007/BF01386390

[25] P. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968. [Online]. Available: http: //ieeexplore.ieee.org/document/4082128/

[26] E. Nikolova, "Approximation algorithms for reliable stochastic combinatorial optimization," in *Proceedings of the 13th International Conference on Approximation: Algorithms and Techniques*, ser. APPROX/RANDOM'10. Berlin, Heidelberg: Springer-Verlag, 2010, p. 338–351.

[27] R. Geisberger, M. Kobitzsch, and P. Sanders, "Route planning with flexible objective functions," in *Proceedings of the Meeting on Algorithm Engineering and Expermiments*, ser. ALENEX '10. USA: Society for Industrial and Applied Mathematics, 2010, p. 124–137.

[28] A. Kotsialos, M. Papageorgiou, C. Diakaki, Y. Pavlis, and F. Middelham, "Traffic Flow Modeling of Large-Scale Motorway Networks Using the Macroscopic Modeling Tool METANET," *IEEE Transaction on Intelligent Transportation Systems*, vol. 3, no. 4, 2002. [Online]. Available: https://ai2-s2-pdfs.s3.amazonaws.com/491b/caedeb5f14c0e2bf41756120f0da9d8e3477.pdf

[29] E. D. Miller-Hooks and H. S. Mahmassani, "Least Expected Time Paths in Stochastic, Time-Varying Transportation Networks," *Transportation Science*, vol. 34, no. 2, pp. 198–215, may 2000. [Online]. Available: http://pubsonline.informs.org/doi/abs/10.1287/trsc.34.2.198.12304

[30] Z. Cao, H. Guo, J. Zhang, F. Oliehoek, and U. Fastenrath, "Maximizing the probability of arriving on time: A practical q-learning method," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, ser. AAAI'17. AAAI Press, 2017, p. 4481–4487.

[31] A. V. Goldberg, H. Kaplan, and R. F. Werneck, "Reach for a: Efficient point-to-point shortest path algorithms," in *Workshop on Algorithm Engineering and Experiments*, 2006, pp. 129–143.

[32] R. J. Gutman, "Reach-based routing: A new approach to shortest path algorithms optimized for road networks." in *Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments*. SIAM, 2004, pp. 100–111.

[33] R. Bauer, M. Baum, I. Rutter, and D. Wagner, *On the Complexity of Partitioning Graphs for Arc-Flags*. Journal of Graph Algorithms and Applications, January 2013.

[34] M. Haklay and P. Weber, "OpenStreetMap: User-Generated Street Maps," *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, oct 2008. [Online]. Available: http://ieeexplore.ieee.org/document/4653466/

[35] D. Feillet, P. Dejax, M. Gendreau, and C. Gueguen, "An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems," *Networks*, vol. 44, no. 3, pp. 216–229, oct 2004. [Online]. Available: http://doi.wiley.com/10.1002/net.20033

[36] UDOT, "UDOT: Utah Department of Transportation," SLC, UTA, 2020. [Online]. Available: https://www.udot.utah.gov/main/f?p=100:6:0::::V,T:,1

[37] D. Long, U. ICAPS 2006 (16 2006.06.06-10 Windermere, and U. International Conference on Automated Planning and Scheduling (16 2006.06.06-10 Windermere, *Optimal route planning under uncertainty*. AAAI Press, 2006. [Online]. Available: http://dl.acm.org/citation.cfm?id=3037122

[38] A. M. Campbell, M. Gendreau, and B. W. Thomas, "The orienteering problem with stochastic travel and service times," *Annals of Operations Research*, vol. 186, no. 1, pp. 61–81, jun 2011. [Online]. Available: http: //link.springer.com/10.1007/s10479-011-0895-2

[39] H. C. LAU, W. YEOH, P. VARAKANTHAM, and D. T. NGUYEN, "Dynamic Stochastic Orienteering Problems for Risk-Aware Applications," *Uncertainty in Artificial Intelligence: Proceedings of the Twenty-Eighth Conference: August 15-17 2012, Catalina Island, United States*, aug 2012. [Online]. Available: http://ink.library.smu.edu.sg/sis{ }research/1610

[40] P. Besnard, S. Hanks, and K. Larson, *Uncertainty in artificial intelligence : proceedings of the Eleventh Conference (1995) : August 18-20, 1995*. Morgan Kaufmann Publishers, 1995. [Online]. Available: http://dl.acm.org/citation.cfm?id= 2074158.2074219

[41] G. Bachman, L. Narici, and E. Beckenstein, *Fourier and Wavelet Analysis*, ser. Universitext. New York, NY: Springer New York, 2000. [Online]. Available: http://link.springer.com/10.1007/978-1-4612-0505-0

[42] D. Basu and R. G. Laha, "On Some Characterizations of the Normal Distribution," *Sankhya: The Indian Journal of Statistics(1954)*, vol. 13, pp. 359–362, 1954. [Online]. Available: https://www.jstor.org/stable/25048183

[43] K. Ahmadi and V. H. Allan, "Scalable stochastic path planning under congestion," in *Proceedings of the 13th International Conference on Agents and Artificial Intelligence*. SCITEPRESS, 2021, pp. 454–463. [Online]. Available: https://doi.org/10.5220/0010394104540463

[44] A. Loder, L. Ambühl, M. Menendez, and K. W. Axhausen, "Understanding traffic capacity of urban networks," *Scientific reports*, vol. 9, no. 1, pp. 1–10, 2019.

[45] B. Büchel and F. Corman, "Review on statistical modeling of travel time variability for road-based public transport," *Frontiers in Built Environment*, vol. 6, p. 70, 2020. [Online]. Available: https://www.frontiersin.org/article/10.3389/fbuil.2020.00070

[46] Y. Fan and Y. Nie, “Optimal Routing for Maximizing the Travel Time Reliability,” *Networks and Spatial Economics*, vol. 6, no. 3, pp. 333–344, Sep. 2006. [Online]. Available: https://doi.org/10.1007/s11067-006-9287-6

[47] Y. M. Nie and X. Wu, “Shortest path problem considering on-time arrival probability,” *Transportation Research Part B: Methodological*, vol. 43, no. 6, pp. 597–613, Jul. 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0191261509000174

[48] S. Samaranayake, S. Blandin, and A. M. Bayen, “Speedup Techniques for the Stochastic on-time Arrival Problem,” in *ATMOS*, 2012.

[49] R. Bauer, T. Columbus, I. Rutter, and D. Wagner, “Search-space size in contraction hierarchies,” *Theoretical Computer Science*, vol. 645, Jul. 2016.

[50] B. Yang, J. Dai, C. Guo, C. S. Jensen, and J. Hu, “Pace: A path-centric paradigm for stochastic path finding,” *The VLDB Journal*, vol. 27, no. 2, p. 153–178, Apr. 2018. [Online]. Available: https://doi.org/10.1007/s00778-017-0491-4

[51] J. Hu, C. Guo, B. Yang, and C. S. Jensen, “Stochastic weight completion for road networks using graph convolutional networks,” in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, 2019, pp. 1274–1285.

[52] K. Ahmadi and V. H. Allan, “Congestion-aware stochastic path planning and its applications in real world navigation,” in *Proceedings of the 13th International Conference on Agents and Artificial Intelligence, ICAART 2021*. SCITEPRESS, 2021, pp. 947–956. [Online]. Available: https://doi.org/10.5220/0010267009470956

[53] K. Ahmadi and V. Allan, “Stochastic path finding under congestion,” in *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*. SCITEPRESS, 2017, pp. 135–140. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8560774

[54] S. C. Frederik Lardinois, Jochen Topf, “OpenStreetMap,” *UIT Cambridge*, 2011. [Online]. Available: https://www.goodreads.com/work/best_book/9555000-openstreetmap-die-freie-weltkarte-nutzen-und-mitgestalten

[55] J. Macqueen, “Some methods for classification and analysis of multivariate observations,” in *5-th Berkeley Symposium on Mathematical Statistics and Probability*, 1967, pp. 281–297. [Online]. Available: https://digitalassets.lib.berkeley.edu/math/ucb/text/math_s5_v1_article-17.pdf

[56] Y. Cheng, “Mean shift, mode seeking, and clustering,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 8, pp. 790–799, 1995.

[57] R. Tibshirani, G. Walther, and T. Hastie, “Estimating the number of clusters in a dataset via the gap statistic,” vol. 63, pp. 411–423, 2000.

[58] E. Parzen, “On Estimation of a Probability Density Function and Mode,” *The Annals of Mathematical Statistics*, vol. 33, no. 3, pp. 1065 – 1076, 1962. [Online]. Available: https://doi.org/10.1214/aoms/1177704472

[59] N. R. Smith, P. N. Zivich, L. M. Frerichs, J. Moody, and A. E. Aiello, "A guide for choosing community detection algorithms in social network studies: The question alignment approach," *American Journal of Preventive Medicine*, vol. 59, no. 4, pp. 597–605, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0749379720302166

[60] R. Clark and M. Macdonald, "Eigenvector-based community detection for identifying information hubs in neuronal networks," *bioRxiv*, 2018. [Online]. Available: https://www.biorxiv.org/content/early/2018/10/30/457143

[61] P. Pons and M. Latapy, "Computing communities in large networks using random walks," *Journal of Graph Algorithms and Applications*, pp. 191–218, 2006. [Online]. Available: https://link.springer.com/chapter/10.1007/11569596_31

[62] S. E. Garza and S. E. Schaeffer, "Community detection with the Label Propagation Algorithm: A survey," *Physica A: Statistical Mechanics and its Applications*, vol. 534, p. 122058, Nov. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378437119312026

[63] Z. Yang, R. Algesheimer, and C. Tessone, "A Comparative Analysis of Community Detection Algorithms on Artificial Networks," *Scientific Reports*, vol. 6, Aug. 2016.

[64] R. W. Floyd, "Algorithm 97: Shortest path," *Commun. ACM*, vol. 5, no. 6, p. 345, Jun. 1962. [Online]. Available: https://doi.org/10.1145/367766.368168

[65] G. Sabidussi, "The centrality index of a graph," *Psychometrika*, vol. 31, pp. 581–603, 1966. [Online]. Available: https://link.springer.com/article/10.1007/BF02289527#citeas

[66] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[67] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0377042787901257

[68] K. Ahmadi and V. H. Allan, "Smart city: Application of multi-agent reinforcement learning systems in adaptive traffic management," in *2021 IEEE International Smart Cities Conference (ISC2)*, 2021, pp. 1–7. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9562951

[69] C. Simon and M. Tagliabue, "Feeding the behavioral revolution: Contributions of behavior analysis to nudging and vice versa," *Journal of Behavioral Economics for Policy*, vol. 2, no. 1, pp. 91–97, March 2018. [Online]. Available: https://ideas.repec.org/a/beh/jbepv1/v2y2018i1p91-97.html

[70] J. Li and A. Deshpande, "Maximizing expected utility for stochastic combinatorial optimization problems," *2011 IEEE 52nd Annual Symposium on Foundations of*

Computer Science, Oct 2011. [Online]. Available: http://dx.doi.org/10.1109/FOCS.2011.33

[71] B. C. R. Rota and M. Simic, "Traffic flow optimization on freeways," *Procedia Computer Science*, vol. 96, pp. 1637–1646, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S187705091632021X

[72] Y. Liu, S. Blandin, and S. Samaranayake, "Stochastic on-time arrival problem in transit networks," *Transportation Research Part B: Methodological*, vol. 119, pp. 122 – 138, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0191261518306702

[73] J. Pan, I. S. Popa, K. Zeitouni, and C. Borcea, "Proactive vehicular traffic rerouting for lower travel time," *IEEE Transactions on Vehicular Technology*, vol. 62, no. 8, pp. 3551–3568, 2013.

[74] E. F. Grumert and A. Tapani, "Bottleneck mitigation through a variable speed limit system using connected vehicles," *Transportmetrica A: Transport Science*, vol. 16, no. 2, pp. 213–233, 2020. [Online]. Available: https://doi.org/10.1080/23249935.2018.1547332

[75] L. R. Ford and D. R. Fulkerson, "Constructing maximal dynamic flows from static flows," *Operations Research*, vol. 6, no. 3, pp. 419–433, 1958. [Online]. Available: http://www.jstor.org/stable/167028

[76] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[77] E. Walraven, M. T. Spaan, and B. Bakker, "Traffic flow optimization: A reinforcement learning approach," *Engineering Applications of Artificial Intelligence*, vol. 52, pp. 203–212, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0952197616000038

[78] S. Padakandla, "A survey of reinforcement learning algorithms for dynamically varying environments," *ACM Computing Surveys*, vol. 54, no. 6, july 2021. [Online]. Available: https://doi.org/10.1145/3459991

[79] K. Yau, J. Qadir, H. Khoo, M. Ling, and P. Komisarczuk, "A survey on reinforcement learning models and algorithms for traffic signal control," *ACM Computing Surveys*, vol. 50, no. 3, Jun. 2017.

[80] A. Paul and S. Mitra, "Deep reinforcement learning based traffic signal optimization for multiple intersections in its," in *2020 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, 2020, pp. 1–6.

[81] W. Zhou, M. Yang, M. Lee, and L. Zhang, "Q-learning-based coordinated variable speed limit and hard shoulder running control strategy to reduce travel time at freeway corridor," *Transportation Research Record*, vol. 2674, no. 11, pp. 915–925, 2020. [Online]. Available: https://doi.org/10.1177/0361198120949875

[82] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[83] C. Boutilier, "Planning, learning and coordination in multiagent decision processes," in *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge*, ser. TARK '96. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996, p. 195–210.

[84] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236

[85] R. Bellman, "On the theory of dynamic programming," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 38, no. 8, p. 716, 1952.

[86] Y. Chen, P. Mehrotra, N. K. S. Samala, K. Ahmadi, V. Jivane, L. Pang, M. Shrivastav, N. Lyman, and S. Pleiman, "A multiobjective optimization for clearance in walmart brick-and-mortar stores," *INFORMS Journal on Applied Analytics*, vol. 51, no. 1, pp. 76–89, feb 2021.

[87] A. Nowé and T. Brys, "A gentle introduction to reinforcement learning," in *Scalable Uncertainty Management*, S. Schockaert and P. Senellart, Eds. Cham: Springer International Publishing, 2016, pp. 18–32.

[88] J. Zhang, H. Chang, and P. Ioannou, "A simple roadway control system for freeway traffic," in *2006 American Control Conference*, 2006, pp. 6 pp.–.

[89] H. van Hasselt, *Reinforcement Learning in Continuous State and Action Spaces*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 207–251.

[90] D. Ma, B. Zhou, X. Song, and H. Dai, "A deep reinforcement learning approach to traffic signal control with temporal traffic pattern mining," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–12, 2021.

[91] P. J. Huber, *Robust Estimation of a Location Parameter*. New York, NY: Springer New York, 1992, pp. 492–518. [Online]. Available: https://doi.org/10.1007/978-1-4612-4380-9_35

[92] K. Ahmadi and V. H. Allan, "Checking the reliability of information sources in recommendation based trust decision making," in *PRIMA 2015: Principles and Practice of Multi-Agent Systems*, Q. Chen, P. Torroni, S. Villata, J. Hsu, and A. Omicini, Eds. Cham: Springer International Publishing, 2015, pp. 367–382.

[93] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[94] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” *J. Mach. Learn. Res.*, vol. 13, no. null, p. 281–305, Feb. 2012.

[95] K. Ramamohanarao, H. Xie, L. Kulik, S. Karunasekera, E. Tanin, R. Zhang, and E. B. Khunayn, “Smarts: Scalable microscopic adaptive road traffic simulator,” *ACM Trans. Intell. Syst. Technol.*, vol. 8, no. 2, Dec. 2016. [Online]. Available: https://doi.org/10.1145/2898363

CURRICULUM VITAE

Kamilia Ahmadi

Education

- PhD: Computer Science, Artificial Intelligence, (2014, 2021), Utah State University, GPA: 4/4
- Management Essentials, Business administration and management – General, Harvard Business School Online, 2020.
- MS: Computer Science, Artificial Intelligence (2011, 2014), Utah State University, GPA: 3.87/4.
- BS: Computer Engineering, (2002-2007), Shiraz University, Shiraz, Iran.

Professional Skills

- Management skills: team leadership, strategic thinking, conflict resolution, delegation, data-driven decision making.
- Data Analysis and Machine Learning: Classical machine learning, reinforcement learning, deep learning, causal inference, experiment design, information retrieval.
- Programming languages: Python , C, Java, SQL and BigQuery.

Selected Projects

- Developing speed elasticity engine to measure the demand elasticity of Walmart items due to the change in delivery channel.

- Causal inference to measure the impact of main drivers of sale.
- Personalizing product selection for targeting customers via push notification
- Hyperlocal modelling: Personalized product offerings to the particular preferences of customers of each store.
- Enhancing the ranking of search results based on store sales data in Walmart Search Engine
- Query clustering for enhancing the results of search engine (Thomson Reuters Summer Internship).
- Simulation for modelling Smart City: Intelligent stochastic path planning simulation for large scale multi agent systems (PhD project).

Work Experience

- Senior Data Science Manager II / Principal Data Scientist (X6)/ Walmart Labs - Personalization, (Dec 2020 present)
- Staff Data Scientist (X5) – Technical Lead Manager of multiple projects, Walmart Labs, (Jan 2019 Dec 2020)
- Senior Statistical Analyst (X4), Walmart Labs, (May 2017 Dec 2018)
- Data Scientist Intern, Thomson Reuters Corporation, NLP Group, (April 2016 September 2016)
- NLP/Data Engineer, Pardazeshgaran Pardis Corporation, Tehran, Iran. (Jan 2007 – August 2010).

Accomplishments and Awards

- Semifinalist of Frantz Edelman Award, Informs 2019.

- Walmart Labs iD8 2018 (the Associate Incubation Bootcamp), Semifinalist
- Utah State University Merit-based Graduate Senate Enhancement Award, April 2015, given to the top students of the program.
- Best Poster Award, Semi-annual graduate reception, CS Department, USU, Spring 2016.
- Best Graduate Research Assistant Award, Semi-annual graduate reception, CS Department, USU, Fall 2015.

Publications

- K. Ahmadi and V. H. Allan, "Practical City Scale Stochastic Path Planning with Pre-Computation", in journal of Lecture Notes in Artificial Intelligence, Springer, October, 2021
- K. Ahmadi and V. H. Allan, "Smart City: Application of Multi-agent Reinforcement Learning Systems in adaptive Traffic Management", IEEE International Smart Cities Conference 2021, September, 2021
- Y. Chen, P. Mehrotra, K. Ahmadi, M. Shrivastav, and S. Pleiman, "A Multi-objective Optimization for Clearance in Walmart Brick-and-Mortar Stores", Informs Journal of Advanced Analytics, Feb 2021.
- K. Ahmadi and V. H. Allan, "Scalable Stochastic Path Planning for City Scale Graphs", 13th International Conference on Agents and Multi-Agents Systems, Feb 2021.
- K. Ahmadi and V. H. Allan, "Congestion-aware Stochastic Path Planning and its Applications in Real World Navigation" 13th International Conference on Agents and Multi-Agent Systems, Feb 2021.

- K. Ahmadi and V. H. Allan, “Trust-Based Decision Making in A Self-Adaptive Agent Organization”, in journal of ACM Transaction on Autonomous and Adaptive Systems (TAAS), April 2016

- K. Ahmadi, and V. H. Allan, “Checking the Reliability of Information Sources in Recommendation-Based Trust Decision Making”, in Principles and Practices of Multi-Agent Systems (PRIMA 2015), October 2015, Bertinoro, Italy.