Spatiotemporal Big Data Analytics for Future Mobility

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Reem Ali

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Adviser: Shashi Shekhar

May, 2019

Acknowledgements

I would like to express my deepest gratitude to my adviser, Professor Shashi Shekhar, for his guidance, support, understanding and wisdom. Not only did he teach me how to be a better researcher, but he also taught me how to be a better teacher. I have never walked into his office, class, or group meeting feeling confused or anxious without coming out feeling more confident, focused, and positive. I would like to extend my deepest appreciation to Professor William Northrop and his former student, Dr. Andrew Kotz, for the rich and fruitful collaboration and insights which helped shape my research and made it more meaningful. I would also like to thank my collaborators from Ford Motor Company, Dr. Shounak Athavale and Dr. Eric Marsman, for their insightful suggestions during our numerous conference call meetings. I am extremely grateful to my committee members, Professor Vipin Kumar, Professor Mohamed Mokbel, and Professor William Northrop, for contributing their time to serve on my committee and for providing me with their invaluable feedback on my research.

I thank all the members of the Spatial Computing research group for being extremely supportive colleagues. You have given me the sense of a real community. Many thanks to Dr. Viswanath Gunturi and Dr. Xun Zhou for their advice and mentoring during my first years. I would also like to thank Kim Koffolt for revising my paper drafts and for all the valuable editing advice.

Last but not least, I am eternally grateful to my parents. None of this would have been possible without their continuous support, encouragement, and belief in me. I am deeply indebted to my husband, Amr, for his support and patience throughout this journey, and my son, Yousof, for bearing with me through the hectic deadlines and for being such a little independent man. Thanks to my baby, Ali, for making this final year more enjoyable and special. Words cannot express my gratitude to Eman Ramadan, the friend who has contributed everything from technical consultations and therapy to meals and baby sitting. I also thank my friends, Sara Morsy, Heba El Dakiky and Sara Aktham for listening, for their advice, and for being my second family in Minnesota. To all my great teachers, thank you for your efforts which made every step easier for me. To all authors of great art, thank you for the inspiration and for lifting me up through hard times.

Dedication

To the most understanding parents, Kawther Haggag and Yousry Amer.
To Mohamed, my first mentor and only brother.
To my husband, Amr, who has put the time to be my cheerleader in the midst of working on his own dissertation.
To my sons, Yousof and Ali, may you always and forever keep learning.

Abstract

Recent years have witnessed the explosion of spatiotemporal big data (e.g. GPS trajectories , vehicle engine measurements, remote sensing imagery, and geotagged tweets) which has a potential to transform our societies. Terabytes of earth observation data are collected every day from thousands of places across the world. Modern vehicles are increasingly equipped with rich sensors that measure hundreds of engine variables (e.g., emissions, fuel consumption, speed, etc) annotated with timestamps and location data for every second of the vehicle's trip. According to reports by McKinsey and Cisco, leveraging such data is potentially worth hundreds of billions of dollars annually in fuel savings. Spatiotemporal big data are also enabling many modern technologies such as on-demand transportation (e.g. Uber, Lyft). Today, the on-demand economy attracts millions of consumers annually and over $50 billion in spending. Even more growth is expected with the emergence of self-driving cars. However, spatiotemporal big data are of volume, velocity, variety, and veracity that exceed the capability of common spatiotemporal data analytic techniques.

My thesis investigates spatiotemporal big data analytics that address the volume and velocity challenges of spatiotemporal big data in the context of novel applications in transportation and engine science, future mobility, and the on-demand economy. The thesis proposes scalable algorithms for mining "Non-compliant Window Co-occurrence Patterns", which allow the discovery of correlations in spatiotemporal big data with a large number of variables. Novel upper bounds were introduced for a statistical interest measure of association to efficiently prune uninteresting candidate patterns. Case studies with real world engine data demonstrated the ability of the proposed approaches to discover patterns which are of interest to engine scientists. To address the high velocity challenge, the thesis explored online optimization heuristics for matching supply and demand in an on-demand spatial service broker. The proposed algorithms maximize the matching size while also maintaining a balanced provider utilization to ensure robustness against variations in the supply-demand ratio and that providers do not drop out. Proposed algorithms were shown to outperform related work on multiple performance measures. In addition, the thesis proposed a scalable matching and scheduling algorithm for an on-demand pickup and delivery broker for moving consumers with multiple candidate delivery locations and time intervals. Extensive evaluation showed that the proposed approach yields significant computational savings without sacrificing the solution quality.

Contents

Acknowledgements **i**

Dedication **iii**

Abstract **iv**

List of Tables **x**

List of Figures **xii**

1 Introduction **1**

1.1 Spatiotemporal Big Data Analytics 1

1.2 Illustrative Application Domain: Transportation and Future Mobility . 2

1.3 Challenges 3

1.3.1 Volume 4

1.3.2 Velocity 4

1.3.3 Variety 4

1.3.4 Veracity 4

1.4 Thesis Contributions 5

2 Discovering Non-compliant Window Co-occurrence Patterns **8**

2.1 Introduction 8

2.2 Basic Concepts and Problem Statement 14

2.2.1 Basic Concepts 14

2.2.2 Interest Measure: Temporal cross-K function 15

2.2.3 Problem Statement 16
2.3 Preliminary Results 18
2.3.1 Naive Approach 19
2.3.2 Key Ideas Behind MTNMiner 19
2.3.3 MTNMiner: A Multi-Parent Tracking Approach for Mining NWC patterns 22
2.4 Proposed Approach 27
2.4.1 BDNMiner: A Bi-Directional approach for mining NWC patterns 28
2.4.2 Tightening the Local and Lattice Upper Bounds 35
2.5 Theoretical Evaluation 41
2.5.1 Correctness of BDNMiner 41
2.5.2 Completeness of BDNMiner 41
2.6 Case Study 43
2.7 Experimental Evaluation 47
2.7.1 Experimental Setup 48
2.7.2 Experimental Results 48
2.8 Conclusion and Future Work 53

3 ULAMA: A Utilization-Aware Matching Approach for Robust On-Demand Spatial Service Brokers 55
3.1 Introduction 55
3.1.1 Overview of On-Demand Spatial Service Brokers 56
3.1.2 Challenges 57
3.1.3 Contributions 58
3.1.4 Scope and Outline 59
3.2 Related Work 60
3.3 Basic Concepts and Problem Statement 62
3.3.1 Basic Concepts 63
3.3.2 Problem Definition 65
3.3.3 Problem Example 66
3.3.4 Complexity Analysis 70
3.3.5 Alternative Problem Formulations 72

3.4 Proposed Approach 72
3.4.1 Novel Supply and Supply-Demand Ratio Aware Heuristics 73
3.4.2 Consumer-Priority-Based Greedy Matching 76
3.4.3 Time Complexity of Proposed Approach 79
3.4.4 Offline Integer Programming Formulation for the OSSP Problem 81
3.5 Experimental Evaluation 83
3.5.1 Discrete-Event Simulation Framework 84
3.5.2 Experimental Design and Dataset 86
3.5.3 Experimental Results 88
3.5.4 Summary of Experimental Results 99
3.6 Conclusion 100
3.7 Future Work 101

4 pkgRendezvous: A Flexible On-Demand Pickup and Delivery Broker for Moving Consumers 103
4.1 Introduction 103
4.2 Basic Concepts and Problem Statement 107
4.2.1 Basic Concepts 107
4.2.2 Problem Definition 109
4.2.3 Problem Example 110
4.3 Proposed Approach 112
4.3.1 Baseline Method 112
4.3.2 pkgRendezvous: A computationally efficient approach for solving FLOPDMC 122
4.4 Experimental Evaluation 128
4.4.1 Discrete-Event Simulation Framework 129
4.4.2 Experimental Design and Dataset 130
4.4.3 Experimental Results 133
4.5 Conclusion and Future Work 141

5 Conclusion and Future Directions 142
5.1 Key Results 142
5.2 Short Term Future Directions 143

5.3 Long Term Future Directions 145

References **146**

Appendix A. Experimental Results for *pkgRendezvous* on Didi Chuxing Ride-Hailing Data **158**

A.1 Experimental Design 158

A.2 Experimental Results 159

List of Tables

2.1 Related work of the NWC pattern discovery problem 11

2.2 Table of Notations 17

2.3 Candidate NWC patterns for the input in Figure 2.4 18

2.4 MTNMiner Bottleneck Analysis 25

2.5 Interesting CO_2-related NWC patterns (smaller indexes indicate smaller values) 44

2.6 Interesting NO_x-related NWC patterns (smaller indexes indicate smaller values) 46

3.1 Outputs of different heuristics for problem in Figure 3.4 67

3.2 Output statistics of different heuristics for problem in Figure 3.4 68

3.3 Potential K-proposition-matches for 3 consumers with 3 service providers and K=2 70

3.4 Summary of ULAMA 73

3.5 Provider demand for the problem in Figure 3.5 75

3.6 Outputs of different heuristics for problem in Figure 3.5 76

3.7 Output statistics of different heuristics for problem in Figure 3.5 76

3.8 Table of Notations for the Offline Integer Programs 81

3.9 Synthetic dataset details for experimental goal 1. Total supply is fixed across all *sdr* values. 88

3.10 Synthetic dataset details for experimental goal 2. Total supply is fixed across all *sdr* values. 98

4.1 Consumer requests input for problem in Figure 4.2 112

4.2 Two possible solutions for the problem in Figure 4.2 and Table 4.1 . . . 112

5.1 Taxonomy of thesis contributions with future directions marked in red color. ST indicates short term future directions while LT indicates long term directions. 143

List of Figures

1.1 Spatiotemporal Big Data Analytics Framework 2
2.1 Non-compliant emissions of oxides of nitrogen along a bus route in Minneapolis, MN (best viewed in color) 9
2.2 Example candidate Non-compliant Window Co-occurrence patterns (best viewed in color) 9
2.3 Our new BDNMiner approach versus our preliminary MTNMiner approach 12
2.4 Input data with one non-compliant window (best in color) 15
2.5 Input with two non-compliant windows 25
2.6 MTNMiner Execution trace (Best viewed in color) 26
2.7 Input with two non-compliant windows (best in color) 33
2.8 BDNMiner Execution trace (Best viewed in color) 34
2.9 An example illustrating nodes used for computing the tightened upper bounds for node {a,b,c}) 37
2.10 Execution time with varying T 48
2.11 Execution time for BDNMiner vs. MTNMiner 50
2.11 Execution time for BDNMiner vs. MTNMiner 51
3.1 An on-demand spatial service broker 57
3.2 Classification of Related Work 61
3.3 An example of a consumer submitting a service request and being matched to a service provider 63
3.4 Example input for the OSSP problem 67
3.5 Second example input for the OSSP problem 75
3.6 Experimental Design 87
3.7 Effect of supply-demand ratio *sdr* (by fixing supply and varying demand). 89

3.7 Effect of supply-demand ratio *sdr* (by fixing supply and varying demand). 90
3.8 Effect of $t_{timeout}$ (best viewed in color) 94
3.9 Effect of K (best viewed in color) 94
3.10 Effect of max. travel time $d_{max,r}$ (best viewed in color) 95
3.11 Effect of max. waiting time $w_{max,r}$ (best viewed in color) 95
3.12 Effect of service rate s_p (best viewed in color) 96
3.13 Effect of moving window length l_W (best viewed in color) 96
3.14 Effect of supply-demand ratio *sdr* (by fixing supply and varying demand). The number of total consumer requests (i.e., demand) corresponding to each *sdr* value is shown in Table 3.10. 98
4.1 Classification of related work 105
4.2 Example Spatial Network (Best viewed in color) 111
4.3 Experimental Design 130
4.4 $|R|$ versus avg. query response time (best viewed in color) 133
4.5 $|R|$ versus percent. matched requests (best viewed in color) 133
4.6 $|R|$ versus total travel time (best viewed in color) 133
4.7 $|R|$ versus total waiting time (best viewed in color) 133
4.8 $|V|$ versus avg. query response time (best viewed in color) 135
4.9 $|V|$ versus percent. matched requests (best viewed in color) 135
4.10 $|V|$ versus total travel time (best viewed in color) 135
4.11 $|V|$ versus total waiting time (best viewed in color) 135
4.12 n_D versus avg. query response time (best viewed in color) 136
4.13 n_P versus avg. query response time (best viewed in color) 136
4.14 $(t_{end} - t_{start})$ versus avg. query response time (best viewed in color) . . 136
4.15 α versus percent. matched requests (best viewed in color) 136
4.16 α versus total travel time (best viewed in color) 137
4.17 α versus total waiting time (best viewed in color) 137
4.18 l_G versus avg. query response time (best viewed in color) 137
4.19 l_G versus total travel time (best viewed in color) 137
4.20 l_G versus total waiting time (best viewed in color) 138
4.21 t_{pickup} and $t_{rendezvous}$ vs. avg. query response time (best viewed in color) 138
4.22 sr_p versus percent. matched requests (best viewed in color) 138

4.23 Update interval of $CellVehicleDistTable$ vs. total travel time (best viewed in color) 140
4.24 Update interval of $CellVehicleDistTable$ vs. total waiting time (best viewed in color) 140
4.25 Update interval of $MaxPathTimeTable$ vs. total travel time (best viewed in color) 140
4.26 Update interval of $MaxPathTimeTable$ vs. total waiting time (best viewed in color) 140
A.1 $|R|$ versus avg. query response time (best viewed in color) 160
A.2 $|R|$ versus percent. matched requests (best viewed in color) 160
A.3 $|R|$ versus total travel time (best viewed in color) 160
A.4 $|R|$ versus total waiting time (best viewed in color) 160
A.5 $|V|$ versus avg. query response time (best viewed in color) 161
A.6 $|V|$ versus percent. matched requests (best viewed in color) 161
A.7 $|V|$ versus total travel time (best viewed in color) 161
A.8 $|V|$ versus total waiting time (best viewed in color) 161

Chapter 1

Introduction

1.1 Spatiotemporal Big Data Analytics

Recent years have witnessed the explosion of spatiotemporal big data such as GPS trajectories, vehicle engine measurements, temporally detailed roadmaps, remote sensing imagery, geo-social media, geotagged tweets, and historical on-demand transactions. Such datasets are of volume, velocity, and variety that exceed the capability of common spatial computing platforms and techniques. Spatiotemporal big data analytics refer to the process of analyzing spatiotemporal big data to uncover information such as hidden patterns and trends and allow informed predictions and decision making. Figure 1.1 illustrates the different components of the spatiotemporal data analytics framework. At the bottom of the stack, spatiotemporal big data infrastructure provide a reliable system for the management of the large data volumes and support spatial/spatiotemporal data types, operations and indexes that help improve the I/O cost of queries by retrieving only the relevant portions of the data files. Representative efforts in that area include SpatialHadoop [1], Spatiotemporal Hadoop [2], Hadoop-GIS [3], GeoSpark [4], and Simba [5]. Spatiotemporal data science [6, 7, 8] encompasses algorithms that extract useful information from spatiotemporal big data. Examples of these techniques include spatial/spatiotemporal outliers detection, algorithms for mining associations and teleconnections, change patterns , spatiotemporal hotspots, partitions and summarization, predictive models, and spatiotemporal optimization models. Spatiotemporal data-driven sciences, as shown on top of the stack, refer to the different research fields with applications that employ spatiotemporal big data analytics to solve spatiotemporal big data problems. For instance, climate scientists use predictive models with big remote sensing data to understand and predict climate change. Transportation scientists use summarization techniques with GPS trajectory data to understand travel behaviors of individuals for improving transportation network planning.

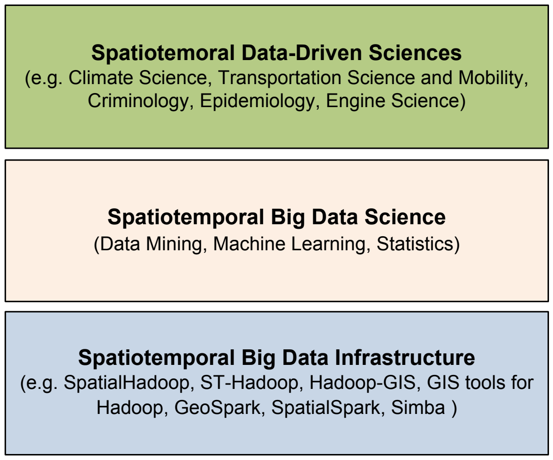

Spatiotemoral Data-Driven Sciences
(e.g. Climate Science, Transportation Science and Mobility, Criminology, Epidemiology, Engine Science)
Spatiotemporal Big Data Science
(Data Mining, Machine Learning, Statistics)
Spatiotemporal Big Data Infrastructure
(e.g. SpatialHadoop, ST-Hadoop, Hadoop-GIS, GIS tools for Hadoop, GeoSpark, SpatialSpark, Simba)

Figure 1.1: Spatiotemporal Big Data Analytics Framework

1.2 Illustrative Application Domain: Transportation and Future Mobility

Transportation accounts for over a third of U.S. green house gas (GHG) emissions and is one of the fastest growing sources of U.S. GHG since 1990, resulting in adverse effects on the earth's atmospheric temperature. In addition, transportation contributes to hundreds of thousands of premature deaths annually due to air pollution (e.g., oxides of nitrogen (NO_x)). It also accounts for a vast majority of U.S. petroleum consumption. Thus, reducing harmful vehicle emissions and improving vehicles fuel efficiency are among the most urgent and important challenges facing our society. Conventional methods in transportation science model and evaluate fuel-consumption and emissions in the lab under controlled conditions (e.g. using predefined standard drive cycles). However, as illustrated by the Volkswagen emissions scandal [9] as well as fines to other manufacturers [10, 11], these methods are not adequate to model emissions and fuel-consumption during real-world driving. A key challenge in this domain is to develop an understanding of the combustion and vehicle behavior under real-world driving conditions towards the design of technologies to reduce emissions and/or fuel-consumption.

Recent years have seen the emergence of modern connected vehicles with rich sensors that measure hundreds of engine variables (e.g., emissions, fuel consumption, speed, etc) annotated with frequent timestamps and location data (e.g., GPS coordinates). From these datasets we can also access the spatiotemporal context (e.g., weather, elevation, traffic, etc) associated with a vehicle's trip. According to reports by McKinsey and Cisco [12, 13], leveraging such big data is potentially worth hundreds of billions of dollars annually in fuel savings. Hence, these spatial big datasets can be leveraged for optimizing engine operation based on actual real-world driving conditions rather than the limited conditions tested in laboratories.

Apart from environmental friendliness and improved safety, connected vehicles can also be used to improve travelers' overall mobility experience by helping to reduce traffic congestion through better signal and traffic control, and allowing on-demand transportation services (e.g. Uber and Lyft). Today, the on-demand economy attracts millions of consumers annually and over $50 billion in spending. Even more growth is expected with the emergence of autonomous vehicle technologies. In the future, it is anticipated that fewer vehicles will be privately owned. Instead, people will use fleet vehicles owned by transportation services. Users can then request a vehicle when needed and the driver-less vehicle would show up. This shift will also lead to the emergence of novel on-demand services where moving consumers riding autonomous vehicles can request products and/or services on demand and the vehicle would take them to the nearest service provider. Opportunities raised by the emerging connected and autonomous vehicles create a growing need for novel spatiotemporal data analytics that can overcome the challenges raised by spatiotemporal big data.

1.3 Challenges

Spatiotemporal big data pose several challenges for classical spatiotemporal analytic techniques:

1.3.1 Volume

One of the main challenges of spatiotemporal big data is the huge data volume. For instance, GPS traces produce 10^{14} data items per year [14]. Engine measurement datasets record hundreds of engine variables for every second of the vehicle's trip. On-demand ride-hailing services collect hundreds of terabytes of data every day [15]. These vast amounts of data represent a major challenge for spatiotemporal data analytic techniques where unsatisfactory computational performance may result particularly for complex use cases involving identifying correlations with a huge number of candidate patterns, non-mononotic statistical interest measures, multi-objective optimizations and complex queries.

1.3.2 Velocity

Spatiotemporal big data are also arriving at a tremendous velocity. For instance, modern connected cars produce 5 GB/hour of vehicle sensor data [16]. On-demand ridehailing services receive billions of route planning queries daily [15]. This high velocity poses challenges for common spatiotemporal big data infrastructure to ingest and manage the incoming data and avoid frequent data loss. Scalable spatiotemporal data science techniques are also needed to allow the analysis of the streaming data at high velocity.

1.3.3 Variety

Variety refers to the need for fusing multiple data sources. Today, there exists many different sources of structured and unstructured spatiotemporal big data including MRI images, geotagged tweets, multivariate trajectories from moving vehicles, as well as exogenous spatiotemporal data sources such as traffic information, weather, and elevation maps which also need to be imposed on spatiotemporal graph models of the road network. The variety of data sources require a huge effort from users to preprocess the data for the use of spatiotemporal big data analytic techniques.

1.3.4 Veracity

Veracity refers to the quality, trustworthiness, and uncertainty issues that may be present in the incoming data. For instance, historical trajectories may include inaccurate

GPS coordinates during areas of lost satellite signals. Similarly, engine measurement data collected from moving vehicles may include added noise due to privacy concerns. Parts of individual trajectories may also be suppressed for privacy. Spatiotemporal big data analytic techniques need to account for such uncertainty to be able to extract useful patterns with high accuracy.

1.4 Thesis Contributions

This thesis focuses on addressing some of the above challenges by proposing novel pattern mining algorithms (i.e., non-compliant window co-occurrence pattern discovery) from spatiotemporal big data as well as scalable spatiotemporal optimization approaches for matching supply (i.e., fixed and moving service providers) and demand (i.e., moving consumers) in on-demand spatial service brokers (i.e., utilization-aware matching for robust on-demand spatial service brokers, flexible on-demand pickup and delivery broker for moving consumers). Below, each chapter is briefly introduced.

- **Chapter 2** discusses a novel pattern mining algorithm called "Non-compliant Window Co-occurrence Pattern Discovery" which addresses the volume challenge of spatiotemporal big data. Informally, given a set of trajectories annotated with measurements of physical variables, the problem of Non-compliant Window Co-occurrence (NWC) pattern discovery aims to determine temporal signatures in the explanatory variables which are highly associated with windows of undesirable behavior in a target variable. NWC discovery is important for societal applications such as eco-friendly transportation (e.g. identifying engine signatures leading to high greenhouse gas emissions) and industrial process control (e.g. understanding failure patterns). Challenges of designing a scalable algorithm for NWC discovery include the non-monotonicity of popular spatiotemporal statistical interest measures of association such as the cross-K function which renders the anti-monotone pruning based algorithms (e.g. Apriori) inapplicable for such interest measures. In this chapter, we first propose two upper bounds for the cross-K function and a top-down multi-parent tracking approach that uses these bounds for filtering out uninteresting candidate patterns and then applies a minimum support (i.e. frequency) threshold as a post-processing step to filter out chance patterns. We also propose a novel bi-directional pruning approach (BDNMiner) that combines top-down pruning based on the cross-K function threshold with bottom-up pruning based on the minimum support threshold to efficiently mine NWC patterns. Case studies with real world engine data demonstrates the ability of the proposed approach to discover patterns which are interesting to engine scientists. Experimental evaluation on real-world data show that the proposed approach yields substantial computational savings.

- **Chapter 3** addresses the data velocity challenge by proposing efficient online optimization heuristics for robustly matching supply and demand in on-demand spatial service brokers. More specifically, the chapter investigates an on-demand spatial service broker for suggesting service provider propositions and the corresponding time of service to mobile consumers while meeting the consumer's maximum travel time and wait time constraints. The goal of the broker is to maximize the number of matched requests while also keeping the "eco-system" functioning and robust against variations in the supply-demand ratio by engaging as many service providers as possible and maintaining providers' participation to ensure that providers do not drop out. This problem is important because of its many related societal applications in the on-demand and sharing economy (e.g., on-demand ride-hailing services, on-demand food delivery, etc). Challenges of this problem include the need to satisfy many conflicting requirements of the broker, consumers and service providers and the high computational complexity since solving the problem for a number of available consumers at any time instant is NP-hard. Related work in spatial crowdsourcing and ridesharing has mainly focused on maximizing the number of matched requests and minimizing travel cost, but did not consider the importance of maintaining provider engagement and balancing provider utilization, which could become a priority when the available supply exceeds the demand. In this chapter, we propose a Utilization-Aware Matching Approach (ULAMA) which employs novel provider-centric heuristics for balancing the utilization of providers, and a consumer-priority-based greedy matching algorithm that prioritizes consumers for maximizing the number of matched requests. Experimental results show that our proposed approach outperforms related work by achieving the lowest variance in provider utilization while matching all available providers even when supply greatly exceeds demand. Our approach also achieved a larger number of matched requests, particularly when supply exceeds demand and also when both supply and demand are balanced.

- **Chapter 4** addresses the challenge of the large data volume of on-demand service transactions by discussing a scalable approach for matching and scheduling delivery vehicles and mobile consumers in an on-demand pickup and delivery broker. In this context, we investigate a flexible on-demand pickup and delivery broker in which moving consumers provide their itinerary including multiple possible delivery locations and the corresponding time interval during which they will be available at each location. The goal of the broker is to deliver the consumer requests at a place and time where consumers are available, while maximizing the number of matched requests and minimizing the delivery vehicles travel and waiting times. The problem is important because of many everyday issues facing delivery services including package loss, package theft, user inconvenience, and revenue loss resulting from rescheduled deliveries when a package arrives at an empty place where no one is available to receive it. Challenges of this problem include the multiple conflicting broker objectives and the high computational complexity due to the exponential number of possible schedules, and the multiple possible pickup and delivery locations per request. Related work on the dynamic pickup and delivery problem with time windows and on-demand ridesharing has only focused on delivery to a single fixed delivery location or a single destination; it has not considered moving consumers with multiple possible delivery locations and time intervals. In this chapter, we propose *pkgRendezvous*, a matching and scheduling algorithm that relies on an early termination filter and an all-insertions cost lower-bound pruning filter with a grid-based lookup table for efficiently enumerating candidate schedules. Experimental results show that our proposed approach yields significant computational savings without sacrificing solution quality.

- **Chapter 5** concludes the key thesis findings and identifies open directions for future research.

Chapter 2

Discovering Non-compliant Window Co-occurrence Patterns

2.1 Introduction

Given a set of trajectories annotated with measurements of physical variables, the Non-compliant Window Co-occurrence (NWC) pattern discovery problem aims to determine temporal signatures in the explanatory variables which are highly associated with windows of undesirable behavior in a target variable (e.g. non-compliance with some standard). For instance, consider Figure 2.1, which shows portions of trajectories of a metro transit bus in Minneapolis-St. Paul, MN, USA. In these trajectories, each point is annotated with physical measurements such as engine power, engine revolutions per minute (RPM), wheel speed, elevation and engine emissions. The red color marks temporal windows within the trajectories where a target variable (emissions of oxides of nitrogen (NO_x) in this example) shows a non-compliant behavior (i.e., the average emissions within the windows exceed US EPA regulations [17]). As shown in the figure, some journeys show this non-compliant behavior, while others do not. NWC discovery aims to determine the underlying temporal signatures of the measured physical variables which are highly associated with those windows of elevated NO_x emissions. These signatures (aka "patterns") represent sequences defined on one or more physical variables that either coincide with or occur within a prespecified time lag from a non-compliant window. Figure 2.2 shows a non-compliant window <2,6>, of length 5 sec, in which the NO_x

emissions exceed the US EPA standard of 0.267 gm/kW-h. As shown in the figure, patterns of high acceleration and increase in elevation co-occur with this non-compliance behavior and thus would be considered as candidate patterns by the NWC discovery problem.

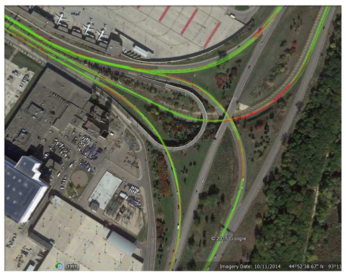


Figure 2.1: Non-compliant emissions of oxides of nitrogen along a bus route in Minneapolis, MN (best viewed in color)

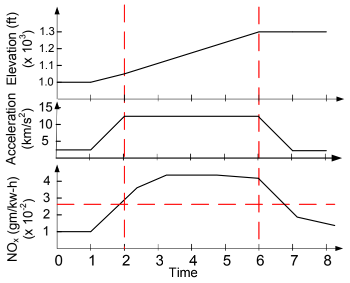


Figure 2.2: Example candidate Non-compliant Window Co-occurrence patterns (best viewed in color)

Illustrative Application Domain: Discovering NWC patterns is important to several scientific and societal applications such as eco-friendly transportation (e.g. discovering engine behaviors leading to high greenhouse gas emissions), detection of engine signatures associated with engine malfunctions (e.g. patterns of sudden unintended acceleration [18]) which can help save people's lives, and industrial process control (e.g. understanding patterns of failure in an industrial process [19]). In this work we use eco-friendly transportation as our illustrative application domain. Current efforts in the field of engine research are aimed at reducing harmful vehicle emissions such as NO_x and carbon dioxide (CO_2) due to their adverse effects on human health and the environment [20, 21]. Despite recent advances in emissions reduction technologies and stricter standards imposed by regulatory agencies, vehicles are emitting at rates higher than their certified limit under real world driving conditions [22, 23]. However, these discrepancies are not a result of vehicles failing certification. In some instances vehicles have been found using emissions cheating devices [24], but the majority of the discrepancies are a result of a certification test not accurately reflecting real-world vehicle use. Therefore, identifying engine variable signatures co-occurring with elevated NO_x and CO_2 emissions in the real-world is key to understanding the cause of the excess emissions. Furthermore, the ability of the NWC discovery method to discover less frequent patterns from large datasets would be well suited for identifying vehicles which contain emissions defeat devices by highlighting instances where the cheating emissions reduction systems were turned on. Additionally, the time lag in co-occurrence patterns must be considered when analyzing vehicle systems due to the large timescale of some vehicle interactions. For instance, as engine load increases, engine temperature (a key factor in NO_x production) may rise at a slower rate due to the heat capacity and inertia of the engine and coolant and hence the non-compliant NO_x emissions might not occur until a few seconds after the engine load started to increase. Due to the extremely short timescale of combustion and the high frequency of its occurrence, most engine variables vary on a very small timescale except for temperature variables which could take a few seconds to a couple minutes depending on the temperature gradient. Therefore, when studying small fixed non-compliant windows (i.e. a 5 sec window length), it is safe to assume that larger scale temperature changes had already occurred, while other explanatory variables and smaller scale temperature changes exhibit change on a small time scale with no lag between them. In addition, engine measurement datasets are usually collected at an aggregate of a higher frequency (i.e. the actual observations usually happen on the 10-50 Hz timescale while the data is being presented on the 1 Hz timescale). Thus, changes in engine variables will usually happen simultaneously (except for bulk temperature changes which may have a longer time scale) which means that interesting changes in explanatory variables usually occur within and along the non-compliant window and the specified lag value.

To measure the strength of an association between a pattern and non-compliant windows, a spatio-temporal statistical measure is preferred in order to provide a statistical interpretation of the output patterns. The cross-K function [25, 26] is a popular spatio-temporal statistical measure which is often used to measure the interaction between pairs of events in space and time. The cross-K function can express how much the association between a given pattern and non-compliant windows deviates from the assumption of their independence. Additionally, the cross-K function is not sensitive to the prevalence of the output pattern and hence can capture signatures that are less frequent but are highly associated with non-compliant windows. In addition, a low minimum support (i.e. frequency) threshold can still be used to filter out chance or spurious patterns (e.g., patterns which might have occurred only once in the input data and that occurrence happened to be near a non-compliant window, leading to a misleading high value for its cross-K function). Therefore, using both the cross-K function threshold and a low minimum support threshold allows capturing non-spurious engine signatures that are highly associated with non-compliant windows even when they are not prevalent in the input dataset.

Challenges: Designing an algorithm for NWC discovery that captures statistically meaningful patterns while maintaining computational scalability is challenging for the following reasons: First, domain-preferred spatio-temporal statistical association measures (e.g., cross-K function) lack monotonicity: a pattern representing an engine signature over multiple variables may be interesting even though its component single-variable signatures are not. For instance, the increase of both engine RPM and brake torque might be more strongly associated with increased NO_x than the increase of engine RPM alone. This property renders Apriori-based pruning inapplicable for such interest measure. Second, there are a huge number of candidate patterns to consider. For each non-compliant window, the number of associated candidate patterns is exponential in the number of variables. This includes all combinations of one, two, three, etc., variables. Third, the data volume is potentially huge due to the large number of variables over a long time series.

Table 2.1: Related work of the NWC pattern discovery problem

Temporal Co-occurrence/Association Patterns		Interest Measure	
		Frequent (i.e. High Support) Patterns	Low Support High Confidence/ Correlation Patterns
Rule Antecedent	A Set or Subsequence of Event Types	[27, 28]	[29, 30, 31, 32]
	(Multivariate) Contiguous Sequence	[33, 34, 35, 36]	Our proposed work

Limitations of Related Work: Related work for the NWC discovery problem mainly consists of literature on mining multi-dimensional temporal association rules [33, 34, 35, 36]. In these rules, a consequent occurs within T time points of an antecedent (a single or multi-dimensional sequence). However, these works, similar to frequent pattern mining [27, 28] have mainly focused on finding the most frequent patterns using a minimum support threshold and Apriori-like pruning methods that rely on the anti-monotone property of the support measure. By contrast, in the NWC discovery problem, rare associations can still be interesting since they can reveal patterns that are highly associated with non-compliant windows but have low support. The statistical interest measure used to capture these patterns does not have this anti-monotone property and hence Apriori-based pruning cannot be applied for this interest measure.

Some other studies in the literature have also addressed the problem of mining rare (i.e. low support) co-occurrence/association patterns with a high confidence threshold [29, 30, 31, 32]. However, these methods only focus on associations between single events and do not model associations between contiguous sequences (e.g., a temporal signature of engine variables co-occurring with non-compliant windows). For instance, they would not be able to capture the association of a continuous acceleration or braking pattern with a window of elevated emissions as shown in our case study. In addition, none of these methods except [32] guarantees completeness. Table 2.1 shows a classification of the related work.

In our preliminary work [37], we proposed two upper bounds for the cross-K function which are cheaper to compute than the computation of the exact cross-K function. We also proposed a top-down Multi-parent Tracking approach for mining NWC patterns (MTNMiner) that uses the proposed upper bounds for pruning uninteresting NWC patterns. MTNMiner was experimentally validated and a case study was provided for showing its ability to find meaningful patterns. This work extends our previous work by proposing a novel bi-directional algorithm (BDNMiner) that improves the scalability of our preliminary MTNMiner algorithm by combining cross-K function pruning

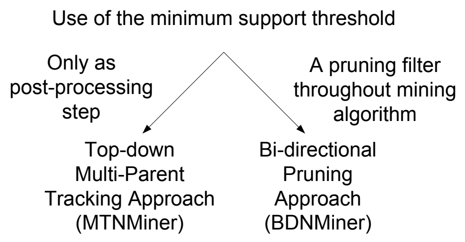

Use of the minimum support threshold
Only as post-processing step
A pruning filter throughout mining algorithm
Top-down Multi-Parent Tracking Approach (MTNMiner)
Bi-directional Pruning Approach (BDNMiner)

Figure 2.3: Our new BDNMiner approach versus our preliminary MTNMiner approach

with minimum support pruning simultaneously. Figure 2.3 summarizes the differences between the two algorithms.

Contributions: This work makes the following new contributions: (1) We propose a bi-directional approach for mining NWC patterns (BDNMiner) using both cross-K function and minimum support pruning simultaneously. (Section 2.4.1) (2) We also propose a method for tightening the cross-K function upper bounds in our BDNMiner for efficiently pruning uninteresting patterns. (Section 2.4.2) (3) We analytically prove the correctness and completeness of BDNMiner. (4) We present a new case study to evaluate the effectiveness of BDNMiner in finding statistically meaningful engine patterns that are associated with non-compliant CO_2 emissions in transit buses. (5) We provide an experimental evaluation using real-world data and show that BDNMiner yields substantial computational savings compared to our preliminary work [37].

Scope and Outline: The process of mining association rules from time series data in continuous domains typically consists of first discretizing the time series values and then discovering the interesting associations [33, 34, 35]. In this work, we focus on the problem of discovering interesting co-occurrence patterns. We do not address the problem of choosing the most suitable discretization technique. Instead, we assume that the discretization intervals for each variable are given as an input to this problem, possibly using representations suggested in [36, 38, 39, 40]. Additionally, in this work we only focus on temporal co-occurrence patterns. Spatial aspects of non-compliant window co-occurrence patterns will be explored more thoroughly in future work. We also only considered using a fixed lag value for all explanatory variables and assumed that all patterns have the same length as the non-compliant windows.

The rest of the chapter is organized as follows: Section 2.2 presents the basic concepts followed by a formal problem definition for the NWC pattern discovery problem. Section 2.3 reviews our preliminary approach towards addressing this problem [37]. The proposed bi-directional BDNMiner algorithm is presented in Section 2.4. Section 2.5 presents a theoretical evaluation of the correctness and completeness of BDNMiner. Section 2.6 presents case studies for using the BDNMiner algorithm on real-world engine datasets collected from transit buses. The experimental evaluation is covered in Section 2.7. Finally, Section 2.8 concludes the chapter and discusses future work.

2.2 Basic Concepts and Problem Statement

2.2.1 Basic Concepts

Definition 1. ***An event***: *Given a variable v, an event $e_i(v)$ is a reading where v falls within a predefined range [v_i,v_{i+1}).*

For example, a set of events $E(v) = \{e_1(v),\ e_2(v),\ ..,\ e_m(v)\}$ can be defined for the wheel speed variable v where $e_1(v)$ indicates that wheel speed $\in$ [0, 5) km/h, $e_2(v)$ indicates that wheel speed $\in$ [5, 10) km/h, and so on.

Definition 2. ***A multi-variate event trajectory (MET)***: *Given a set of explanatory variables V and a target variable y, a MET is a sequence of multivariate points $p_t = (p_t^1, p_t^2, ..., p_t^{|V|}, y_t)$, $1 \leq t \leq \tau$, where t is a timestamp of p_t, τ is the trajectory length, p_t^k is an event defined for variable $v_k \in V, 1 \leq k \leq |V|$, and $y_t \in \mathbb{R}$.*

Figure 2.4 shows an example of a MET of length τ=8, defined over two explanatory variables V=$\{v_1$:engine power, v_2:engine RPM$\}$, where $E(v_1)$=$\{a_1,\ a_2,\ a_3\}$ and $E(v_2)$=$\{b_1,\ b_2,\ b_3\}$ are their corresponding sets of events, and a target variable y of NO_x emissions.

Definition 3. ***An event-sequence S(v)***: *Given a variable v, an event-sequence $S(v)$ is a sequence of events $e_i(v)$ that are temporally contiguous in a MET.*

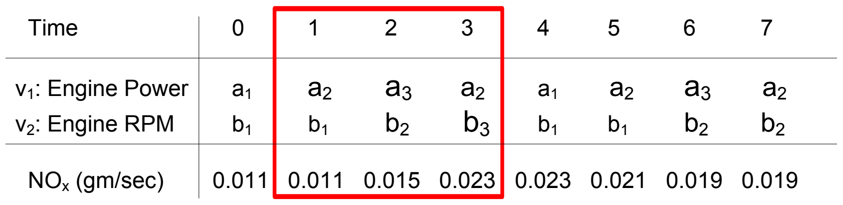
For example, in Figure 2.4, $a_2a_3a_2$ is an event sequence for engine power.
Next, before we present the definition of a non-compliant window, we first need to distinguish between two types of functions, namely, local functions and zonal functions. Local functions are functions that determine the output at each time instant based on the attribute value at this time instant. For example, computing the engine load at a given time instant as a function of the engine power at that time instant is a local function. Zonal functions employ aggregate operators over time instants in a longer interval. For example, determining the average NO_x emissions within a 10 sec window is a zonal function.

Definition 4. ***A non-compliant window (W_N)***: *Given a MET m, a zonal function F defined over the target variable y of m and a window length L, and a threshold h, a non-compliant window W_N=<t_i,t_j>is a time interval in m, of length L, where $F(y)$*

>h. The length L is defined as the number of time instants within the window, i.e. L = t_j - t_i+ 1.

For example, given the MET in Figure 2.4, and a function F defined as the percentage of increase in NO$_x$ between the start and end of a 3 sec window, and a threshold of 100%, the window <1,3>is a non-compliant window since $\frac{0.023-0.011}{0.011}$ = 109%>100%.

Definition 5. ***A Non-compliant Window Co-occurrence (NWC) pattern****: Given a MET m defined over a set of explanatory variables V and a target variable y, and a time lag δ, an NWC pattern C is a set of equal-length event-sequences $\{S_i(u_i) \mid u_i \in U$, $U \subseteq V$ and $1 \leq i \leq |U|\}$, that started at the same time point, and within a time lag δ preceding the start of a non-compliant window in m. Length(C) denotes the length of the event-sequences in C and is equal to the non-compliant window length L. Dim(C) denotes the dimensionality of pattern C (i.e. number of variables in C), where Dim(C)= $|U|$.*



Time	0	1	2	3	4	5	6	7
v_1: Engine Power	a_1	a_2	a_3	a_2	a_1	a_2	a_3	a_2
v_2: Engine RPM	b_1	b_1	b_2	b_3	b_1	b_1	b_2	b_2
NO$_x$ (gm/sec)	0.011	0.011	0.015	0.023	0.023	0.021	0.019	0.019

Figure 2.4: Input data with one non-compliant window (best in color)

For example, in Figure 2.4, given a time lag δ=1 sec, we can identify 6 NWC patterns as listed in Table 2.3. The first three patterns coincide with the non-compliant window, while the last three patterns precede the window by a lag of 1 sec. Patterns with IDs =1, 2, 4 and 5 have a dimensionality of 1 since they are defined on only one variable, while patterns 3 and 6 have a dimensionality of 2.

2.2.2 Interest Measure: Temporal cross-K function

In this work, the temporal cross-K function, a purely temporal form of the space-time cross-K function [25, 26], is used as a statistical measure to express how much the association between a given pattern and non-compliant windows deviates from independence. A temporal cross-K function measuring the association between an NWC pattern, C, and the occurrence of non-compliant windows, W_N, at a time lag δ is calculated as follows: $K_{C,W_N}(\delta) = \lambda^{-1}_{W_N}$ *E[number of non-compliant windows starting within time δ from the start of an instance of C]*, where λ_{W_N} is the expected number of non-compliant window events per unit time. Under the assumption of independence between the occurrences of pattern C and the non-compliant windows, $K_{C,W_N}(\delta)$ is equal to $(\delta + 1)$. Whenever $K_{C,W_N}(\delta)$ is greater than $\delta + 1$, this indicates an association between the pattern and the non-compliant behavior, with higher values indicating a stronger association. According to [41], $K_{C,W_N}(\delta)$ can be estimated by:

$$\begin{aligned}\hat{K}_{C,W_N}(\delta) &= \lambda_{W_N}^{-1} \sum_i \sum_j \frac{I(0 \leq d(C_i, W_{Nj}) \leq \delta)}{|C|} \\ &= \frac{T}{|W_N||C|} \sum_i \sum_j I(0 \leq d(C_i, W_{Nj}) \leq \delta) \end{aligned} \tag{2.1}$$

where $d(C_i, W_{Nj})$ is the distance between the start of instance C_i of pattern C and the start of the non-compliant window W_{Nj}; I(.) is an indicator function that assumes a value of 1 if $0 \leq d(C_i, W_{Nj}) \leq \delta$, and a value of 0 otherwise; T= $\sum\limits_{allMETs} \tau$ (and is referred to as the time series length in this chapter), and $|W_N|$ and $|C|$ are, respectively, the number of non-compliant windows and the number of instances (i.e., cardinality) of pattern C across all METs (i.e., the time series). Hence, $\hat{K}_{C,W_N}(\delta)$ can be written as:

$$\hat{K}_{C,W_N}(\delta) = \frac{T \times |C \overset{\delta}{\bowtie} W_N|}{|W_N||C|} \tag{2.2}$$

where $|C \overset{\delta}{\bowtie} W_N|$ denotes the cardinality of the temporal join set between the instances of pattern C and non-compliant windows, W_N, such that an instance C_i and W_{Nj} are only joined if C_i preceded W_{Nj} by time t where $0 \leq t \leq \delta$. For simplicity, in the rest of this chapter, we will refer to $|C \overset{\delta}{\bowtie} W_N|$ as the cardinality of the join set of C.

2.2.3 Problem Statement

The problem of discovering Non-compliant Window Co-occurrence (NWC) patterns can be formally expressed as follows:

Given:

1. A set M of multivariate event trajectories (METs)
2. A set of non-compliant windows W_N (i.e. time intervals on M), where each window is of length L
3. A time lag δ,
4. A temporal cross-K function threshold ϵ, and
5. A minimum support threshold *minsupp*

Find: All NWC patterns C where $\hat{K}_{C,W_N}(\delta) > \epsilon$.

Objective: Reduce computational cost.

Constraints:

1. All output patterns should have a support at least equal to *minsupp*.
2. All METs in M are sampled at equal intervals with the same sampling rate.
3. Correctness and completeness are guaranteed.

The *minsupp* threshold is mainly used to discard chance/spurious patterns and reduce the number of patterns output to the user, however, it should generally be set to a very low value to allow the discovery of both rare as well as frequent patterns. Table 2.2 summarizes the notations used in the above problem statement.

Table 2.2: Table of Notations

Symbol	Description
M	set of multivariate event trajectories (METs)
W_N	set of non-compliant windows
L	non-compliant window length
δ	time lag
ϵ	temporal cross-K function threshold
minsupp	minimum support threshold
C	an NWC pattern
$\hat{K}_{C,W_N}(\delta)$	cross-K function of pattern C at time lag δ

Assumptions: In our above problem formulation, we have made the following simplifying assumptions:

1. All explanatory variables are assumed to have the same lag value δ.
2. The length of a non-compliant window is exactly the same as the candidate patterns length.

Example: Figure 2.4 shows an example for the input to the NWC discovery problem. The input consists of a MET defined on two explanatory variables: v_1:engine power and v_2:engine RPM, and a target variable: NO$_x$ emissions in gm/sec. E(v_1) = $\{a_1,a_2,a_3\}$ and E(v_2) = $\{b_1, b_2, b_3\}$ are the set of events defined for the engine power and engine RPM variables, respectively. One non-compliant window <1,3>of length L = 3 sec is input and is marked by a red rectangle as shown in the figure. Hence, $|W_N| = 1$. The $\hat{K}_{C,W_N}(\delta)$ threshold ϵ is set to 5, δ is set to 1 sec, and *minsupp* is set to 1/8 (for illustration purposes only). The aim is to find all NWC patterns meeting the $\hat{K}_{C,W_N}(\delta)$ and *minsupp* thresholds. Table 2.3 shows all the candidate NWC patterns. The first 3 patterns are those coinciding with the non-compliant window <1,3>, while the next 3 patterns are the patterns preceding the window by a lag of 1 sec. Columns 3, 4, 5 and 6 show the number of occurrences of each pattern in the time series (i.e. pattern cardinality), the cardinality of the join set between instances of this pattern and non-compliant windows, the pattern support, and the value of the interest measure, respectively.

Table 2.3: Candidate NWC patterns for the input in Figure 2.4

ID	Candidate Pattern C	| C |	$\lvert C \overset{\delta}{\bowtie} W_N \rvert$	support(C)	$\hat{K}_{C,W_N}(1)$	Is output?
1	$\{a_2a_3a_2\}$	2	1	1/4	4	NO (4 <5)
2	$\{b_1b_2b_3\}$	1	1	1/8	8	YES (8 >5)
3	$\{a_2a_3a_2,\ b_1b_2b_3\}$	1	1	1/8	8	YES (8 >5)
4	$\{a_1a_2a_3\}$	2	1	1/4	4	NO (4 <5)
5	$\{b_1b_1b_2\}$	2	1	1/4	4	NO (4 <5)
6	$\{a_1a_2a_3,\ b_1b_1b_2\}$	2	1	1/4	4	NO (4 <5)

For example, the first pattern in the table occurred twice at time instants 1 and 5 (i.e. $|C|$=2). That is, it has a support = $\frac{|C|}{T}$ = 2/8 = 1/4. However, only one of those occurrences was associated with a non-compliant window: the pattern instance at t=1 coincided with the non-compliant window <1,3>(i.e. $|C \overset{1}{\bowtie} W_N|$=1). Hence, for this pattern, $\hat{K}_{C,W_N}(\delta) = \hat{K}_{C,W_N}(1) = \frac{T \times |C \overset{1}{\bowtie} W_N|}{|W_N||C|} = \frac{8\times1}{1\times2} = 4<5$. However, the second pattern occurred only once at t=1, where it coincided with a non-compliant window. Hence for this pattern, $\hat{K}_{C,W_N}(1) = \frac{T \times |C \overset{1}{\bowtie} W_N|}{|W_N||C|} = \frac{8\times1}{1\times1}$ = 8>5 and support(C) = 1/8 $\geq$ *minsupp*. As shown in Table 2.3, only patterns 2 and 3 have an interest measure (cross-K function) exceeding ϵ and support at least equal to *minsupp*, and thus these are the final output patterns as indicated in column 7.

2.3 Preliminary Results

In this section, we first present the naive approach for solving the NWC discovery problem [37]. Then, we review our preliminary **M**ulti-Parent **T**racking approach for mining **N**WC patterns (MTNMiner) with its cross-K function upper bound pruning filters.

2.3.1 Naive Approach

The naive approach starts by finding all non-compliant windows in the given time series (i.e. the collection of input METs) using a sliding window of the same length as the given non-compliant window length. Then, for each non-compliant window in a MET m, we enumerate all temporal windows in the MET that started within time t preceding this non-compliant window, where $0 \leq t \leq \delta$. Finally, for each of these temporal windows we enumerate all the candidate NWC patterns. Each pattern C is enumerated by calculating its cardinality $|C|$ using a single linear scan of the time series. Whenever an instance of the pattern is found, the algorithm examines the non-compliant windows table to count the number of windows that are within δ sec from this pattern. Hence, for a pattern C, both $|C|$ and $|C \overset{\delta}{\bowtie} W_N|$ are calculated using a single linear scan. Finally, if the pattern satisfies the *minsupp* threshold, its cross-K function is calculated and the pattern is output if the measure exceeds the user-specified threshold ϵ.

Note that while enumerating the non-compliant windows and their corresponding candidate patterns, no non-compliant window or pattern is allowed to overlap two different METs. In addition, if the input METs belong to different moving objects (e.g. different vehicles), NWC patterns in a MET of one object should not be associated with a non-compliant window in a MET of another object. To achieve this, the non-compliant window table also stores the object ID to differentiate between the non-compliant windows of the different objects.

2.3.2 Key Ideas Behind MTNMiner

In this subsection, we review the three key ideas behind the MTNMiner algorithm before presenting the algorithm in the following subsection.

Key Idea 1: Local Upper Bound of $\hat{K}_{C,W_N}(t)$:

First, we define a subset/superset relation between NWC patterns.

Definition 6. ***Subset and superset patterns****: Given two NWC patterns, C=$\{S_i(u_i) : u_i \in U, 1 \leq i \leq |U|\}$ and C'=$\{S_i(q_i) : q_i \in Q, 1 \leq i \leq |Q|\}$, then, C' is said to be a* ***subset*** *of C iff: (1)Length(C') = Length(C). (2) $Q \subseteq U$. (3) For every $S_i(q_i) \in$ C', we have $S_i(q_i) \in C$. Similarly, C is said to be a* ***superset*** *of C' (i.e. superset(C')).*

Definition 7. ***Local upper bound****: Given an NWC pattern C=$\{S_i(u_i) \mid u_i \in U, U \subseteq V$ and $1 \leq i \leq |U|\}$ and a time lag δ, the* ***local upper bound*** *of $\hat{K}_{C,W_N}(\delta)$, denoted as $UB_{local}(\hat{K}_{C,W_N}(\delta))$, can be computed as follows:*

$$UB_{local}(\hat{K}_{C,W_N}(\delta)) = \frac{T}{|W_N|} \times \frac{Upper_{Loc}(|C \overset{\delta}{\bowtie} W_N|)}{Lower(|C|)}$$

$$\textit{where: } Upper_{Loc}(|C \overset{\delta}{\bowtie} W_N|) = \min_{\{S_i\} \in C, 1 \le i \le Dim(C)} (|\{S_i\} \overset{\delta}{\bowtie} W_N|)$$
$$\textit{and } Lower(|C|) = |\textit{superset(C)}| \tag{2.3}$$

Note that $Upper_{Loc}(|C \overset{\delta}{\bowtie} W_N|)$ is an upper bound of $|C \overset{\delta}{\bowtie} W_N|$ which exists in the numerator of $\hat{K}_{C,W_N}(\delta)$. It is computed using the minimum join set cardinality of all subset patterns of C that consist of only one event-sequence (i.e. one-variable subset patterns). $Lower(|C|)$ is a lower bound of $|C|$, which is in the denominator of $\hat{K}_{C,W_N}(\delta)$, and is equal to the cardinality of any superset pattern of C. Next, we prove that $UB_{local}(\hat{K}_{C,W_N}(\delta))$ is an upper bound of $\hat{K}_{C,W_N}(\delta)$.

Lemma 1. *Given an NWC pattern C and a time lag δ, $Upper_{Loc}(|C \overset{\delta}{\bowtie} W_N|)$ is an upper bound of $|C \overset{\delta}{\bowtie} W_N|$*

Proof. *For every NWC pattern $\{S_i\}$ consisting of a single event-sequence where $\{S_i\} \subseteq C$, $1 \le i \le Dim(C)$, we have $|\{S_i\}| \ge |C|$, where $|\{S_i\}|$ and $|C|$ are the cardinality of the patterns $\{S_i\}$ and C in the time series, respectively. Since $\{S_i\} \subseteq C$, we also have $|\{S_i\} \overset{\delta}{\bowtie} W_N| \ge |C \overset{\delta}{\bowtie} W_N|$, $\forall\ 1 \le i \le Dim(C)$. Then, $Upper_{Loc}(|C \overset{\delta}{\bowtie} W_N|) = \min_{\{S_i\} \in C, 1 \le i \le Dim(C)} (|\{S_i\} \overset{\delta}{\bowtie} W_N|) \ge |C \overset{\delta}{\bowtie} W_N|$.* ■

Lemma 2. *Given an NWC pattern C and a time lag δ, $Lower(|C|)$ is a lower bound of $|C|$.*

Proof. *Any superset pattern of C has a cardinality smaller than or equal to C. Therefore, Lower($|C|$) $= |\textit{superset(C)}| \le |C|$.* ■

Theorem 1. *Given an NWC pattern C and a time lag δ, $UB_{local}(\hat{K}_{C,W_N}(\delta))$ is an upper bound of $\hat{K}_{C,W_N}(\delta)$.*

Proof. *Using Lemmas 1 and 2, we have $\hat{K}_{C,W_N}(\delta) = \frac{T}{|W_N|} \times \frac{|C \overset{\delta}{\bowtie} W_N|}{|C|} \le \frac{T}{|W_N|} \times \frac{Upper_{Loc}(|C \overset{\delta}{\bowtie} W_N|)}{Lower|C|} = UB_{local}(\hat{K}_{C,W_N}(\delta))$* ■

Now, since $UB_{local}(\hat{K}_{C,W_N}(\delta))$ is an upper bound of $\widehat{K_{C,W_N}}(\delta)$, then if this upper bound is less than the cross-K function threshold ϵ, pattern C will not be output and hence there is no need to compute the actual cardinality of the pattern or the cardinality of its join set.

Key Idea 2: Lattice Upper Bound of $\hat{K}_{C,W_N}(t)$:

We use a second upper bound for the $\widehat{K_{C,W_N}}(\delta)$ of a pattern C, denoted as $UB_{lattice}(\hat{K}_{C,W_N}(\delta))$. Although this bound is less tight than the local upper bound of a pattern, $UB_{lattice}(\hat{K}_{C,W_N}(\delta))$ has a conditional monotone property. Based on that property, if $UB_{lattice}(\hat{K}_{C,W_N}(\delta))$ is less than ϵ, then the lattice upper bound for all subset patterns of C is also less than ϵ and so they can be completely pruned without calculating their upper bounds.

Definition 8. ***Lattice upper bound****: Given an NWC pattern C=$\{S_i(u_i) \mid u_i \in U, U \subseteq V$ and $1 \leq i \leq |U|\}$ and a time lag δ, the* ***lattice upper bound*** *of $\hat{K}_{C,W_N}(lag)$, denoted as $UB_{lattice}(\hat{K}_{C,W_N}(\delta))$, can be computed as follows:*

$$UB_{lattice}(\hat{K}_{C,W_N}(\delta)) = \frac{T}{|W_N|} \times \frac{Upper_{Lat}(|C \overset{\delta}{\bowtie} W_N|)}{Lower(|C|)}$$
$$\textit{where: } Upper_{Lat}(|C \overset{\delta}{\bowtie} W_N|) = \max_{\{S_i\}\in C, 1\leq i\leq Dim(C)}(|\{S_i\} \overset{\delta}{\bowtie} W_N|)$$
$$\textit{and } Lower(|C|) = |superset(C)| \tag{2.4}$$

Theorem 2. *Given an NWC pattern C and a time lag δ, $UB_{lattice}(\hat{K}_{C,W_N}(\delta))$ is an upper bound of $\hat{K}_{C,W_N}(\delta)$.*

Conditional Monotone Property for the Lattice Upper Bound:

Next we present Lemma 3 which describes a conditional monotone property for the lattice upper bound. Proofs of Theorem 2 and Lemma 3 can also be found in [37].

Lemma 3. *Given an NWC pattern C and a time lag δ, $UB_{lattice}(\hat{K}_{C,W_N}(\delta))$ is monotonically decreasing with decreasing Dim(C) if $Lower(|C|)$ is kept monotonically increasing. In other words, given two NWC patterns C and C' where C' $\subset$ C, then if $Lower(|C'|) \geq Lower(|C|)$, then $UB_{lattice}(\hat{K}_{C',W_N}(\delta)) \leq UB_{lattice}(\hat{K}_{C,W_N}(\delta))$.*

Key Idea 3: Efficient Calculation of the Pattern Cardinality:

A more efficient method to calculate the pattern cardinality is to preprocess the time series to create a *startingEdge* index. This index is a hash table where the key is two events that occurred consecutively in time i.e., $s_1 \rightarrow s_2$. The value is a list of all the time instants where this edge appeared in the input time series. A separate index is kept for each of the input variables. To calculate the cardinality of a pattern, we use the first two consecutive events (i.e., the first edge in the pattern) as the key, and retrieve the corresponding time instants where this edge occurred from the hash table. Then, we only search the time series at these time instants to count the cardinality of the pattern.

2.3.3 MTNMiner: A Multi-Parent Tracking Approach for Mining NWC patterns

The MTNMiner algorithm starts by finding and then iterating through all non-compliant windows. For each window, it enumerates patterns starting at t time points preceding that window, where $0 \leq t \leq \delta$. In addition, MTNMiner uses the key ideas introduced in the previous subsection to efficiently traverse the candidate patterns enumeration space. For each value of t preceding a non-compliant window $<t_i, t_j>$, a lattice data structure is used to represent all the patterns starting at t, as shown in Figure 2.6a. The lattice nodes represent all the possible patterns within the window $<t_i - t, t_j - t>$. Each node is labeled with the list of variables in the pattern it belongs to. For example, consider the input MET shown in Figure 2.5. For the window <0,2>, the lattice node labelled $\{a, b\}$ represents the pattern defined by the first two variables in that window, namely $\{a_1a_2a_3, b_1b_1b_2\}$.

Within the lattice of each window, MTNMiner starts by enumerating all leaf nodes representing one-variable patterns, and stores the join-set cardinality of these nodes in an array $LeafJoinSetCount$. Then, a top-down breadth first traversal/search (BFS) is performed to enumerate the rest of the lattice nodes while applying the previously defined upper bounds. Since each node has multiple parents, a node can be pruned through the lattice upper bound of any of its parent nodes. Therefore, a node is inserted into the BFS queue for enumeration only if all its parents were already visited and none had a lattice upper bound $>\epsilon$. Hence, each node keeps track of the number of its unvisited parents (i.e. $unVisitedParents$). This also avoids adding duplicate copies of a node to the queue through the node's multiple parents. In addition, each node stores the following information: (1)$supersetCount$: the maximum cardinality found so far of a superset pattern of this node; and (2)$isPruned$: a flag to indicate if the node was already pruned through one of its ancestor nodes. Initially, for each node n, $isPruned$ is set to $False$, $unVisitedParents$ is set to the number of parent nodes of n, and $supersetCount$ is set to 1 since we are sure that there is at least one instance of the root node pattern in the current window, and this root node pattern is a superset of all the patterns in that window. An $enumeratedPatterns$ table is used to store the patterns already enumerated. This table also stores the cardinalities of the leaf nodes' patterns and their join sets. Finally, the algorithm uses a queue to perform a Breadth First Traversal for the lattice nodes.

Algorithm 1 MTNMiner

```
1: $enumeratedPatterns \leftarrow \{\}$
2: Queue queue $\leftarrow \{\}$
3: $startingEdgeIndex \leftarrow$ CreateStartingEdgeIndexFromMETs
4: $lattice \leftarrow$ Create and initialize lattice
5: for each window $w$=<$t_i, t_j$>in $W_N$ do          ▷ iterate through all non-compliant windows
6:     for t := $\delta$ to 0 do          ▷ iterate from 0 to the max lag $\delta$ preceding $w$
7:         $latticeCp \leftarrow$ CreateDeepCopy($lattice$)
8:         $LeafJoinSetCount \leftarrow$ EnumerateOneVariableNodes(latticeCp,enumeratedPatterns)
9:         queue.enqueue(latticeCp.root)
10:        while queue not empty do
11:            Node node $\leftarrow$ queue.dequeue()
12:            enumerateWithUpperBoundPruning(latticeCp,node,queue,w,t,$\delta$)

13: function enumerateWithUpperBoundPruning(lattice,n,queue,w,t,$\delta$)
14:     if $UB_{lattice}(\hat{K}_{\text{n.C},W_N}(\delta)) \leq \epsilon$ then PruneAllNodeSubsets(n,lattice)
15:     else if $UB_{local}(\hat{K}_{\text{n.C},W_N}(\delta)) \leq \epsilon$ then
16:         for each unpruned non-leaf child node $ch$ of n do
17:             $ch.supersetCount \leftarrow$ max($ch.supersetcount$,$n.supersetcount$)
18:             Check if n is last visited parent of ch, then queue.enqueue($ch$)
19:     else          ▷ no pruning occurred
20:         $C \leftarrow$ expandPattern($n$)
21:         if $C$ not in $enumeratedPatterns$ then
22:             $[|C|, |C \overset{\delta}{\bowtie} W_N|] \leftarrow$ Calculate cardinalities using $startingEdgeIndex$
23:             $enumeratedPatterns$.put($C$)
24:             if $\frac{|C|}{T} \geq minsupp$ and $\hat{K}_{C,W_N}(\delta) > \epsilon$ then Output C.
25:             for each unpruned non-leaf child node $ch$ of $n$ do
26:                 $ch.supersetCount \leftarrow$ max($ch.supersetcount$,$|C|$)
27:                 Check if n is last visited parent of ch, then queue.enqueue($ch$)
28:         else          ▷ C already enumerated
29:             PruneAllNodeSubsets(n)
```

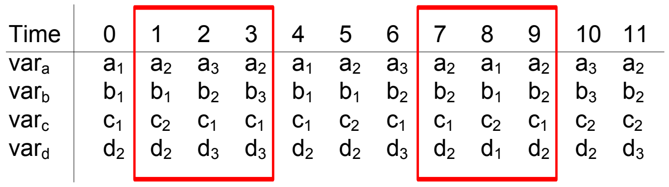
Algorithm 1 shows the pseudo code of MTNMiner. First, the algorithm starts by initializing the used data structures (lines 1-4). Next, the pattern enumeration step is performed (lines 5-12). The algorithm iterates through all temporal windows starting within a time lag t preceding a non-compliant window, where $0 \leq t \leq \delta$. For each temporal window, MTNMiner starts by creating a copy of the initial lattice to enumerate the patterns in that window (line 7). Pattern enumeration proceeds in two phases: **Phase 1 (line 8)** enumerates the patterns represented by all the leaf nodes. Each pattern is expanded by retrieving it from the input time series. If the pattern was already enumerated, its cardinality and join set cardinality are retrieved from the *enumeratedPatterns* table and used to calculate its cross-K function. The join set cardinality is also stored in the $LeafJoinSetCount$ array. Otherwise, the pattern cardinality and its join set cardinality are calculated and stored with the pattern in the *enumeratedPatterns* table. Additionally, the join set cardinality is stored in the $LeafJoinSetCount$ array to be used in calculating the upper bounds for the rest of the lattice nodes. In **phase 2 (lines 9-12):** the algorithm performs a top-down breadth first traversal starting from the root node of the lattice and continuing until the queue is empty. For each node in the queue, the function $EnumerateWithUpperBoundPruning(.)$ (lines 13-29) is called, as follows.

The $EnumerateWithUpperBoundPruning(.)$ function starts by calculating the lattice upper bound of the node using the maximum join set cardinality of all the one-variable subsets of the node (already stored in $LeafJoinSetCount$) and the value of $supersetCount$ of the node. If the lattice upper bound is $\leq \epsilon$, all subset patterns of this node (i.e. all descendant nodes) are marked as pruned (line 14). If not, the local upper bound is calculated. If this bound is $\leq \epsilon$ (line 15), then the cost of enumerating the pattern represented by this node (i.e., expanding it and calculating its cardinality and the cardinality of its join set) is saved. However, we still need to examine the children of this node (lines 16-18). For each child node not marked as pruned, we set its $supersetCount$ variable to the maximum of its current $supersetCount$ value and the $supersetCount$ of its parent node. Then, we decrease the number of $unVisitedParents$ for the child by one and if this was the last visited parent (i.e. unVisistedParents = 0), we insert the child node into the queue. Finally, if the local upper bound of the node was greater than ϵ, then we have to enumerate this node (lines 19-29). First the node is expanded by retrieving the actual pattern from the time series. If the pattern was already enumerated (lines 28-29), all its subset nodes are marked as pruned. Otherwise (lines 21-27), the cardinalities of the pattern and its join set are calculated using the *startingEdge* index, and the pattern is inserted into the *enumeratedPatterns* table. If the pattern satisfies the *minsupp* threshold, its cross-K function is calculated and if this value exceeds ϵ, the pattern is output (line 24). Finally (lines 25-27), the child nodes are treated in the same way as described before; however, in this case, the *supersetCount* of each child is set to the maximum of its current value and the cardinality computed for the parent node (line 26). As the *supersetCount* value of each node increases, the lattice upper bound becomes tighter.

Bottleneck Analysis: Table 2.4 shows a break-down of the running time of MTNMiner without pruning. A MET with T =50,000 points is used, with L= 5 sec, δ = 1 sec, ϵ = 15, and *minsupp*=0.01% threshold is specified. As shown in Table 2.4, the main bottleneck is calculating the cardinality of candidate patterns and their join set. By comparison, the time required for copying and traversing the actual lattice (in addition to all other tasks) is negligible. Hence, our pruning strategies focus on avoiding this cardinality computation cost. It should also be noted that although for each enumerated window, a lattice is created with nodes representing all candidate patterns within that window, only one lattice at a time is kept in memory.

Table 2.4: MTNMiner Bottleneck Analysis

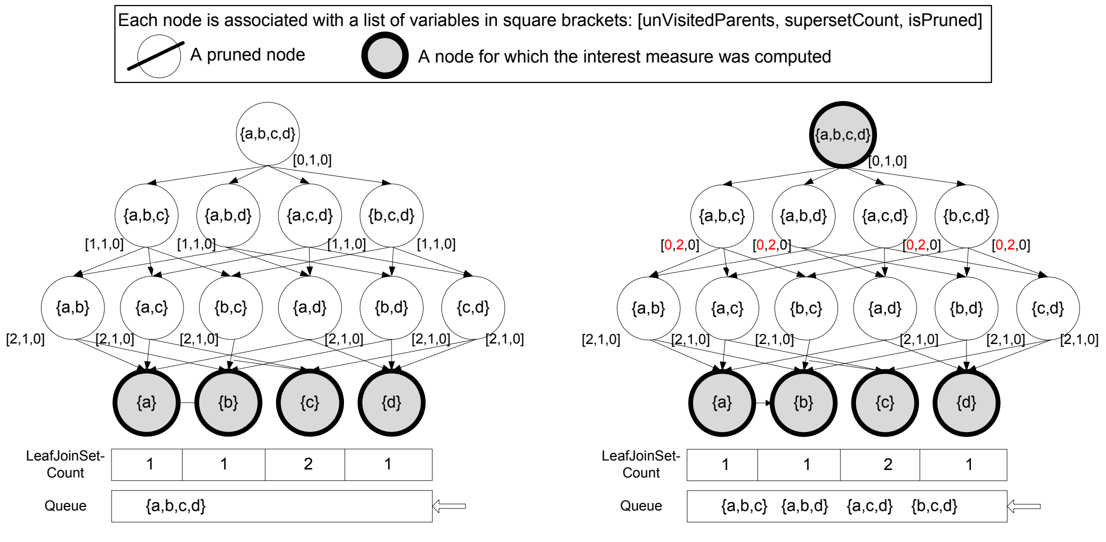
No. of variables	Cardinality Counting Time	Other Tasks Time	Total Time
6	267.2 sec	3.1 sec	270.3 sec
8	769.2 sec	9.3 sec	778.5 sec
10	3772 sec	39 sec	3811 sec



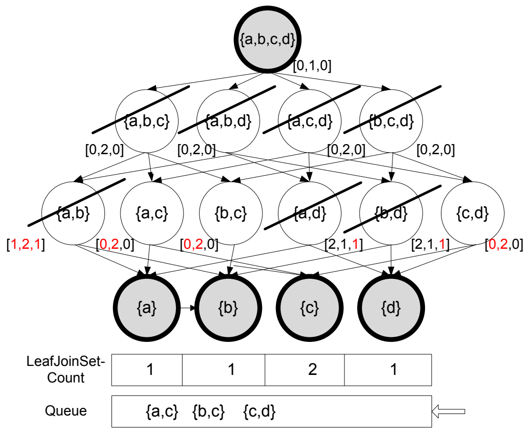
Time	0	1	2	3	4	5	6	7	8	9	10	11
var_a	a_1	a_2	a_3	a_2	a_1	a_2	a_3	a_2	a_1	a_2	a_3	a_2
var_b	b_1	b_1	b_2	b_3	b_1	b_1	b_2	b_2	b_1	b_2	b_3	b_2
var_c	c_1	c_2	c_1	c_1	c_1	c_2	c_1	c_1	c_2	c_1	c_2	c_2
var_d	d_2	d_2	d_3	d_3	d_2	d_2	d_3	d_2	d_1	d_2	d_2	d_3

Figure 2.5: Input with two non-compliant windows

Execution Trace: Figure 2.6 shows an example run of MTNMiner for the input data shown in Figure 2.5. The MET is of length T=12 and has two non-compliant windows of length L = 3 sec, namely, <1,3>and <7,9>. The cross-K function threshold is set to $\epsilon = 3.5$, $\delta = 1$ sec and *minsupp*=1/6 (for illustration). For brevity, the execution trace shows only the enumeration of candidates within one window <0,2>, which started 1 sec before the non-compliant window <1,3>. Similar enumerations will be done for the windows <1,3>, <6,8>and <7,9>.


Each node is associated with a list of variables in square brackets: [unVisitedParents, supersetCount, isPruned]
A pruned node
A node for which the interest measure was computed
{a,b,c,d}
[0,1,0]
{a,b,c} {a,b,d} {a,c,d} {b,c,d}
[1,1,0]
[0,2,0]
{a,b} {a,c} {b,c} {a,d} {b,d} {c,d}
[2,1,0]
{a} {b} {c} {d}
LeafJoinSet-Count
1 1 2 1
Queue
{a,b,c,d}
{a,b,c} {a,b,d} {a,c,d} {b,c,d}

(a) Step 1 (b) Step 2



(c) Step 3

Step	Node	| C |	$\lvert C \overset{1}{\bowtie} W_N \rvert$	$\hat{K}_{C,W_N}(1)$
1	{a}	3	1	2
	{b}	2	1	3
	{c}	3	2	4
	{d}	3	1	2
2	{a,b,c,d}	2	1	3
3	{a,b,c}	-	-	-
	{a,b,d}	-	-	-
	{a,c,d}	-	-	-
	{b,c,d}	-	-	-

(d) Interest measure computations

Figure 2.6: MTNMiner Execution trace (Best viewed in color)

Figure 2.6 shows the lattice for window <0,2>after executing each step of the algorithm, where one whole level is enumerated at every step. Figure 2.6a shows the lattice created for window <0,2>after enumerating the leaf-nodes and calculating their cross-K function. Their join set cardinalities are shown in the array $LeafJoinSetCount$. Figure 2.6d shows the computed values at step 1, in which only the pattern of node $\{c\}$ is output, namely $\{c_1c_2c_1\}$, since its cross-K function equals $4>\epsilon$ and its support equals 3/12>1/6. Next, the root node is inserted in the queue. Figure 2.6b shows the lattice after enumerating the root node by calculating its lattice upper bound (= $\frac{12}{2} \times \frac{\max\{1,1,2,1\}}{1} = 12 > \epsilon$) and local upper bound (= $\frac{12}{2} \times \frac{\min\{1,1,2,1\}}{1} = 6 > \epsilon$). Since both values exceed ϵ, the actual cardinalities of the root pattern and its join set are calculated. Then for its child nodes, the $unVisitedParents$ variable is decremented, their $supersetCount$ is set to the actual cardinality of the root node pattern just computed (changes are marked in red), and the child nodes are inserted into the queue. In Figure 2.6c, the lattice upper bound is calculated for the first node in the queue, $\{a, b, c\}$. Although its value exceeds ϵ, the local upper bound value of 3 is less than ϵ and hence the node is pruned (from Theorem 1). Then, the $supersetCount$ of its child nodes is set to its $supersetCount$ value and their $unVisitedParents$ are decremented. Node $\{a, b, d\}$, which is next in the queue, is enumerated by calculating its lattice upper bound. Since the bound is equal to 3 $<\epsilon$, the node is pruned and its three child nodes are also marked as pruned (from Theorem 2 and Lemma 3). The enumeration continues similarly for all the nodes in this level where nodes $\{a, c, d\}$ and $\{b, c, d\}$ will also be pruned through their local upper bounds (=3$<\epsilon$). Finally, the next level of nodes will be enumerated similarly until the queue is empty.

In the next section, we will present a new proposed approach for NWC pattern mining that enhances the scalability of MTNMiner by combining cross-K function pruning using the lattice and local upper bounds with minimum support pruning.

2.4 Proposed Approach

In this section, we present our proposed **B**i-**D**irectional approach for mining **N**WC patterns (BDNMiner). We start by describing the bi-directional traversal algorithm and the termination condition for ending the pattern search. Then, we describe a method for calculating tighter local and lattice upper bounds using the new proposed traversal for efficiently pruning uninteresting patterns.

2.4.1 BDNMiner: A Bi-Directional approach for mining NWC patterns

The main idea behind BDNMiner is using both the cross-K function and *minsupp* thresholds simultaneously for pruning. To allow pruning based on the cross-K function threshold (i.e. using the lattice and local upper bounds) the algorithm has to perform a top-down traversal of the lattice nodes as previously illustrated in the MTNMiner algorithm. On the other hand, the support measure has a well-known anti-monotone property [42] which indicates that the support of a pattern is always smaller than or equal to the support of its subset patterns. Hence, if the support of a given pattern does not pass the *minsupp* threshold, all its superset patterns can be pruned since they will never pass the threshold either. Therefore, pruning based on the *minsupp* threshold requires a bottom-up traversal of the lattice moving from leaf nodes toward the root node. Thus, our proposed BDNMiner algorithm uses a hybrid bi-directional traversal where one level of the lattice is enumerated from each direction at a time. In the bottom-up traversal, nodes are enumerated and can be pruned using the *minsupp* threshold. In that case, all the parent nodes of the pruned node are also marked as "pruned using *minsupp*". The top-down traversal starts from the top-level and prunes nodes using the lattice and local upper bounds as illustrated in MTNMiner. In Section 2.4.2, we will also show how this bi-directional traversal strategy can be used to compute tighter local and lattice upper bounds where the pattern cardinalities computed during the bottom-up traversal can be used in calculating the upper bounds during the top-down traversal.

The lattice data structure used by BDNMiner can be shown in Figure 2.8a. Unlike MTNMiner, the links between the nodes in this lattice are bi-directional links. Thus, each node has pointers to its parent nodes as well as its children nodes. Additionally, each node has a pointer to its next sibling node in the same lattice to allow the enumeration of a whole level at a time. An array is created with length equal to the number of levels in the lattice where each location in the array stores a pointer to the first node in the corresponding level in the lattice. In addition, each node maintains the following information: (1)$supersetCount$: the maximum cardinality found so far of a superset pattern of this node; and (2)$isPruned$: an integer to indicate if the node was pruned by using the *minsupp* threshold (=1), or the lattice upper bound (=2), or both (=3) or is unpruned yet (i.e. default value=0).

Algorithm 2 BDNMiner

```
1: $enumeratedPatterns \leftarrow \{\}$
2: $startingEdgeIndex \leftarrow$ CreateStartingEdgeIndexFromMETs
3: $lattice \leftarrow$ Create and initialize lattice
4: for each window $w$=<$t_i, t_j$>in $W_N$ do        ▷ iterate through all non-compliant windows
5:     for t := $\delta$ to 0 do        ▷ iterate from 0 to the max lag $\delta$ preceding $w$
6:         $latticeCp \leftarrow$ CreateDeepCopy($lattice$)
7:         $nextBottomLevel \leftarrow 0$
8:         $nextTopLevel \leftarrow latticeCp.dimensionsNum - 1$        ▷ enumerate the leaf-node level
9:         $LeafJoinSetCount \leftarrow$ EnumerateOneVariableNodes(latticeCp,enumeratedPatterns)
10:        if all one variable nodes have $\frac{|C|}{T} \leq minsupp$ then continue
11:        else
12:            $nextBottomLevel \leftarrow nextBottomLevel$+1
13:        enumerateRootNode(latticeCp,latticeCp.root,w,t,$\delta$,$nextTopLevel$)
14:        $isAllLevelPrunedByMinsupp \leftarrow$ false
15:        while (!$isAllLevelPrunedByMinsupp$) and ($nextBottomLevel \leq nextTopLevel$) do
16:            for each node in $nextTopLevel$ do        ▷ enumerate one level from top
17:                enumerateWithUpperBoundPruning(latticeCp,node,w,t,$\delta$)
18:            $nextTopLevel \leftarrow nextTopLevel$-1
19:            if $nextBottomLevel \leq nextTopLevel$ then   ▷ top-down and bottom-up traversals did not meet
20:                $isAllLevelPrunedByMinsupp \leftarrow$ true
21:                for each node in $nextBottomLevel$ do        ▷ enumerate one level from top
22:                    isPruned $\leftarrow$ enumerateWithMinsuppPruning(latticeCp,node,w,t,$\delta$,)
23:                    if not isPruned then
24:                        $isAllLevelPrunedByMinsupp \leftarrow$ false
25:                $nextBottomLevel \leftarrow nextBottomLevel$+1

26: function enumerateWithUpperBoundPruning(lattice,n,w,t,$\delta$)
27:     if n is marked as pruned by $UB_{lattice}$ then return
28:     if n is marked as pruned by $minsupp$ then
29:         for each non-leaf child node $ch$ of n not marked as pruned by $UB_{lattice}$ do
30:             $ch.supersetCount \leftarrow$ max($ch.supersetcount$,$n.supersetcount$)
31:         return
32:     if $UB_{lattice}(\hat{K}_{\text{n.C},W_N}(\delta)) \leq \epsilon$ then PruneAllNodeSubsets(n,lattice)
33:     else if $UB_{local}(\hat{K}_{\text{n.C},W_N}(\delta)) \leq \epsilon$ then
34:         for each non-leaf child node $ch$ of n not marked as pruned by $UB_{lattice}$ do
35:             $ch.supersetCount \leftarrow$ max($ch.supersetcount$,$n.supersetcount$)
36:     else        ▷ no pruning occurred
37:         $C \leftarrow$ expandPattern($n$)
38:         if $C$ not in $enumeratedPatterns$ then
39:             $[|C|, |C \overset{\delta}{\bowtie} W_N|] \leftarrow$ Calculate cardinalities using $startingEdgeIndex$
40:             $enumeratedPatterns$.put($C$)
41:             if $\frac{|C|}{T} \geq minsupp$ and $\hat{K}_{C,W_N}(\delta) > \epsilon$ then Output C.
42:             for each non-leaf child node $ch$ of $n$ not marked as pruned by $UB_{lattice}$ do
43:                 $ch.supersetCount \leftarrow$ max($ch.supersetcount$,$|C|$)
44:         else PruneAllNodeSubsets(n)
```

```
45: function ENUMERATEWITHMINSUPPPRUNING(lattice,n,w,t,$\delta$)
46:     if n is marked as pruned by $minsupp$ then return true
47:     if n is marked as pruned by $UB_{lattice}$ then return false
48:     $C \leftarrow$ expandPattern($n$)
49:     if $C$ not in $enumeratedPatterns$ then
50:         $[|C|, |C \overset{\delta}{\bowtie} W_N|] \leftarrow$ Calculate cardinalities using $startingEdgeIndex$
51:         $enumeratedPatterns$.put($C$)
52:         if $\frac{|C|}{T} \geq minsupp$ then
53:             if $\hat{K}_{C,W_N}(\delta) > \epsilon$ then Output C.
54:         else PRUNEALLNODESUPERSETS(n) and return true
55:     else        ▷ C already enumerated          ▷ use pattern support to check if its supersets can be pruned
56:         if $\frac{|C|}{T} \geq minsupp$ then  PRUNEALLNODESUPERSETS(n) and return true
57:     return false
```

Algorithm 2 shows the pseudo code of BDNMiner. Lines 1-6 are very similar to MTNMiner where the algorithm starts by creating a lattice for the patterns starting at t time points preceding each non-compliant window, where $0 \leq t \leq \delta$. Line 7 initializes the variable $nextBottomLevel$ to zero to indicate the next level to be enumerated by the bottom-up traversal (i.e. leaf-node level). Similarly, line 8 initializes the variable $nextTopLevel$ to the root node level which is the next level to be enumerated by the top-down traversal. Next, the pattern enumeration step is performed in three phases (lines 9-25). **In phase 1 (lines 9-12)**, patterns represented by all the leaf nodes are enumerated by calculating their cardinalities, join set cardinalities and cross-K function(line 9). The join set cardinalities are also stored in the $LeafJoinSetCount$ array. If the support of a pattern does not exceed the $minsupp$ threshold, all its supserset patterns are marked as "pruned using $minsupp$". Line 10 terminates the search in the current window if all leaf nodes were pruned using $minsupp$. Otherwise, the $nextBottomLevel$ variable is incremented to point to the next level (line 12). **In phase 2 (line 13)**, the root node is enumerated by calculating its cardinality and join set cardinality and evaluating its cross-K function if the $minsupp$ threshold is passed. The join set cardinality of the root is then propagated to the $supsersetCount$ variable of all its child nodes and the $nextTopLevel$ variable is decremented. **In phase 3 (lines 14-25)**, the algorithm continues enumerating one level of nodes from each direction at a time. The top-down enumeration is illustrated in lines 16 to 18 by calling the function $EnumerateWithUpperBoundPruning(.)$ for each enumerated node. The bottom-up enumeration occurs in lines 20-25 by calling the function $EnumerateWithMinsuppPruning(.)$. The algorithm continues until the two traversals cross each other (i.e., meet at the same level) or until a complete level is marked as "pruned using *minsupp*" during the bottom-up traversal (line 15). The correctness of this termination condition will be shown in Lemma 4 below.

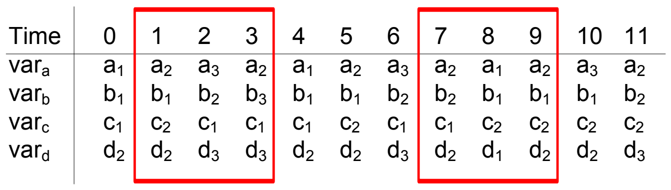
In the $EnumerateWithUpperBoundPruning(.)$ function (lines 26-44), if the node is already marked as pruned by the lattice upper bound (line 27), the function terminates since all its child nodes should already be marked as pruned. However, if it was marked as pruned by *minsupp*, its $supersetCount$ is propagated to its child nodes similar to MTNMiner (lines 28-31). Otherwise, the node is unpruned. Hence, its lattice and local upper bounds are calculated and dealt with in a similar way as in MTNMiner (lines 32-44).

In the $EnumerateWithMinsuppPruning(.)$ function (lines 45-57), the enumeration terminates immediately if the node was marked as pruned either by *minsupp* or the lattice upper bound. Otherwise, the pattern is expanded by retrieving it from the input time series. If the pattern was previously enumerated (lines 55-57), its cardinality is retrieved. Then, if the pattern's support is <*minsupp*, all its superset patterns (i.e. all parent nodes) are marked as pruned using *minsupp*. On the other hand, if the pattern was not previously enumerated, its cardinality and join set cardinality are calculated using the *startingEdge* index. Once again, if the pattern's support is <*minsupp*, all its superset patterns are marked as pruned using *minsupp*. Otherwise, if its cross-K function exceeded ϵ, the pattern is output to the user.

Definition 9 (Termination condition:)**.** *In BDNMiner, the bi-directional search in a given window (i.e. lattice) terminates when any or both of the following two conditions hold:*
(1) $nextBottomLevel > nextTopLevel$
(2) A complete level has been marked as "pruned using minsupp" during the bottom-up traversal

Lemma 4. *The termination condition in Definition 9 is correct.*

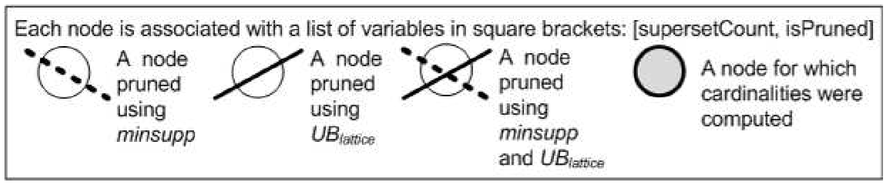
Proof. *To prove Lemma 4, we need to prove that for a lattice representing the patterns in a given window, all patterns must have already been visited or pruned if the termination condition holds. Consider each of the following cases in which the termination condition holds:* ***Case 1:*** $nextBottomLevel > nextTopLevel$. *In this case, the next* *level to be enumerated by the bottom-up traversal is higher than the next level to be enumerated by the top-down traversal. Therefore, each level in the lattice has been already visited by at least one of the two traversals. Hence, all nodes in the lattice have already been visited (A).* ***Case 2: A complete level has been marked as "pruned using*** *minsupp*" ***during the bottom-up traversal****. Due to the anti-monotone property of the support measure, if a pattern's support is less than minsupp, all its superset patterns' support must also be less than minsupp. Therefore, if a complete level was marked as "pruned using minsupp" during the bottom-up traversal, this implies that all their superset patterns are also pruned (see lines 54 and 56 in Algorithm 2). Therefore, all the lattice levels above the pruned level are also pruned. In addition, since the bottom-up traversal already pruned this level, then all levels below it have already been visited by the bottom-up traversal. Therefore, all nodes in the lattice have already been visited or pruned (B). Therefore, from (A) and (B), the termination condition is correct.* ■

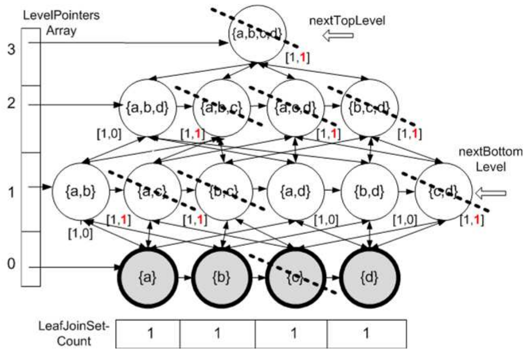


Time	0	1	2	3	4	5	6	7	8	9	10	11
var_a	a_1	a_2	a_3	a_2	a_1	a_2	a_3	a_2	a_1	a_2	a_3	a_2
var_b	b_1	b_1	b_2	b_3	b_1	b_1	b_2	b_2	b_1	b_1	b_1	b_2
var_c	c_1	c_2	c_1	c_1	c_1	c_2	c_1	c_1	c_2	c_2	c_2	c_2
var_d	d_2	d_2	d_3	d_3	d_2	d_2	d_3	d_2	d_1	d_2	d_2	d_3

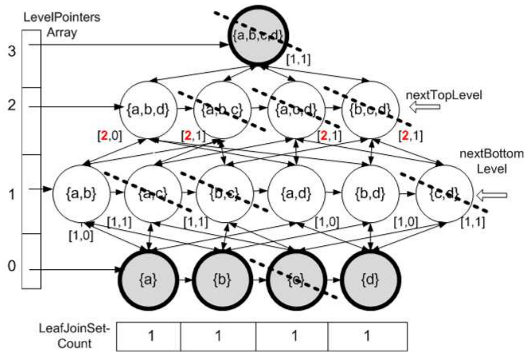
Figure 2.7: Input with two non-compliant windows (best in color)

Execution Trace: Figure 2.8 shows an example run of BDNMiner for the input data shown in Figure 2.7. The MET is of length T=12 and has two non-compliant windows of length L = 3 sec, namely <1,3>and <7,9>. The cross-K function threshold is set to $\epsilon = 3.5$, $\delta = 1$ sec and *minsupp* = 1/4 (for illustration). Again, for brevity, the execution trace shows only the enumeration of candidates within one window <0,2>, which started 1 sec before the non-compliant window <1,3>. Figure 2.8a shows the lattice created for window <0,2>after enumeration of the leaf-nodes. Their join set cardinalities are shown in the array *LeafJoinSetCount*. Since the pattern of node $\{c\}$ has support $= \frac{2}{12} <$ *minsupp*, node $\{c\}$ and all its parent nodes are marked as "pruned using *minsupp*". The *nextBottomLevel* variable is also incremented to point to the next higher level.

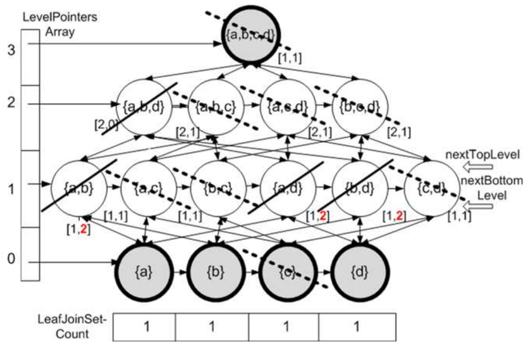

Each node is associated with a list of variables in square brackets: [supersetCount, isPruned]
A node pruned using minsupp
A node pruned using UB_lattice
A node pruned using minsupp and UB_lattice
A node for which cardinalities were computed



(a) Step 1



(b) Step 2



(c) Step 3

Step	Node	$\mid C \mid$	$\lvert C \overset{1}{\bowtie} W_N \rvert$	$\hat{K}_{C,W_N}(1)$
1	{a}	3	1	2
	{b}	3	1	2
	{c}	2	1	-
	{d}	3	1	2
2	{a,b,c,d}	2	1	-
3	{a,b,c}	-	-	-
	{a,b,d}	-	-	-
	{a,c,d}	-	-	-
	{b,c,d}	-	-	-

(d) Interest measure computations

Figure 2.8: BDNMiner Execution trace (Best viewed in color)

Next, Figure 2.8b shows the lattice after enumerating the root node by calculating its cardinality and join set cardinality. The root node is enumerated in order to allow the propagation of its cardinality (=2) to the $supersetCount$ variables of its child nodes. The $nextTopLevel$ variable is also decremented to point to the next lower level. Then, the algorithm starts enumerating one level from each direction until the termination condition is reached. Figure 2.8c shows the lattice after the top-down traversal enumerates the $nextTopLevel$. Since node $\{a, b, d\}$ is the only remaining unpruned node in this level, its lattice upper bound is calculated. Since that bound is equal to $= \frac{12}{2} \times \frac{\max\{1,1,1\}}{2} = 3 < \epsilon$, the node is pruned and its three child nodes are also marked as pruned (from Theorem 2 and Lemma 3). Next, the *nextTopLevel* variable is again decremented and the enumeration continues using the bottom-up traversal. The algorithm terminates when *nextBottomLevel* exceeds *nextTopLevel* in the following step.

2.4.2 Tightening the Local and Lattice Upper Bounds

For a given NWC pattern C=$\{S_i(u_i) \mid u_i \in$ U, U $\subseteq$ V and $1 \leq i \leq |U|\}$ and a time lag δ, the local upper bound (see Definition 7) and lattice upper bound (see Definition 8) were defined using an upper bound on $|C \overset{\delta}{\bowtie} W_N|$, namely $Upper_{Loc}(|C \overset{\delta}{\bowtie} W_N|)$ and $Upper_{Lat}(|C \overset{\delta}{\bowtie} W_N|)$ respectively. These $Upper_{Loc}(|C \overset{\delta}{\bowtie} W_N|)$ and $Upper_{Lat}(|C \overset{\delta}{\bowtie} W_N|)$ bounds were calculated using the minimum and maximum join set cardinality, respectively, of all subset patterns of C that consist of only ***one*** event-sequence (i.e. ***one-variable subset patterns***). Hence, in our preliminary MTNMiner algorithm, all leaf nodes are enumerated first and their join set cardinalities are computed. These cardinalities are then used for calculating the local and lattice upper bounds for all the nodes in the lattice.

In our proposed BDNMiner approach, the local and lattice upper bounds calculated in the top-down traversal can be further tightened by using the join set cardinalities of the nodes at the last level enumerated by the bottom-up traversal rather than only the leaf-node level. For instance, if the last level enumerated by the bottom-up traversal is the level of two-variable nodes (i.e. level 1), the local and lattice upper bounds for node $\{a, b, c\}$ for instance can be calculated using the minimum and maximum join set cardinalities, respectively, of all two-variable subset patterns of node $\{a, b, c\}$ (i.e., patterns of nodes $\{a, b\}$, $\{a, c\}$, and $\{b, c\}$.

To allow the above modification, the bottom-up traversal stores the join set cardinalities of its enumerated nodes in a hash map, $JoinSetCountMap$. The key to this hash map is the node label, e.g., $\{a, b\}$ and the value is the computed join set cardinality of the node. Note that if a node was marked as pruned (i.e using the lattice upper bound), its join set cardinality will not be computed during the bottom-up traversal and hence will be missing from the $JoinSetCountMap$.

Now, we formally define the tightened local and lattice upper bounds and prove their correctness as shown below.

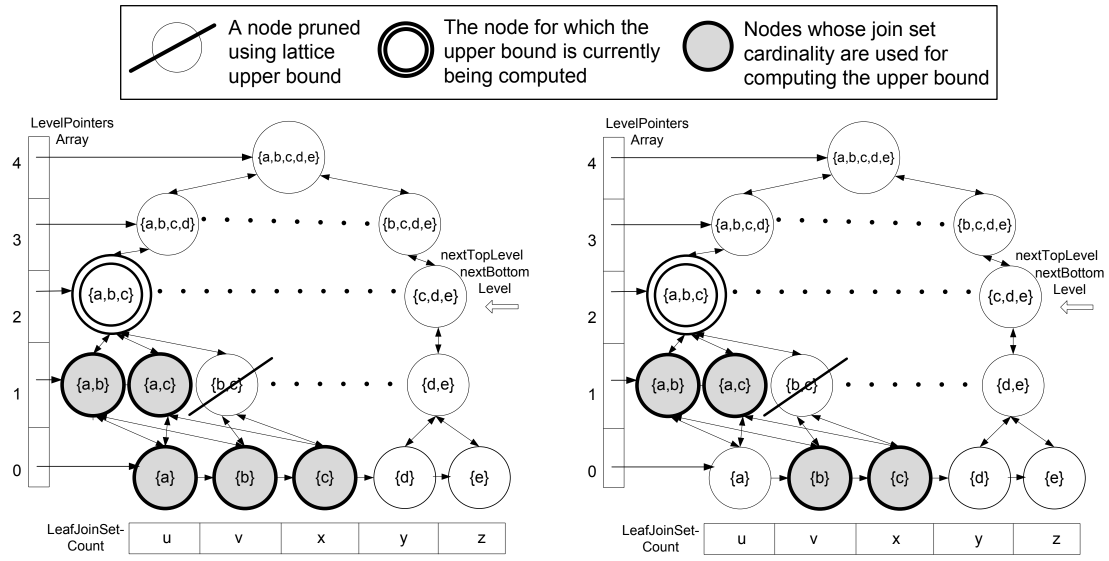
Algorithm 3 Computing $Upper_{TiLoc}(|C \overset{\delta}{\bowtie} W_N|)$ for a node

```
1: function UPPERTILOC(node,LastBottomLevel,JoinSetCountMap)
2:     $subsetLabels$ ← Create all node labels representing subset patterns of node whose dimensionality equals lastBottomLevel+1
3:     $minJoinSetCount$ ← Find minimum join set cardinality of all one-variable subsets of node
4:     for each $lbl$ in $subsetLabels$ do
5:         if $JoinSetCountMap(lbl) \neq$ null then
6:             minJoinSetCount ← MIN(JoinSetCountMap(lbl),minJoinSetCount)
7:     return minJoinSetCount
```

Definition 10. ***Tightened local upper bound**: For the NWC pattern C, the **tightened local upper bound** of $\hat{K}_{C,W_N}(\delta)$, denoted as $UB_{TightLocal}(\hat{K}_{C,W_N}(\delta))$, can be computed as follows:*

$$\begin{aligned} UB_{TightLocal}(\hat{K}_{C,W_N}(\delta)) &= \frac{T}{|W_N|} \times \frac{Upper_{TiLoc}(|C \overset{\delta}{\bowtie} W_N|)}{Lower(|C|)} \\ &= \frac{T}{|W_N|} \times \frac{Upper_{TiLoc}(|C \overset{\delta}{\bowtie} W_N|)}{|superset(C)|} \end{aligned} \tag{2.5}$$

The tightened local upper bound for a pattern C is similar to the definition of the local upper bound (Definition 7) except for the part of the numerator representing the upper bound of $|C \overset{\delta}{\bowtie} W_N|$, namely $Upper_{TiLoc}(|C \overset{\delta}{\bowtie} W_N|)$, which is calculated based on Algorithm 3. The algorithm takes three inputs: the node being enumerated (i.e. node of pattern C), the index of the last level enumerated by the bottom-up traversal, and the $JoinSetCountMap$ storing the join set cardinalities of enumerated nodes. As shown in the algorithm, $Upper_{TiLoc}(|C \overset{\delta}{\bowtie} W_N|)$ is calculated using the minimum of the join set cardinalities of all one-variable subset patterns of C and the join set cardinalities of any subset pattern of C that lies in the last level enumerated by the bottom-up traversal (i.e. patterns whose dimensionality equals the number of variables in the $lastBottomLevel$). Figure 2.9a illustrates the nodes used for computing the $Upper_{TiLoc}(|C \overset{\delta}{\bowtie} W_N|)$ for node {a,b,c} assuming that level 1 (i.e. the level of two-variable nodes) is the last level enumerated from the bottom of the lattice and that the node {b,c} has been marked as pruned. Thus, the grey shaded nodes are the nodes whose join set cardinalities are used in the computation, and the minimum of their join set cardinalities is returned as the value for $Upper_{TiLoc}(|C \overset{\delta}{\bowtie} W_N|)$. Since node {b,c} was marked as pruned, its join set cardinality is not used. Next, we show that the tightened local upper bound is a correct upper bound of the cross-K function interest measure.


A node pruned using lattice upper bound
The node for which the upper bound is currently being computed
Nodes whose join set cardinality are used for computing the upper bound
LevelPointers Array
4
3
2
1
0
{a,b,c,d,e}
{a,b,c,d}
{b,c,d,e}
{a,b,c}
{c,d,e}
{a,b}
{a,c}
{b,c}
{d,e}
{a}
{b}
{c}
{d}
{e}
nextTopLevel
nextBottom Level
LeafJoinSet-Count
u
v
x
y
z

(a) Nodes used for computing $Upper_{TiLoc}(|C \overset{\delta}{\bowtie} W_N|)$ for node {a,b,c}

(b) Nodes used for computing $Upper_{TiLat}(|C \overset{\delta}{\bowtie} W_N|)$ for node {a,b,c}

Figure 2.9: An example illustrating nodes used for computing the tightened upper bounds for node {a,b,c})

Lemma 5. *Given an NWC pattern C and a time lag δ, $Upper_{TiLoc}(|C \overset{\delta}{\bowtie} W_N|)$ is an upper bound of $|C \overset{\delta}{\bowtie} W_N|$.*

Proof. *To prove this lemma, we need to prove that for a pattern C, Algorithm 3 returns an upper bound of $|C \overset{\delta}{\bowtie} W_N|$. Algorithm 3 starts by computing the minimum join set cardinality of all one-variable subsets of C, namely $\{S_i\}$, where $\{S_i\} \subseteq C$, $1 \leq i \leq Dim(C)$ and stores it in the variable minJoinSetCount (line 3). Since all subsets of C have a join set cardinality at least equal to that of C, we now have $minJoinSetCount \geq |C \overset{\delta}{\bowtie} W_N|$. Now, let lastBottomLevel be the the last level enumerated by the bottom-up traversal. From Algorithm 3, we know that the value of minJoinSetCount only changes if a subset pattern of C lying at lastBottomLevel was enumerated and its the join set cardinality is smaller than the current value of minJoinSetCount (lines 4 to 6). Again, since all subset patterns of C have a join set cardinality at least equal to that of C, therefore minJoinSetCount (i.e. the returned value) will always remain greater than or equal to $|C \overset{\delta}{\bowtie} W_N|$. Thus, Algorithm 3 will always return an upper bound of $|C \overset{\delta}{\bowtie} W_N|$* ■

Theorem 3. *Given an NWC pattern C and a time lag δ,* $UB_{TightLocal}(\hat{K}_{C,W_N}(\delta))$ *is an upper bound of* $\hat{K}_{C,W_N}(\delta)$*.*

Proof. *From Theorem 1, we already know that* UB_{local} *is an upper bound of* $\hat{K}_{C,W_N}(\delta)$*. Now, the* $UB_{TightLocal}$ *definition is similar to the* UB_{local} *definition except for replacing the* $Upper_{Loc}(|C \overset{\delta}{\bowtie} W_N|)$ *term by the* $Upper_{TiLoc}(|C \overset{\delta}{\bowtie} W_N|)$ *term for upper bounding* $|C \overset{\delta}{\bowtie} W_N|$*. Since from Lemma 5, we know that* $Upper_{TiLoc}(|C \overset{\delta}{\bowtie} W_N|)$ *is also an upper bound of* $|C \overset{\delta}{\bowtie} W_N|$*, then* $UB_{TightLocal}(\hat{K}_{C,W_N}(\delta))$ *is also an upper bound of* $\hat{K}_{C,W_N}(\delta)$*.*

■

We now define the tightened lattice upper bound as follows:

Definition 11. ***Tightened lattice upper bound****: For the NWC pattern C, the* ***tightened lattice upper bound*** *of* $\hat{K}_{C,W_N}(\delta)$*, denoted as* $UB_{TightLattice}(\hat{K}_{C,W_N}(\delta))$*, can be computed as follows:*

$$\begin{aligned} UB_{TightLattice}(\hat{K}_{C,W_N}(\delta)) &= \frac{T}{|W_N|} \times \frac{Upper_{TiLat}(|C \overset{\delta}{\bowtie} W_N|)}{Lower(|C|)} \\ &= \frac{T}{|W_N|} \times \frac{Upper_{TiLat}(|C \overset{\delta}{\bowtie} W_N|)}{|superset(C)|)} \end{aligned} \tag{2.6}$$

The tightened lattice upper bound for a pattern C is also similar to the definition of the lattice upper bound (Definition 8) except for the part of the numerator representing the upper bound of $|C \overset{\delta}{\bowtie} W_N|$, namely $Upper_{TiLat}(|C \overset{\delta}{\bowtie} W_N|)$, which is calculated based on Algorithm 4. In this algorithm, the bound is calculated using the maximum of the join set cardinalities of all subset patterns of C that lie in the last level enumerated from the bottom of the lattice. If any of these subset patterns does not exist in the *JoinSetCountMap*, all the one-variable patterns of this subset are used instead. This is performed to maintain the pruning power of the lattice upper bound (i.e., the ability to prune all node descendants if the node's lattice upper bound does not exceed the cross-K function threshold) as will be proved in Lemma 6 below. Figure 2.9b shows an example for computing $Upper_{TiLat}(|C \overset{\delta}{\bowtie} W_N|)$ for node {a,b,c} where the shaded nodes are the nodes whose join set cardinalities are retrieved, and their maximum is returned. Since node {b,c} has been marked as pruned, its join set cardinality cannot be found in the *JoinSetCountMap* and thus the join set cardinalities of its one-variable subset patterns, namely {b} and {c} are used instead.

Algorithm 4 Computing $Upper_{TiLat}(|C \overset{\delta}{\bowtie} W_N|)$ for a node

```
1: function UpperTiLat(node,LastBottomLevel,JoinSetCountMap)
2:     $subsetLabels$ ← Create all node labels representing subset patterns of node whose dimensionality equals lastBottomLevel+1
3:     $maxJoinSetCount$ ← -∞                                   ▷ very small value
4:     for each $lbl$ in $subsetLabels$ do
5:         if $JoinSetCountMap(lbl) \neq$ null then
6:             maxJoinSetCount ← MAX(JoinSetCountMap(lbl),maxJoinSetCount)
7:         else
8:             $maxLeafJoinSetCount$ ← Find maximum join set cardinality of all one-variable subset nodes of lbl
9:             $maxJoinSetCount$ ← MAX(maxLeafJoinSetCount,JoinSetCountMap(lbl))
10:    return maxJoinSetCount
```

Next, we show that the tightened lattice upper bound is a correct upper bound of the cross-K function interest measure.

Theorem 4. *Given an NWC pattern C and a time lag δ, $UB_{TightLattice}(\hat{K}_{C,W_N}(\delta))$ is an upper bound of $\hat{K}_{C,W_N}(\delta)$.*

Proof. *From Theorem 2, we already know that $UB_{lattice}$ is an upper bound of $\hat{K}_{C,W_N}(\delta)$. The $UB_{TightLattice}$ definition is similar to the $UB_{lattice}$ definition except for replacing the $Upper_{Lat}(|C \overset{\delta}{\bowtie} W_N|)$ by the $Upper_{TiLat}(|C \overset{\delta}{\bowtie} W_N|)$ term. Therefore, to prove this theorem, we only need to prove that $Upper_{TiLat}(|C \overset{\delta}{\bowtie} W_N|)$ is also an upper bound of $|C \overset{\delta}{\bowtie} W_N|$. Thus, we need to prove that for a pattern C, Algorithm 4 returns an upper bound of $|C \overset{\delta}{\bowtie} W_N|$. Now, let lastBottomLevel be the the last level enumerated by the bottom-up traversal. Algorithm 4 sets the variable $maxJoinSetCount$ to the maximum join set cardinality of the subset patterns of C lying at lastBottomLevel (line 6). If a subset of C at lastBottomLevel has been pruned and thus its join set cardinality was not found in $JoinSetCountMap$, the maximum join set cardinality of its one-variable subsets are used instead (line 8). Since all subsets of C (including one-variable subsets and subsets lying at lastBottomLevel) have a join set cardinality at least equal to that of C, therefore $maxJoinSetCount$ (i.e. return value) is always greater than or equal to $|C \overset{\delta}{\bowtie} W_N|$. Thus, Algorithm 4 will always return an upper bound of $|C \overset{\delta}{\bowtie} W_N|$.* ■

In the following lemma, we prove that the tightened lattice upper bound has a conditional upper bounding property which allows pruning all descendants of a node whose tightened lattice upper bound does not exceed the cross-K function threshold provided that the descendant node lies at a level above the last level enumerated by the bottom-up traversal.

Lemma 6. Conditional Upper Bounding Property for the Tightened Lattice Upper Bound: *Given an NWC pattern C and a time lag δ, then for any pattern C' $\subset$ C, lying above the last level enumerated by the bottom-up traversal, $UB_{TightLattice}(\hat{K}_{C,W_N}(\delta))$ is also an upper bound of $\hat{K}_{C',W_N}(\delta)$.*

Proof. *Let lastBottomLevel be the last level enumerated by the bottom-up traversal. Let $SubsetLabels_C$ be the set of all node labels representing subset patterns of C lying at lastBottomLevel. Based on Algorithm 4, $Upper_{TiLat}(|C \overset{\delta}{\bowtie} W_N|)$ is set to the maximum of the join set cardinalities of nodes in $SubsetLabels_C$ which have not been pruned and the the join set cardinalities of one-variable subsets of the nodes in $SubsetLabels_C$ which have been pruned. Since for every node $x \in SubsetLabels_C$, the join set cardinality of all one-variable subsets of x is greater than or equal to the join set cardinality of x, therefore we have $Upper_{TiLat}(|C \overset{\delta}{\bowtie} W_N|) \geq \max\limits_{x \in SubsetLabels_C}(|x.pattern \overset{\delta}{\bowtie} W_N|)$.(A)*

Now for any pattern C' $\subset$ C, lying above lastBottomLevel, let $SubsetLabels_{C'}$ be the set of all node labels representing subset patterns of C' lying at lastBottomLevel. Since C' $\subset$ C, then for every $y \in SubsetLabels_{C'}$, y also belongs to $SubsetLabels_C$.(B)

From (A) and (B), we get $Upper_{TiLat}(|C \overset{\delta}{\bowtie} W_N|) \geq \max\limits_{x \in SubsetLabels_C}(|x.pattern \overset{\delta}{\bowtie} W_N|) \geq \max\limits_{y \in SubsetLabels_{C'}}(|y.pattern \overset{\delta}{\bowtie} W_N|)$.(C)

In addition, for every y in $SubsetLabels_{C'}$, we have $|y.pattern \overset{\delta}{\bowtie} W_N|) \geq |C' \overset{\delta}{\bowtie} W_N|$ since y is a subset of C'.(D)

Therefore, from (C) and (D), we get:

$Upper_{TiLat}(|C \overset{\delta}{\bowtie} W_N|) \geq \max\limits_{x \in SubsetLabels_C}(|x.pattern \overset{\delta}{\bowtie} W_N|)$

$\geq \max\limits_{y \in SubsetLabels_{C'}}(|y.pattern \overset{\delta}{\bowtie} W_N|) \geq |C' \overset{\delta}{\bowtie} W_N|$ *(E).*

In addition, from Definition 11, $Lower(|C|) = |superset(C)| \leq |C'|$.(F)

Therefore, from (E) and (F), we get:

$UB_{TightLattice}(\hat{K}_{C,W_N}(\delta)) = \frac{T}{|W_N|} \times \frac{Upper_{TiLat}(|C \overset{\delta}{\bowtie} W_N|)}{Lower(|C|)} \geq \frac{T}{|W_N|} \times \frac{|C' \overset{\delta}{\bowtie} W_N|}{|C'|} = \hat{K}_{C',W_N}(\delta)$.

■

2.5 Theoretical Evaluation

In this section, we formally prove the correctness and completeness of our proposed BDNMiner algorithm.

2.5.1 Correctness of BDNMiner

Lemma 7. *The BDNMiner algorithm is correct. Correctness means that every NWC pattern C discovered by the algorithm has* $\hat{K}_{C,W_N}(\delta)>\epsilon$ *and support* $\geq$ *minsupp.*

Proof. *According to Algorithm 2, the BDNMiner algorithm only outputs an NWC pattern in any of the following lines: line 9, line 13, line 41 and lines 52-53. In lines 9 and 13, the EnumerateOneVariableNodes() and EnumerateRootNode() functions, respectively, evaluate the pattern support and cross-K function interest measure and the pattern is output only if its interest measure* $\hat{K}_{C,W_N}(\delta)$ *exceeds* ϵ *and its support is at least equal to the minsupp threshold. Similarly, lines 41 and 52-53 explicitly ensure that the pattern cross-K function exceeds* ϵ *and the pattern support is at least equal to the minsupp threshold. Therefore, the BDNMiner algorithm is correct.* ■

2.5.2 Completeness of BDNMiner

Lemma 8. *The BDNMiner algorithm is complete. Completeness means that every NWC pattern C with* $\hat{K}_{C,W_N}(\delta)>\epsilon$ *and support* $\geq$ *minsupp is reported by BDNMiner.*

Proof. *According to Algorithm 2 (lines 3-6), BDNMiner creates lattice nodes for all patterns that start within time t from all non-compliant windows, where* $0 \leq t \leq \delta$*. Hence, according to Definition 5, all possible NWC patterns have a corresponding node in the created lattices. Then, the BDNMiner algorithm performs a bi-directional traversal where one complete level is enumerated from each direction at a time until the termination condition is satisfied (lines 7-25). Hence, since the termination condition is correct (from Lemma 4), all created nodes are either visited for enumeration or pruned. Therefore, for the algorithm to be complete, we only need to prove that no pruning occurs for any node whose pattern has* $\hat{K}_{C,W_N}(\delta)>\epsilon$ *and support* $\geq$ *minsupp. In Algorithm 2, a node n representing an NWC pattern C is pruned only in one of the following six cases:* ***Case 1: (lines 32 and 33)*** *its lattice upper bound* $\leq \epsilon$ *(line 32) or local upper* *bound $\leq \epsilon$ (line 33). In this case, according to Theorems 2 and 1 respectively (or Theorems 4 and 3 if the tightened lattice and local upper bounds were used), $\hat{K}_{C,W_N}(\delta) \leq \epsilon$ and n can be pruned.* ***Case 2: (also in line 32)*** *the lattice upper bound of one of its ancestor nodes $\leq \epsilon$. From Lemma 3, if the lattice upper bound of one of the ancestors of $n \leq \epsilon$, then the lattice upper bound of $n \leq \epsilon$, and hence its $\hat{K}_{n.C,W_N}(\delta)$ is also $\leq \epsilon$ if the $supersetCount$ is kept monotonically increasing. So, now we only need to prove that $supsersetCount$ monotonically increases as we go down the lattice and the dimensionality of the patterns decrease. According to Algorithm 2, the $supersetCount$ of a node is either kept the same or is set to a larger value from one of its parent or ancestor nodes (lines 30, 35 and 43). Hence, $supersetCount$ never decreases as we go down the lattice and thus the condition holds. Also, if the tightened lattice upper bound was used in line 32, from Lemma 6, the tightened lattice upper bound of a node is also an upper bound on all its descendant nodes that are not already enumerated by the bottom-up traversal. Hence, if the bound of an ancestor node of n is $\leq \epsilon$, then the cross-K function of the descendant node n is also $\leq \epsilon$ and n can be pruned. A node n can also be pruned by:* ***Case 3: (lines 44 and 55)*** *the pattern was already in the enumeratedPatterns table. So, clearly the pattern has been already considered.* ***Case 4: (also line 44)*** *one of its ancestor nodes was in the enumeratedPatterns table. This case happens when an ancestor node pattern has already been enumerated before in another window. Hence, all its descendant patterns have already been considered in that window and there is no need to reconsider those patterns in the current window and thus n can be pruned.* ***Case 5: (line 54)*** *the pattern support is <minsupp so n can be pruned.* ***Case 6: (lines 54 and 56)*** *one of its descendant nodes had a pattern support <minsupp. Due to the anti-monotone property of the support measure, if a pattern is infrequent (i.e. has support <minsupp), then all its parent nodes are also infrequent [42]. Hence, if a descendant node of n has support <minsupp, then the pattern in n must also have a support <minsupp and can be pruned.* ■

2.6 Case Study

Patterns of non-compliant CO_2 emissions: With ever mounting evidence of human caused global climate change, decreasing the production of heat trapping or "green-house" gases (GHG) has become a top priority of researchers around the world [43]. One of the key human produced GHGs which makes up nearly 83% of all heat trapping gasses is carbon dioxide or CO_2 [44] meaning reduction of CO_2 emissions is pertinent and necessary. CO_2 emissions from engine powered vehicles are directly proportional to the quantity of fuel used; therefore CO_2 emissions reduction must be accomplished through reduced fuel use. Furthermore, with over 250 million vehicles registered in the US [45], a small improvement would have a major impact.

To evaluate the effectiveness of the BDNMiner algorithm in detecting non-compliant emissions behavior, we conducted a case study of engine CO_2 emissions using real-world sensor data collected from on board a transit bus in the Minneapolis-St. Paul area, USA. The dataset measured several engine and environmental variables at a rate of 1 Hz. Data points covered roughly 19 days ($\approx$ 176 trips) on three routes with different average speeds to build a dataset representative of transit bus operation and ensure that the data was not biased by a specific route. The mass-specific CO_2 emissions were calculated from the diesel fueling rate, a reasonable assumption for diesel vehicles [46]. With CO_2 as the focus, the non-compliant windows of CO_2 emissions were defined as windows of length $L = 5$ sec in which the average CO_2 in gm/kW-h exceeded the Environmental Protection Agency (EPA) vocational standard threshold ($avgCO2$) of 800 [47] and the percentage of increase in CO_2 exceeded $PincT = 100\%$. We used a temporal cross-K function threshold $\epsilon = 55$, $\delta = 1$ sec, and $minsupp = 0.001\%$.

The variables tested in this study were the engine crank shaft rotations per minute (Engine RPM), exhaust gas recirculation flow rate (EGR kgph), exhaust gas pressure, wheel speed, engine brake power, GPS elevation change, vehicle acceleration, engine fuel rate and engine torque demand difference (Torque Demand Diff) as these are likely to influence CO_2 production and exhibit change within the specified window length and time lag. Most parameters had equal length windows; however a modified window was necessary for engine RPM because of the disproportionate amount of low and high idle periods with narrow RPM bands. Engine idle operation is essentially the default states for the engine when the vehicle is stopped and no engine power is required. Therefore identifying when idling occurs is necessary for understanding what the engine is doing.

Table 2.5: Interesting CO_2-related NWC patterns (smaller indexes indicate smaller values)

ID	NWC Pattern C	$\hat{K}_{C,W_N}(1)$	support(C)
1	Engine RPM: $\{s_3\ s_3\ s_3\ s_2\ s_{22}\}$ Wheel speed: $\{w_0 w_0 w_0 w_0 w_0\}$ EGR kgph: $\{g_0 g_0 g_0 g_0 g_0\}$	62.64	0.0021%
2	Engine RPM: $\{s_4\ s_2\ s_2\ s_2\ s_2\}$ Wheel speed: $\{w_3\ w_2\ w_2\ w_2\ w_1\}$	62.64	0.0011%
3	Fuel rate: $\{f_0\ f_1\ f_2\ f_3\ f_5\}$ Brake Power: $\{p_{16}\ p_{16}\ p_{17}\ p_{17}\ p_{17}\}$ EGR kgph: $\{g_0 g_0 g_0 g_0 g_0\}$	62.64	0.0011%
4	Engine RPM: $\{s_4\ s_3\ s_2\ s_2\ s_2\}$ Wheel speed: $\{w_3\ w_3\ w_2\ w_2\ w_1\}$ Brake Power: $\{p_{16}\ p_{16}\ p_{17}\ p_{17}\ p_{17}\}$	62.64	0.0011%
5	Wheel speed: $\{w_3\ w_3\ w_2\ w_2\ w_1\}$ Fuel rate: $\{f_0\ f_1\ f_2\ f_3\ f_4\}$ Brake Power: $\{p_{16}\ p_{16}\ p_{17}\ p_{17}\ p_{17}\}$	62.64	0.0011%
6	Fuel rate: $\{f_5\ f_6\ f_5\ f_7\ f_{14}\}$ Acceleration: $\{a_{34}\ a_{34}\ a_{34}\ a_{34}\ a_{35}\}$	62.64	0.0013%
7	Wheel speed: $\{w_0 w_0 w_0 w_0 w_0\}$ Fuel rate: $\{f_6\ f_6\ f_6\ f_6\ f_{15}\}$ Acceleration: $\{a_{34}\ a_{34}\ a_{34}\ a_{34}\ a_{35}\}$ Elevation change: $\{e_{100}\ e_{100}\ e_{100}\ e_{100}\ e_{100}\}$ EGR kgph: $\{g_0 g_0 g_0 g_0 g_0\}$	62.64	0.0012%
8	Wheel speed: $\{w_0 w_0 w_0 w_0 w_0\}$ EGR kgph: $\{g_0 g_0 g_0 g_0 g_0\}$ Torque Demand Diff: $\{t_{15}\ t_{15}\ t_{15}\ t_{19}\ t_{23}\}$	62.64	0.0011%
9	Acceleration: $\{a_{34}\ a_{34}\ a_{34}\ a_{34}\ a_{33}\}$ EGR kgph: $\{g_0 g_0 g_0 g_0 g_0\}$ Torque Demand Diff: $\{t_{18}\ t_{17}\ t_{16}\ t_{16}\ t_{15}\}$	62.64	0.0013%

BDNMiner identified 29,510 non-compliant windows and 1,067 patterns. The algorithm distinguished multiple instances of elevated emissions corresponding to the engine reacting to external disturbances such as the driver slowing down or accelerating. Table 2.5 shows a select number of interesting output patterns with the highest cross-K function values where elevated emissions were detected. Pattern 1 shows an instance where the bus engine RPM transitions from the low idle window to a slightly slower engine speed and then jumps up to a high engine speed all while the vehicle wheel speed is in the 0 to 5 km/h range. This is indicative of the driver shifting the bus into drive and accelerating from a break or end-of-route point. Pattern 2 illustrates a series where a transition from a higher speed to a slower speed causes high emissions, and pattern 3 illustrates instances where change in engine power output also results in elevated emissions. Interestingly, patterns 4 and 5 show two engine signatures where elevated emissions are detected as the vehicle is slowing and the engine power production is increasing. At initial glance, this behavior is counter intuitive since a slowing vehicle should not need power to decelerate. However, when taking into account the dynamics of vehicle systems, this behavior can be explained by vehicle accessory power consumption (i.e. water pump, alternator and power steering). These accessories require continuous power to operate. As the vehicle slows, decreasing power comes from the wheels, requiring the engine to compensate. This behavior is further explained by the steady increase in fueling in pattern 5.

Patterns 6 and 7 provide two scenarios where elevated emissions co-occur with an abrupt increase in bus fueling rate and acceleration. This scenario is indicative of the driver accelerating, which results in an increase in CO_2 while the engine works to increase its power output. It is possible that acceleration patterns of different drivers would show up as "signatures" in the fueling rates much like the differences in fueling between patterns 6 and 7. Lastly, two interesting, yet contrasting patterns which could help explain the other cases are patterns 8 and 9. Unlike previous patterns, these are examples of the bus reacting to changes in the demand on the vehicle from external sources (e.g., the air conditioning turns on requiring more engine torque). Both of these cases are dependent on the change in torque demand difference which quantifies the discrepancy between the desired and actual engine output. However, pattern 8 depends on increasing demand difference which may be a result of a vehicle accessory turning on (e.g., air conditioning, power steering, heating fans). Contrarily, pattern 9 depends on decreasing demand difference possibly arising from the bus coasting towards a stop. The occurrence of these two patterns indicates that the increased emissions is a result of the vehicle's system adapting to change. This confirms the findings of others [48, 49] and should prompt engine researchers to investigate strategies for optimally handling these specific transient cases. Such improvements would help in the effort to reduce

CO_2 emissions.

Patterns of non-compliant NO_x emissions: Using the same dataset, we conducted a second case study on NO_x emissions. NO_x is a harmful type of engine emission whose inhalation is detrimental to lung function and increases the health risks for sensitive populations. Additionally, atmospheric NO_x emissions are precursors of the harmful ground level ozone and acid rain formation [21]. In this case study, the non-compliant windows of NO_x emissions were defined as windows of length $L = 5$ sec in which the average NO_x in gm/kW-h exceeded the Environmental Protection Agency (EPA) test threshold ($avgNO_xT$) of 0.267 [17] and the percentage increase in NO_x exceeded $PincT$ = 100%. We used a temporal cross-K function threshold $\epsilon = 15$, $\delta = 2$ sec, and $minsupp = 0.01\%$. The variables used for this case study included engine RPM, engine torque, engine power, wheel speed, and acceleration which can typically influence the increase in NO_x. Additionally, since the production of NO_x is heavily dependent on temperature [46], the engine intake temperature, coolant temperature, and selective catalytic reduction (SCR) system intake temperature were also added to the list of prescribed variables. Except for the coolant temperature variable which is not expected to show a significant change within a window length (L) of 5 sec and a lag time (δ) of 2 sec, the remaining variables selected for this study on NOx emissions are expected to show change within the specified window length and lag parameters. Similar to the CO_2 emission case, most of the variables had equal length intervals; however for the engine RPM, additional modified windows were also created to account for the narrow windows of engine idling.

Table 2.6: Interesting NO_x-related NWC patterns (smaller indexes indicate smaller values)

ID	NWC Pattern C	$\hat{K}_{C,W_N}(2)$	support(C)
1	Wheel speed: $\{w_0\ w_0\ w_0\ w_1\ w_2\}$	21.57	0.66%
2	Engine RPM: $\{s_1\ s_2\ s_3\ s_3\ s_3\}$ Engine power: $\{r_5 r_5 r_5 r_5 r_5\}$ Wheel speed: $\{w_0 w_0 w_0 w_0 w_0\}$ Acceleration: $\{a_{16}\ a_{16}\ a_{17}\ a_{17}\ a_{17}\}$	16.28	0.01006%
3	Engine RPM: $\{s_1\ s_1\ s_2\ s_3\ s_3\}$ Engine power: $\{r_5 r_5 r_5 r_5 r_5\}$ Wheel speed: $\{w_1\ w_0 w_0 w_0 w_0\}$	17.15	0.011%

The number of identified non-compliant windows was 98,290, generating 1,159 NWC patterns. Analysis of the output shows that BDNMiner was able to correctly identify the high NO$_x$ association with low engine load and slow speed driving that was previously found by Misra et al. [22]. The association between slow driving speed and high NO$_x$ is shown in the first row of Table 2.6. This NWC pattern illustrates that accelerating at speeds between 0 and 15 km/h is highly associated with elevated NO$_x$ conditions. The output pattern shown in the second row of Table 2.6 illustrates the association between high NO_x output and low engine load. In this instance, the wheel speed was between 0 and 5 km/h, and the engine load was around 10% of the rated load as indicated by the low engine power bin which would constitute a low load condition. These findings confirm that NWC pattern discovery can correctly identify patterns associated with high NO$_x$. A particularly interesting finding is also shown in the last row of Table 2.6. In this NWC pattern, the wheel speed appears to decrease from the start of the window, while the engine RPM appears to increase substantially, resulting in what can be thought of as a counter intuitive vehicle operation. A potential explanation of this case could be the effect of some factors such as a down-shift in the transmission. Further investigation would be needed to understand the true cause of this finding, but having these windows identified provides engine researchers with a specific starting point and an insight into the parameters involved.

2.7 Experimental Evaluation

The goal of our experiments was to evaluate the performance of the proposed BDNMiner algorithm compared to our previous MTNMiner algorithm. The naive approach was not tested due to its limited scalability as previously shown in [37]. The evaluation was performed on real-world data by varying and observing the effect of the following workload parameters: time series length T, number of variables $|V|$, temporal cross-K function threshold ϵ, minimum support threshold $minsupp$, time lag value δ, NWC pattern length L (i.e., non-compliant window length), and the number of non-compliant windows. Three candidate algorithms were included in our analysis:

- MTNMiner: The multi-parent tracking algorithm discussed in section 2.3.3.
- BDNMiner-LO: This is a version of the BDNMiner algorithm where the lattice

and local upper bounds are calculated similar to the MTNMiner algorithm (i.e. as proposed in Section 2.3.2). LO indicates that the bounds are calculated using the join set cardinalities of the **l**eaf nodes **o**nly.

- BDNMiner-LBL: This is the BDNMiner algorithm using the tightened lattice and local upper bounds proposed in Section 2.4.2. LBL indicates that the bounds are calculated using the join set cardinalities of the **l**ast **b**ottom **l**evel that was enumerated.

2.7.1 Experimental Setup

Experiments were performed using the real-world dataset used in the NO_x case study with a time series of length T=100,000 points. The non-compliant windows were defined as windows of length 5 sec in which the average of NO_x emissions in gm/kW-h exceeded the EPA standard threshold ($avgNO_xT$) of 0.267, and the percentage of increase in NO_x exceeded $PincT$ = 100%. The default parameter values were: T = 50,000 points, $|V| = 8$, $\epsilon = 15$, $\delta = 2$ sec, $L = 5$ sec, $minsupp$ = 0.005%, $avgNO_xT$ = 0.267 and $PincT$ = 100%, unless stated otherwise. Algorithms were implemented using the Java programming language. All experiments were run on a machine with an Intel Xeon Quad Core 3.00 GHz processor with 64 GB RAM.

2.7.2 Experimental Results

In this subsection, we focus on evaluating the algorithmic refinements of the BDNMiner algorithm as compared to MTNMiner. However, we also evaluated the effect of the *startingEdge* index by running two versions of the naive approach in order to separate the effect of the index from all the other pruning filters. The first version of the naive approach uses a linear scan of the data to calculate the cardinality of each pattern and its join set, while the other version uses the *startingEdge* index. Figure 2.10 shows the execution times of both versions. As can be seen, the

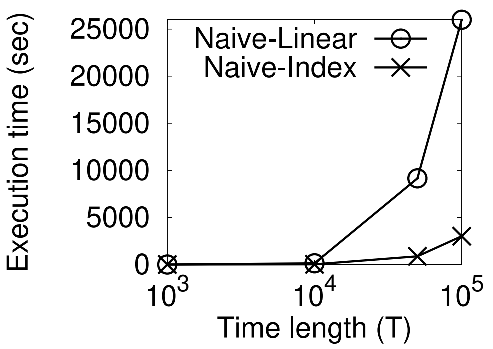


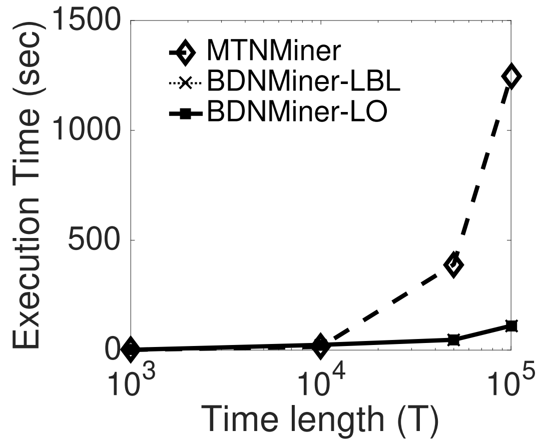
Figure 2.10: Execution time with varying T

startingEdge index leads to substantial computational savings by reducing the time required for cardinality counting. At T=10^5 points, the naive approach using the *startingeEdge* index was 8.5 times faster than the linear scan version. Consequently, we used the *startingEdge* index as the method for cardinality counting in all of our experiments.

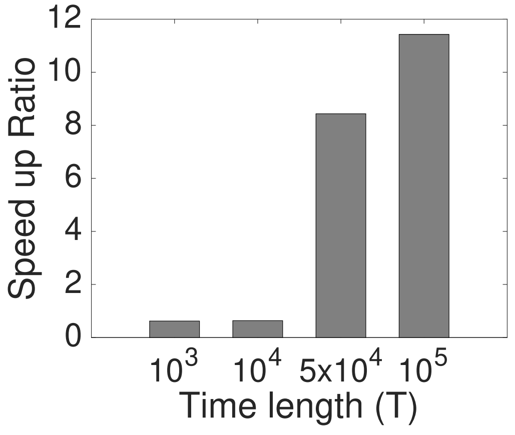
Effect of time series length (T): We ran the BDNMiner (both versions) and MTNMiner algorithms on subsets of the dataset with 1000, 10,000, 50,000 and 100,000 points where each subset was a contiguous set of trips. Figure 2.11a shows the execution times for both algorithms and Figure 2.11b shows the corresponding speedup ($=\frac{\texttt{MTNMiner execution time}}{\texttt{BDNMiner-LBL execution time}}$). As can be seen, both versions of the BDNMiner algorithm significantly outperform the MTNMiner algorithm. We can also see that although BDNMiner-LBL uses tighter upper bounds, its execution time is very similar to the BDNMiner-LO algorithm. This can be attributed to the additional overhead in calculating the tightened upper bounds where all subsets of dimensionality equal to the number of variables in the last bottom level enumerated have to be generated for each node. Overall, the computational savings of BDNMiner increase as the length of the time series increases. At 100,000 points, BDNMiner is more than an order of magnitude faster than the MTNMiner.

Effect of the number of variables ($|V|$): Figure 2.11c shows the execution times for both algorithms as the number of variables increases and Figure 2.11d shows the corresponding speedup. While the execution time of the MTNMiner algorithm increases exponentially, the growth of the BDNMiner execution time is much slower due to the use of minimum support pruning. At $|V|$=14 variables, BDNMiner is 48 times faster than MTNMiner. Figure 2.11e compares the execution times of BDNMiner-LBL and BDNMiner-LO only to clarify their distinction. Although both algorithms have very similar execution times, BDNMiner-LBL runs slightly faster as the number of variables increases due to using tighter upper bounds.

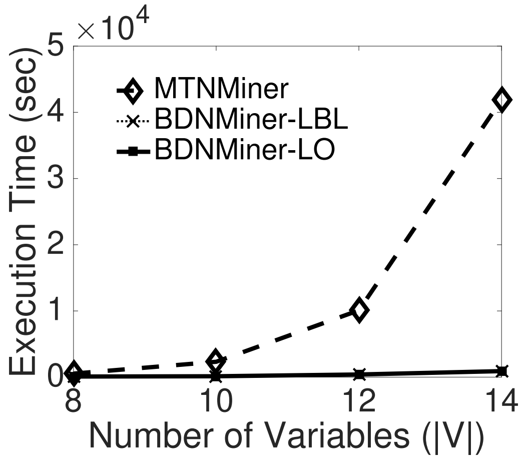
Effect of temporal cross-K function threshold (ϵ): Figure 2.11f shows the execution times for MTNMiner and both versions of BDNMiner as the temporal cross-K function threshold ϵ increases. As can be seen, the execution time for MTNMiner significantly decreases with the increase in ϵ. This happens because when the threshold of the interest measure increases most of the candidate patterns can be pruned.



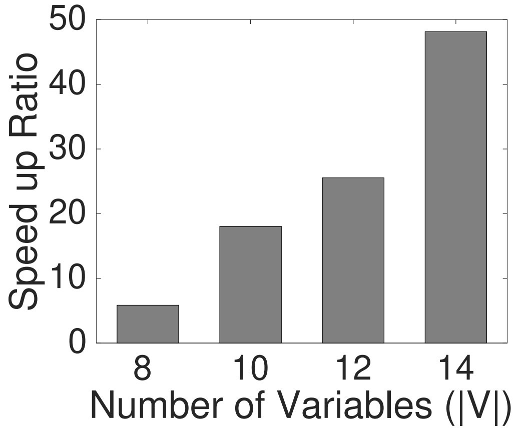
(a) Execution time with varying T



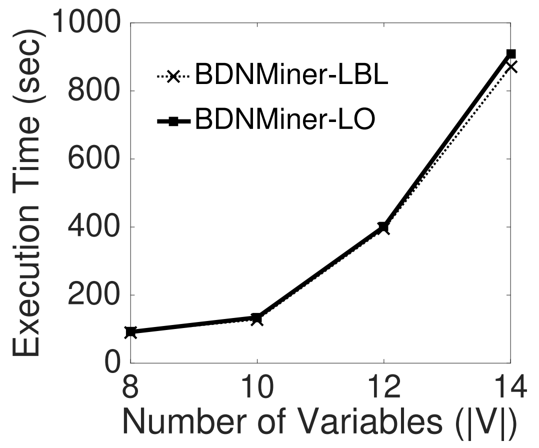
(b) Speedup with varying T



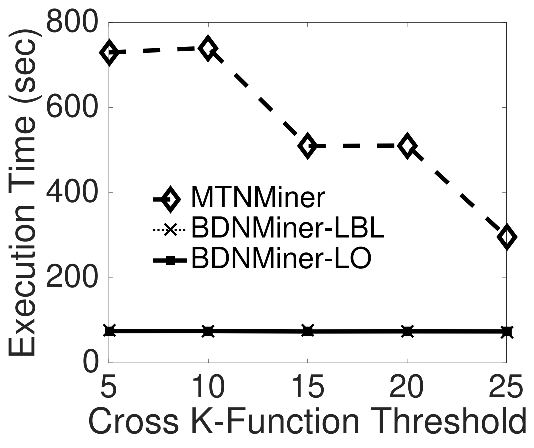
(c) Execution time with varying $|V|$



(d) Speedup with varying $|V|$

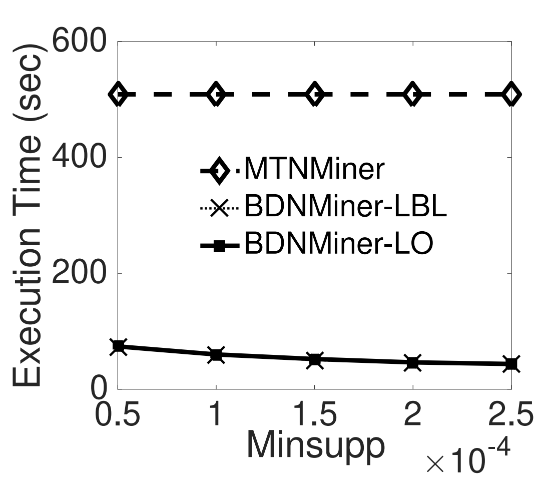


(e) Execution time with varying $|V|$

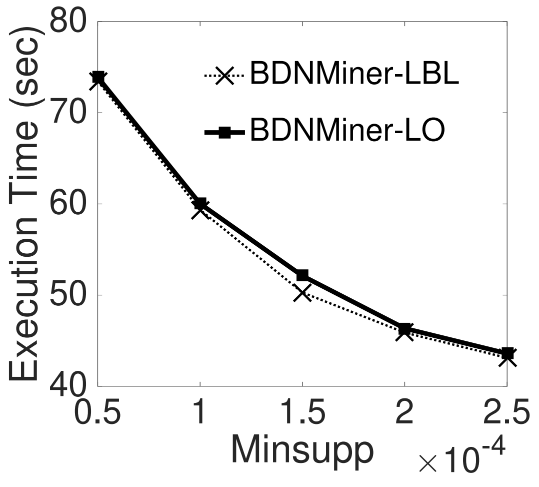


(f) Execution time with varying ϵ

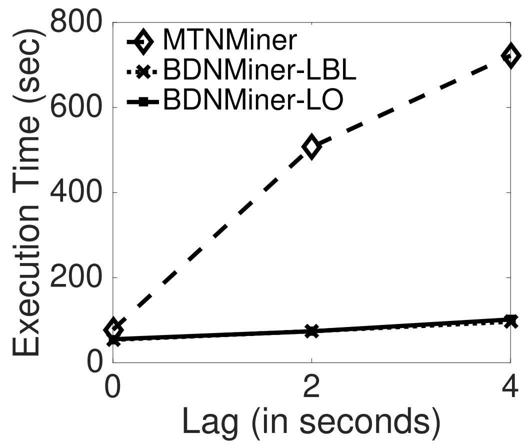
Figure 2.11: Execution time for BDNMiner vs. MTNMiner



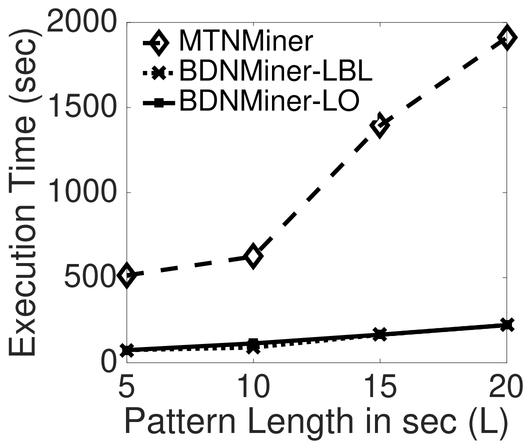
(g) Execution time with varying *minsupp*



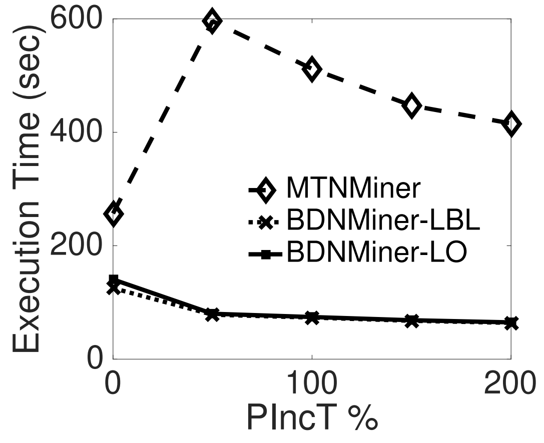
(h) Execution time with varying *minsupp*



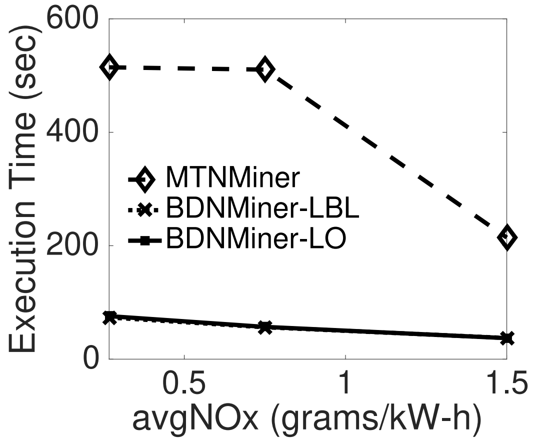
(i) Execution time with varying δ



(j) Execution time with varying L



(k) Execution time with varying *PIncT*



(l) Execution time with varying $avgNO_xT$

Figure 2.11: Execution time for BDNMiner vs. MTNMiner

On the other hand, the execution times of both versions of BDNMiner only slightly decrease with the increase in ϵ and both versions have the same trend. This indicates that the minimum support pruning also plays a dominant role in BDNMiner's execution time savings.

Effect of minimum pattern support threshold (*minsupp*): Figure 2.11g shows the execution times of MTNMiner, BDNMiner-LO and BDNMiner-LBL as *minsupp* increases from 0.005% to 0.025%. The increase in the minimum support threshold has no effect on MTNMiner. Its computational cost remains constant since the algorithm only uses *minsupp* as a post-processing step before a pattern is output to the user. By contrast, the execution time of both BDNMiner versions decreases with the increase in the *minsupp* threshold since more patterns can be pruned during the bottom-up traversal. Figure 2.11h compares the execution times of BDNMiner-LBL and BDNMiner-LO only to illustrate the difference in performance. As can be seen, BDNMiner-LBL runs slightly faster than BDNMiner-LO due to the use of the tighter lattice and lower upper bounds.

Effect of time lag (δ): To observe the effect of the maximum time lag between a pattern and a non-compliant window, we measured the execution times with δ varying from 0 to 4 secs. Figure 2.11i shows that a larger δ value results in an increase in the execution time for both algorithms. The reason is that more time is needed to enumerate the larger number of temporal windows preceding each non-compliant window. Nevertheless, both versions of BDNMiner consistently outperform MTNMiner with computational savings increasing as the value of δ increases.

Effect of pattern length (L): Figure 2.11j shows the effect of pattern length on the MTNMiner, BDNMiner-LO and BDNMiner-LBL algorithms. As the pattern length increases, the execution time of all algorithms increase due to the increase in the cost of calculating the pattern cardinality. Nevertheless, both versions of BDNMiner always outperform MTNMiner.

Effect of the number of non-compliant windows: The effect of the number of non-compliant windows was studied by varying two variables which control the non-compliant window definition, namely, $PIncT$ and $avgNO_xT$. Figure 2.11k shows the execution times as $PIncT$ increases from 0% to 200%. The computational cost of MTN-Miner decreases as $PIncT$ was increased from 50% to 200%. This is due to the decrease in the number of non-compliant windows, which reduces the total number of patterns enumerated. However, at $PIncT = 0\%$, the number of non-compliant windows was very high, resulting in a large decrease in the cross-K function values for all the candidate patterns. This occurred because the cardinality of non-compliant windows $|W_N|$ lies in the cross-K function's denominator and no output patterns were produced. As a result, most patterns were pruned by MTNMiner in this case, leading to a large reduction in execution time. However, as $PIncT$ increased from 0% to 50%, the number of non-compliant windows decreased substantially (from 12,931 to 6,026 windows), leading to higher values of the cross-K function and less pruning. Then, as $PIncT$ increased to 100%, the number of non-compliant windows exhibited a smaller decrease (from 6,026 to 4,845 windows). At this smaller decrease, the reduction in the overall computation pattern enumeration time was higher than the pruning lost by the increase of the cross-K function values, leading to an overall reduction in execution time. For both versions of the BDNMiner algorithm, the execution time decreases as $PIncT$ increases due to the decrease in the number of non-compliant windows and consequently the number of enumerated patterns. We can also see that between 0% and 50%, BDNMiner does not show the same trend as MTNMiner since BDNMiner's execution time was less affected by the changes in the cross-K function values due to the simultaneous use of the minimum support pruning filter.

Figure 2.11l shows the execution times when varying $avgNO_xT$ from the EPA standard threshold of 0.267 up to 1.5 gm/kW-h. Similar to the effect of $PIncT$, as $avgNO_xT$ increases, fewer non-compliant windows are identified, leading to a decrease in the execution time for both BDNMiner and MTNMiner. However, as before, BDNMiner always performs significantly better.

2.8 Conclusion and Future Work

This work explored the problem of Non-compliant Window Co-occurrence (NWC) pattern discovery in relation to an important real-world application, eco-friendly transportation. The NWC discovery problem is challenging due to the large number of candidate patterns, large data volume and the lack of monotonicity in the temporal cross-K function used to measure the interestingness of a pattern. In this work, we proposed a bi-directional pruning approach for mining NWC patterns (BDNMiner) which uses both cross-K function and minimum support pruning simultaneously. We also proposed a method for calculating tighter bounds for the cross-K function in BDNMiner as opposed to the bounds proposed in our preliminary work. For large datasets, the proposed BDNMiner algorithm was shown to be an order of magnitude faster. We also presented two case studies using engine measurement data that showed the effectiveness of the proposed algorithm in finding patterns of interest to engine scientists and which motivate future engine research.

In the future, we plan to relax some of our assumptions in the NWC problem by exploring patterns of variable lengths, particularly for datasets collected at higher frequencies, and considering different lag values for the different explanatory variables. In addition, we plan to explore more spatial aspects of the NWC discovery problem (e.g. the effect of left/right turns on non-compliant engine emissions). Moreover, we will investigate the discovery of statistically significant NWC patterns as well as a parallel formulation for BDNMiner to further enhance its scalability.

Chapter 3

ULAMA: A Utilization-Aware Matching Approach for Robust On-Demand Spatial Service Brokers

3.1 Introduction

The increasing proliferation of mobile technologies such as smart phones has led to non-traditional business models and the appearance of new marketplaces as evidenced by the emerging on-demand and sharing economy. Today, the on-demand economy attracts millions of consumers annually and over $50 billion in spending [50]. Success stories include many on-demand ride-hailing services; food delivery services such as Instacart [51]; and other types of on-demand services such as bike rental (e.g., Spinlister [52]), home services (e.g., TaskRabbit [53]) and beauty services (e.g., StyleBee [54]), etc. These new marketplaces benefit consumers by increasing their access to services while reducing investment costs in resources and infrastructure (e.g., road and parking space) and other societal costs (e.g., greenhouse gas emissions, fuel consumption) through collaborative consumption that allows meeting larger demand from consumers via efficient management of the available supply.

In this chapter we investigate an on-demand spatial service broker for identifying commerce opportunities between mobile consumers on the road and registered service providers (e.g., restaurants, grocery stores, hair salons, etc). The broker receives service requests from moving consumers on the spatial network and matches these requests to stationary service providers while satisfying the constraints of both the consumers and the service providers. This is a challenging problem due to the conflicting requirements of the broker (e.g. maximizing profit), consumers (e.g. minimizing travel and waiting times) and service providers (e.g. maximizing their matching size) and the need to satisfy these requirements while consumer requests are unknown in advance.

Related work for this problem includes matching algorithms proposed for spatial crowdsourcing systems as well as online ridesharing systems. These algorithms focused on the goal of maximizing the number of matched requests or assigned tasks. However, they may result in an unbalanced matching where some providers/drivers are continuously assigned most of the work while other providers/drivers remain idle. This situtation may particularly arise at times when the available demand gets closer to or below the provided supply, and thus may result in service providers switching to other service brokers and reducing the robustness of the on-demand broker against such variations in the supply-demand ratio. Therefore, in this chapter we propose ULAMA, a UtiLization-Aware Matching Approach that focuses on balancing the provider utilization while still maximizing the number of matched requests. We extensively evaluate our approach using synthetic datasets with real-world characteristics and compare its solution quality to offline integer programming formulations where all consumer requests are known in advance.

3.1.1 Overview of On-Demand Spatial Service Brokers

Given a set of service providers defined by their locations and service rates, a set of dynamically arriving consumer service requests and a number of required propositions K, an on-demand spatial service broker (illustrated in Figure 3.1) matches each consumer request to K service provider propositions and presents the corresponding estimated time of service for each proposition. In this context, a spatial service refers to a category of services where the locations of consumers and service providers are critical in the matching process since consumers need to drive or walk to get the service. The service provider propositions assigned to a consumer must meet consumer constraints such as the maximum acceptable travel time and maximum acceptable waiting time before service, while not violating the providers' supply constraints. The goal of the broker is to maximize the number of matched requests. In addition, the broker has to keep the "eco-system" functioning by engaging as many service providers as possible and ensuring that opportunities to fulfill consumer requests are equitably distributed among providers. For instance, in an on-demand ride-hailing service, service providers (i.e., drivers) that are consistently not assigned any rides may eventually decide to leave the system or switch to another broker, resulting in overall service degradation. Hence, on-demand ride-hailing services encourage drivers to stay on the road by trying to avoid periods of idleness, as well as using other incentives such as alerting them when they are so close to meeting a personal earning target [55]. The importance of balancing provider utilization increases and becomes an especially high priority when the available supply exceeds demand, and engaging service providers becomes more challenging.

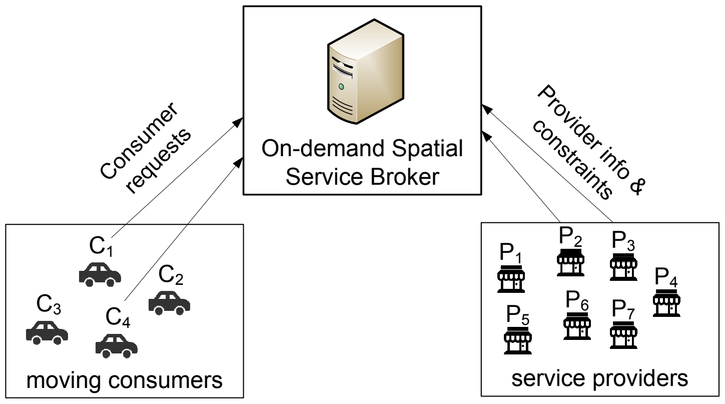


Figure 3.1: An on-demand spatial service broker

3.1.2 Challenges

Designing an on-demand spatial service broker is challenging for the following reasons: First, the broker needs to satisfy many conflicting requirements. For instance, the broker aims to maximize the number of matched requests for maximizing its profit while simultaneously keeping the eco-system functioning by balancing the utilization of the available service providers. The broker also needs to satisfy the conflicting requirements of consumers (e.g., in terms of travel costs and wait times) and service providers (e.g., maximize the number of each provider's assigned requests).

A second challenge is that consumers' requests are unknown in advance. Therefore, minimizing the variance of provider utilization at each time instant does not guarantee the optimal minimum variance of provider utilization that can be achieved if all requests were known in advance. For instance, suppose we have two service providers P_1 and P_2 with a utilization of 1/2 and 1/3 respectively. To minimize the variance in provider utilization at the current time instant, one may assign all the incoming service requests to P_2 to increase its utilization. However, if the future service requests are all in the neighborhood of P_2 and the demand for P_1 declines, then assigning the current requests to P_1, although appearing as a suboptimal decision at the current time instant, may lead to a more overall balanced utilization with a smaller variance.

A third challenge is that the relationship between available supply and demand in on-demand business models, as depicted by the supply-demand ratio, exhibits spatio-temporal heterogeneity. Hence, a matching strategy that works well for one time and/or location may not work as well for other times or locations with different supply-demand ratios. Finally, given a number of consumer requests and a list of candidate propositions that satisfy the constraints of each request, finding the set of K-propositions that maximizes the number of matched requests is an NP-hard problem (as discussed in Section 3.3.4).

3.1.3 Contributions

In our prior work [56], we formally defined the problem of On-demand Spatial Service Propositions (OSSP) and proposed a new category of service provider-centric heuristics that focused on increasing the number of engaged service providers. However, these heuristics did not account for the supply capacity (i.e. service rate) of each service provider and did not consider the temporal heterogeneity of demand in a typical day, which limits their ability to fully balance the providers utilization. To address the limitations of our prior work, this work makes the following new contributions:

1. We prove that solving the OSSP problem for a set of available consumer requests at any given time instant is an NP-hard problem (Section 3.3.4).

2. We propose ULAMA, a Utilization-Aware Matching Approach that minimizes the variance in provider utilization while meeting the conflicting requirements of the broker, consumers and service providers. This approach consists of the following contributions:

   (a) Novel provider-supply and supply-demand ratio aware heuristics, namely, Least Utilized First and Least Recent Demand-Supply Ratio First, for minimizing the variance in provider utilization (Section 3.4.1).

   (b) A consumer-priority based greedy matching algorithm with a conflict-aware consumer prioritization strategy that uses the above heuristics while also prioritizing consumers during matching to maximize the number of matched requests (Section 3.4.2).

3. We propose three offline integer programming formulations, which differ primarily in the objective function, to derive bounds on the solution quality of the proposed approach by assuming that all consumer requests are known in advance (Section 3.4.4).

4. We experimentally evaluate our proposed approach using synthetic datasets with real-world characteristics (Section 3.5). Our experimental results show that our proposed approach outperforms our prior and related work on multiple performance measures, including a smaller variance in provider utilization and a higher average utilization for the lowest 10% utilized providers. In addition, the proposed approach also achieved a larger number of matched requests compared to our prior and related work when supply exceeds demand and when both supply and demand are balanced.

3.1.4 Scope and Outline

In this work, we focus on the problem of designing an on-demand spatial service broker that matches incoming consumer requests with service provider propositions while satisfying consumer and supply constraints and keeping the eco-system alive. However, learning consumer preferences for different service providers (e.g., preferred stores, meals, or cuisines) and incorporating them into the matching algorithm is considered outside the scope of this work. We also assume that the broker and service providers are driven by the number of matched service requests (i.e., transactions) which is a proxy for their profit and that profits are fixed for all service requests. Additionally, this work does not model the temporal evolution in the behavior of consumers and service providers which may be critical to the robustness and survivability of the eco-system.

The rest of the chapter is organized as follows: Section 3.2 discusses the related work and its limitations. Section 3.3 presents some basic concepts and the formal definition for the OSSP problem. Then, it analyzes the problem complexity by presenting a proof of NP-hardness. Section 3.4 presents the details of our proposed approach. The experimental evaluation is given in Section 3.5. Section 3.6 presents our conclusion. Finally, Section 3.7 discusses our future work.

3.2 Related Work

The related work for this problem falls into two categories. The first category is spatial crowdsourcing [57, 58, 59, 60, 61, 62]. In a spatial crowdsourcing system, the matching server dynamically receives information about available tasks as well as requests from workers who are ready to work, and then matches the workers to these tasks. Each worker specifies his constraints such as the region in which he can accept tasks and the maximum number of tasks he is willing to perform. Different objective functions have been used by the server. For instance, in [61], the goal was to maximize the spatial/temporal diversity of spatial tasks such as taking videos or photos of a landmark from different directions and at different times of the day. In [62], the goal was to recruit a subset of the worker vehicles within a given limited budget such that the spatial and temporal coverage of the vehicles for the study region are maximized.

A more closely related class of spatial crowdsourcing systems is those systems where the server (i.e. broker) assigns the tasks to the workers with the objective of maximizing the number of assigned tasks [57, 58, 59, 60]. In addition, the server tries to minimize the distance traveled by the workers by giving a higher priority to task-worker assignments with the “Least Travel Cost” (LTC). In [57], another idea was proposed for prioritizing the tasks to be assigned, namely, the “Least Location Entropy Priority” (LLEP). In this method, the entropy of each task location is computed based on the number of worker visits to that location. Tasks with lower entropy indicate fewer visits (i.e., tasks in worker-sparse areas) and are given higher priority since tasks in areas with higher worker densities are more likely to be assigned in the future. This allows the broker to maximize the number of tasks assigned in the future when more workers arrive. The second category of related work is online ridesharing systems [63, 64, 65, 66, 67, 68] in which trip requests are dynamically matched to vehicles while satisfying the waiting and service time constraints of the passenger and possibly the maximum detour distance specified by the drivers. In these systems, an incoming trip request is matched to the vehicle that adds the “Least Travel Cost”.

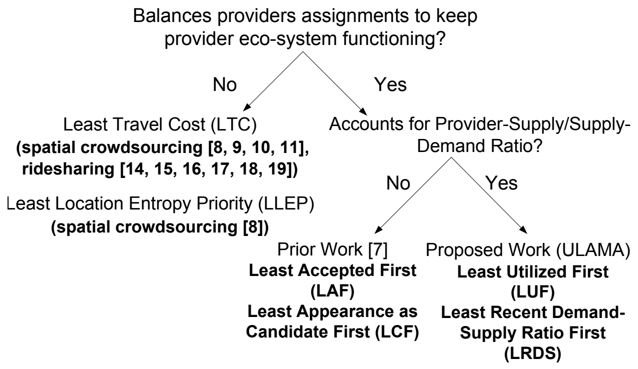

Balances providers assignments to keep provider eco-system functioning?
No
Yes
Least Travel Cost (LTC)
(spatial crowdsourcing [8, 9, 10, 11], ridesharing [14, 15, 16, 17, 18, 19])
Least Location Entropy Priority (LLEP)
(spatial crowdsourcing [8])
Accounts for Provider-Supply/Supply-Demand Ratio?
No
Yes
Prior Work [7]
Least Accepted First (LAF)
Least Appearance as Candidate First (LCF)
Proposed Work (ULAMA)
Least Utilized First (LUF)
Least Recent Demand-Supply Ratio First (LRDS)

Figure 3.2: Classification of Related Work

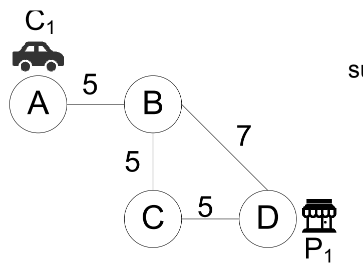
While the related work discussed above has focused on maximizing the number of matched requests (or tasks) and minimizing the total travel time, these works did not consider the need to keep the eco-system functioning by maximizing the number of engaged service providers (e.g., workers in spatial crowdsourcing systems or drivers in ridesharing systems) and balancing the distribution of requests among these providers. In our prior work [56], we proposed a new category of service provider-centric heuristics, namely, Least Accepted First (LAF) and Least Appearance As Candidate First (LCF), for increasing the number of engaged service providers while meeting the conflicting requirements of the broker, consumers and service providers. The LAF heuristic aims to increase the number of matched service providers by giving higher priority to service providers that have received the fewest number of acceptances by consumers so far. The LCF heuristic also aims to increase the matching size by giving a higher priority to providers that have been least considered as a candidate proposition by consumers. A service provider is considered as a candidate proposition if it lies within the maximum acceptable travel time of a consumer and its next available service time satisfies the consumer's maximum acceptable waiting time constraint. Hence, the intuition behind LCF is to favor service providers in regions with fewer originating requests since these providers have a smaller probability of being matched to consumers. In addition, LCF may favor service providers with longer service times since such providers are more likely to have difficulty meeting the consumers waiting time constraints and thus have a smaller probability of being matched.

Our experimental results showed that when the available supply exceeds demand, our prior work heuristics can achieve a larger number of matched requests, a larger number of matched service providers, and a more even distribution of demand among providers. However, our prior work still had the following limitations. First, it only focused on evenly distributing the demand, but may still fail to balance the utilization of service providers with different supply capacities (i.e., service rates) since providers with a smaller number of assignments will always be favored even if they have near full utilization. Second, it did not consider the temporal heterogeneity of demand in a typical day. Therefore, providers with decreasing demand may eventually be underutilized. In this work, we extend our previous work by proposing a utilization-aware matching approach with novel provider-centric heuristics that account for provider supply and achieve a higher utilization balance among service providers as compared to our prior work. Figure 3.2 shows a classification of our closely related work.

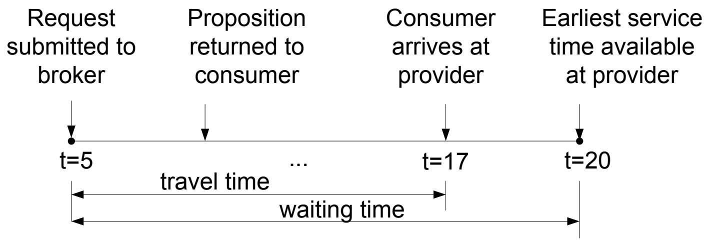
3.3 Basic Concepts and Problem Statement

In this section we define some basic concepts and formally define the OSSP problem.

3.3.1 Basic Concepts



(a) A spatial network with one consumer and one service provider


Request submitted to broker
Proposition returned to consumer
Consumer arrives at provider
Earliest service time available at provider
t=5
...
t=17
t=20
travel time
waiting time

(b) Event timeline for the request of consumer C_1

Figure 3.3: An example of a consumer submitting a service request and being matched to a service provider

Definition 12. *A* ***Spatial Network*** $G = (N, E)$ *consists of a node set* N *representing road intersections and an edge set* E *representing road segments. Each node* $n \in N$ *is associated with a pair of real numbers (latitude, longitude) representing the spatial location of the node. Edge set* E *is a subset of the cross product* $N \times N$*. Each element* $e = (u, v) \in E$ *is an edge that joins node* u *to node* v*, and is associated with a scalar value representing the travel time cost along that edge.*

Definition 13. *A* ***Service Provider*** $p = (id_p\ , loc_p, s_p)$ *is a provider that is registered in the spatial service broker system. Each provider is associated with a provider id* id_p *, a node representing the provider's location in the spatial network* loc_p*, and the provider's service rate* s_p *representing the number of requests that can be served per hour.*

Definition 14. *A* ***Consumer Request*** *represents a request* $r = (cid_r, a_r, loc_r, d_{max,r}, w_{max,r})$ *from a mobile consumer on the spatial network. Each request is associated with a consumer id* cid_r*, the arrival time of the request* a_r*, a node representing the consumer's current location in the spatial network* loc_r*, the consumer's maximum acceptable travel time* $d_{max,r}$*, and the consumer's maximum acceptable waiting time before service* $w_{max,r}$*, where the consumer's travel and waiting times are defined as follows:*

Consumer travel time $d(r, p)$*: Given a spatial network* G*, a consumer request* r*, and a service provider* p*, the travel time of consumer* cid_r *to provider* p *is the shortest travel time in* G *from the location at which the consumer submitted his service request* loc_r *to the provider's location* loc_p *in* G*.*

Consumer waiting time $w(r, p)$**:** *Given a consumer request r, and a service provider p that is matched to serve this request by the on-demand broker, the waiting time for consumer* cid_r *before being served at provider p is equal to the elapsed time between the submission of the consumer's request to the broker and the time at which p is scheduled to start serving consumer* cid_r*.*

For instance, consider the spatial network in Figure 3.3a with one consumer C_1 and one service provider P_1. Edges are labeled with the travel time along the edge in minutes. The travel time of consumer C_1 to provider P_1, assuming the request is submitted at node A, is equal to 12 min (i.e. the travel time along the shortest path ABD). Assume that consumer C_1 has submitted the service request at time t=5, and that provider P_1's earliest available service time is at t=20. Therefore, if C_1 is matched to P_1, C_1's waiting time will be equal to 15 min. The travel and waiting times for consumer C_1 are illustrated on the timeline shown in Figure 3.3b.

Definition 15. *A* ***Service Provider Proposition*** *is a proposition that the broker provides in response to a consumer's request for service. It is defined as a 5-tuple* $(r, p, d, t_{start}, t_{timeout})$ *where r represents a consumer request, p represents a service provider proposed for serving request r, d represents the shortest travel time from* loc_r *to* loc_p*,* t_{start} *represents the time at which provider p will start serving consumer* cid_r*, and* $t_{timeout}$ *represents the time duration during which the broker keeps the service time for request r reserved at provider p, starting from the time instant at which the proposition is provided to the consumer. If the proposition is accepted within* $t_{timeout}$*, the consumer is matched to this provider. Otherwise, the reserved service slot is released and the consumer request is not matched.*

For instance, the proposition (R_1, P_2, 12, 20, 3) matches request R_1 to be served at provider P_2. The shortest travel time from the consumer's location at which the request was issued to the provider is 12 min and the consumer can start getting service at this provider at t=20. The proposition will expire in 3 minutes if not accepted by the consumer. Hence, the broker may provide a set of suggested propositions to a consumer. The consumer can then accept one of these propositions within the given timeout duration.

Definition 16. ***Provider Utilization****: The utilization of a provider p, denoted as*

u_p, is defined as the ratio between the number of propositions accepted by consumers from this provider and the total number of service slots held by the provider. Thus, the utilization of provider p can be computed as follows:

$$u_p = \frac{\texttt{number of propositions accepted by consumers from provider P}}{s_p \times \texttt{ simulation time in hours}}$$

3.3.2 Problem Definition

The OSSP problem can be expressed as follows:

Given:

1. A spatial network G
2. A set P of service providers in G.
3. A set R of consumer requests arriving dynamically from consumers in G
4. A number of required propositions K
5. A timeout duration $t_{timeout}$

Find: K service provider propositions for each $r_i \in R$

Objectives:

- Broker-centric: Maximize the number of matched requests
- Provider-centric: Maximize the number of matched providers and minimize the variance of provider utilization

Constraints:

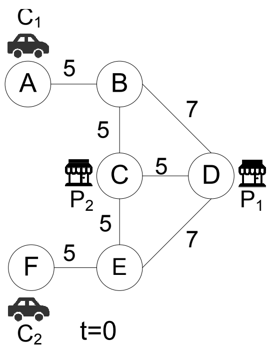
1. For each output proposition $(r, p, d, t_{start}, t_{timeout})$, we have $d \leq d_{max,r}$ and $t_{start} - a_r \leq w_{max,r}$
2. For each provider p, the number of propositions with which p is associated per hour $\leq s_p$
3. For each proposition $(r, p, d, t_{start}, t_{timeout})$, we have $d \leq t_{start} - t_{prop}$, where t_{prop} represents the time at which the proposition is presented to the consumer.

In this problem, for each consumer request, the broker finds K service propositions which satisfy the consumer maximum travel time and waiting time constraints (as shown in the first constraint) and returns them to the consumer. If no K propositions satisfying the consumer constraints are found, the consumer request is not matched. The fifth input is the timeout duration $t_{timeout}$. After a consumer receives K service provider propositions, he/she needs to accept a proposition within $t_{timeout}$ to guarantee the time of service t_{start} presented with each proposition. While matching consumer requests, the broker has to also respect the service rate constraint for each service provider (second constraint). In addition, a consumer cannot be matched to start service at a given provider before he can actually arrive at that provider's location (third constraint).

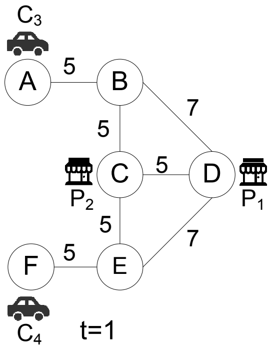
The objective of the broker is to maximize the number of matched requests in order to increase the number of transactions and maximize profits. In addition, the broker aims to match as many service providers as possible and to minimize the variance in the utilization of all service providers. This objective ties to the application domain requirement of keeping the "eco-system" functioning by incentivizing providers to stay in the system.

3.3.3 Problem Example

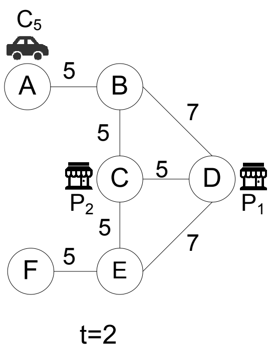
Figure 3.4 shows an example input for the OSSP problem. The weights shown on the edges of the road network refer to the travel time (in minutes) along the edge. Two service providers P_1 and P_2 are located at nodes D and C respectively. The service rates for providers P_1 and P_2 are 4 requests/hr and 8 requests/hr for the current hour, respectively. For simplicity, the number of required propositions (K) is set to 1, the timeout interval length is also set to 1 min, and all consumers are assumed to have flexible travel time and waiting time constraints. In other words, for each arriving consumer, P_1 and P_2 can both be considered as candidate propositions. As shown in Figure 3.4, at the first time instant (t=0), two consumers, namely C_1, located at node A, and C_2 at node F submit requests to the broker to receive service propositions. At t=1, two other requests arrive, one from C_3 at node A and one from C_4 at node F. At t=2, a single request arrives (from C_5 at node A). Finally, a single request arrives at t=3 (C_6 at node F).



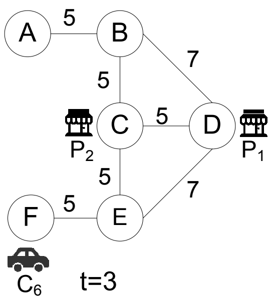
(a) Requests arriving at t=0



(b) Requests arriving at t=1



(c) Requests arriving at t=2



(d) Requests arriving at t=3

Figure 3.4: Example input for the OSSP problem

Table 3.1: Outputs of different heuristics for problem in Figure 3.4

Heuristic	Matching at t=0	Matching at t=1	Matching at t=2	Matching at t=3	Final Matching (grouped by provider)
Least Travel Cost (LTC)	(C1, P2) (C2, P2)	(C3, P2) (C4, P2)	(C5, P2)	(C6, P2)	P2 ← {C1, C2, C3, C4, C5, C6}
Least Location Entropy Priority (LLEP)	(Arbitrary)	(Arbitrary)	(Arbitrary)	(Arbitrary)	(Arbitrary)
Least Accepted First (LAF)	(C1, P1) (C2, P1)	(C3, P2) (C4, P2)	(C5, P1)	(C6, P2)	P1 ← {C1, C2, C5}, P2 ← {C3, C4, C6}
Least Appearance as Candidate First (LCF)	(Arbitrary)	(Arbitrary)	(Arbitrary)	(Arbitrary)	(Arbitrary)
Least Utilized First (LUF)	(C1, P1) (C2, P1)	(C3, P2) (C4, P2)	(C5, P2)	(C6, P2)	P1 ← {C1, C2}, P2 ← {C3, C4, C5, C6}
Least Recent Demand-Supply Ratio First (LRDS)	(C1, P2) (C2, P2)	(C3, P2) (C4, P2)	(C5, P2)	(C6, P2)	P2 ← {C1, C2, C3, C4, C5, C6}

Table 3.2: Output statistics of different heuristics for problem in Figure 3.4

Heuristic	Final Matching (grouped by provider)	No. matched requests	No. matched providers	Providers Utilization & Its Standard Deviation
Least Travel Cost (LTC)	P2 ← {C1, C2, C3, C4, C5, C6}	6	1	u1 = 0, u2 = 6/8 = 3/4, σ = 0.53
Least Location Entropy Priority (LLEP)	(Arbitrary)	6	1 or 2	(Arbitrary)
Least Accepted First (LAF)	P1 ← {C1, C2, C5}, P2 ← {C3, C4, C6}	6	2	u1 = 3/4, u2 = 3/8, σ = 0.265
Least Appearance as Candidate First (LCF)	(Arbitrary)	6	1 or 2	(Arbitrary)
Least Utilized First (LUF)	P1 ← {C1, C2}, P2 ← {C3, C4, C5, C6}	6	2	u1 = 2/4 = 1/2, u2 = 4/8 = 1/2, σ = 0
Least Recent Demand-Supply Ratio First (LRDS)	P2 ← {C1, C2, C3, C4, C5, C6}	6	1	u1 = 0, u2 = 6/8 = 3/4, σ = 0.53

Table 3.1 shows different possible outputs by applying several heuristics from the related and proposed work. The table shows the matching decision made by each heuristic at every time instant and the final overall matching per service provider. For instance, the output of the LTC heuristic is shown in the first row. At each time instant, consumers are assigned to the nearest service provider. Thus, P_2 ends up being assigned all the available requests (assuming consumers waiting and travel time constraints are satisfied) since all requests have shorter travel times to P_2 than to P_1. Table 3.2 shows the objective function values for the outputs of the different heuristics. For instance, LTC was able to match all six available requests, but all matches were to a single provider (i.e., P_2) whose utilization is now 6/8 while P_1's utilization remains zero. The standard deviation (σ) of the providers utilization in this case is equal to 0.53, which indicates the large variance.

The second row in Table 3.1 shows the output of the LLEP strategy. In this case, at every time instant, consumers in provider-sparse areas are given higher priority since they would have a smaller entropy. For example, at t=1, each consumer has two service providers nearby, P_1 and P_2. The location entropy of each consumer C_i is equal to $Entropy(C_i) = -\sum_{p_i \in Prov(C_i)} pr_{C_i}(p_i) \times log_2(pr_{C_i}(p_i))$, where $Prov(C_i)$ is the set of providers in the neighborhood of C_i and $pr_{C_i}(p_i) = \frac{1}{|Prov(C_i)|}$. Hence, both consumers have a location entropy equal to $-[\frac{1}{2}log_2(\frac{1}{2}) + \frac{1}{2}log_2(\frac{1}{2})] = 1$. Since both consumers have the same location entropy (i.e., tie), any of the two consumers can be matched first, and the matching will be arbitrary since each consumer can be matched to either P_1 or P_2. Similarly, the matching is also arbitrary for the other time instants. Therefore, as shown in Table 3.2, this matching strategy can match all six requests, but may result in either 1 or 2 matched providers and an arbitrary utilization for each of them. Based on the choice of the providers at every time instant, the utilization of P_1 may vary from 0 to 1 while the utilization of P_2 may vary from 0 to 6/8.

The output of the LAF heuristic is shown in the third row in Table 3.1, where a higher priority is given to service providers with fewer prior acceptances from consumers. Let $a_{i,t}$ be the number of accepted requests at provider P_i at time t. At t=0, both providers have no accepted requests so far (i.e., $a_{1,0} = a_{2,0}$) so each of the two consumers can be matched arbitrarily to any candidate provider. Suppose that both consumers C_1 and C_2 were matched to P_1 as shown in the third row and second column. Since the timeout interval length is set to 1, all matched consumers need to accept their propositions before the timeout at t=1. Also, since K is set to 1, each consumer is matched to only one service provider and thus we can simply assume that both C_1 and C_2 accept the single propositions to which they were matched. Now at t=1, we have $a_{1,1} = 2$ while $a_{2,1} = 0$ (since P_2 is not yet accepted by any consumers). Therefore, when requests by C_3 and C_4 arrive at t=1, they are matched to provider P_2 which is currently the least accepted provider. Now at t=2, we have $a_{1,2} = a_{2,2} = 2$, so an arriving request from consumer C_5 can be matched to either P_1 or P_2. Suppose that C_5 is matched to P_1. As a result, C_6 will be matched to P_2 at t=3. Table 3.2 (third row) shows that our LAF heuristic was able to match all consumer requests and engage both service providers. It also achieved a higher provider utilization balance than LTC; the standard deviation of providers utilization in this case is only 0.265.

The output of the LCF heuristic is also shown in the fourth row in Table 3.1. LCF prioritizes service providers with the fewest occurrences as a candidate match. Hence, the LCF heuristic will match consumers arbitrarily to P_1 and P_2 since for this example all consumers were assumed to have both P_1 and P_2 as candidate propositions.

As shown in this example, both LTC and LLEP discussed in related work are consumer- or broker-centric heuristics that do not aim at increasing the number of matched providers or balancing the distribution of requests among them. We can also see how our provider-centric LAF heuristic from prior work achieved a more balanced provider utilization when compared to the related work. In Section 3.4 we will show how our novel proposed heuristics can achieve even a higher provider utilization balance while also maximizing the number of matched requests.

3.3.4 Complexity Analysis

In this subsection, we show that solving the OSSP problem for a given set of consumers at any time instant is NP-hard. Consider an example with three consumer requests arriving at time t=0, and three service providers P_1, P_2, P_3 whose service rates (i.e., supply) satisfy the maximum acceptable travel time and maximum acceptable waiting time for the three consumers. Let the number of required propositions per consumer (K) equal 2. A potential match for consumer C_i is denoted as (C_i, S_i) where S_i represents a set of K propositions that can be matched to consumer C_i. We refer to this potential match as a *K-proposition-match.* For instance, $(C_1, \{P_2, P_3\})$ indicates a potential match for consumer C_1 where it is matched to two propositions involving providers P_2 and P_3. Table 3.3 shows the potential K-proposition-matches for each of the three consumer requests.

Table 3.3: Potential K-proposition-matches for 3 consumers with 3 service providers and K=2

Consumer	Potential K-propositions-matches
C_1	$(C_1, \{P_1, P_2\})$, $(C_1, \{P_1, P_3\})$, $(C_1, \{P_2, P_3\})$
C_2	$(C_2, \{P_1, P_2\})$, $(C_2, \{P_1, P_3\})$, $(C_2, \{P_2, P_3\})$
C_3	$(C_3, \{P_1, P_2\})$, $(C_3, \{P_1, P_3\})$, $(C_3, \{P_2, P_3\})$

Identifying the subset of K-proposition-matches from Table 3.3 that maximizes the number of matched consumers is a challenging problem since an on-demand service broker could receive a large number of requests with a large number of candidate providers for each request. Therefore, we now prove that this $OSSP_t$ problem (i.e., the OSSP problem for every time instant t) is NP-hard by reduction from the maximum 3-dimensional matching problem. Our proof follows the proof of NP-hardness for the spatial crowdsourcing problem presented in [58].

Definition 17. ***Maximum 3-dimensional Matching Problem*** *Let X, Y, and Z be finite, disjoint sets, and let T be a subset of $X \times Y \times Z$. That is, T consists of triples (x, y, z) such that $x \in X$, $y \in Y$, and $z \in Z$. Now $M \subseteq T$ is a 3-dimensional matching if for any two distinct triples (x1, y1, z1) $\in M$ and (x2, y2, z2) $\in M$, we have x1 $\neq$ x2, y1 $\neq$ y2, and z1 $\neq$ z2. The maximum 3-dimensional matching problem is to find a 3-dimensional matching $M \subseteq T$ that maximizes the number of triples in M (i.e., $\mid M \mid$).*

Now we define a special problem instance of $OSSP_t$, namely $OSSP_{t,1}$, where we assume that every service provider can only be assigned to one consumer at time t (for instance by choosing a service time for each provider that exceeds the maximum acceptable waiting time of consumers). We start by proving that $OSSP_{t,1}$ is an NP-hard problem.

Lemma 9. *The $OSSP_{t,1}$ problem is NP-hard.*

Proof. *To prove this lemma, we first provide a reduction of the Maximum 3-dimensional matching problem to the $OSSP_{t,1}$ problem in polynomial time. That is, given an instance I_M of the maximum 3-dimensional matching problem, we show that there is an instance I_O of the $OSSP_{t,1}$ problem where the solution to I_O can be converted into a solution to I_M in polynomial time. Now consider an instance I_M where each of the sets X, Y and Z have n elements. Let T be a subset of X x Y x Z. Then, the solution to I_M requires finding a set $A \subseteq T$ such that $\mid A \mid$ is maximized. Next, we describe a mapping from I_M to I_O. To achieve this mapping, for every element in X, we create a consumer request and for every element in Y and Z, we create a service provider such that we now have n consumers and 2n service providers. Every consumer has a set of potential K-proposition-matches M_i where $M = \cup_{i=1}^{|C|} M_i$ and every potential K-proposition-match in $\cup_{i=1}^{|C|} M_i$ will be in the form $(C_x, < P_y P_z >)$, where $0 < x \leq n$, $0 < y \leq n$ and $n < z \leq 2n$. To solve I_O, we need to find the set $B \subseteq M$ such that B is the largest 3-dimensional matching (i.e., a set of maximum cardinality in which no two K-proposition-matches contradict). This means that for every 2 K-proposition-matches in B, say $(C_{x_1}, < P_{y_1} P_{z_1} >)$ and $(C_{x_2}, < P_{y_2} P_{z_2} >)$, $C_{x_1} \neq C_{x_2}$, $P_{y_1} \neq P_{y_2}$ and $P_{z_1} \neq P_{z_2}$. Thus, if B is the solution to I_O, then the solution to I_M representing the set A with maximum cardinality can be created by replacing every consumer C_x with the* *corresponding element x, every provider P_y with the corresponding element y, and every provider P_z with the corresponding element z.*

Lemma 10. *The $OSSP_t$ problem is NP-hard.*

Proof. *By restriction from $OSSP_{t,1}$: since the $OSSP_{t,1}$ is a special problem instance of $OSSP_t$ and is proved to be an NP-hard problem by Lemma 9, therefore, the $OSSP_t$ problem is NP-hard.*

3.3.5 Alternative Problem Formulations

In our OSSP problem definition, every consumer specifies a number of constraints including a maximum acceptable travel time and a maximum acceptable waiting time. However, different alternatives for consumer constraints can also be used without affecting the computational structure of the problem. For instance, the maximum acceptable travel time could be replaced by a maximum acceptable detour distance from a given destination. Similarly, service providers may also provide a time series of daily or weekly service rates since service rates may vary over the day and from weekdays to weekends.

3.4 Proposed Approach

Our prior work focused on increasing the number of matched providers by evenly distributing the demand (i.e., available requests) among providers or favoring providers with smaller demand. However, given that different providers may have different service rates (i.e., supply), the LAF and LCF heuristics may not be fully capable of balancing the utilization of the different providers since they do not account for the variation in the providers supply. In addition, these heuristics do not consider the temporal heterogeneity of demand (e.g., due to changes in day and night population of a region). Therefore, providers with decreasing demand may eventually be underutilized.

In this section we address the limitations of our prior work by proposing ULAMA, a **U**ti**L**ization-**A**ware **M**atching **A**pproach that focuses on minimizing the variance in provider utilization while still maximizing the number of matched requests. Our proposed approach consists of the following several contributions (summarized in Table 3.4).

First, we propose two novel supply and supply-demand ratio aware heuristics for maximizing the number of engaged providers while balancing provider utilization. We then propose a consumer-priority-based greedy algorithm that employs the proposed heuristics while also prioritizing consumers in a way that maximizes their chances of being matched. We also propose a conflict-aware prioritization strategy that can be used by the greedy matching algorithm to account for conflicting candidate propositions during matching. Finally, we present three offline integer programming formulations for the OSSP problem that can be used to derive bounds on the solution quality of our proposed approach.

Table 3.4: Summary of ULAMA

Objective	Maximize No. Matched Consumers	Maximize No. Matched Providers & Minimize Utilization Variance
Proposed Work	Conflict-aware Consumer Prioritization (CAP)	Novel Heuristics: • Least Utilized First (LUF) • Least Recent Demand-Supply Ratio First(LRDS)
	A consumer-priority-based Greedy Matching Algorithm	

3.4.1 Novel Supply and Supply-Demand Ratio Aware Heuristics

Least Utilized First (LUF)

This heuristic aims to minimize the variance in the utilization of the different providers by giving higher priority to service providers that have been least utilized so far. Hence, each proposition from a provider is assigned a score equal to the provider's current utilization, which indicates the gap between the number of requests accepted from the provider and the supply capacity of that provider. During matching, candidate propositions for each consumer are sorted in increasing order of their assigned scores, allowing service providers who have lower utilization to be matched first.

Consider once again the example from Figure 3.4. At t=0, both providers have zero utilization (i.e., $u_1 = u_2 = 0$) and so each consumer can be matched arbitrarily to any candidate provider. Suppose that both consumers C_1 and C_2 are matched to P_1 as shown in the fifth row and second column of Table 3.1. Now P_1's utilization score becomes 2/4 while the utilization score of P_2 remains zero. Therefore, at t=1 both consumers are matched to P_2 since it has the least utilization, and u_2 increases to 2/8. Since the utilization of P_2 is still less than P_1, the broker will also match the request from consumer C_5 at t=2 to provider P_2 whose utilization now increases to 3/8. Finally, at t=3, consumer C_6 is also matched to P_2 and now the utilization of both providers become equal as shown in the fifth row of Table 3.2. In this example, we can see that the LUF heuristic was able to achieve the same matching size (i.e., six requests) while engaging both service providers and achieving a zero standard deviation for provider utilization (i.e., the least possible variance).

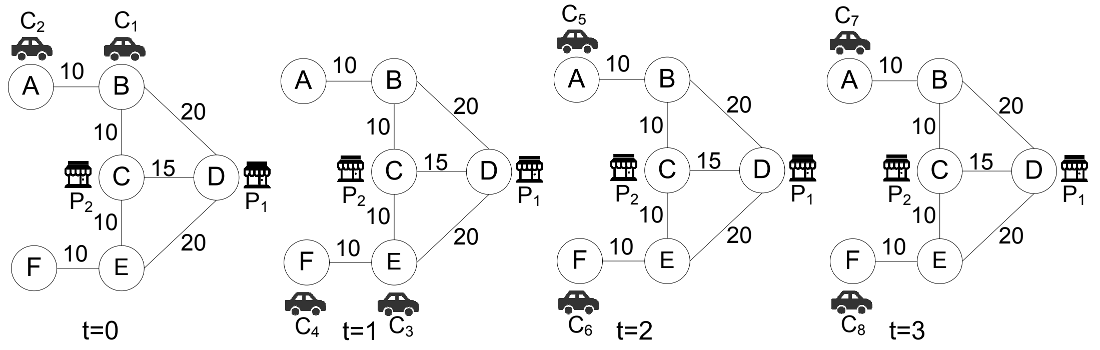
Least Recent Demand-Supply Ratio First (LRDS)

While LUF favors providers with a larger gap between the number of accepted requests at a provider and its supply, the LRDS heuristic observes providers' demand patterns over a recent time window and favors providers with a larger gap between their recent demand and supply. In LRDS, each proposition from a provider is assigned a score equal to the ratio between the demand on that provider (i.e., number of times the provider satisfied the maximum acceptable travel time constraint of a consumer request) during a recent moving time horizon and its supply during the same window length. Candidate propositions for each consumer are then sorted in an increasing order of their scores, allowing providers with larger gaps between their demand and supply to be matched first. Hence, this heuristic accounts not only for providers' supply, but also for the spatio-temporal heterogeneity of demand. For instance, a service provider might have high demand during lunch hours (e.g., a restaurant on university campus), but lower demand in the evening. By observing the changes in demand over a moving time horizon, LRDS can thus favor providers with declining demand to avoid them being eventually underutilized. We also note that favoring providers with the least recent demand-supply ratio is the same as favoring providers with the highest recent supply-demand ratio. However, in this heuristic we choose to compute the demand-supply ratio of each provider to avoid division by zero when a provider has no recent demand.

Let us return to the example in Figure 3.4. Assume a moving time window of length $l = 1$ hour. Let DS_i denote the demand-supply ratio of provider P_i during this most recent window (i.e., hour). Since every provider is considered a candidate proposition for the two requests (i.e., C_1 and C_2) at t=0, we have $DS_1 = 2/4$ while $DS_2 = 2/8$.

Therefore, LRDS favors provider P_2 with the least recent demand-supply ratio. For this example, we will notice that P_2 will always be favored by LRDS since it consistently has a lower demand-supply ratio. This occurs since both P_1 and P_2 have the same demand patterns where each arriving consumer could be assigned to either of them.

Now, consider another example of the OSSP problem as shown in Figure 3.5. Let P_1 and P_2 be two service providers with service rates of 4 requests/hr and 6 requests/hr respectively. For simplicity, we also assume that the number of required propositions (K) is set to 1, and the timeout interval length is set to 1 min. The figure shows that two consumer requests arrive at every time instant. However, for this example, suppose that P_1 and P_2 have two different demand patterns (e.g., due to differences in the maximum acceptable travel time for consumers). This scenario is illustrated in Table 3.5 where , for instance, at t=0, provider P_2 can be considered a candidate proposition for both C_1 and C_2, while P_1 can only be matched to C_1 at the same time instant. The table also shows that the demand for P_1 declines at t=2 and t=3 while the demand for P_2 remains constant.



(a) Requests arriving at t=0 (b) Requests arriving at t=1 (c) Requests arriving at t=2 (d) Requests arriving at t=3

Figure 3.5: Second example input for the OSSP problem

Table 3.5: Provider demand for the problem in Figure 3.5

	t=0	t=1	t=2	t=3
Demand per Provider	P1: C1 P2: C1, C2	P1: C3 P2: C3, C4	P1: - P2: C5, C6	P1: - P2: C7, C8

Table 3.6: Outputs of different heuristics for problem in Figure 3.5

Heuristic	Matching at t=0	Matching at t=1	Matching at t=2	Matching at t=3	Final Matching (grouped by provider)
Least Utilized First (LUF)	$u_1 = 0$ $u_2 = 0$	$u_1 = 1/4$ $u_2 = 1/6$	$u_1 = 1/4$ $u_2 = 3/6$	$u_1 = 1/4$ $u_2 = 5/6$	P1 ← {C1}, P2 ← {C2, C3, C4, C5, C6, C7}
	(C1, P1) (C2, P2)	(C3, P2) (C4, P2)	(C5, P2) (C6, P2)	(C7, P2)	
Least Recent Demand-Supply Ratio First (LRDS)	$DS_1 = 1/4$ $DS_2 = 2/6$	$DS_1 = 2/4$ $DS_2 = 4/6$	$DS_1 = 2/4$ $DS_2 = 6/6$	$DS_1 = 2/4$ $DS_2 = 8/6$	P1 ← {C1, C3}, P2 ← {C2, C4, C5, C6, C7, C8}
	(C1, P1) (C2, P2)	(C3, P1) (C4, P2)	(C5, P2) (C6, P2)	(C7, P2) (C8, P2)	

Table 3.7: Output statistics of different heuristics for problem in Figure 3.5

Heuristic	Final Matching (grouped by provider)	# matched requests	# matched providers	Providers Utilization & Its Standard Deviation
Least Utilized First (LUF)	P1 ← {C1}, P2 ← {C2, C3, C4, C5, C6, C7}	7	2	u1 = 1/4, u2 = 6/6 = 1, σ = 0.53
Least Recent Demand-Supply Ratio First (LRDS)	P1 ← {C1, C3}, P2 ← {C2, C4, C5, C6, C7, C8}	8	2	u1 = 2/4 = 1/2, u2 = 6/6 = 1, σ = 0.35

Table 3.6 shows the outputs for the LUF and LRDS heuristics. For LUF (first row), the utilization value of both providers *before* matching is shown in every column. Similarly, for LRDS (second row), the value of the recent demand-supply ratio DS_i is shown at every time instant. Table 3.7 compares the matching results for both heuristics. The table shows that in this example LRDS outperforms LUF in both the matching size and the provider utilization balance. The reason is that LRDS takes into account the limited demand available for P_1 and hence favors P_1 during matching at t=1. This also saved some service capacity in P_2 which receives higher demand, and thus results in an overall higher number of matched requests by LRDS.

3.4.2 Consumer-Priority-Based Greedy Matching

ULAMA consists of a consumer-priority-based greedy matching algorithm that applies the proposed heuristics discussed above while also prioritizing the available requests for maximizing the number of matched consumers. This algorithm (outlined in Algorithm 5) consists of three main phases: **candidate evaluation**, **consumer prioritization**, and **matching**. In the *candidate evaluation phase* (lines 1 to 6), the algorithm finds the set of candidate propositions for each available consumer by first finding the shortest path between the consumer and all service providers. For the set of service providers within the maximum acceptable travel time of that consumer, we check if the provider's next available service time also satisfies the consumer's maximum acceptable waiting time. When both constraints are satisfied, the provider is added as a candidate proposition for that consumer. Each candidate proposition is then assigned a score that is computed based on one of the provider-centric heuristics discussed in Section 3.4.1. A priority queue is used to store the candidate propositions in the order of their scores. Once all candidate propositions are identified for each consumer, the algorithm goes into a *consumer prioritization phase* (line 7). In this phase, consumers are sorted in an ascending order based on the number of candidate propositions they have. Consumers with a smaller number of candidate propositions are thus given higher priority since they have less probability of being matched later. If two consumers have the same number of candidate propositions, the consumer with a shorter waiting time constraint is matched first since he/she is less likely to be matched than other consumers with longer waiting times. Finally, in the *matching phase* (lines 8 to 12), the algorithm iterates through consumer requests in the new sorted order and selects the first K propositions from the consumer's priority queue that satisfy the consumer's maximum acceptable waiting time constraint. The waiting time constraint is rechecked during the matching phase since the next available service time of a provider in a candidate proposition may still violate the consumer's wait time constraint if that provider has been already matched to another consumer. Once a consumer can be matched to K propositions, the next available service time for the providers of these propositions is updated to reserve the service time up to a given timeout interval. The algorithm then returns the set of matched propositions for all the matched consumers.

Algorithm 5 Consumer-Priority-Based Greedy Matching

Input:

(1) A set of consumer requests (*arrivalEventsList*)
(2) List of service providers
(3) Matching heuristic, e.g., LUF, LRDS

Output:

A set of matched propositions for each consumer

Algorithm:

```
    {Candidate Evaluation Phase:}
 1: for each consumer request $r$ in $arrivalEventsList$ do
 2:     $nearProviders \leftarrow$ Find providers satisfying $d_{max,r}$
 3:     $candidtatePropositions[r] \leftarrow$ Find providers in $nearProviders$ satisfying $w_{max,r}$
 4:     for each $proposition$ in $candidtatePropositions[r]$ do
 5:         $proposition.score \leftarrow$ Calculate score based on input heuristic
 6:         $priorityQueue[r] \leftarrow proposition$
    {Consumer Prioritization Phase:}
 7: Sort consumer requests in $arrivalEventsList$    ▷ consumers with smaller number of candidates in
    $candidtatePropositions$ are put first, and ties are broken in favor of shorter $w_{max,r}$
    {Matching Phase:}
 8: for each consumer request $r$ in sorted $arrivalEventsList$ do
 9:     $matchedPropositions[r] \leftarrow$ Find K propositions in $priorityQueue[r]$ that still satisfy consumer's $w_{max,r}$,
    otherwise return an empty set.
10:     if $matchedPropositions[r]$ not empty then
11:         for each proposition $prop$ in $matchedPropositions[c]$ do
12:             $prop.provider.nextAvailableServiceTime \leftarrow prop.provider.nextAvailableServiceTime +$
    $prop.provider.serviceTime$
13: return $matchedPropositions$
```

Conflict-Aware Prioritization Strategy

The matching algorithm discussed above prioritizes consumers based on their number of candidate propositions. However, a consumer with many candidate propositions that gets lower priority may end up being unmatched since these propositions may not satisfy the consumer's wait time if their service providers were already matched to other consumers that appeared earlier in the sorted list. Thus, we propose a conflict-aware prioritization (CAP) strategy which prioritizes consumers based not only on their number of candidates propositions, but also on the number of non-conflicting propositions for each consumer. Below, we define some basic concepts and then present how the CAP strategy works.

Definition 18. ***Conflict Score of a Provider****: Given a set of consumer requests R at time t, and a set of candidate propositions S_i identified for each request $r_i \in R$, the conflict score of a provider p_j is the number of times that p_j appeared in all sets of candidate propositions $\cup_{i=1}^{|R|} S_i$.*

Definition 19. *A* ***Non-conflicting candidate*** *is a candidate proposition $(r, p, d, t_{start}, t_{timeout})$ where provider p has a conflict score of 1.*

The CAP strategy intersects with all three phases of the greedy algorithm described in this section.

- **During the Candidate Evaluation Phase:** The conflict score of each provider is computed by incrementing the score of a provider whenever it is added as a candidate proposition for a consumer request.
- **During the Consumer Prioritization Phase:** Consumers are sorted by prioritizing consumers with a smaller number of non-conflicting candidates. This allows consumers with no non-conflicting candidates to be matched first to avoid matching other requests to their high conflict propositions. When two consumers have the same number of non-conflicting candidates, we apply several tie breakers: First consumers with a smaller number of candidates are put first. In the case of a tie, consumers with a larger sum of conflict scores over all their propositions are selected. Finally, consumers with smaller waiting times are selected
- **During the Matching Phase:** We apply one of the proposed heuristics discussed in Section 3.4.1. However, when two propositions have the same score (e.g., two providers have the same utilization while applying LUF) the proposition whose provider has a smaller conflict score is selected to increase the probability of other consumers being matched later in the list.

3.4.3 Time Complexity of Proposed Approach

Let n be the number of simultaneous consumer requests in the queue, m be the number of service providers, V and E be the set of nodes and edges in the spatial network respectively, K be the number of required propositions per consumer request, and l be the length of the moving window employed by the LRDS heuristic.

Therefore, the time complexity of ULAMA using the LUF heuristic is O(n$|E| log|V|+$

$nmlogm$). The first term in this cost model represents the cost of running Dijkstra's algorithm for each of the n consumers in the queue to find the shortest travel time between the consumer and all service providers in O($|E|log|V|$) during the candidate evaluation phase. Then, after identifying the candidate propositions for each consumer, a priority queue is created for that consumer to store the computed score for each proposition, where the maximum number of propositions is equal to the number of service providers m. The creation of the priority queues results in an additional cost of O($nmlogm$), the second term in our cost model. This is then followed by the consumer prioritization phase where consumers are sorted in O($nlogn$). We note that the cost of creating the priority queues in the previous phase dominates the cost of sorting. In addition, it also dominates the cost of the matching phase where the K candidate propositions with the smallest scores are selected from the priority queues for each consumer (i.e. O($Knlogm$)). Hence, the total time complexity of ULAMA using LUF is O($n|E|log|V| + nmlogm$).

The time complexity of ULAMA with the LRDS heuristic is similar to the case of LUF, but with an additional term of O(nml) which represents the cost of computing the recent demand for each provider within the moving window of length l. For each of the m providers, the recent demand computation is O(nl). Hence, the total time complexity of ULAMA using LRDS is O($n|E|log|V| + nmlogm + nml$) which is still linear in the number of consumer requests in the queue.

Finally, we note that using the conflict-aware prioritization strategy involves a time complexity of O(nm) which represents the cost of computing the number of non-conflicting candidates and sum of conflicts for each of the n consumers in the queue before sorting, but this cost is still dominated by the cost of creating the priority queue for all consumers (O ($nmlogm$)) and thus can be ignored in our cost model.

3.4.4 Offline Integer Programming Formulation for the OSSP Problem

Table 3.8: Table of Notations for the Offline Integer Programs

Symbol	Description
N	Number of consumers
M	Number of providers
T	Number of simulated time steps. It is equal to the sum of the maximum consumer's arrival time and the maximum consumer's waiting time constraint
K	Number of required propositions per consumer
a_i	Arrival time of consumer C_i
$d_{max,i}$	Maximum acceptable travel time constraint of consumer C_i
$w_{max,i}$	Maximum acceptable waiting time constraint of consumer C_i
$d_{i,j}$	Shortest travel time between consumer C_i and provider P_j
s_j	Service rate (per hour) for provider P_j
v_j	Service time of provider P_j
$simTimeHours$	Simulation time in hours

We developed three Integer Programming formulations for the offline version of the OSSP problem, where all consumer requests are assumed to be known in advance. These formulations are used to derive bounds on the solution quality of our proposed approach. Therefore, the three formulations differ primarily in their objective functions, where the first objective aims to maximize the number of matched requests, the second aims to maximize the number of matched providers and the third aims to minimize the standard deviation of the providers utilization. Table 3.8 summarizes the notations used in our integer programs (IPs). Our main decision variables, denoted as x_{ijt} are defined as follows:

$$x_{ijt} = \begin{cases} 1, & \text{if consumer } i \text{ is assigned to start service at provider } j \text{ at time } t \\ 0, & \text{otherwise} \end{cases}$$

$$\forall i = 1, 2, ..., N;\ \forall j = 1, 2, ..., M;\ and\ \ \forall t = 0, 1, ..., T$$

Next, we present our offline IPs:

Objective for the First Offline IP Maximizing the Number of Matched Requests (OIPMaxReq):

$$Max \sum_i \sum_j \sum_t x_{ijt}$$

Objective for the Second Offline IP Maximizing the Number of Matched Providers (OIPMaxProv):

$$Max \sum_{j} (\sum_{i} \sum_{t} x_{ijt})^{1/2}$$

Objective for the Third Offline IP Minimizing the Standard Deviation of Providers Utilization (OIPMinSTD):

$$Min \operatorname*{STDEV}_{j=1}^{M} (\frac{\sum_i \sum_t x_{ijt}}{s_j \times simTimeHours})$$

subject to:

$$x_{ijt} \in \{0,1\}, \qquad \forall i = 1,2,...,N; \forall j = 1,2,...,M; and\ \forall t = 0,1,...,T \tag{3.1}$$

$$\sum_{j} \sum_{t} x_{ijt} = y_i K, \qquad \forall i = 1,2,...,N \tag{3.2}$$

$$y_i \in \{0,1\}, \qquad \forall i = 1,2,...,N \tag{3.3}$$

$$\sum_{t} x_{ijt} \leq 1, \qquad \forall i = 1,2,...,N; \forall j = 1,2,...,M \tag{3.4}$$

$$d_{ij} \sum_{t} x_{ijt} \leq d_{max,i}, \qquad \forall i = 1,2,...,N; \forall j = 1,2,...,M \tag{3.5}$$

$$\sum_{t=a_i+w_{max,i}+1}^{T} x_{ijt} = 0, \qquad \forall i = 1,2,...,N; \forall j = 1,2,...,M \tag{3.6}$$

$$\sum_{l=t}^{t+vj-1} \sum_{i} x_{ijl} \leq 1, \qquad \forall j = 1,2,...,M; and\ \forall t = 0,1,...,T \tag{3.7}$$

$$\sum_{t=0}^{a_i+d_{ij}-1} x_{ijt} = 0, \qquad \forall i = 1,2,...,N; \forall j = 1,2,...,M \tag{3.8}$$

As already noted, the first objective aims to maximize the matching size in order to derive an upper bound for the number of matched consumers assuming that all requests are known in advance (i.e., solving an offline version of the OSSP problem). Similarly the second objective aims to maximize the number of matched service providers. A function with concave returns is used by raising the number of propositions per provider to a power less than 1. This allows the optimization algorithm to favor matchings where the same number of propositions come from more service providers rather than fewer service providers. Finally, the third objective function aims to minimize the standard deviation of the utilization of all providers. The constraints for all three integer programs are the same except for the third program (balancing providers utilization) where an additional constraint is used as will be discussed next. Constraint 3.1 enforces the binary nature of x_{ijt}. Constraints 3.2 and 3.3 ensure that each matched consumer is assigned K propositions. Constraint 3.4 ensures that a consumer does not receive more than one proposition from the same service provider. Constraints 3.5 and 3.6 ensure that the propositions matched to a consumer meet his maximum acceptable travel time and maximum acceptable waiting time constraints respectively. Constraint 3.7 ensures that, for each provider, no two consumers are matched to the same service time slot. In this case, we assume a single server per provider. However, it is easy to extend the formulation to multiple servers by keeping track of the number of servers (i.e., queues) per service provider. Constraint 3.8 ensures that a consumer is not assigned to start service at a provider before he can reach that provider's location. Finally, the following additional constraint is added only for the third objective:

$$\sum_i \sum_j \sum_t x_{ijt} \geq \sum_i \sum_j \sum_t x^*_{ijt}$$

where x^*_{ijt} is the optimal value for x_{ijt} obtained from solving OIPMaxReq. This constraint is used to put a threshold on the number of matched propositions to avoid degeneration into the trivial solution where no matches occur and a minimum standard deviation of zero is returned.

All constraints presented above are linear constraints. The first objective function is also linear, while the second and third objectives result in convex integer optimization problems. We also note that the number of variables and the number of constraints are both O(N M T), where N is the number of consumers, M is the number of service providers, and T is the number of simulated time steps.

3.5 Experimental Evaluation

Our experimental goals are two-fold: First, we want to compare the performance of our proposed approach to our prior and related work. Particularly, we want to answer the following questions: (1) Does our utilization-aware provider-centric approach improve fairness to (i.e. equity among) providers in terms of the utilization balance? (2) Does a provider-centric approach reduce the business volume? Second, we also want to evaluate the solution quality of our proposed approach by comparing it to the solution of an offline integer program which assumes that all consumer requests are known in advance. The following candidate algorithms were included in our analysis. Note that the prefix "*U-*" is used to indicate the different variations of **U**LAMA to easily differentiate the proposed work from our prior and related work.

- U-LUF: ULAMA with the **L**east **U**tilized **F**irst heuristic.
- U-LRDS: ULAMA with the **L**east **R**ecent **D**emand-**S**upply Ratio First heuristic.
- U-LUF-R: ULAMA with the **L**east **U**tilized **F**irst heuristic while using the Least **R**ecent Demand-Supply Ratio First heuristic as a tie-breaker.
- U-LUF-C: ULAMA with the **L**east **U**tilized **F**irst heuristic and the **C**onflict-Aware Prioritization strategy.

We note that the first three candidates are applied using the consumer-priority based greedy approach discussed in Section 3.4.2 (i.e., sorting consumers based on the number of candidate propositions), while the last candidate employs the conflict-aware prioritization strategy discussed in Section 3.4.2.

For our first experimental goal, we compared against the following related work strategies: (a) Least Travel Cost (LTC), (b) Least Location Entropy Priority (LLEP), (c) Least Accepted First (LAF), and (d) Least Appearance As Candidate First (LCF).

For evaluating the LLEP strategy proposed in [57], we have also used a grid index as suggested in [57] to divide the spatial region into grid cells for computing the location entropy. Service providers lying in the same grid cell of a consumer are used for calculating the location entropy of that consumer.

For our second experimental goal, we used the three offline IP formulations, namely, OIPMaxReq, OIPMaxProv, and OIPMinSTD, with the three objective functions discussed in Section 3.4.4.

3.5.1 Discrete-Event Simulation Framework

We simulate the interactions between arriving consumer requests and the on-demand spatial service broker using a discrete-event simulation framework [69] whose outline is shown in Algorithm 6. A priority queue stores the simulation events in the order of their arrival times. The simulation starts by generating all consumer arrival events and storing them in the queue. Each event carries the arrival time of the consumer's request, maximum acceptable travel time, and maximum acceptable waiting time. The queue also carries another type of event, namely, proposition acceptance events when a user accepts a service provider proposition from among the propositions suggested by the broker. The simulation loop then starts and continues until the queue is empty (lines 5 to 19).

Algorithm 6 Simulation Algorithm

```
 1: $providersList \leftarrow$ Initialize list of service providers
 2: $roadNetworkGraph \leftarrow$ Initialize edges and vertices of road network
 3: Queue $queue \leftarrow$ Insert consumer request arrival events
 4: $clock \leftarrow 0$
 5: while $queue$ not empty do
 6:     $acceptanceEvents \leftarrow$ Get all proposition acceptance events with an event arrival time $\leq$ clock
 7:     $arrivalEvents \leftarrow$ Get all consumer arrival events with an event arrival time $\leq$ clock
 8:     for each event $e$ in $acceptanceEvents$ do
 9:         for each proposition in $e.propositions$ do
10:             if $proposition \neq e.acceptedProposition$ then
11:                 $proposition.provider.nextAvailableServiceTime \leftarrow proposition.provider.nextAvailableServiceTime - proposition.provider.serviceTime$
12:     $matchedProp \leftarrow$ ConsumerPriorityBasedGreedyMatching($clock$, $arrivalEvents$, $roadNetworkGraph$)
13:     for each consumer proposition set $match$ in $matchedProp$ do
14:         $acceptedProposition \leftarrow$ randomly select a proposition from $match.propositions$
15:         $timeBeforeAcceptance \leftarrow$ randomly generate time between 0 and $t_{timeout}$
16:         Create an acceptance event e with arrival time = $clock + timeBeforeAcceptance$ and store $acceptedProposition$, and $match.propositions$
17:         queue.enqueue(e)
18:     Update simulation statistics
19:     $clock \leftarrow clock + 1$
20: Output Simulation Statistics
```

For each iteration, the simulation dequeues all proposition acceptance and consumer arrival events with arrival times before or at the current clock time. Proposition acceptance events are handled first (lines 8 to 11). For each of these events, the next available service time of all the consumer's unaccepted prepositions is updated to release the service time units reserved for that consumer. Then, the broker matches the available consumer requests to service providers by calling the consumer-priority-based greedy algorithm (Algorithm 5) (line 12). For every matched request, a corresponding acceptance event is generated and added into the queue to simulate a consumer accepting one of the suggested propositions (lines 13 to 17). Finally, the simulation performance measures are updated and the clock advances to the next time instant.

We note that while our study employed a discrete-event simulation framework to model the interactions between consumers and the broker, there has been other approaches in the literature using mathematical formulas and the queuing theory for modeling supply and demand [70, 71, 72]. However, these methods usually pose restrictive assumptions on the distribution of parameters and/or the network topology for developing closed forms. Optimization methods (e.g., integer programming [63]) have also been used; however, they do not scale to large datasets and thus would violate the real-time on-demand requirement of this problem. In our work, we simulated different supply and demand scenarios using synthetic datasets designed to approximate real-world characteristics since we did not have access to real datasets which are usually collected as proprietary data.

3.5.2 Experimental Design and Dataset

Our experimental design is illustrated in Figure 3.6. We generated synthetic datasets with real-world characteristics as captured by real service provider locations and the use of real-world population data for the city of Minneapolis, MN. Population data was used for generating the origin locations of consumer's requests. For comparison with related and prior work, we used a dataset of 120 restaurants (i.e., service providers) in the city of Minneapolis. The following procedure was used by the simulator to generate the supply and demand: First, providers' service rates were generated using a random number that was uniformly distributed over the range [$minS$, $maxS$]. Then, given a value for the supply-demand ratio sdr to simulate, we generated a number of consumer requests per hour such that:

$$sdr = \frac{\text{sum of service rates per hour of all providers (i.e., } \sum_{p \in P} s_p)}{\text{total number of requests in that hour}},$$

where $\mid P \mid$ is the set of all service providers.

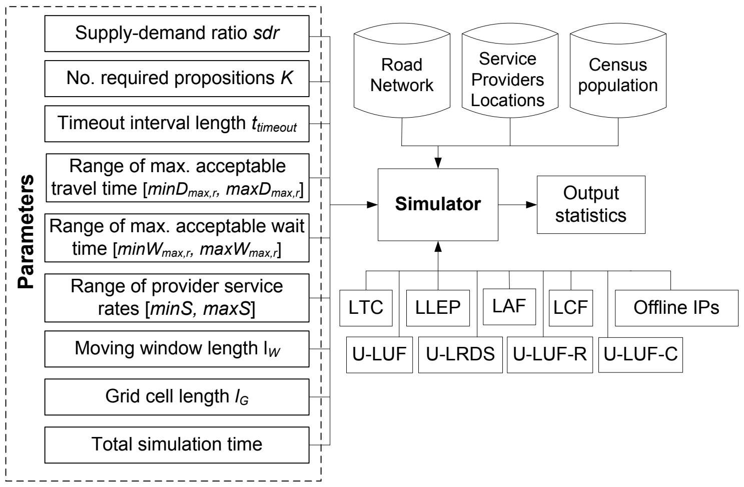

Parameters
Supply-demand ratio sdr
No. required propositions K
Timeout interval length $t_{timeout}$
Range of max. acceptable travel time [$minD_{max,r}$, $maxD_{max,r}$]
Range of max. acceptable wait time [$minW_{max,r}$, $maxW_{max,r}$]
Range of provider service rates [minS, maxS]
Moving window length l_W
Grid cell length l_G
Total simulation time
Road Network
Service Providers Locations
Census population
Simulator
Output statistics
LTC
LLEP
LAF
LCF
Offline IPs
U-LUF
U-LRDS
U-LUF-R
U-LUF-C

Figure 3.6: Experimental Design

The locations at which consumers submitted their service requests were selected randomly from the set of nodes in the spatial network such that the spatial distribution of the generated requests follows the real population density distribution in the city. Hence, the number of requests originating from each node in the network was proportional to the ratio of that node's population to the total city population. Experiments simulated 10 hours of operation assuming 5 lunch hours and 5 dinner hours for each restaurant. The locations of requests during lunch hours were generated in proportion to the day population of the nodes, while the locations of the dinner requests were proportional to the night population. Consumers' maximum acceptable travel times and maximum acceptable waiting times were generated using random numbers that were uniformly distributed over the ranges [$minD_{max,r}$, $maxD_{max,r}$] and [$minW_{max,r}$, $maxW_{max,r}$] respectively. The arrival time of each request generated in a given hour was also uniformly distributed over that hour duration. Although arrival of requests in the real-world may not exhibit uniform distribution, we account for this by simulating a wide range of values for the supply-demand ratio, thus simulating different demand patterns. The experiments were performed using the road network provided by the Minnesota Department of Transportation [73], and U.S. Census population data and were run on a machine with an Intel Core i5 2.3 GHz processor and 8 GB RAM. All algorithms were implemented in the Java programming language and the integer programming models were implemented using the CVX optimization package in Matlab.

We analyzed the performance of our proposed algorithms by varying and observing the effect of the following workload parameters: the supply-demand ratio sdr, the timeout duration $t_{timeout}$, the number of required propositions K, the minimum and maximum values for the provider service rate ($minS$ and $maxS$ respectively), the minimum and maximum values for the maximum acceptable travel time constraint ($minD_{max,r}$ and $maxD_{max,r}$ respectively), the minimum and maximum values for the maximum acceptable waiting time constraint ($minW_{max,r}$ and $maxW_{max,r}$ respectively), the moving window length l_W used by the U-LRDS and U-LUF-R algorithms, and the grid cell length l_G.

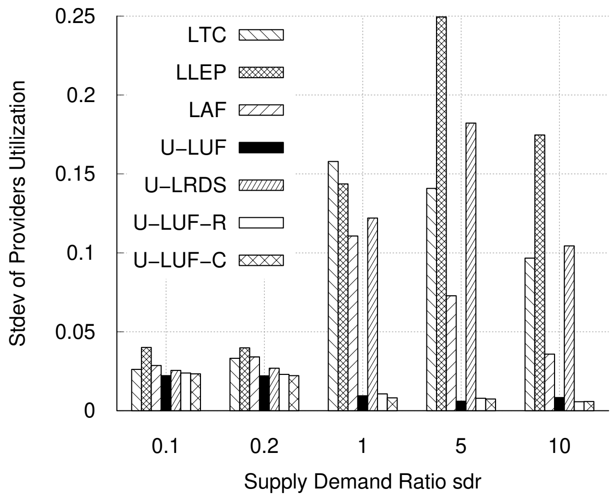
3.5.3 Experimental Results

Evaluating Algorithms Performance Relative to Prior and Related Work:

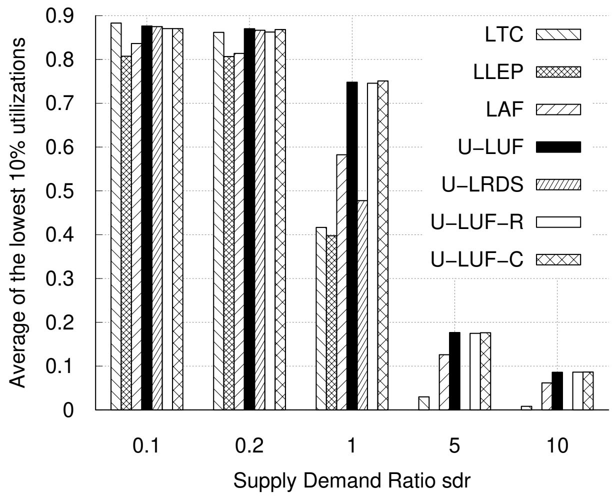
For this set of experiments, the default parameter values were set as follows: $K = 3$, $t_{timeout} = 2$ min, $minS = 5$ requests/hr, $maxS = 15$ requests/hr, $minD_{max,r} = 8$ min and $maxD_{max,r} = 25$ min, $minW_{max,r} = 10$ min, $maxW_{max,r} = 25$ min, $l_W = 30$ min, and $l_G = 2000$ m, unless stated otherwise. Table 3.9 shows the details of the datasets generated for the different sdr values

Table 3.9: Synthetic dataset details for experimental goal 1. Total supply is fixed across all sdr values.

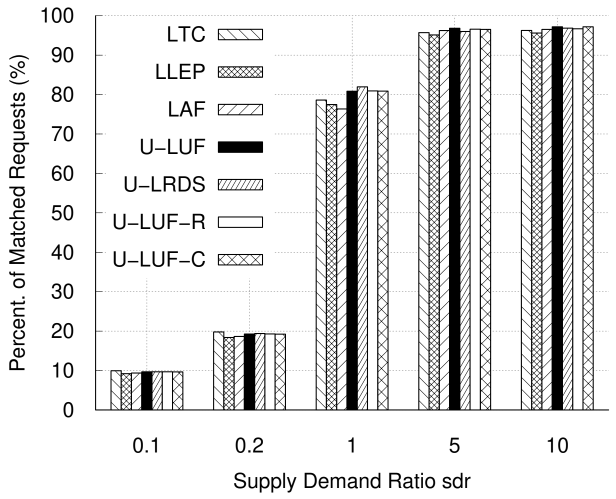
Supply-demand ratio sdr	No. of consumer requests	No. of service providers
0.1	121600	120
0.2	60800	120
1	12160	120
5	2430	120
10	1210	120



(a) STDEV of providers utilization

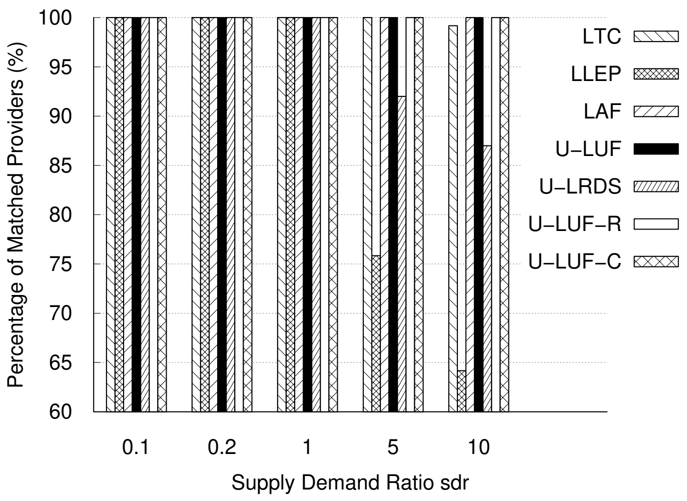


(b) Average of the lowest 10% provider utilizations

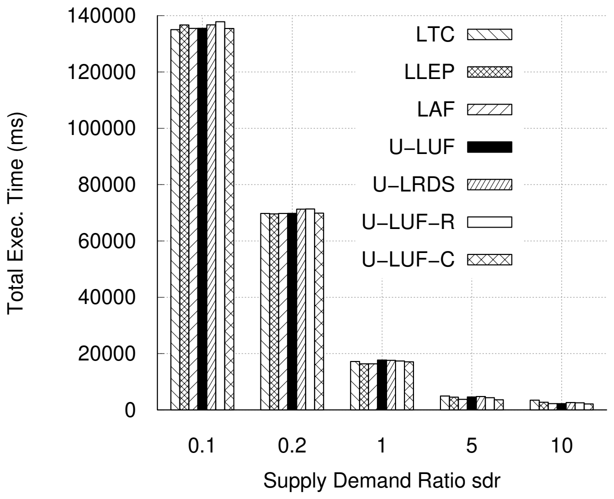


(c) Percentage of matched requests

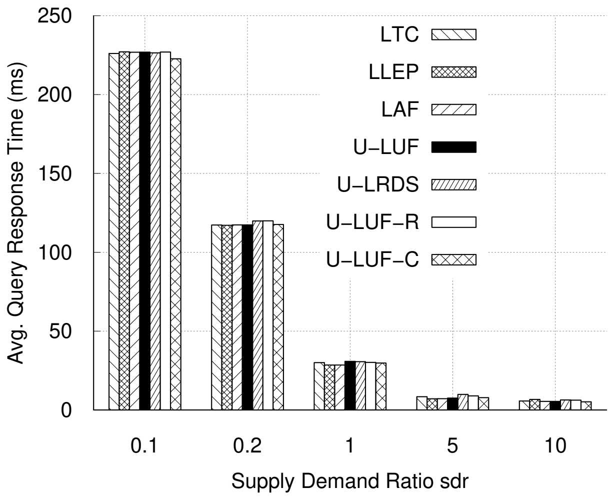
Figure 3.7: Effect of supply-demand ratio *sdr* (by fixing supply and varying demand).



(d) Percentage of matched service providers



(e) Total execution time



(f) Average query response time

Figure 3.7: Effect of supply-demand ratio *sdr* (by fixing supply and varying demand).

Effect of supply-demand ratio (*sdr*): To evaluate the effect of the supply-demand ratio on the performance of the different algorithms we ran our simulation for the *sdr* values of 0.1 (demand greatly exceeds supply), 0.2 (supply $<$demand), 1 (supply $\approx$ demand i.e., balanced), 5 (supply $>$demand) and 10 (supply greatly exceeds demand). For each value, the generated supply was fixed and the number of generated requests varied as shown in Table 3.9.

Standard deviation of providers utilization: Figure 3.7a shows the effect of *sdr* on the standard deviation of the providers utilization. We note that the LCF heuristic results were consistently dominated by LAF and were therefore eliminated from the plots to improve readability. Generally, a lower standard deviation indicates a better utilization balance. As seen in the figure, our proposed LUF-based algorithms (i.e., U-LUF, U-LUF-R and U-LUF-C) always achieve a more balanced provider utilization. More specifically, we can see that U-LUF achieves within 0.08 to 0.78 of the standard deviation of our prior LAF heuristic, and within 0.04 to 0.85 of the standard deviation of LTC. The gap between U-LUF and other related/prior work increased at $sdr \geq 1$ since balancing provider utilization with limited demand becomes more challenging.

We also observe that when demand exceeds supply, U-LRDS also achieves a lower standard deviation than our prior LAF heuristic as well as other related work (i.e. 0.79 to 0.88 of the standard deviation of LAF). However, as demand decreases at larger *sdr* values, the performance of U-LRDS declines since it keeps assigning those providers with smaller demand-supply ratios regardless of their current utilization, while leaving other providers underutilized. At the same time, we can also observe that using U-LRDS as a tie-breaker for U-LUF (i.e., in U-LUF-R) results in the least standard deviation of 0.0056 at $sdr = 10$ since at this value many providers remain unutilized. In this case, favoring providers with larger supply, by using U-LRDS as a tie-breaker, results in smaller utilization values (since supply is in the denominator of the utilization formula) which makes the utilization of those favored providers closer to the zero utilization of the many unutilized providers.

Average of the lowest 10% provider utilizations: Figure 3.7b shows the effect of *sdr* on the average utilization of the lowest 10% of providers. A larger average is desirable as it indicates better performance for the least utilized providers. We observe that when demand exceeds supply (i.e., $sdr = 0.1$ and $sdr = 0.2$, all proposed algorithms have a high and very comparable performance to related work (i.e., LTC) since the large demand results in most providers being almost fully utilized. However, as *sdr* increases and less demand arrives, our proposed LUF-based algorithms greatly outperform both prior and related work. In the case of balanced supply and demand, all LUF-based algorithms achieved 28% higher average utilization than our prior LAF (which performed best among related work). Also when supply exceeds demand, LUF-based algorithms achieved up to 40% higher average. This is due to their ability to balance the utilization of all providers by being aware of their service rates (i.e., supply). We also observe, however, that as *sdr* increases, U-LRDS is outperformed by our prior LAF heuristic since it again favors the same set of providers with low demand-supply ratios regardless of their current utilization and thus has a smaller improvement for the utilization of the least utilized providers.

Percentage of matched requests: Figure 3.7c shows the effect of *sdr* on the percentage of matched requests. At small *sdr* values, most of the requests could not be matched since the demand was much larger than the available supply. As *sdr* increases, so did the percentage of matched requests. Although one may intuitively think that a provider-centric approach may reduce the business volume unlike other broker-centric (e.g. LLEP) and consumer-centric heuristics (e.g. LTC), this figure shows that when supply and demand were balanced (sdr=1) and also when supply exceeded demand (i.e., *sdr*= 5, *sdr*=10), all our proposed algorithms have slightly outperformed related and prior work heuristics. For instance, at *sdr*=1, the percentage of matched requests by U-LRDS was 3.3% higher than LTC (the highest among related work) while our LUF-based algorithms (i.e. U-LUF, U-LUF-R and U-LUF-C) were 2.3% higher. This separation decreases as supply exceeds demand (at *sdr*=5, *sdr*=10) where U-LUF was 1% higher than LTC. The use of the consumer prioritization phase and accounting for the current utilization of service providers (e.g., in U-LUF, U-LUF-C) and/or the gap between the recent demand and supply of each provider (e.g., in U-LRDS, U-LUF-R) allowed a better distribution of the available demand and a more efficient use of the available supply. Although in theory, all requests can be matched at *sdr*=1, the broker might not actually be able to match all requests if consumers' maximum acceptable travel times and wait times are short.

We also note that U-LUF-C slightly outperformed U-LUF at *sdr*=1 (less than 1%)

due to the use of the conflict-aware prioritization strategy. However, this difference disappeared at higher *sdr* values since as supply exceeds demand, a smaller number of consumer requests are being matched simultaneously, leading to fewer conflicts among the candidate propositions, rendering the effect of conflict-aware prioritization less significant.

Percentage of matched providers: Figure 3.7d shows the effect of *sdr* on the percentage of matched providers. When demand exceeds supply or both are balanced, all the compared algorithms can match all the available service providers. However, as supply exceeds demand (e.g., *sdr*=10), only provider-centric algorithms (LAF, U-LUF, U-LUF-R and U-LUF-C) are able to match 100% of the service providers. These are followed by LTC matching 99% of the providers (at sdr=10) and then LLEP matching only up to 76%. We can also observe that U-LRDS is the only exception among provider-centric algorithms that did not engage 100% of the providers at large values of *sdr*. The reason is that U-LRDS will always favor those service providers which have a large gap between their supply and demand regardless of their actual utilization (unlike U-LUF), and since supply exceeds demand (i.e., at large *sdr*), the demand is not enough to cover other service providers.

Total execution time and average query response time: Figure 3.7e shows the total execution time as *sdr* increases. Since the number of generated requests decreases with the increase in *sdr* (as indicated in Table 3.9), the total execution time also decreases. As shown in the figure, all algorithms had very similar execution times with less than 140 sec at $sdr = 0.1$. Figure 3.7f shows the effect on the average query response time per consumer. Again, for a larger number of requests (i.e., smaller *sdr*), at each clock, a large group of requests is being processed simultaneously, which results in a longer response time per request. As the number of requests decreases at larger *sdr* values, the response time decreases since only a small number of requests is handled simultaneously. We note that at *sdr*=0.1 (i.e. an average of 200 simultaneous requests), all algorithms achieve an average query response time of less than 250 ms.

To further illustrate the scalability of our approach, we also evaluated its performance for different queue sizes (i.e. different number of consumer requests arriving simultaneously). Our results have shown that our proposed algorithms can scale up to a large number of simultaneous requests. For 1000 simultaneous requests, our proposed algorithms resulted in an average query response time of 1 sec. For 10,000 simultaneous requests, the average query response time was 10 sec. This is a very large number of simultaneous requests when compared to actual numbers of ride requests released by a major ride-hailing company [74] showing less than 170 requests per minute (i.e. 10,000 requests in a whole hour) during the busiest hours in New York City.

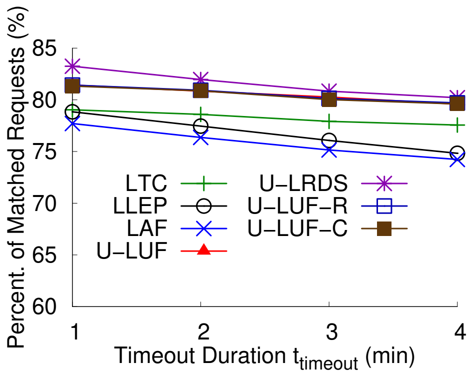


Figure 3.8: Effect of $t_{timeout}$ (best viewed in color)

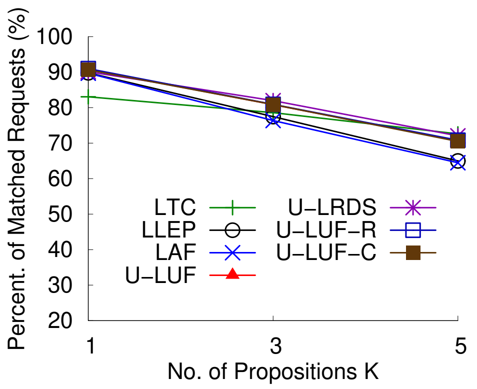


Figure 3.9: Effect of K (best viewed in color)

Effect of timeout duration ($t_{timeout}$): Figure 3.8 shows the effect of the timeout duration on the percentage of matched requests. We observe that as $t_{timeout}$ increases, the percentage of matched requests decreases. This is because the service time for each consumer matched to K proposed providers remain reserved until the consumer accepts one of the propositions. The time taken to accept a proposition can extend up to $t_{timeout}$; hence increasing $t_{timeout}$ increases the time it takes for the reserved capacity to be assigned to another consumer, thus reducing the number of matched requests. We can also observe that U-LRDS consistently matches a higher percentage of matched consumers (i.e. 2.6% to 4.2% higher than LTC which was the best performing among related work), followed by the LUF-based algorithms (2% higher than LTC). These results accord with the results from Figure 3.7c when supply and demand are balanced (i.e., sdr=1).

Effect of number of required propositions (K): Figure 3.9 shows that increasing the number of required propositions K decreases the percentage of matched requests. A larger value of K implies that each request has to be matched to a larger number of service providers, with service times being reserved at all matched providers until a proposition is accepted or timeout occurs. Thus, increasing K reduces the number of requests that can be matched since more service capacity is being reserved per request. At $K \leq 3$, our proposed algorithms outperformed all related work heuristics (1% to 3.3% higher). This separation does not appear at K=5 as the heuristics start to perform similar to LTC since the larger number of propositions result in a demand that is significantly higher than the available supply, which also accords with results from Figure 3.7c when demand exceeds supply.

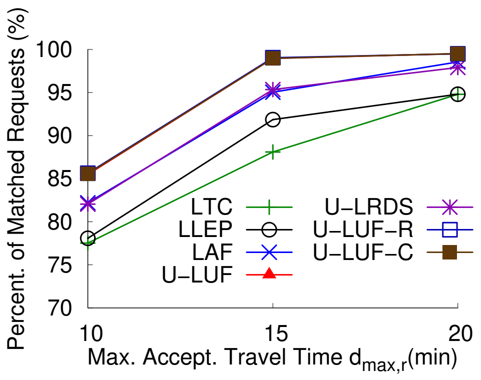


Figure 3.10: Effect of max. travel time $d_{max,r}$ (best viewed in color)

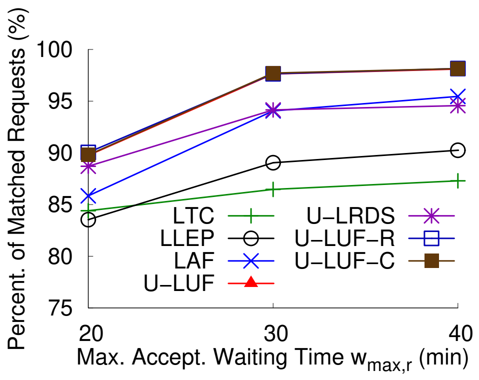


Figure 3.11: Effect of max. waiting time $w_{max,r}$ (best viewed in color)

Effect of maximum acceptable travel time ($d_{max,r}$): In this experiment, all consumers were assigned an equal max acceptable waiting time of 30 min, and their maximum acceptable travel time $d_{max,r}$ was varied as shown in Figure 3.10. Results showed that increasing $d_{max,r}$ increases the percentage of matched requests. With greater travel time constraints, there are more service providers available for consumers to be matched to. For instance, the percentage of matched requests for U-LUF increased from 85.5% at $d_{max,r} = 10$ min to 99.5% at $d_{max,r} = 20$ min. Our LUF-based algorithms still consistently outperformed related work (i.e. up to 4% higher than LAF at $d_{max,r} = 15$ min). We can also notice that the rate of increase in the percentage of matched requests slows for larger $d_{max,r}$ values since the maximum acceptable waiting time places another constraint on the number of service providers that can be matched to a given request.

Effect of maximum acceptable waiting time($w_{max,r}$): In this experiment, consumers were assigned a maximum acceptable travel time between $minD_{max,r} = 10$ min and $maxD_{max,r} = 20$ min. The maximum acceptable waiting time $w_{max,r}$ was varied as shown in Figure 3.11. We can observe that increasing the maximum acceptable waiting time value increases the percentage of matched requests since more candidate service providers become available for consumers to be matched to. For instance, the percentage of matched requests for U-LUF increased from 89.7% at $w_{max,r} = 20$ min to 98% at $w_{max,r} = 40$ min. The rate slows for larger values of $w_{max,r}$ since the maximum acceptable travel time remains a constraint on request matching. We again note that our LUF-based algorithms still consistently outperformed related work (i.e. 2.6% to 4% higher than LAF at $w_{max,r} = 40$ min and 20 min respectively).

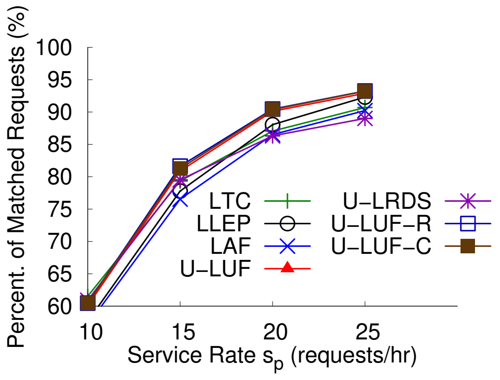


Figure 3.12: Effect of service rate s_p (best viewed in color)

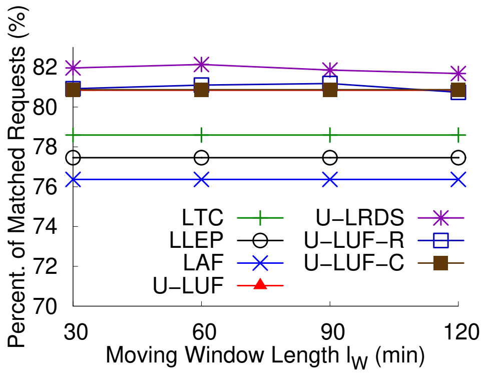


Figure 3.13: Effect of moving window length l_W (best viewed in color)

Effect of providers service rates (s_p): In this experiment, the total number of generated requests (i.e., total demand) was fixed at 18,000 requests, and the minimum and maximum provider service rate were varied from 10 to 25 requests/hr. As shown in Figure 3.12, higher service rates means more requests can be matched per unit time and thus the percentage of matched requests increases. For instance, the percentage of matched requests for U-LUF increased from 60% at $s_p = 10$ requests/hr to 93% at $s_p = 25$ requests/hr. We also note that all LUF-based algorithms consistently outperformed related work (e.g. U-LUF-R and U-LUF-C were up to 4.2% higher in the percentage of matched requests compared to related work).

Effect of moving window length (l_W): Figure 3.13 shows the effect of increasing the moving window length l_W used by U-LRDS and U-LUF-R. As l_W increases from 30 min to 60 min, we notice a very slight increase in the percentage of matched requests (less than 0.2%) since a relatively larger window length may allow a more accurate observation for the demand on each service provider. However, as l_W increases further, we notice that the percentage of matched requests decreases back since a very large window length may not capture the recent demand patterns for service providers and thus may not be helpful for predicting what the demand will be like in the near future.

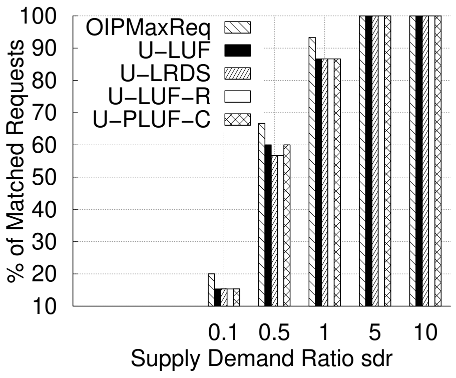
We also evaluated the effect of increasing the cell length l_G of the grid index used in computing the location entropy of consumers in the LLEP strategy. Our results showed that changing l_G had no significant impact on the percentage of matched requests by LLEP, and LLEP's ranking among other algorithms remained the same for the different values.

Evaluating Solution Quality Compared to an Offline Optimized Solution:

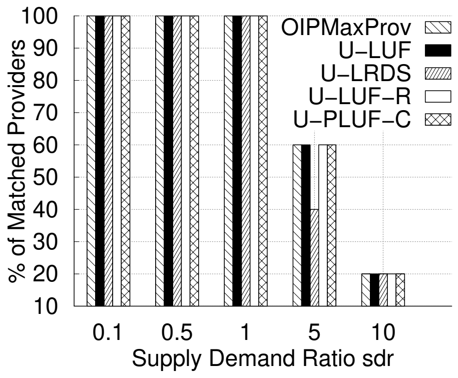
We compared the solution quality of our proposed algorithms to the solution obtained from our three offline integer program formulations, namely, OIPMaxReq, OIPMax-Prov, and OIPMinSTD, where all requests are assumed to be known in advance. Due to the limited scalability of integer programming approaches given the large number of variables and constraints, we compared our solution quality to an offline optimized solution using only a randomly selected subset of 5 providers and 60 time steps (i.e., consumers arrive during a 30 min interval and have a maximum acceptable waiting time of 30 min). The solutions were compared in terms of the percentage of matched requests, the percentage of matched providers and the standard deviation of providers utilization, at different values of the supply-demand ratio sdr. For each value, the generated supply was fixed and the number of generated requests varied as shown in Table 3.10. The default parameter values were set as follows: $minS$ = 4 requests/hr, $maxS$ = 8 requests/hr, $minD_{max,r}$ = 20 min and $maxD_{max,r}$ = 30 min, $minW_{max,r}$ = 20 min, $maxW_{max,r}$ = 30 min, l_W = 30 min, and l_G = 2000 m, unless stated otherwise. For this experiment, we also set $K = 1$ and $t_{timeout} = 0$ min since offline approaches do not simulate a timeout interval and thus consumers are not allowed to select among multiple propositions. Consumers' travel time and wait time constraints were set to relatively large values due to the smaller number of service providers simulated in this case.

Table 3.10: Synthetic dataset details for experimental goal 2. Total supply is fixed across all *sdr* values.

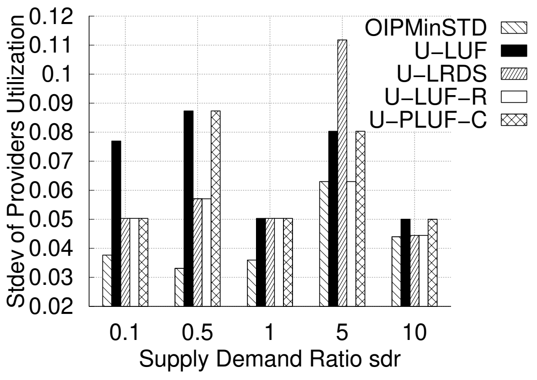
Supply-demand ratio *sdr*	No. of consumer requests	No. of service providers
0.1	150	5
0.5	30	5
1	15	5
5	3	5
10	1	5



(a) Percentage of matched requests



(b) Percentage of matched service providers



(c) STDEV of providers utilization

Figure 3.14: Effect of supply-demand ratio *sdr* (by fixing supply and varying demand). The number of total consumer requests (i.e., demand) corresponding to each *sdr* value is shown in Table 3.10.

Percentage of matched requests: Figure 3.14a shows the percentage of matched requests as the supply-demand ratio *sdr* increases. When demand exceeds supply, our proposed algorithms achieved between 4.7% to 10% lower than the offline upper bound on the matching size achieved by OIPMaxReq (which is 20% and 66.7% for $sdr = 0.1$ and $sdr = 0.5$ respectively). For balanced supply and demand, proposed algorithms are 6.7% lower than the percentage of matched requests of OIPMaxReq. When supply exceeds demand, this gap decreases and our proposed algorithms achieved the offline upper bound by matching all consumer requests.

Percentage of matched providers: Figure 3.14b shows the percentage of matched providers at different *sdr* values. It can be seen that for all values of *sdr*, all algorithms (except U-LRDS at $sdr = 5$) achieve the upper bound obtained from OIPMaxProv. When supply exceeds demand (i.e., sdr=5), U-LRDS tends to overload the same set of service providers which have a large supply-demand ratio; since demand is not enough to cover all providers in this case, U-LRDS returns an overall smaller percentage of matched providers (i.e. 20% lower than the offline upper bound).

Standard deviation of provider utilization: Figure 3.14c shows the standard deviation (stdev) of provider utilization at different *sdr* values. When demand exceeds supply, U-LRDS and U-LUF-R achieved the lowest standard deviation (less than 1.7x the offline stdev of OIPMinSTD, i.e., 2.9x the offline variance). For balanced supply and demand, the gap is reduced and all algorithms achieved 1.39x the optimal stdev (i.e., twice the offline variance). As supply exceeds demand, U-LUF-R achieved the same minimum offline stdev. Since U-LRDS favors service providers with higher supply, using it as a tie breaker with U-LUF reduces the stdev as it results in a small increase in utilization and hence the utilization of the favored providers is closer to that of unutilized providers resulting in an overall smaller utilization variance.

3.5.4 Summary of Experimental Results

Our experimental results show that our proposed approach, particularly with the Least Utilized First heuristic, can achieve the lowest variance in provider utilization while matching all the available providers even when supply highly exceeds the demand. Results also show that the average utilization of the lowest 10% of providers in our proposed approach is 28% higher than our prior work [56] in the case of balanced supply-demand, and up to 40% higher when supply exceeds the demand. Using the Least Recent Demand-Supply Ratio First heuristic as a tie-breaker resulted in the least utilization variance when supply greatly exceeds demand. In addition, our approach is also able to achieve a slightly larger matching size compared to our prior and related work when supply exceeds demand and when both supply and demand are balanced.

We also compared the results of our approach to the offline solution using a set of five service providers and supply-demand ratios varying from 0.1 to 10. Our results show that our approach can achieve within 4.7% to 10% of the offline upper bound of the matching size while engaging the same number of service providers. When supply exceeds demand, our approach achieved the same percentage of matched requests as the offline solution. Finally, our approach also achieved the same variance of provider utilization as the offline solution when supply exceeds demand, and less than three times that variance for other supply-demand scenarios.

As shown from the above results, our proposed approach provides a more balanced provider utilization while meeting all the other conflicting matching requirements of a broker to avoid the inequitable distribution of opportunities to meet demand among service providers. Additionally our approach also ensures that this balanced provider utilization does not come at the expense of reducing the business volume since the percentage of matched requests was even higher particularly when supply is greater than or equal to demand. Hence, the advantage of this approach can be most leveraged for situations where demand and supply are balanced and also when supply exceeds the available demand, since at these times keeping all service providers engaged becomes more challenging and thus the probability of providers leaving or switching to other service brokers increases.

3.6 Conclusion

This work explored the problem of On-demand Spatial Service Propositions where an on-demand spatial service broker aims to maximize the number of matched consumer requests while keeping the provider eco-system functioning by engaging a large number of service providers and balancing their utilization. We proposed ULAMA, a Utilization-Aware Matching Approach that employs novel provider-supply aware heuristics, namely, Least Utilized First and Least Recent Demand-Supply Ratio First, for minimizing the variance in provider utilization, and a consumer-priority based greedy matching algorithm that uses these heuristics while prioritizing consumers during matching to maximize the number of matched requests. Our experimental evaluation showed that our proposed approach outperformed the related work on multiple performance measures including a smaller variance in provider utilization, a higher average utilization for the lowest 10% utilized providers, and a larger number of matched requests when supply is greater than or equal to the available demand.

3.7 Future Work

Reducing inequity in cities is recognized as one of the most pressing challenges in today's world for building sustainable cities and communities according to the United Nation's Sustainable Development Goals [75]. It is also recognized by many US and international agencies as important for health, economic development and social stability [76, 77], and cities are increasingly interested in equity as a core attribute of sustainability [78]. For instance, as part of the US Department of Transportation smart city challenge, the city of Columbus proposed a plan for connecting under-served neighborhoods to jobs and services which could lower child mortality rates in these neighborhoods [79]. Ride-hailing services are currently cooperating with cities to hire workers from struggling neighborhoods [80].

On-demand spatial brokers may potentially help reduce inequity in cities by accounting for income inequality across different neighborhoods. In our future work, we plan to investigate how the proposed approach could be employed to reduce inequity in cities possibly by giving preference to service providers from economically disadvantaged areas among otherwise similarly ranked providers. In addition, we plan to extend the OSSP problem to allow mobile service providers and incorporate consumer ratings into the matching process. Aside from balancing providers utilization for increasing the participation of service providers, some studies [81, 82, 83] have also considered the problem of self-regulating the imbalance in supply and demand systems through the adjustment of pricing mechanisms. In the future, we plan to model the variation in profit across different types of consumer requests and explore the effect of pricing models on the incentives of service providers. We hope to explore how the broker can adapt to spatio-temporal variations in market conditions and the evolving behavior of consumers and service providers.

Chapter 4

pkgRendezvous: A Flexible On-Demand Pickup and Delivery Broker for Moving Consumers

4.1 Introduction

In today's world, the on-demand economy has emerged as an unconventional digital marketplace that leverages technology for connecting consumers with service providers, providing them with fast and convenient access to goods and services. With high demand from users, the on-demand economy will continue to revolutionize many traditional industries including transportation, healthcare, and many professional services. Besides the most famous on-demand ride-hailing services, on-demand home delivery services for online products are also experiencing a tremendous growth in recent years. Examples include same day shopping cart delivery services (e.g. Amazon Prime Now [84]), grocery delivery (e.g. InstaCart [85], AmazonFresh [86]), and package delivery (e.g. Roadie [87]).

A common challenge that faces delivery services is possible package theft, as well as package damage. In 2016, 11 million U.S. homeowners experienced a stolen package [88, 89] with theft incidents especially high during holiday seasons when people expect more packages. This has led companies such as Amazon and Walmart to explore other delivery options such as using smart locks that allow delivery personnel to leave packages inside the home [90, 91], a solution that still raises other privacy concerns. The risk of package theft is also an inconvenience that forces many people to stay at home or leave work early on the day they are expecting a package [89]. Moreover, packages arriving at empty houses may need to be rescheduled which increases consumers wait time and delivery company losses.

In this work, we investigate a flexible on-demand pickup and delivery broker that allows moving consumers to receive packages at a place and time where they are available during the day. Consumers submit their service requests to the broker, consisting of their specific order and a set of candidate rendezvous (i.e., delivery) locations with the corresponding time intervals during which the consumer is available at each location. Given a set of dynamically arriving consumer service requests, a set of service locations representing possible pickup locations, and a set of delivery vehicles, the broker matches each request to a delivery vehicle, schedules the selected pickup and delivery locations for that request into the assigned vehicle's schedule and returns the selected rendezvous/delivery location to the consumer. The goal of the broker is to maximize the number of matched requests while also minimizing the travel and wait times for the delivery vehicles.

Challenges: Designing a flexible on-demand pickup and delivery broker for moving consumers poses several challenges. First, the broker needs to satisfy multiple conflicting objectives including maximizing the number of matched requests, and minimizing vehicle travel and wait times, while also guaranteeing a good user experience by minimizing the response time to consumers. Second, finding the best schedule for a new pickup and delivery location while also minimizing vehicles' travel and wait times is computationally expensive due to the exponential number of possible schedules and the large number of vehicles. Third, there is an increased computational complexity due to the multiple possible pickup (i.e., service locations) and delivery locations and times for each consumer request.

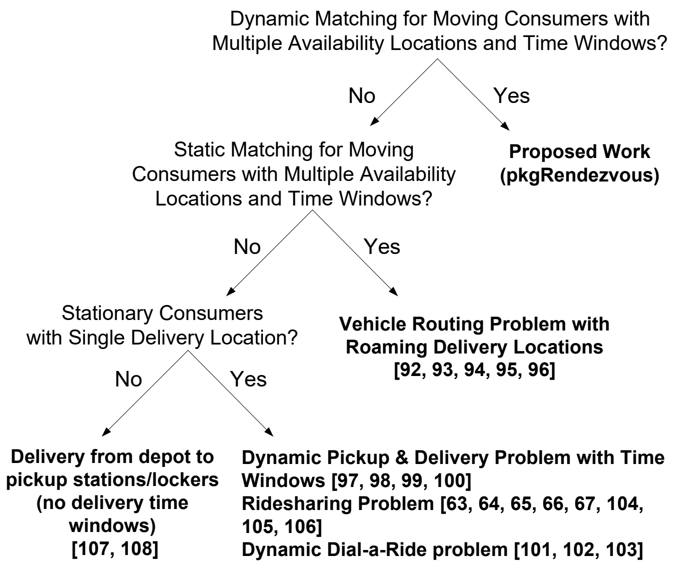

Dynamic Matching for Moving Consumers with Multiple Availability Locations and Time Windows?
No
Yes
Static Matching for Moving Consumers with Multiple Availability Locations and Time Windows?
Proposed Work (pkgRendezvous)
No
Yes
Stationary Consumers with Single Delivery Location?
Vehicle Routing Problem with Roaming Delivery Locations [92, 93, 94, 95, 96]
No
Yes
Delivery from depot to pickup stations/lockers (no delivery time windows) [107, 108]
Dynamic Pickup & Delivery Problem with Time Windows [97, 98, 99, 100]
Ridesharing Problem [63, 64, 65, 66, 67, 104, 105, 106]
Dynamic Dial-a-Ride problem [101, 102, 103]

Figure 4.1: Classification of related work

Related work: Figure 4.1 shows a classification of the related work of the flexible on-demand pickup and delivery problem for moving consumers. The closest category of related work is the vehicle routing problem with roaming delivery locations [92, 93, 94, 95, 96]. In this problem, the input is a fleet of m vehicles with known capacities, and a set of consumer requests each associated with multiple candidate delivery locations at which the consumer will be available. Each delivery location is associated with a time window representing the earliest and latest possible times for delivery at this location. Vehicles are allowed to arrive at a delivery location before its earliest associated time, but not after the latest time. The goal is to find a set of delivery routes and delivery times such that every customer receives a single delivery by a single vehicle while minimizing the total routing cost and without violating the vehicle capacity constraints. However, these works assume a static setting where all requests are known in advance. Also, a feasible solution must deliver all available requests. This setting is different from the dynamic setting of an on-demand pickup and delivery broker where requests are served dynamically upon arrival. Additionally, most of these works assume a single depot for the pickup of all requests. All delivery routes start and end at the depot.

Another category of related work includes the literature on the dynamic pickup and delivery problem with time windows (PDPTW), the dynamic Dial-a-Ride problem

(DARP), and the ridesharing problem. In these problems, stationary consumers submit service requests such that each request is associated with a single delivery location or a single destination. In the PDPTW [97, 98, 99, 100], the input is similar to the vehicle routing problem where a fleet of m vehicles with known capacities are responsible for the pickup and delivery of goods. However, consumer requests arrive dynamically, and each request is associated with a pickup node and a destination node. The pickup and destination nodes are also associated with a time window representing the earliest and latest possible times for pickup and delivery at the node respectively. The goal is to schedule the pickup and delivery for each request in a way that minimizes the total routing cost. The DARP [101, 102, 103] and ridesharing [104, 105, 63, 64, 65, 66, 67, 106] problems differ from PDPTW in the fact that the consumers themselves are picked up and dropped off instead of goods. Therefore, additional constraints are added to minimize consumer inconvenience such as limiting the total length of the ride and consumers' wait time before pickup. However, the three problems in this category assume a single pickup and a single delivery location per request.

In addition, few studies have addressed the problem of delivery from a single depot to multiple possible parcel lockers [107, 108]. In these works delivery locations do not represent consumer availability locations and thus these delivery locations are not associated with time windows.

In contrast, our proposed work focuses on dynamically arriving consumer requests that are associated with multiple possible rendezvous (i.e., delivery) locations and time intervals, while also allowing multiple possible pickup locations depending on the type of service requested.

Contributions: This work makes the following contributions: (1) We formally define the problem of Flexible On-demand Pickup and Delivery for Moving Consumers (FLOPDMC). (2) We propose a matching and scheduling algorithm, *pkgRendezvous*, for an on-demand pickup and delivery broker. Our proposed approach relies on two pruning filters: (a) an early termination condition for reducing the scheduling time, and (b) an all-insertions cost lower bound pruning filter with a grid-based lookup table for pruning high cost candidate pickup and delivery location pairs. (3) We analytically prove the correctness of the proposed pruning filters utilized by our approach. (4) We provide an experimental evaluation using synthetic data with real-world characteristics and show that our proposed approach yields significant computational savings compared to the baseline method.

Scope and Outline: This work focuses on solving the FLOPDMC problem where consumer requests are picked up and delivered at one of the consumer's candidate delivery locations. Information about the candidate delivery locations and corresponding time intervals may be acquired from consumer calendars, mined from the consumer's historical GPS trajectories, or directly specified by the user. However, this work assumes that this information is input to the problem, and therefore does not address the problem of identifying the candidate delivery locations and time intervals for each consumer request.

The rest of the chapter is organized as follows: Section 4.2 presents the basic concepts followed by a formal problem definition and example for the flexible on-demand pickup and delivery problem. Section 4.3 presents our proposed approach. The experimental evaluation is covered in Section 4.4. Finally, Section 4.5 concludes the chapter and discusses future work.

4.2 Basic Concepts and Problem Statement

In this section we define some basic concepts and formally define the FLexible On-demand Pickup and Delivery Problem for Moving Consumers (FLOPDMC).

4.2.1 Basic Concepts

Definition 20. *A **Spatial Network** $G = (N, E)$ consists of a node set N representing road intersections and an edge set E representing road segments. Each node $n \in N$ is associated with a pair of real numbers (latitude, longitude) representing the spatial location of the node. Edge set E is a subset of the cross product $N \times N$. Each element $e = (u, v) \in E$ is an edge that joins node u to node v, and is associated with a scalar value representing the travel time cost along that edge.*

Definition 21. *A **Service Location** $p = (id_p, loc_p, sr_p)$ is a service provider or store that is registered in the on-demand broker system and acts as a pickup location. Each service location is associated with an id id_p, a node representing its location in the* *spatial network loc_p, and a service rate sr_p representing the number of requests that can be served/picked up per hour.*

Definition 22. *A* ***Consumer Request*** *represents a request r = (cid_r, a_r, $type_r$, I_r) from a consumer on the spatial network. Each request is associated with a consumer id cid_r, the arrival time of the request a_r, the requested order or service type $type_r$, and a consumer itinerary I_r. The consumer itinerary I_r consists of a set of triples $< d, t_{start}, t_{end} >$ where d is a node representing a candidate delivery (i.e., rendezvous) location in the spatial network, and t_{start} and t_{end} represent the start and end times of the corresponding time interval during which consumer cid_r will be available and staying at location d during the day.*

Definition 23. *A* ***Delivery Vehicle*** *is a vehicle that is available for picking up and delivering consumers requests. Each vehicle v = (id_v , loc_v, s_v, r_v) is associated with a vehicle id id_v, a node representing the current location of the vehicle in the spatial network loc_v, a schedule s_v which consists of a temporally ordered sequence of nodes representing pickup and delivery locations of consumer requests matched to the vehicle, and a route r_v representing the minimum cost path (i.e., path with shortest travel time) along the spatial network starting at the vehicles' current location loc_v and visiting each node in s_v in order.*

Definition 24. ***Vehicle Total Travel time*** $t(v)$***:*** *Given a spatial network G, the total travel time of vehicle v is the sum of travel times along all edges on the vehicle's route r_v.*

Definition 25. ***Vehicle Total Waiting Time*** $w(v)$***:*** *The total waiting time for vehicle v is the sum of the waiting time at each pickup and delivery node visited in the vehicle's schedule s_v. Given a consumer request r, and a vehicle matched to pick up and deliver that request from nodes p and d respectively, the wait time at the pickup node p is equal to the time between the arrival of vehicle v at node p and the earliest possible pickup time for request r at p. Similarly, the wait time at the delivery node d is the time between the arrival of vehicle v at node d and t_{start} which represents the start of the interval during which the consumer will be available for delivery at d.*

4.2.2 Problem Definition

The FLOPDMC problem can be formulated as follows:
Given:

1. A spatial network G.
2. A set P of service locations in G.
3. A set V of delivery vehicles in G.
4. A set R of consumer requests arriving dynamically.
5. Pickup and Rendezvous time durations. (t_{pickup} and $t_{rendezvous}$ respectively).

Find: A proposed vehicle and delivery location for each request $r\ in\ R$
Objectives:

- Broker-centric: Maximize the number of matched requests
- Provider-centric: Minimize the vehicles' total travel and waiting times

Constraints:

1. For each service location $p \in P$, the number of matched requests per hour $\leq sr_p$.
2. Each vehicle must visit each pickup location on its schedule for a duration $\geq t_{pickup}$, such that the pickup duration occurs after the earliest available pickup time for the request at that location.
3. Each vehicle must visit each delivery location on its schedule for a duration $\geq t_{rendezvous}$, such that the rendezvous duration occurs during the consumer's availability interval at this location [t_{start},t_{end}] as specified by the consumer's itinerary.
4. Each request is served by at most one vehicle.
5. When a request is matched to a vehicle, the request's pickup and delivery locations must both be visited by the same vehicle, and the pickup node has to occur before the delivery node on the vehicle's schedule.
6. Consumer itineraries may contain only candidate delivery locations where consumers are staying for a duration ≥ 1 hour.

In this problem, for each consumer request, the broker matches the request to a delivery vehicle, and selects one of the candidate service locations for the request pickup, and one of the candidate delivery locations from the consumer's itinerary, while satisfying the service location service rate and pickup constraints (as shown in constraints 1 and 2 in the problem definition), and the consumer's availability times (constraint 3). For a matched request, the broker schedules the selected pickup and delivery locations into the vehicle's current schedule while ensuring that the pickup location is scheduled first on the vehicle's route (constraints 4 and 5). A vehicle may arrive at a delivery location before the start of the consumer's availability interval at that location, but may only leave after satisfying the rendezvous duration during that interval (constraint 3). Similarly, a vehicle may arrive at a pickup node before the start of the pickup time. The sixth constraint is only added for practicality purposes since very short delivery windows may easily be disrupted due to traffic and road conditions; however, a different cutoff value may also be chosen. If no vehicle satisfying the above constraints is found, the consumer request is not matched.

The objective of the broker is to maximize the number of matched requests to maximize profits. In addition, the broker aims to minimize the total travel time of the delivery vehicles to minimize fuel consumption and other vehicle associated costs. Finally, the broker also aims to minimize the vehicles' waiting time to avoid idle times on the vehicles' schedules and reduce the schedule total length. The objectives are assumed to be in a lexicographic order of the number of matched requests, total travel time, and total waiting time, such that a solution with a larger number of matched requests is better than another solution with a smaller number of matched requests but a shorter travel time. In the case of autonomous delivery vehicles, minimizing the waiting time becomes even less important since no human drivers are involved.

4.2.3 Problem Example

Figure 4.2 shows an example input for the FLOPDMC problem. The weights shown on the edges of the road network refer to the travel time (in minutes) along the edge. Four service locations are shown on the spatial network: two bookstores and two grocery stores. For simplicity, we assume a single delivery vehicle located at node B. We also assume that the service locations have high service rates such that the requested orders will be ready for pickup by the time the vehicle arrives at the service location. Let t_{pickup} $= t_{rendezvous} =$ 10 min. Table 4.1 shows two consumer requests that are submitted to the broker at 5:00 pm. Consumer C_1 requests a book delivery, while C_2 requests the delivery of a grocery shopping cart. Each consumer specified two candidate delivery locations (i.e., office and gym; and home and mall) as well as the time interval during which he/she will be available at each location.

Table 4.2 shows two possible solutions for this problem. The schedule column shows the sequence of nodes that will be visited for the pickup and delivery of each of the consumer requests, together with the time at which the delivery vehicle will arrive at and leave from the node. The last two columns show the total travel time and total waiting time for each solution. As shown in the table, both solutions successfully match the two requests, but the second solution provides a shorter travel time and is thus preferred over Solution 1.

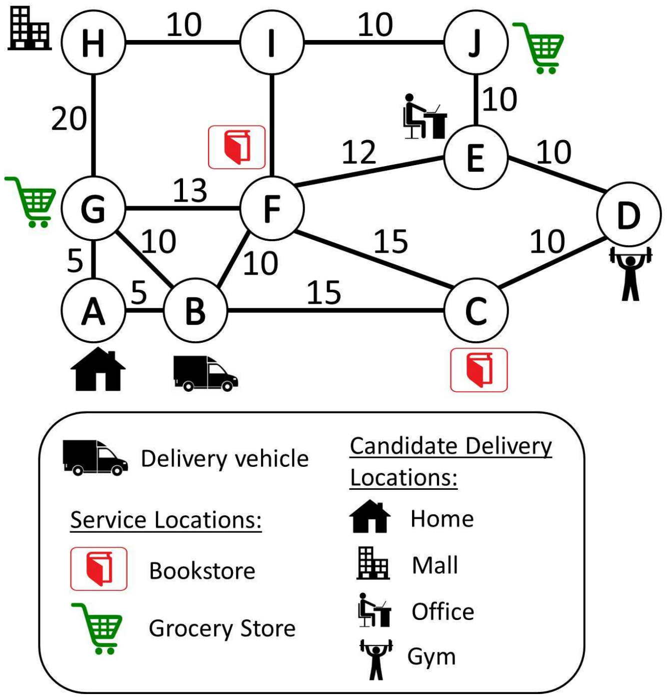


Figure 4.2: Example Spatial Network (Best viewed in color)

Table 4.1: Consumer requests input for problem in Figure 4.2

Consumer	Service type	Candidate Delivery Location	Delivery Node	t_{start}	t_{end}
C1	Book delivery	Office	E	5:00	6:30
		Gym	D	6:40	7:40
C2	Same day shopping cart delivery	Home	A	5:00	6:10
		Mall	H	6:40	7:40

Table 4.2: Two possible solutions for the problem in Figure 4.2 and Table 4.1

	Schedule				Total travel time	Total wait time
	Location	Node	Arrival Time	Leave Time		
Solution 1	Bookstore (C1 Pickup)	F	5:10	5:20	52 min	18 min
	Office (C1 Delivery)	E	5:32	5:42		
	Grocery Store (C2 Pickup)	J	5:52	6:02		
	Mall (C2 Delivery)	H	6:22	6:50		
Solution 2	Grocery Store (C2 Pickup)	G	5:10	5:20	45 min	25 min
	Home (C2 Delivery)	A	5:25	5:35		
	Bookstore (C1 Pickup)	C	5:55	6:05		
	Gym (C1 Delivery)	D	6:15	6:50		

4.3 Proposed Approach

We begin by describing a baseline method for solving the FLOPDMC problem. Then, we present our proposed *pkgRendezvous* approach which leverages pruning filters for minimizing the computational cost.

4.3.1 Baseline Method

Algorithm 7 presents a baseline method for solving the FLOPDMC problem. For each request, the algorithm identifies the candidate pickup and delivery locations. The cross product of the two candidate sets represents the set of all possible pickup and delivery location pairs that can be used to schedule the given request into one of the current vehicles schedules. The goal of the algorithm is to find the vehicle schedule that can satisfy the request with the minimum increase in total vehicle travel and wait times (i.e. minimum cost insertion schedule). To achieve this goal, the algorithm adapts a two-phase approach used in on-demand ridesharing [104, 105], namely a vehicle searching phase (line 5) followed by a vehicle scheduling phase (line 6). In the vehicle searching phase, the algorithm finds all candidate vehicles that can visit the pickup and delivery nodes before the end of the consumer's availability interval by leveraging a grid index of the vehicles. In the vehicle scheduling phase, for each candidate vehicle, the algorithm finds the best way to insert the pickup and delivery nodes within the vehicle's current schedule by choosing the min cost insertion way. Insertion into a vehicle's schedule has to satisfy the feasibility constraints as detailed in Section 4.3.1. The scheduling phase finally returns the schedule with the min cost insertion among all candidate vehicles. The searching and scheduling phases are repeated for every possible pickup and delivery pair, and the vehicle associated with the minimum scheduling cost is assigned to serve the given request. Details of the vehicle indexing and vehicle searching and scheduling phases are presented next.

Vehicle Indexing

To quickly identify the list of candidate vehicles for a given pickup and delivery node pair, a spatio-temporal grid index is built by partitioning the spatial network into grid cells [105]. For each grid cell g_i, the node that is closest to the geographical center of the cell is selected as the anchor node for that cell, denoted as c_i. Travel times between all node pairs in the spatial network are precomputed and stored in a matrix SP where each element $sp(i, j)$ denotes the travel time cost of the shortest travel time path from node i to node j in order to reduce the expensive shortest travel time path computations. Each grid cell g_i stores two lists: (a) a temporally ordered grid cell list $g_i.cellList$, which is a list of all other grid cells sorted in ascending order of the travel time from their anchor nodes to the anchor node c_i of g_i, and (b) a vehicle list $g_i.vecList$, which stores the IDs of all vehicles that are currently located in g_i or will enter g_i in the near future (i.e., within a 2 hour window) according to the vehicle's schedule. The timestamp t_{entry} at which the vehicle will arrive in g_i is also stored. This spatio-temporal index is updated whenever a vehicle leaves a grid cell or is scheduled to enter a grid cell. In addition, the timestamps t_{entry} for that vehicle in the grid cells at which the vehicle is indexed are also updated.

Algorithm 7 Baseline Method

Input: G: spatial network, P: Service locations, V: Delivery vehicles, r: consumer request, *index*: Vehicles grid Index, *clock*: current time

Output: Assigned vehicle v_{min} with updated minimum cost insertion schedule $s_{v,min}$

Algorithm:

```
1: $minCost \leftarrow \infty$
2: $candidPickup \leftarrow$ Find all service locations in $P$ providing $type_r$
3: $candidDelivery \leftarrow I_r$                                   ▷ triples $< d, t_{start}, t_{end} >$
4: for each pickup and delivery pair $(p_i, d_j, t_{j\_start}, t_{j\_end})$ in $candidPickup \times candidDelivery$ do
5:     $candidateVehicles \leftarrow$ Find all vehicles from $index$ that can reach $p_i$ and $d_j$ before their deadlines   ▷ vehicle searching phase
6:     $< v,\ s_v,\ cost\_s_v > \leftarrow$ Find min cost insertion schedule for $p_i$ and $d_j$ in $candidateVehicles$   ▷ vehicle scheduling phase
7:     if $cost\_s_v < minCost$ then
8:         $minCost \leftarrow cost\_s_v$
9:         $v_{min} \leftarrow v$
10:        $s_{v,min} \leftarrow s_v$
11: if $minCost \neq \infty$ then Assign request r to vehicle $v_{min}$ and update its schedule to $s_{v,min}$
12: else if P thenrint “request cannot be matched”
```

Vehicle Searching

In a ridesharing problem setting, pickup nodes can be associated with a latest possible pickup deadline based on the consumer’s specified maximum waiting time. In contrast, pickup nodes in the FLOPDMC problem are associated with earliest possible pickup times based on the earliest time at which the order is ready for pickup, but no explicit latest possible pickup time (i.e. deadline) is specified since, as long as a payment method is specified, service locations may hold the order at least until their location’s closing time. However, delivery nodes have deadlines based on the end of the consumer’s availability interval at that location. Hence, given a pickup and delivery node pair (p_i, d_j) with the associated availability interval [t_{start}, t_{end}])at d_j, the deadlines for reaching both p_i and d_j by the same vehicle v can be defined as follows:

$$d_j.deadline = t_{end} - t_{rendezvous} \tag{4.1}$$

$$p_i.deadline = d_j.deadline - sp(p_i, d_j), \tag{4.2}$$

where $sp(p_i, d_j)$ is the travel time along the shortest travel time path from p_i to d_j.

Now, to identify the set of candidate vehicles for scheduling the pair (p_i, d_j), the searching phase starts by retrieving the grid cell g_p at which node p_i is located. Then, the list $g_i.cellList$ is scanned in ascending order to retrieve all grid cells from which g_p is reachable before $p_i.deadline$. Each grid cell g_l that satisfies the following condition will be retrieved:

$$clock + sp(c_l, c_p) \leq p_i.deadline \tag{4.3}$$

where $sp(c_l, c_p)$ is the travel time along the shortest travel time path from the anchor node of grid cell g_l to the anchor node of grid cell g_p.

The above condition guarantees that only grid cells from which a vehicle can travel to the pickup node grid cell before the pickup deadline will be retrieved, assuming that each grid cell collapses to its anchor node. Finally, candidate vehicles are identified from the lists *vecList* of each of the retrieved grid cells as well as the grid cell of the pickup node. To find the candidate vehicles from a grid cell g_l, the searching phase returns all vehicles in g_l.vecList that satisfy the condition:

$$t_{entry} + sp(c_l, c_p) \leq p_i.deadline \tag{4.4}$$

This condition guarantees that the pickup node deadline allows time for the vehicle to enter the grid cell g_l and then move from that grid cell to the pickup node grid cell (assuming that each grid cell collapses to its anchor node).

Vehicle Scheduling

The goal of the vehicle scheduling phase is to identify which vehicle among the candidate vehicles can be used to schedule the given pickup and delivery node pair with the minimum increase in vehicle travel and waiting times. To achieve this, the scheduling phase needs to attempt all possible insertion points for the pickup and delivery nodes into a vehicle's schedule. In addition, the scheduling algorithm may also attempt to reorder the nodes on the current vehicle's schedule, subject to feasibility constraints, to minimize the vehicle's travel and waiting times. However, finding the optimal order of nodes is known to be an NP-hard problem [65] and thus performing the ordering step can degrade the user experience in an on-demand service where consumers expect quick response times. Therefore, we assume that the order of the nodes in the schedule is fixed, an assumption that has often been used in on-demand services [65, 105]. The scheduling algorithm determines the the best pickup and delivery node insertion points in each candidate vehicle's schedule by choosing the min cost insertion way. The scheduling phase finally returns the schedule with the min cost insertion among all candidate vehicles.

Algorithm 8 presents an outline for the scheduling phase. For each candidate vehicle, the algorithm attempts all possible insertion points for the pickup and delivery nodes in the vehicle's schedule such that the delivery node is visited after the pickup node but not before it. The insertion is performed through a call to the method $Insert(s_v,$ p_i, d_j, k, $l)$ on line 5. This method starts by inserting the pickup node p_i at point k in the schedule. The insertion has to satisfy the feasibility constraints as detailed in Section 4.3.1. These constraints guarantee that the resulting schedule is a valid schedule in which all nodes will be visited within the allowed time windows. If p_i is successfully inserted, the delivery node d_j is inserted at point l while also satisfying the relevant feasibility constraints for the delivery node insertion. Finally, the method returns the updated schedule and associated insertion cost (as described below) to the scheduling algorithm, and the vehicle and schedule with the min overall cost are selected.

Algorithm 8 Scheduling Phase in Baseline Method

Input: $candidVehicles$: candidate vehicles set , p_i: pickup node to be inserted, d_j: delivery node to be inserted

Output: If a feasible insertion was found, algorithm returns a vehicle v_{min} and its updated schedule $updated\_s_{min}$ with p_i and d_j, and the insertion cost $minCost$. Otherwise, algorithm returns null

Algorithm:

```
1:  $minCost \leftarrow \infty$
2:  for each vehicle $v$ in $candidVehicles$ do
3:      for each insertion point $k$ in schedule $s_v$ do
4:          for each insertion point $l > k$ in schedule $s_v$ do
5:              $< updated\_s_v, cost > \leftarrow$ Insert($s_v$, $p_i$, $d_j$, $k$, $l$)
6:              if ($updated\_s_v \neq null$) AND ($cost < minCost$) then
7:                  $v_{min} \leftarrow v$
8:                  $updated\_s_{min} \leftarrow updated\_s_v$
9:                  $minCost \leftarrow cost$
10: if $minCost \neq \infty$ then
11:     return $< v_{min}, updated\_s_{min}, minCost >$
12: else if r theneturn null
```

Insertion Cost Function: The scheduling phase aims to find the vehicle schedule that minimizes both vehicle travel and waiting times. To this end, $Insert()$ employs an insertion cost function that utilizes a weighted linear combination of two terms as proposed in the best performing heuristic in [109]. Let $s_v^{'}$ be the updated schedule of vehicle v after inserting pickup node p and delivery node d into insertion points k and l of s_v respectively. The cost function for this insertion is defined using Equation 4.5:

$$cost(s_v, p, d, k, l) = \alpha\delta_{tt}(s_v, p, d, k, l) + (1 - \alpha)(PF(p, k) + PF(d, l)), \tag{4.5}$$

where $\delta_{tt}(s_v, p, d, k, l)$ is the difference between the total travel time of schedule s_v and $s_v^{'}$ and is defined using Equation 4.6, and $PF(p, k)$ and $PF(d, l)$ are the push forward functions for the pickup and delivery nodes respectively. The push forward functions are defined in equations 4.7 and 4.8 and are used to minimize the vehicle waiting or idle time. α is a scalar value that represents the weight of minimizing the total travel time versus the total idle time in a schedule such that $0 \leq \alpha \leq 1$.

$$\delta_{tt}(s_v, p, d, k, l) = \sum_{i=1}^{i=|s_v^{'}|-1} sp(s_v^{'}[i], s_v^{'}[i + 1]) - \sum_{i=1}^{i=|s_v|-1} sp(s_v[i], s_v[i + 1]), \tag{4.6}$$

where $s_v[i]$ and $s_v^{'}[i]$ are the nodes at index i in schedule s_v and $s_v^{'}$ respectively.

$$PF(p,k) = \begin{cases} 0, & \text{if } p \text{ inserted at end of } s_v \\ pickupStart(succ_p, s_v^{'}) - pickupStart(succ_p, s_v), & \text{if } succ_p \text{ is a pickup node} \\ rendezvousStart(succ_p, s_v^{'}) - rendezvousStart(succ_p, s_v), & \text{if } succ_p \text{ is a delivery node} \end{cases} \tag{4.7}$$

where $succ_p$ is the successor node of p in the schedule s_v; $pickupStart(succ_p, s_v)$ is the time at which the pickup duration t_{pickup} starts at the node $succ_p$ in schedule s_v, assuming it is a pickup node; and $rendezvousStart(succ_p, s_v)$ is the time at which the rendezvous duration $t_{rendezvous}$ starts at node $succ_p$ in schedule s_v, assuming it is a delivery node.

Similarly, $PF(d,l)$ is defined in Equation 4.8. Based on Equation 4.7 and 4.8, the push forward value is always greater than or equal to zero.

$$PF(d,l) = \begin{cases} 0, & \text{if } d \text{ inserted at end of } s_v \\ pickupStart(succ_d, s_v^{'}) - pickupStart(succ_d, s_v), & \text{if } succ_d \text{ is a pickup node} \\ rendezvousStart(succ_d, s_v^{'}) - rendezvousStart(succ_d, s_v), & \text{if } succ_d \text{ is a delivery node} \end{cases} \tag{4.8}$$

where $succ_d$ is the successor node of d in the schedule s_v.

The push forward function indicates the amount of delay in the pickup or rendezvous start time that occurs at the node that follows the node just inserted. Whenever a new pickup or delivery node is inserted into a schedule, the pickup or rendezvous start time at the following nodes may exhibit a delay depending on the amount of wait time that was expected at that node. The longer the initial waiting time at the following node $succ_p$, the smaller the push forward of node p, denoted as $PF(p, k)$ will be. The reason is that the time used for visiting the inserted node p will use some or all of the wait time at $succ_p$ in the schedule. Since a lower insertion cost is always preferred, incorporating the push forward in the insertion cost function prioritizes insertion points where long waiting times occur since at these points the push forward values are smaller, which in turn minimizes the total vehicle idle time.

Feasibility Constraints

Feasibility constraints refer to constraints 2, 3 and 5 in the FLOPDMC problem definition in Section 4.2.2. These constraints ensure that the output vehicle schedules are feasible since (a) each pickup node will be visited for a pickup duration that starts after the request is ready for pickup, (b) each delivery node is visited for a rendezvous duration during the consumer's availability interval at that node, and (c) the pickup node is visited before the delivery node (i.e. precedence rule). Condition (c) is ensured by the nature of the scheduling algorithm. Therefore, in this section, we focus on the first two conditions (a) and (b).

An insertion call $Insert(s_v, p, d, k, l)$ has to iterate through each node in the schedule s_v to ensure that all constraints are satisfied for each node. If a constraint is violated at one node, the $Insert()$ call terminates. To reduce the computational cost of this feasibility checking, each node n in a vehicle's schedule stores the following four variables:

- $n.arr_e$: the earliest possible arrival time at node n such that the visit to all the previous nodes in the schedule is feasible.
- $n.lv_e$: the earliest possible leaving time from node n such that the visit to all the previous nodes in the schedule is feasible and the pickup (or rendezvous) time is feasibly completed at node n.
- $n.lv_l$: the latest possible leaving time from node n such that the visit to all the following nodes in the schedule becomes feasible.
- $n.arr_l$: the latest possible arrival time at node n such that the pickup (or rendezvous) time can be feasibly completed at node n and the visit to all the following nodes in the schedule is also feasible.

Additionally, each pickup node stores the earliest available pickup time $t_{readyForPickup}$

for the given request, and each delivery node stores the start and end of the consumer's availability interval $[t_{start}, t_{end}]$. Now, to check the feasibility of inserting a pickup or delivery node into a schedule, the first step is to set the above four variables for the node to be inserted.

Setting the variables for inserting a pickup node p in schedule s_v: Equations 4.9, 4.10, 4.11 and 4.12 are used to set the variables of the pickup node to be inserted:

$$p.arr_e = \begin{cases} clock + sp(loc_v, p), & \text{if } p \text{ is inserted as first node in } s_v \text{ or } s_v \text{ is empty} \\ x_i.lv_e + sp(x_i, p), & \text{if } p \text{ is inserted after node } x_i \text{ in } s_v, \end{cases} \tag{4.9}$$

where *clock* is the current time.

$$p.lv_e = max(p.arr_e, t_{readyForPickup}) + t_{pickup} \tag{4.10}$$

The above equation ensures that pickup only occurs after the request is ready for pickup even if the vehicle arrives earlier at node p.

$$p.lv_l = \begin{cases} \infty, & \text{if } p \text{ is inserted last in } s_v \text{ or } s_v \text{ is empty} \\ x_{i+1}.arr_l - sp(p, x_{i+1}), & \text{if } p \text{ is inserted before node } x_{i+1} \text{ in } s_v \end{cases} \tag{4.11}$$

Note that if p is inserted at the end of the schedule, its lv_l variable is set to ∞ only until the delivery node for the request is also inserted. Then the lv_l value will be updated.

$$p.arr_l = p.lv_l - t_{pickup} \tag{4.12}$$

Setting the variables for inserting a delivery node d in schedule s_v: Equations 4.13, 4.14, 4.15 and 4.16 are used to set the variables of the delivery node to be inserted:

$$d.arr_e = x_i.lv_e + sp(x_i, d), \tag{4.13}$$

where x_i is the node preceding the inserted node d in s_v.

$$d.lv_e = max(d.arr_e, d.t_{start}) + t_{rendezvous} \tag{4.14}$$

Equation 4.14 ensures that delivery only occurs after the start of the consumer's availability interval at d even if the vehicle arrives earlier at d.

$$d.lv_l = \begin{cases} \infty, & \text{if } d \text{ is inserted last in } s_v \\ x_{i+1}.arr_l - sp(d, x_{i+1}), & \text{if } d \text{ is inserted before node } x_{i+1} \text{ in } s_v \end{cases} \tag{4.15}$$

In Equation 4.15, the value of ∞ indicates that the vehicle has reached the last node on its schedule and is free to move around until a new request is assigned to it, or in the case of an autonomous vehicle, the vehicle may remain at that location until it is matched to another request.

$$d.arr_l = \begin{cases} d.t_{end} - t_{rendezvous}, & \text{if } d \text{ is inserted last in } s_v \\ min(x_{i+1}.arr_l - sp(d, x_{i+1}), d.t_{end}) - t_{rendezvous}, & \text{if } d \text{ is inserted before node} \\ & x_{i+1} \text{ in } s_v \end{cases} \tag{4.16}$$

Finally, for the insertion of the pickup/delivery node n at a given location to be feasible, the following two conditions must hold true:

1. $n.arr_e \leq n.arr_l$

2. $n.lv_e \leq n.lv_l$

The above conditions ensure that the vehicle can arrive at the inserted node before the last possible arrival time, thereby guaranteeing that pickup/delivery at the node is feasible. They also ensure that the vehicle can leave the inserted node before the last time it has to leave for the rest of the schedule to be feasible. Since both conditions can be checked locally (i.e. by retrieving only the inserted node and its direct neighbors) in O(1), they reduce the cost of feasibility checking particularly in the case when the insertion point is found to be infeasible. If the insertion point was feasible, the node is inserted into the schedule. In addition, the following variables will be updated

- arr_e and lv_e of all the nodes following the inserted node in the schedule.
- arr_l and lv_l of all nodes preceding the inserted node in the schedule

4.3.2 pkgRendezvous: A computationally efficient approach for solving FLOPDMC

Our proposed pkgRendezvous algorithm follows the outline of the two-phase approach shown in Algorithm 7. However, to minimize the scheduling computational cost, the *pkgRendezvous* algorithm applies two pruning filters to reduce the search space of candidate schedules. Sections 4.3.2 and 4.3.2 present the details of the proposed filters.

Early Termination Filter

During the scheduling phase (Algorithm 8), calls to $Insert()$ are issued for the given pickup and delivery node to be inserted using all possible insertion combinations (i.e. all insertion points) for each candidate vehicle. Executing $Insert(s_v, p, d, k, l)$, starts by inserting the node p at insertion point k in s_v. If the insertion satisfies the feasibility constraints, the method continues to insert node d at insertion point l.

Let $nodeCost(s_v, p, k)$ denote the cost of inserting node p at insertion point k of schedule v. We define this cost in Equation 4.17 as follows:

$$nodeCost(s_v, p, k) = \alpha\delta_{tt}(s_v, p, k) + (1 - \alpha)PF(p, k), \tag{4.17}$$

where $\delta_{tt}(s_v, p, k)$ is the difference between the total travel time of schedule s_v and that of s_v after inserting p at point k, and $PF(p, k)$ is computed using Equation 4.7 such that $s_v^{'}$ is the schedule s_v after inserting node p at point k.

Lemma 11. *If $nodeCost(s_v, p, k) > \tau$, then for any node d, $cost(s_v, p, d, k, l)) > \tau$, $\forall$ $l > k$.*

Proof. *Since each vehicle is assumed to travel along the shortest travel time path between any two nodes on its schedule, inserting a new node into a vehicle schedule will never decrease the initial total travel time of that schedule. Therefore we have $\delta_{tt}(s_v, p, k) \geq \delta_{tt}(s_v, p, d, k, l)$, $\forall$ $l > k$ (A). We note that only values of $l > k$ are relevant to our discussion since a delivery node cannot be inserted before its corresponding pickup node. Also, from the definition of the push forward function in equations 4.7 and 4.8, we have $PF(p, k) \geq 0$ and $PF(d, l) \geq 0$ (B). From (A) and (B) and the insertion cost function and $nodeCost(s_v, p, k)$ definitions in equations 4.5* *and 4.17 respectively, we get* $cost(s_v, p, d, k, l) = \alpha\delta_{tt}(s_v, p, d, k, l) + (1-\alpha)(PF(p,k) + PF(d,l)) \geq \alpha\delta_{tt}(s_v, p, k) + (1-\alpha)PF(p,k) = nodeCost(s_v, p, k)\ \forall\ l > k$. *Therefore,* $cost(s_v, p, d, k, l) \geq nodeCost(s_v, p, k)$¿$\tau$*,* $\forall\ l > k$. *Therefore,* $cost(s_v, p, d, k, l)$¿τ*,* $\forall\ l > k$.

Algorithm 7 and Algorithm 8 keep track of the minimum cost of inserting pickup and delivery nodes, denoted as $minCost$, into any schedule obtained so far among all candidate vehicles, pickup and delivery pairs, and insertion points for a given consumer request. Hence, if a call to $Insert(s_v, p, k, d, l)$ finds that the cost of inserting the pickup node p at point k in s_v exceeds $minCost$, the method immediately terminates without scheduling the delivery node. The reason is that, according to Lemma 11 (where τ is set to $minCost$), the resulting insertion cost of this schedule will also exceed $minCost$ and therefore there is no need to consider the schedule. In addition, all calls to $Insert()$ with $l > k$ for this schedule and pickup node are also pruned.

All-Insertions Cost Lower Bound Pruning Filter

Unlike the early termination filter which terminates the scheduling of a delivery node for a given pickup node insertion point, this filter utilizes a lower bound on the insertion cost of *all possible insertion points* for a given pickup and delivery pair and a given schedule. Hence, if this lower bound exceeds the minimum cost insertion obtained so far, the algorithm can completely prune the schedule for the given pickup and delivery node pair without attempting to schedule either the pickup or the delivery nodes at any of its insertion points.

Before presenting the definition of the proposed lower bound, we need to define the following notations: Given a node n in the spatial network, and a schedule s_v, let $minSP(n, s_v)$ denote the minimum of all shortest travel time costs from n to all nodes in s_v. Hence, $minSP(n, s_v)$ can be computed as follows:

$$minSP(n, s_v) = \min_{\forall 1 < i < |s_v|}(sp(n, s_v[i])) \tag{4.18}$$

Similarly, $minSP(s_v, n)$ denotes the minimum of all shortest travel time costs from all nodes in schedule s_v to node n.

Additionally, we define $maxSP(n, s_v)$ and $maxSP(s_v, n)$ as the maximum of all shortest travel time costs from node n to all nodes in s_v, and from all nodes in s_v to node n respectively. Finally, let $maxPathTime(s_v)$ denote the maximum of all shortest travel time costs of the paths between every two consecutive nodes in s_v. Therefore, $maxPathTime(s_v)$ is computed as follows:

$$maxPathTime(s_v) = \max_{\forall 1<i<|s_v|-1}(sp(s_v[i]), s_v[i+1])) \tag{4.19}$$

Using the above notations, we define the "all-insertions detour lower bound" as shown below. For simplicity, we assume in the following definitions of this section that the current vehicle location loc_v is added as the first node in the vehicle's schedule s_v (i.e., $s_v[0]$).

Definition 26 (All-Insertions Detour Lower Bound)**.** *Given a vehicle schedule s_v, and a pickup node p and delivery node d to be inserted in s_v, the all-insertions detour lower bound, denoted as $LB_{AD}(s_v, p, d)$, is computed as follows:*

$$LB_{AD}(s_v,\ p,\ d) = \max(0,\ LB_{AD,p}(s_v)) + \max(0,\ LB_{AD,d}(s_v, p)), \tag{4.20}$$

where

$$\begin{aligned} LB_{AD,p}(s_v) = {} & \min(sp(s_v[|s_v|],\ p), minSP(s_v,\ p) + minSP(p,\ s_v) \\ & - maxPathTime(s_v)) \end{aligned}$$

and

$$\begin{aligned} LB_{AD,d}(s_v,\ p) = {} & \min(sp(s_v[|s_v|],\ d), sp(p,\ d), \min(minSP(s_v,\ d),\ sp(p,\ d)) \\ & + minSP(d,\ s_v) - \max(maxPathTime(s_v),\ maxSP(p,\ s_v))) \end{aligned}$$

Lemma 12. *Given a vehicle schedule s_v, and a pickup node p and delivery node d to be inserted in s_v, $LB_{AD}(s_v, p, d)$ is a lower bound of $\delta_{tt}(s_v, p, d, k, l)$ $\forall$ $1 < k < l \leq |s_v|$.*

Proof. *From Equation 4.6, $\delta_{tt}(s_v, p, d, k, l)$ represents the difference in the total travel time of schedule s_v (i.e. detour in s_v) after inserting nodes p and d at points k and l respectively. Hence, to prove Lemma 12, we need to prove that $LB_{AD}(s_v, p, d)$ is a lower bound of the detour (in travel time) after inserting p and d at any points k and l $\forall$ $1 < k < l \leq |s_v|$. To this end, we start by proving that $LB_{AD,p}(s_v)$ (i.e. found* *inside the first term on the right side of Equation 4.20) represents a lower bound on the detour of inserting p into s_v at any point k, $\forall\ 1 < k \leq |s_v|$ (Part I). We then prove that $LB_{AD,d}(s_v, p)$ (i.e. in the second term on the right side of the equation) is a lower bound on the detour of inserting d at point l into s_v after p is already inserted, $\forall\ 1 < k < l \leq |s_v|$ (Part II).*

First, consider the insertion of node p in s_v at any point k such that $1 < k \leq |s_v|$. In this case, there are two possible scenarios. The first is that p is inserted at the end of s_v. In this scenario, the travel time detour (i.e. increase in travel time) is equal to the travel time from the last node in s_v to p = $sp(s_v[|s_v|], p)$ (A). The second scenario is that p is inserted between any two nodes in s_v. Note that p cannot be inserted as the first node in s_v since the first node represents the current vehicle location. Therefore, in the second scenario, if p is inserted between nodes x_{k-1} and x_k, then the the travel time detour of inserting p equals $sp(x_{k-1},p)$ + $sp(p,x_k)$ - $sp(x_{k-1},x_k)$ (B). For any k where $1 < k \leq |s_v|$, we have $minSP(s_v,p) \leq sp(x_{k-1},p)$, and $minSP(p,s_v) \leq sp(p,x_k)$ (from Equation 4.18) (C). We also have $maxPathTime(s_v) \geq sp(x_{k-1},x_k)$ (from Equation 4.19) (D). Therefore, from (B), (C) and (D), we have $minSP(s_v,p)$ + $minSP(p,s_v)$ - $maxPathTime(s_v)$ as a lower bound on the travel time detour in the second scenario (E). Hence, from (A) and (E) of both scenarios, we get $LB_{AD,p}(s_v)$ = $\min(sp(s_v[|s_v|],\ p), minSP(s_v,\ p) + minSP(p,\ s_v) - maxPathTime(s_v))$ is a lower bound on the detour of inserting p at k, $\forall\ 1 < k \leq |s_v|$. (Part I)

Next, we consider the insertion of node d at any point l in s_v assuming that p has already been inserted at point k, such that $1 < k < l \leq |s_v|$. In this case, we have similar possible scenarios as above. The first scenario is inserting d at the end of s_v. In this case, the travel time detour is $sp(s_v[|s_v|], d)$. However, since p is already inserted, there is a possibility that p is the node at the end of s_v, so the detour becomes $sp(p, d)$. Therefore a lower bound on the detour in this scenario is $min(sp(s_v[|s_v|], d), sp(p, d))$ (F). The second scenario is inserting d between two nodes x_{l-1} and x_l in s_v. In this scenario, the detour equals $sp(x_{l-1},d)$ + $sp(d,x_l)$ - $sp(x_{l-1},x_l)$ (G). However, since p is already inserted, if d is inserted immediately after p, the detour becomes equal to $sp(p, d)$ + $sp(d,x_l)$ - $sp(p,x_l)$ (H). Also, for any l where $1 < k < l \leq |s_v|$, we have $minSP(s_v,d) \leq sp(x_{l-1},d)$, and $minSP(d,s_v) \leq sp(d,x_l)$, and $maxSP(p,s_v) \geq sp(p,x_l)$, and $maxPathTime(s_v) \geq sp(x_{l-1},x_l)$ (by definition of the terms on the left *side of the inequalities) (M). Therefore, from (G), (H) and (M), we have a lower bound on the detour of inserting d in the second scenario* $= \min(minSP(s_v,\ d),\ sp(p,\ d)) + minSP(d,\ s_v) - \max(maxPathTime(s_v),\ maxSP(p,\ s_v)))$ *(N). Hence, from (F) and (N) of both scenarios, we get that* $LB_{AD,d}(s_v, p) = \min(sp(s_v[|s_v|],\ d),\ sp(p,\ d), \min(\ minSP\ (\ s_v,\ d),\ sp(p,\ d)) + minSP(d,\ s_v)$ - $\max(maxPathTime(s_v),\ maxSP(p,\ s_v)))$ *is a lower bound on the detour of inserting d at point l into* s_v *after p is already inserted,* $\forall\ 1 < k < l \leq |s_v|$. *(Part II)*

From (Part I) and (Part II), and since travel time detours are ≥ 0 *given that vehicles always travel along the shortest travel time path between any two nodes, we get that* $LB_{AD}(s_v,\ p,\ d) = \max(0,\ LB_{AD,p}(s_v)) + \max(0,\ LB_{AD,d}(s_v, p))$ *is the lower bound of the travel time detour after inserting p and d at any points k and l* $\forall\ 1 < k < l \leq |s_v|$.

Definition 27 (All-Insertions Cost Lower Bound)**.** *Given a vehicle schedule* s_v*, and a pickup node p and delivery node d to be inserted in* s_v*, the all-insertions cost lower bound, denoted as* $LB_{AC}(s_v, p, d)$*, can be computed as follows:*

$$LB_{AC}(s_v, p, d) = \alpha LB_{AD}(s_v, p, d) \tag{4.21}$$

Corollary 1. *Given a vehicle schedule* s_v*, and a pickup node p and delivery node d to be inserted in* s_v*,* $LB_{AC}(s_v, p, d)$ *is a lower bound of the insertion cost function* $cost(s_v, p, d, k, l)\ \forall\ 1 < k < l \leq |s_v|$.

Proof. *From Lemma 12, we have* $LB_{AD}(s_v, p, d)$ *is a lower bound of* $\delta_{tt}(s_v, p, d, k, l)$ $\forall\ 1 < k < l \leq |s_v|$ *(A). Also, from equations 4.7 and 4.8, a zero value is considered as a lower bound for* $PF(p, k)$ *and* $PF(d, l)$*,* $\forall\ 1 \leq k < l \leq |s_v|$ *(B). Therefore, from (A), (B), and the definitions in Equations 4.5 and 4.21, we get* $cost(s_v, p, d, k, l) = \alpha\ \delta_{tt}(s_v, p, d, k, l) + (1-\alpha)\ (PF(p, k) + PF(d, l)) \geq \alpha\ LB_{AD}(s_v, p, d) = LB_{AC}(s_v, p, d)$, $\forall\ 1 < k < l \leq |s_v|$. *Therefore,* $cost(s_v, p, d, k, l) \geq LB_{AC}(s_v, p, d)$, $\forall\ 1 < k < l \leq |s_v|$*, which proves the corollary.*

Based on the above corollary, the all-insertions cost lower bound can be used to prune all possible insertions of a given pickup and delivery node in a given schedule if the bound exceeds the minimum cost insertion obtained so far. However, the bound computation involves computing several terms (i.e. $minSP(s_v, p)$, $minSP(p, s_v)$, $minSP(s_v, d)$, $minSP(d, s_v)$, $maxSP(p, s_v)$, and $maxPathTime(s_v)$) based on definitions 27 and 26). These terms require iterating through all schedule nodes, which is an expensive overhead cost if done for every bound computation. To reduce this cost we utilize two lookup tables, namely, $CellVehicleDistTable$ and $MaxPathTimeTable$ where the values of these terms are precomputed. However, to allow these precomputations to be tractable, we approximate the travel times to and from each pickup and delivery node with the travel times to and from the anchor nodes of their grid cells (assuming grid cells collapse to their anchor nodes as in Section 4.3.1. The details of each lookup table are discussed below including how and when the table entries are updated.

A) Grid-based Vehicle Distances Lookup Table ($CellVehicleDistTable$) The $CellVehicleDistTable$ augments the grid index proposed in Section 4.3.1. For each grid cell g_i with anchor node c_i, the lookup table stores the following variables for each vehicle v:

- $minSP(s_v, c_i)$
- $minSP(c_i, s_v)$
- $maxSP(c_i, s_v)$

Now for instance, for a given pickup node p and delivery node d, and a vehicle schedule s_v, the value $minSP(s_v, p)$ is approximated by first finding the grid cell g_p in which node p resides and then retrieving the value $CellVehicleDistTable[g_p,\ v].minSP(s_v, c_p)$. Similarly, the value $minSP(d, s_v)$ is approximated using the value of $CellVehicleDistTable[g_d,\ v].minSP(c_d, s_v)$.

The entries in $CellVehicleDistTable$ need to be updated in the following two cases:

- **When a vehicle leaves a pickup/delivery node on its schedule:** In this case a node is removed from the schedule, so the values of $minSP(s_v, c_i)$, $minSP(c_i, s_v)$, $maxSP(c_i, s_v)$ will need to be updated for this vehicle entry in all grid cells. However, this update can be deferred by scheduling it at regular time intervals. The reason is that while the values of $minSP$ and $maxSP$ may change, the new values of $minSP$ will be greater than or equal to the stored value (given that a node was removed from s_v), while the the new value of $maxSP$ will be less than or equal to the stored value. Thus the all-insertions cost lower bound $LB_{AC}(s_v, p, d)$ computed based on the stored values may only get looser but this does not result in false pruning.

- **When a new pickup and delivery node are assigned into a vehicle schedule:** Deferring lookup table updates in this case will cause the value of the all-insertions cost lower bound to be invalid. Therefore, the lookup table values are immediately updated in this case for the assigned vehicle entry in all grid cells.

(B) Maximum Path Travel Time Lookup Table ($MaxPathTimeTable$) This lookup table stores the value of $maxPathTime(s_v)$ for every vehicle v. Updates to the table entries occur in the following two cases:

- **When a vehicle leaves a pickup/delivery node on its schedule:** In this case, a node is removed from the schedule, which may affect the value of the maximum path cost. Again, this update can be deferred by scheduling it at regular time intervals since it does not invalidate the value of the all-insertions cost lower bound.
- **When new pickup and delivery nodes are assigned into a vehicle schedule:** In this case, new paths are added to the schedule, while some old path may also be broken to accommodate the inserted nodes. Therefore, if the new added paths have a cost that is greater than the stored $maxPathTime(s_v)$, the value is immediately updated in O(1). Otherwise, if an existing path was broken for inserting the new nodes, the table entry is scheduled to be updated as a deferred update.

4.4 Experimental Evaluation

Our experimental goals were two-fold: First, we wanted to compare the performance of our proposed approach, *pkgRendezvous*, to the baseline method. Second, we wanted to evaluate how the performance of the proposed approach is affected by variations in different parameters. The following candidate algorithms were included in our analysis.

- BL: the **B**ase**l**ine method.
- PR-ET: the **p**kg**R**endezvous algorithm using only the **E**arly **T**ermination filter discussed in Section 4.3.2.
- PR-LB: the **p**kg**R**endezvous algorithm using only the all-insertions cost **L**ower **B**ound pruning filter discussed in Section 4.3.2.

- PR-All: the **p**kg**R**endezvous algorithm using both pruning filters.

The algorithms' performance was compared based on both efficiency and effectiveness (i.e. solution quality). To compare efficiency, we used the average response time for consumer requests. To compare effectiveness, we used the percentage of matched requests, the total vehicle travel time, and the total vehicle waiting time.

4.4.1 Discrete-Event Simulation Framework

We simulated the interactions between arriving consumer requests and the on-demand pickup and delivery broker using a discrete-event simulation framework [69] whose outline is shown in Algorithm 9. A priority queue stores the consumer arrival events in the order of their arrival times. The simulation starts by generating all consumer request arrival events, representing consumer requests submitted to the broker, and storing them in the queue. The simulation loop then starts and continues until the queue is empty (lines 7 to 12). For each iteration, the simulation dequeues all request arrival events with arrival times before or at the current clock time. Then, all vehicle objects are updated to their current status. The broker then matches the requests to the delivery vehicles by executing the two-phase *pkgRendezvous* algorithm (line 11). Finally, the simulation performance measures are updated, and the clock advances to the next time instant.

Algorithm 9 Simulation Algorithm

```
1: serviceLocList ← Initialize list of service locations
2: vehicles ← Initialize list of delivery vehicles
3: graph ← Initialize edges and vertices of spatial network
4: < index, cellVehicleDistTable, maxPathTimeTable > ← Initialize grid index and lookup tables
5: queue ← Generate consumer request arrival events
6: clock ← 0
7: while queue not empty do
8:     arrivalEvents ← Get all consumer arrival events with an event arrival time ≤ clock
9:     for each vehicle v in vehicles do UPDATEVEHICLELOCATION(v, clock)
10:    for each request req in arrivalEvents do
11:        assignment ← PKGRENDEZVOUS(graph, serviceLocList, vehicles, req, clock,
       index,cellVehicleDistTable, maxPathTimeTable)
12:        Output assignment
       Update simulation statistics clock ← clock + 1
13: Output Simulation Statistics
```

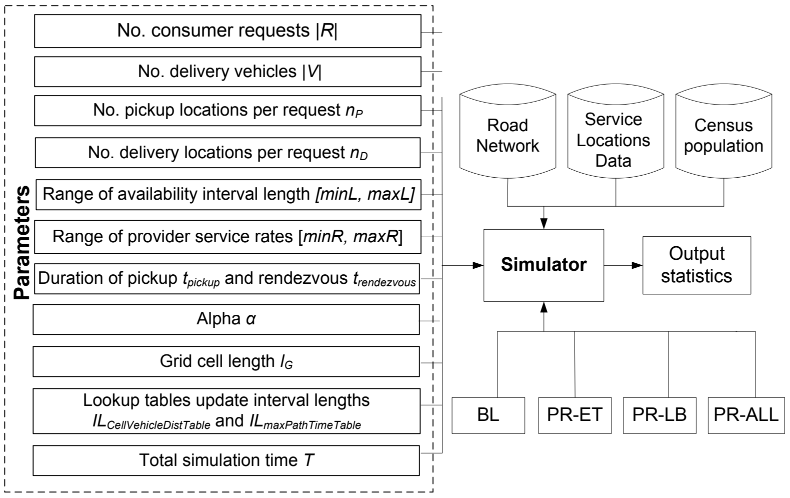

Parameters
No. consumer requests |R|
No. delivery vehicles |V|
No. pickup locations per request n_P
No. delivery locations per request n_D
Range of availability interval length [minL, maxL]
Range of provider service rates [minR, maxR]
Duration of pickup t_{pickup} and rendezvous $t_{rendezvous}$
Alpha α
Grid cell length l_G
Lookup tables update interval lengths $IL_{CellVehicleDistTable}$ and $IL_{maxPathTimeTable}$
Total simulation time T
Road Network
Service Locations Data
Census population
Simulator
Output statistics
BL
PR-ET
PR-LB
PR-ALL

Figure 4.3: Experimental Design

4.4.2 Experimental Design and Dataset

Our experimental design is illustrated in Figure 4.3. We generated synthetic datasets with real-world characteristics as captured by the use of real-world service locations and real-world population data for the city of Minneapolis, MN. The datasets were designed to approximate real-world characteristics due to the lack of real on-demand pickup and delivery request data which is usually collected as proprietary data. Population data was used for generating the candidate delivery locations in the itineraries of consumer requests such that the spatial distribution of consumer delivery locations followed the real population density distribution in the city. We used data for 60 department stores (i.e., service locations) in the city of Minneapolis. The service rates for these service locations were generated using a random number that was uniformly distributed over the range [$minR$, $maxR$]. The initial locations of delivery vehicles were also randomly generated over the spatial network.

Experiments simulated 16 hours of operation for the day. The number of candidate pickup and delivery locations per request (n_P and n_D respectively) were input into the simulator. For each request, the candidate pickup locations were randomly selected from the list of service locations. Then, to generate the delivery locations and the corresponding availability time intervals of a consumer's itinerary we used the following process: First, the last possible request arrival time arr_{last} was computed based on the total simulation time, the number of candidate delivery locations to be generated in consumer itineraries, and the maximum availability interval width using Equation 4.22:

$$arr_{last} = total\_simulation\_time - [n_D \times (maxL + maxNodePairTravelTime)], \quad (4.22)$$

where $maxL$ is the maximum length of consumers' availability intervals at any delivery location, and maxNodePairTravelTime is the maximum travel time between any two nodes in the spatial network.

Equation 4.22 guarantees that the simulation can generate an itinerary for the last arriving consumer request which satisfies the input specifications specified to the simulator. The request arrival time is then randomly generated between the simulation start time and the last possible request arrival time computed from Equation 4.22. For the first delivery location on the consumer's itinerary, the start of the availability interval is set to the request arrival time. The end of the interval is computed by adding the availability interval length, which is uniformly distributed over the range [$minL$, $maxL$]. Next, the location of the delivery location is determined based on the start of the availability interval at that location. If the start time is during daytime, the delivery location is randomly selected in proportion to the daytime population of the nodes in the spatial network such that nodes with higher population are selected for a larger number of requests. Similarly, if the start of the availability interval is during the late evening or night hours, the location is selected in proportion to the nighttime population of the spatial network nodes. To generate the remaining delivery locations on the consumer's itinerary, the start time of the availability interval at each location is first estimated based on the sum of the end of the availability interval at the previous delivery location and the maximum travel time between any two nodes in the spatial network. Using this estimate, the delivery location is randomly selected according to the daytime and nighttime population density as already described. Then, the actual start of the availability interval is computed by adding the availability interval end time at the previous location and the shortest travel time between the two delivery locations. The last three steps are repeated until all delivery locations on the consumer's itinerary are generated with their corresponding availability intervals. Experiments were performed using the road network provided by the Minnesota Department of Transportation [73], and Census population data [110] from which daytime and nighttime population were derived according to [111]. All algorithms were implemented in the Java programming language and run on a machine with an Intel Core i7 2.2 GHz processor and 16 GB RAM.

We analyzed the performance of our proposed algorithms by varying and observing the effect of the following workload parameters: the number of consumer requests $|R|$, the number of delivery vehicles $|V|$, the number of candidate pickup and delivery locations per request (n_P and n_D respectively), the minimum and maximum values for the availability interval length ($minL$ and $maxL$ respectively) at each delivery location, the value of the insertion cost function weight α, the grid cell length l_G, the pickup and rendezvous time durations (t_{pickup} and $t_{rendezvous}$ respectively), the minimum and maximum values for the service rates of service locations ($minR$ and $maxR$ respectively), and the length of the update intervals for the $CellVehicleDistTable$ and $MaxPathTimeTable$ lookup tables ($IL_{CellVehicleDistTable}$ and $IL_{MaxPathTimeTable}$ respectively). The default parameter values were set as follows: $|R|$ = 10,000 requests, $|V| = 1000$ vehicles, $n_P = n_D = 3$ locations, $minL = 1$ hour, $maxL = 3$ hours, $\alpha = 0.75$, $l_G = 1000$ m, $t_{pickup} = t_{rendezvous} = 5$ min, $IL_{CellVehicleDistTable} = IL_{MaxPathTimeTable}$ $= 1$ min, $minR = 20$ requests/hr, and $maxR = 60$ requests/hr, unless stated otherwise.

4.4.3 Experimental Results

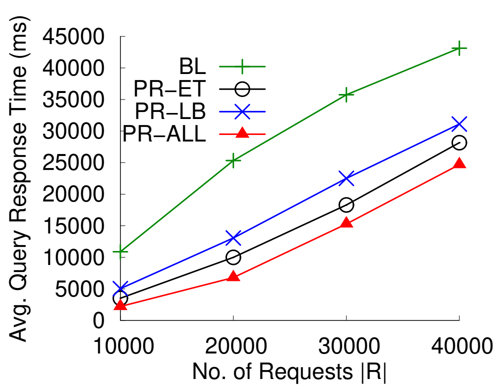


Figure 4.4: $|R|$ versus avg. query response time (best viewed in color)

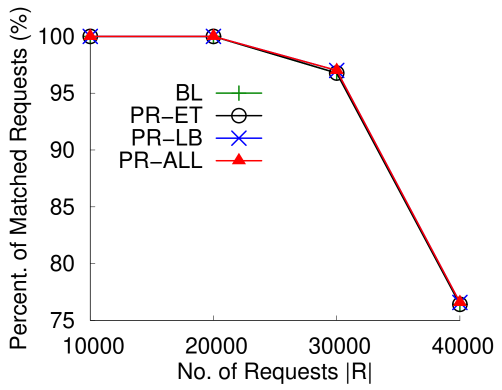


Figure 4.5: $|R|$ versus percent. matched requests (best viewed in color)

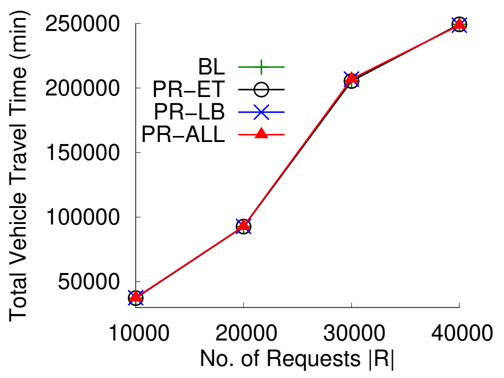


Figure 4.6: $|R|$ versus total travel time (best viewed in color)

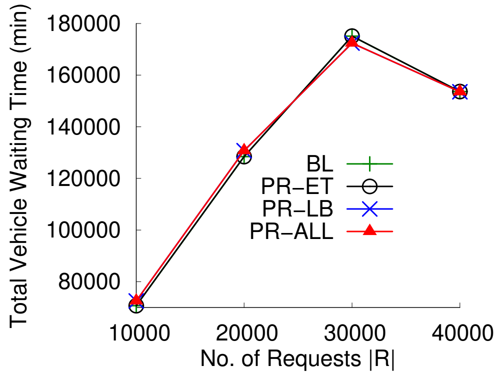


Figure 4.7: $|R|$ versus total waiting time (best viewed in color)

Effect of number of consumer requests ($|R|$): For this experiment, the minimum and maximum service rate values were set to $minR = maxR = 60$ requests/hr to allow more supply for the increased number of simulated requests. Figure 4.4 shows the effect of the number of requests $|R|$ on the average query response time. As can be seen, increasing$|R|$ increases the average query response time since a larger number of requests is being processed simultaneously by the broker. At all request levels, the *pkgRendezvous* algorithm with all its variations consistently outperform the baseline method *BL*. We note that *PR-ET* was faster than *PR-LB* since it does not include the overhead of maintaining and updating the lookup tables. However, the combination of both filters (*PR-ALL*) resulted in the highest computational savings (up to 4.9x as fast as BL).

Figure 4.5 shows the effect of $|R|$ on the percentage of matched requests. As $|R|$ increases, a smaller percentage of requests can be matched by the broker due to the fixed supply of vehicles and service locations. At $|R| = 40{,}000$ requests, a large decrease in the percentage of matched requests occurred due to the saturation of the available supply. From the figure, we can also observe that all algorithms result in a similar percentage of matched requests.

Figure 4.6 and Figure 4.7 show the effect of $|R|$ on the total vehicle travel and waiting times respectively. As $|R|$ increases, the vehicles travel time increases to accommodate the new requests. Similarly, the total vehicle waiting time increases initially as more requests are inserted into the vehicles' schedules, resulting in vehicles waiting at pickup and/or delivery locations. However, as the number of requests increases further, we notice a decrease in the total waiting time. This happens because fuller vehicle schedules and more travel between pickup and delivery locations fills the idle time spent waiting at those locations. From both figures, as well as Figure 4.5 we can see that the *pkgRendezvous* algorithm does not sacrifice solution quality when reducing the computational cost since the results of the algorithm (i.e. percentage of matched requests, total vehicle travel and waiting times are very similar to the baseline method.

Effect of number of delivery vehicles ($|V|$): Figure 4.8 shows the effect of the number of vehicles $|V|$ on the average query response times. As $|V|$ increases, the average query response time also increases due to the larger number of candidate vehicles returned from the vehicle searching phase. We can observe that all variations of the *pkgRendezvous* algorithm consistently outperform the baseline method, with *PR-All* performing up to 3.8x as fast as the baseline method. Figure 4.9 shows the effect of $|V|$ on the percentage of matched requests. As $|V|$ increases, the percentage of matched requests also increases up to 100%. Similarly, the total vehicle travel time initially increases as shown in Figure 4.10, since adding more vehicles allows more requests to be matched which increases the total vehicle travel time. However, as $|V|$ increases further, the total travel time starts decreasing since the the larger number of vehicles provide more possibilities for optimizing vehicle schedules, resulting in a lower total vehicle travel time value. Figure 4.11 shows that the total vehicle waiting time increases with the increase in V. The reason is that as travel time is reduced by the availability of more vehicles, the vehicles spend more idle time at pickup/delivery locations. Again, from Figures 4.9, 4.10 and 4.11, we can see that *pkgRendezvous* returns very comparable solutions to the baseline method. The total travel time of *PR-All* was only slightly higher than *BL* (less than 70 secs per vehicle on average), while achieving smaller total waiting time.

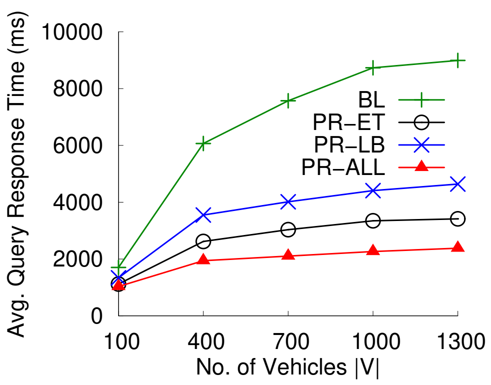


Figure 4.8: $|V|$ versus avg. query response time (best viewed in color)

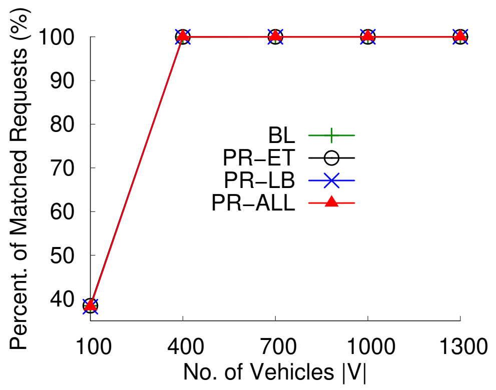


Figure 4.9: $|V|$ versus percent. matched requests (best viewed in color)

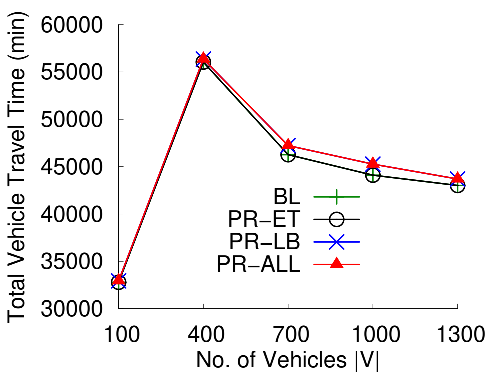


Figure 4.10: $|V|$ versus total travel time (best viewed in color)

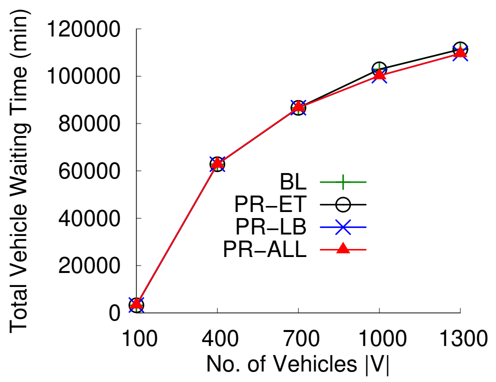


Figure 4.11: $|V|$ versus total waiting time (best viewed in color)

Effect of number of candidate delivery locations per consumer itinerary (n_D): In this experiment, the minimum and maximum length for the consumer availability interval at all delivery locations were set to $minL = maxL = 1$ hour to allow increasing the number of delivery locations per consumer itinerary. The last possible request arrival time arr_{last} was fixed based on $n_D = 5$, so that only the number of candidate delivery locations n_D was varied. Figure 4.12 shows the effect of varying n_D on the average query response time. We observe that as n_D increases, the average query response time increases for all algorithms. Also, as n_D increases, the separation between *pkgRendezvous* and *BL* increases. Again, we can observe that *PR-All* consistently outperforms all other algorithms.

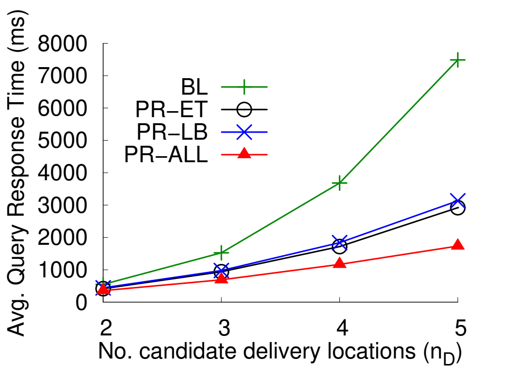


Figure 4.12: n_D versus avg. query response time (best viewed in color)

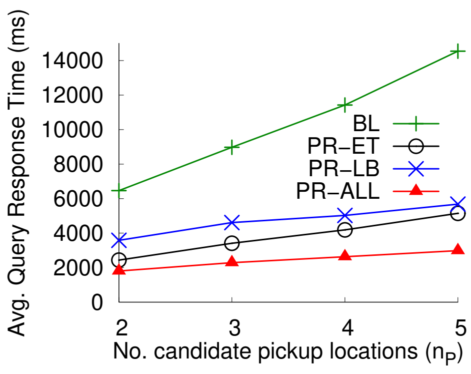


Figure 4.13: n_P versus avg. query response time (best viewed in color)

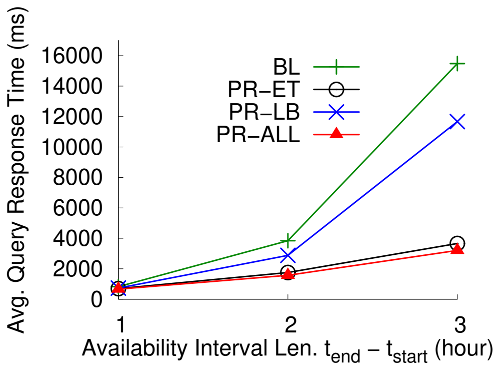


Figure 4.14: $(t_{end} - t_{start})$ versus avg. query response time (best viewed in color)

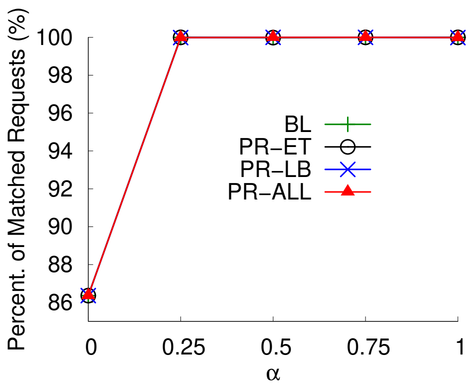


Figure 4.15: α versus percent. matched requests (best viewed in color)

Effect of number of candidate pickup locations per request (n_P): In Figure 4.13, we can observe that as the number of candidate pickup locations per request increases, the average query response time also increases due to the larger scheduling possibilities that need to be examined. Similar to the aforementioned results, *PR-All* also consistently outperforms BL.

Effect of length of consumer's availability interval per delivery location ($t_{end} - t_{start}$): In this experiment, $|V|$ was set to 200 vehicles, the last possible request arrival time arr_{last} was fixed based on $maxL = 3$ hours, and the value of $minL$ and $maxL$ were both varied as shown on the x-axis in Figure 4.14. As can be seen from the figure, as the availability interval length increases, the average query response time also increases. The reason is that with the increase in a consumer's availability time at each delivery location, there is a larger probability that the consumer's request can be feasibly matched to a delivery vehicle, which implies that more requests will get scheduled. The increased scheduling overhead results in increased query response times.

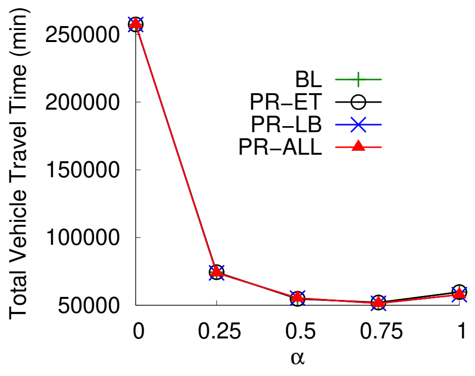


Figure 4.16: α versus total travel time (best viewed in color)

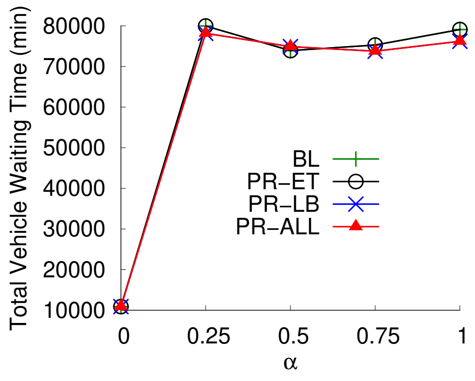


Figure 4.17: α versus total waiting time (best viewed in color)

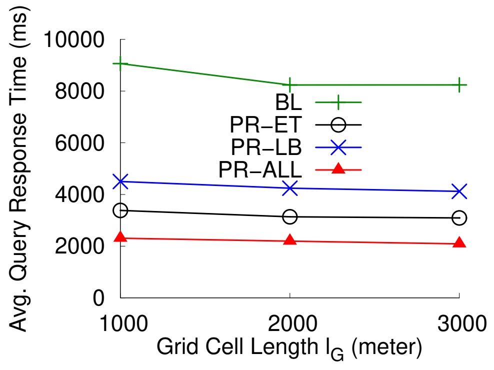


Figure 4.18: l_G versus avg. query response time (best viewed in color)

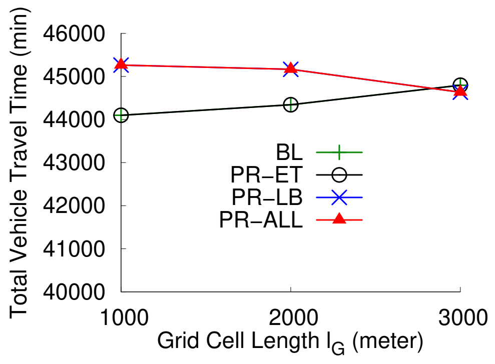


Figure 4.19: l_G versus total travel time (best viewed in color)

Effect of insertion cost function weight (α): Figure 4.15 shows the effect of α

on the percentage of matched requests. At $\alpha = 0$, many requests are unmatched since the part of the cost function that minimizes the travel time in the insertion cost function is eliminated. This results in unoptimized schedules, making it harder to schedule more requests. However, as α increases, the percentage of matched requests also increases. The effect of α on the total vehicle travel and waiting times is shown in Figure 4.16 and Figure 4.17 respectively. We notice that increasing α results in decreasing the total vehicle travel time. The total waiting time greatly increases from α= 0 to α= 0.25 due to the large decrease in travel time, which in turn leaves space for more idle (i.e. waiting) time. The best results occur at $\alpha = 0.75$, where we have the smallest total travel time and total waiting time simultaneously.

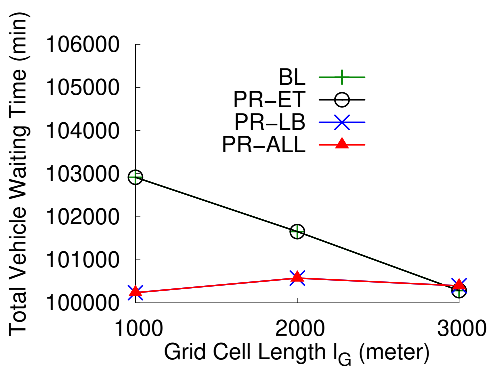


Figure 4.20: l_G versus total waiting time (best viewed in color)

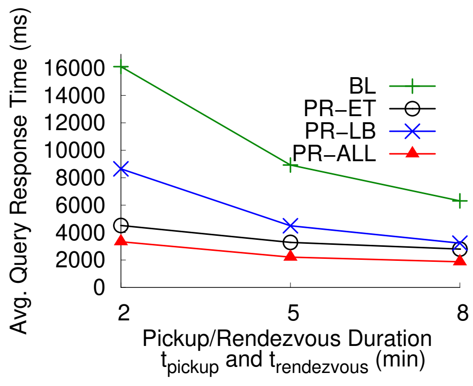


Figure 4.21: t_{pickup} and $t_{rendezvous}$ vs. avg. query response time (best viewed in color)

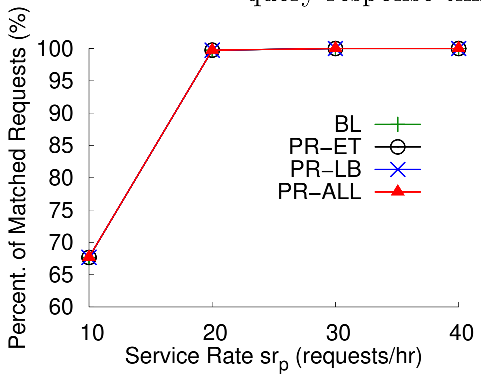


Figure 4.22: sr_p versus percent. matched requests (best viewed in color)

Effect of grid cell length (l_G): As can be seen in Figure 4.18, an increase in l_G results in slightly lower average query response time since the number of grid cells retrieved and updated during vehicle searching decreases. Figure 4.19 and Figure 4.20 show the effect of increasing l_G on the total vehicle travel and waiting times respectively. We can see that for *BL* and *PR-ET*, increasing l_G increases the total travel time. This occurs due to the reduced number of grid cells, which implies that a larger cell size is approximated using an anchor node. This reduces the quality of the approximation used in vehicle searching, resulting in longer travel times and consequently lower idle/waiting times (as more time is spent traveling on the network). However, interestingly, we observe that for *PR-LB* (and also for *PR-All*, which uses the same lower bound filter), the total travel time decreases from $l_G = 2000$ m to $l_G = 3000$ m. This can be attributed to the online nature of the problem (i.e. requests are not known to the broker before their arrival) and the greedy nature of the algorithm. Hence, the algorithm attempts to optimize the objectives based on the current available requests. However, even a suboptimal schedule at the current time instant may result in a better solution in the future as more requests arrive to the broker. We also note that for this experiment, all requests are matched by all candidate algorithms.

Effect of pickup and rendezvous durations (t_{pickup} and $t_{rendezvous}$): In this experiment, both t_{pickup} and $t_{rendezvous}$ were varied as shown in Figure 4.21. As t_{pickup} and $t_{rendezvous}$ increase, we observe that the average query response time decreases. This occurs since the increase in the pickup and rendezvous durations minimizes the number of feasible insertion points for a given pickup and delivery pair into a given schedule. Therefore, as more insertion points become infeasible, the overhead of scheduling the pickup and delivery nodes at these points is eliminated, which reduces the response time for the corresponding requests.

Effect of service rate (sr_p): Figure 4.22 shows the effect of increasing the service rates of service locations on the percentage of matched requests. At sr_p=10 for all service locations, the percentage of matched requests is low since the available supply is much less than the demand (i.e. consumer requests). As sr_p increases, the supply exceeds demand and thus all consumer requests can be matched.

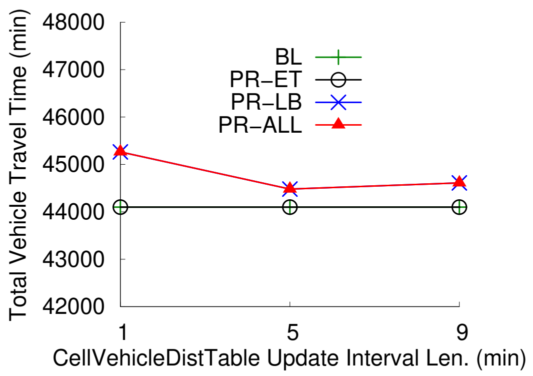


Figure 4.23: Update interval of $CellVehicleDistTable$ vs. total travel time (best viewed in color)

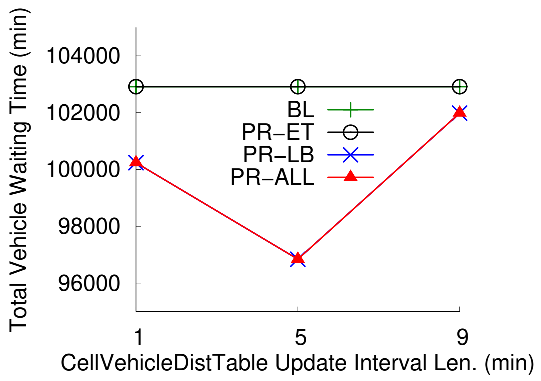


Figure 4.24: Update interval of $CellVehicleDistTable$ vs. total waiting time (best viewed in color)

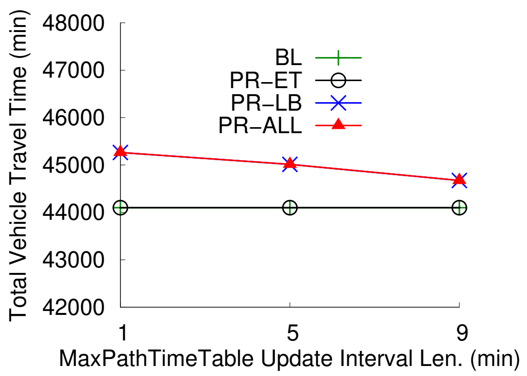


Figure 4.25: Update interval of $MaxPathTimeTable$ vs. total travel time (best viewed in color)

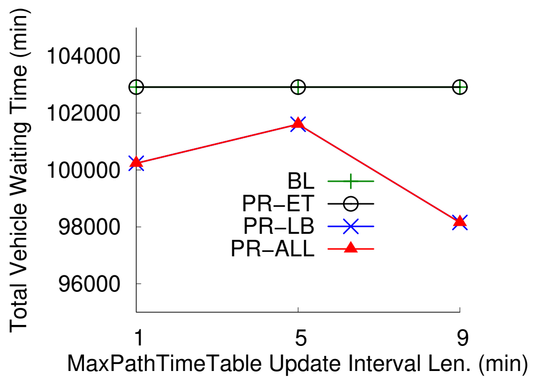


Figure 4.26: Update interval of $MaxPathTimeTable$ vs. total waiting time (best viewed in color)

Effect of lookup table update intervals length ($IL_{CellVehicleDistTable}$ and $IL_{MaxPathTimeTable}$): Figure 4.23 and Figure 4.24 show the effect of the deferred update interval length of the lookup table $CellVehicleDistTable$ on both the total vehicle travel and waiting times respectively. Similarly, Figure 4.25 and Figure 4.26 show the effect of the $MaxPathTimeTable$ update interval length. Generally, a longer interval length implies a lower update frequency. We note that in both experiments, all requests were matched by the broker. As observed in these figures, the total travel and waiting times for both the *BL* and *PR-ET* algorithms remain constant since they do not apply the all-insertions cost lower bound pruning filter, and therefore do not employ any lookup tables. As seen from Figure 4.23 and Figure 4.25, the *PR-LB* and *PR-All* algorithms achieve a total travel time that is within only 0.9% to 2.6% higher than the total travel time of the baseline method (*BL*). In addition, *PR-LB* and *PR-All* achieve even lower total vehicle waiting time than *BL* as shown in Figure 4.24 and Figure 4.26. We can also observe that the total travel time did not increase at $IL_{CellVehicleDistTable}$=5 and $IL_{MaxPathTimeTable}$=5. This suggests that a longer update interval may also be used.

4.5 Conclusion and Future Work

This work explored the problem of Flexible On-demand Pickup and Delivery for Moving Consumers (FLOPDMC) where each consumer request includes an itinerary with multiple possible delivery locations and the corresponding time intervals during which the consumer is available at each location. The on-demand broker matches consumer requests to delivery vehicles with the objectives of maximizing the number of matched requests and minimizing the total vehicle travel and waiting times. We proposed *pkgRendezvous*, a matching and scheduling algorithm with two pruning filters: an early termination condition, and an all-insertions cost lower bound pruning filter with lookup tables for efficiently pruning high cost pickup and delivery pairs. Our experimental evaluation showed that our proposed approach resulted in significant computational savings compared to the baseline method without sacrificing the solution quality. In the future, we plan to investigate a variation of this problem where candidate delivery locations are associated with probabilistic rather than deterministic time intervals. These probabilities may arise in the case where the candidate locations are mined from consumers' historical GPS trajectories.

Chapter 5

Conclusion and Future Directions

5.1 Key Results

Spatiotemporal big data are of volume, velocity, and variety that exceed the capabilities of common spatiotemporal data analytic techniques. This thesis investigates spatiotemporal big data analytics that address the volume and velocity challenges of spatiotemporal big data in the context of novel applications in transportation and engine science, future mobility, and the on-demand economy. The thesis proposes scalable algorithms for mining non-compliant window co-occurrence patterns and introduces novel upper bounds for a non-monotonic statistical interest measure to address the large data volume challenge by pruning uninteresting candidate patterns. To address the high velocity challenge, the thesis proposes novel and scalable online optimization heuristics for matching service providers to dynamically arriving requests from mobile consumers in an on-demand spatial service broker. The proposed heuristics provide a robust matching approach under variations in the supply-demand ratio. A novel matching and scheduling algorithm is also proposed for the case of moving providers (i.e., delivery vehicles) to scale up to a large number of on-demand transactions. Evaluation on real-world data as well as synthetic datasets with real-wold characteristics show that the proposed approaches yield significant computational savings without sacrificing the solution quality. The proposed matching heuristics were also shown to outperform related work on multiple performance measures (i.e., provider utilization variance, percentage of matched requests, average utilization of least utilized providers). A summary of the thesis contributions is shown in Table 5.1, which also illustrates open directions for future work (marked in red).

Table 5.1: Taxonomy of thesis contributions with future directions marked in red color. ST indicates short term future directions while LT indicates long term directions.

	Spatiotemporal Big Data Challenge			
Spatiotemporal Big Data Analytics Model	**Volume**	**Velocity**	**Variety**	**Veracity**
Non-compliant Window Co-occurrence Pattern	• Monotonic upper bounds for a statistical interest measure (Ch. 2) • A Multi-Parent Tracking pruning approach (Ch. 2) • A bidirectional pruning approach (Ch. 2) Future Work: Algorithm parallelization and statistical significance testing (ST)	Online mining of emerging patterns in vehicle measurement big data (ST)	Mining co-occurrences with exogenous data (LT)	Mining meaningful patterns from uncertain data (LT)
Spatiotemporal Optimization	• An early termination filter (Ch. 4) • An all-insertions cost lower-bound pruning filter for efficiently enumerating candidate schedules. (Ch. 4) Future Work: Algorithm parallelization (ST)	Utilization-aware online matching heuristics (Ch. 3)	Modeling consumer preferences from historical transactions (LT)	Modeling uncertain locations mined from historical trajectories (LT)

5.2 Short Term Future Directions

In the short term, I plan to investigate the following directions for further improving the scalability of the proposed approaches and also addressing the velocity challenge faced when mining non-complaint window co-occurrence patterns.

Parallelization of the Proposed Approaches: Due to the growing volume of spatiotemporal big data, parallelization may be considered to further improve the scalability of the proposed approaches. I plan to investigate different alternatives for parallelizing the proposed non-compliant window co-occurrence pattern mining algorithms. A simple approach could be to distribute the processing of different trajectories and/or different non-complaint windows across different nodes. Since the size of the lattice representing the candidate patterns increases exponentially with the number of variables, approaches for parallelizing the processing of the lattice nodes may also be investigated. Similarly, the matching workflow in an on-demand spatial service broker can occur in a distributed manner since supply-demand matching can be considered an embarrassingly parallel workload where consumer requests and service providers can be split into parallel tasks, each representing a geographic region.

Statistical Significance Testing: In this thesis, we looked at statistical interest measures of association (i.e., cross-K function) and developed monotonic upper bounds to efficiently prune uninteresting patterns using a user specified threshold. This ensures that the output patterns are statistically meaningful which helps in communicating the patterns to domain scientists. However, the proposed approach did not ensure the statistical significance of the output patterns. In the future, statistical significance testing need to be incorporated to further help in removing false positive patterns.

Online Mining of Emerging Non-compliant Window Co-occurrence Patterns: Emerging co-occurrence patterns refer to recent co-occurrence patterns that have only recently occurred and persisted in the data. The streaming nature of vehicle sensor data can allow the discovery of new emerging co-occurrence patterns which may help in monitoring engine malfunctions (e.g. unexpectedly high emissions or fuel consumption) and provide engine scientists with insights into possible reasons associated with such malfunctions, thus aiding in vehicle prognostics. However, due to the large data volume and high velocity of the data, discovering such patterns require efficient online and incremental algorithms. The related literature of online association rule mining focuses on identifying frequent itemsets. However, in co-occurrence pattern mining, it is not sufficient to only store the frequent itemsets and their counts since even low support patterns can result in interesting co-occurrence patterns which are highly associated with non-compliant windows. In addition, due to the high dimensionality of the data (i.e. large number of variables), storing the counts of all patterns that appeared in the data is infeasible since the number of patterns can be exponential in the number of variables. Hence, new online and incremental algorithms need to be investigated for addressing such challenges.

5.3 Long Term Future Directions

In the long term, algorithms that address the variety and veracity challenges of spatiotemporal big data need to be investigated. For instance, it is important to develop non-compliant window co-occurrence pattern mining approaches that model the variety of data sources for examining co-occurrences with exogenous parameters such as spatial features (e.g., right and left turns, traffic signals, elevation, etc) as well as weather and traffic information. In addition, engine measurement data may also suffer from uncertainty due to noisy readings or suppression due to privacy preserving protocols. Hence, the veracity challenge also needs to be addressed by developing novel approaches to discover meaningful patterns from such uncertain or incomplete data.

Similarly, for on-demand spatial service brokers, the quality of the matching decisions can also be improved by leveraging other data sources such as historical on-demand transactions to model the consumer preferences and increase the acceptance probability of consumers and service providers. Historical user GPS trajectories can also be mined to identify the frequently visited locations. These locations may also be used as candidate delivery locations with associated probabilities that reflect their visiting frequency instead of having the users explicitly list all candidate locations. In this case, on-demand pickup and delivery brokers need to account for the uncertainty of those locations while matching consumers to delivery vehicles.

References

[1] Ahmed Eldawy and Mohamed Mokbel. Spatial Hadoop. `http://spatialhadoop.cs.umn.edu/`, 2013. Accessed 9 Oct 2013.

[2] Louai Alarabi, Mohamed F Mokbel, and Mashaal Musleh. St-hadoop: A mapreduce framework for spatio-temporal data. *GeoInformatica*, 22(4):785–813, 2018.

[3] Ablimit Aji, Fusheng Wang, Hoang Vo, Rubao Lee, Qiaoling Liu, Xiaodong Zhang, and Joel Saltz. Hadoop gis: a high performance spatial data warehousing system over mapreduce. *Proceedings of the VLDB Endowment*, 6(11):1009–1020, 2013.

[4] Jia Yu, Jinxuan Wu, and Mohamed Sarwat. Geospark: A cluster computing framework for processing large-scale spatial data. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 70. ACM, 2015.

[5] Dong Xie, Feifei Li, Bin Yao, Gefei Li, Liang Zhou, and Minyi Guo. Simba: Efficient in-memory spatial analytics. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1071–1085. ACM, 2016.

[6] Ranga Raju Vatsavai, Auroop Ganguly, Varun Chandola, Anthony Stefanidis, Scott Klasky, and Shashi Shekhar. Spatiotemporal data mining in the era of big spatial data: algorithms and applications. In *Proceedings of the 1st ACM SIGSPATIAL international workshop on analytics for big geospatial data*, pages 1–10. ACM, 2012.

[7] Shashi Shekhar, Zhe Jiang, Reem Ali, Emre Eftelioglu, Xun Tang, Venkata Gunturi, and Xun Zhou. Spatiotemporal data mining: a computational perspective. *ISPRS International Journal of Geo-Information*, 4(4):2306–2338, 2015.

[8] Gowtham Atluri, Anuj Karpatne, and Vipin Kumar. Spatio-temporal data mining: A survey of problems and methods. *ACM Computing Surveys (CSUR)*, 51(4):83, 2018.

[9] Karl Russell Guibert Gates, Jack Ewing and Derek Watkins. New York Times. How Volkswagen Is Grappling With Its Diesel Scandal. `https://goo.gl/gZNEUA`, 2016.

[10] Joseph B. White. The Wall Street Journal. U.S. Fines Hyundai, Kia for Fuel Claims. `https://goo.gl/7COZMj`, 2014.

[11] Jonathan Soble. New York Times. Mitsubishi Admits Cheating on Fuel-Economy Tests. `https://goo.gl/zkKBpn`, 2016.

[12] James Manyika et al. Big data: The next frontier for innovation, competition and productivity. Technical report, McKinsey Global Institute, 2011.

[13] Cisco. 2014 Connected World Technology Final Report. `https://goo.gl/ObzJjs`, 2014.

[14] Michael R Evans, Dev Oliver, KwangSoo Yang, Xun Zhou, Reem Y Ali, and Shashi Shekhar. Enabling spatial big data via cybergis: Challenges and opportunities. In *CyberGIS for Geospatial Discovery and Innovation*, pages 143–170. Springer, 2019.

[15] Didi Chuxing. The Gaia Initiative. `https://outreach.didichuxing.com/research/opendata/en/`, 2019. Accessed 21 mar 2019.

[16] Joe Speed. IoT for V2V and the connected car. `http://www.slideshare.net/JoeSpeed/aw-megatrends-2014-joe-speed`, 2014. Accessed 21 mar 2019.

[17] DieselNet. Heavy-Duty Onroad Engines. https://www.dieselnet.com/standards/us/hd.php, 2015.

[18] Wikipedia. Sudden unintended acceleration. https://goo.gl/OvMi6w, 2015.

[19] Jin Wang and Q Peter He. Multivariate statistical process monitoring based on statistics pattern analysis. *Industrial & Engineering Chemistry Research*, 49(17):7858–7869, 2010.

[20] Krish Vijayaraghavan et al. Effects of light duty gasoline vehicle emission standards in the united states on ozone and particulate matter. *Atmospheric Environment*, 60:109–120, 2012.

[21] U.S. Environmental Protection Agency. Ground level ozone health effects. 2014.

[22] Chandan Misra et al. In-use nox emissions from model year 2010 and 2011 heavy-duty diesel engines equipped with aftertreatment devices. *Environmental science & tech.*, 47(14):7892–7898, 2013.

[23] Office of Transportation & Air Quality. Mpg: Label values vs. corporate average fuel economy (cafe) values label mpg. 2014.

[24] Quirin Schiermeier. The science behind the volkswagen emissions scandal. *Nature*, 24 September 2015.

[25] Peter J Diggle, Amanda G Chetwynd, Roland Häggkvist, and Sarah E Morris. Second-order analysis of space-time clustering. *Statistical methods in medical research*, 4(2):124–136, 1995.

[26] Edith Gabriel and Peter J Diggle. Second-order analysis of inhomogeneous spatio-temporal point process data. *Statistica Neerlandica*, 63(1):43–51, 2009.

[27] Charu C. Aggarwal, Mansurul A. Bhuiyan, and Mohammad Al Hasan. Frequent pattern mining algorithms: A survey. In *Frequent pattern mining*. Springer, 2014.

[28] Wei Shen, Jianyong Wang, and Jiawei Han. Sequential pattern mining. In *Frequent pattern mining*. Springer, 2014.

[29] Edith Cohen et al. Finding interesting associations without support pruning. *Knowledge and Data Engineering, IEEE Transactions on*, 13(1):64–78, 2001.

[30] Tara McIntosh and Sanjay Chawla. High confidence rule mining for microarray analysis. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, 4(4):611–623, 2007.

[31] Wael Zakaria, Yasser Kotb, and Fayed Ghaleb. Mcr-miner: Maximal confident association rules miner algorithm for up/down-expressed genes. *Appl. Math*, 8(2):799–809, 2014.

[32] Yan Huang et al. Mining confident co-location rules without a support threshold. In *Proc. of the 2003 ACM symposium on Applied computing*, pages 497–501, 2003.

[33] Tim Schluter and Stefan Conrad. About the analysis of time series with temporal association rule mining. In *IEEE Symposium on Computational Intelligence and Data Mining*, pages 325–332, 2011.

[34] Sherri K Harms and Jitender S Deogun. Sequential association rule mining with time lags. *Journal of Intelligent Information Systems*, 22(1):7–22, 2004.

[35] Lucia Sacchi, Cristiana Larizza, Carlo Combi, and Riccardo Bellazzi. Data mining with temporal abstractions: learning rules from time series. *Data Mining and Knowledge Discovery*, 15(2):217–247, 2007.

[36] Gautam Das et al. Rule discovery from time series. In *Proceedings of the ACM International Conference on Knowledge and Data Discovery*, pages 16–22, 1998.

[37] Reem Y Ali, Venkata MV Gunturi, Andrew J Kotz, Shashi Shekhar, and William F Northrop. Discovering non-compliant window co-occurrence patterns: A summary of results. In *Advances in Spatial and Temporal Databases*, pages 391–410. Springer, 2015.

[38] C Stuart Daw, Charles Edward Andrew Finney, and Eugene R Tracy. A review of symbolic analysis of experimental data. *Review of Scientific Instruments*, 74(2):915–930, 2003.

[39] Sotiris Kotsiantis and Dimitris Kanellopoulos. Discretization techniques: A recent survey. *GESTS Intl. Trans. on Computer Science and Engineering*, 32(1):47–58, 2006.

[40] Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. Experiencing sax: a novel symbolic representation of time series. *Data Mining and knowledge discovery*, 15(2):107–144, 2007.

[41] Philip M Dixon. Ripley's k function. *Encyclopedia of environmetrics*, 2002.

[42] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.

[43] Bill McKibben. Climate change impacts in the united states: the third national climate assessment, 2014.

[44] U.S. Environmental Protection Agency. Inventory of U.S. Greenhouse Gas Emissions and Sinks: 1990 - 2012. https://www3.epa.gov/climatechange/ghgemissions/usinventoryreport.html, 2014.

[45] F. H. Administration. Annual vehicle distance traveled in miles and related data - 2011. https://www.fhwa.dot.gov/policyinformation/statistics/2011/pdf/vm1.pdf, 2014.

[46] Stephen R Turns. *An Introduction to Combustion: Concepts and Applications*, volume 287. McGraw-hill New York, 3rd edition, 2012.

[47] U.S. Government Publishing Office. 40 cfr ch. u section 1036.108. http://goo.gl/fg5NyV, 2015.

[48] Ke Fang, Zongyan Li, Andrew Shenton, David Fuente, and Bo Gao. Black box dynamic modeling of a gasoline engine for constrained model-based fuel economy optimization. Technical report, SAE Technical Paper, 2015.

[49] Dennis N Assanis, Zoran S Filipi, Scott B Fiveland, and Michalis Syrimis. A predictive ignition delay correlation under steady-state and transient operation of a direct injection diesel engine. *Journal of Engineering for Gas Turbines and Power*, 125(2):450–457, 2003.

[50] Charles Colby and Kelly Bell. Harvard Business Review. The On-Demand Economy Is Growing, and Not Just for the Young and Wealthy. `http://goo.gl/LyOlHl`, April 2016.

[51] Instacart. `https://www.instacart.com/`.

[52] Spinlister. `https://www.spinlister.com/`.

[53] TaskRabbit. `https://www.taskrabbit.com/`.

[54] StyleBee. `https://www.stylebee.com/`.

[55] NOAM SCHEIBER. The New York Times. How Uber Uses Psychological Tricks to Push Its Drivers' Buttons. `https://www.nytimes.com/interactive/2017/04/02/technology/uber-drivers-psychological-tricks.html?_r=0`, April 2017.

[56] Reem Y Ali, Emre Eftelioglu, Shashi Shekhar, Shounak Athavale, and Eric Marsman. Supply-demand ratio and on-demand spatial service brokers: a summary of results. In *Proceedings of the 9th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, pages 7–12. ACM, 2016.

[57] Leyla Kazemi and Cyrus Shahabi. Geocrowd: enabling query answering with spatial crowdsourcing. In *Proc. of the 20th SIGSPATIAL Intl. Conference on Advances in Geographic Information Systems*, pages 189–198. ACM, 2012.

[58] Leyla Kazemi et al. Geotrucrowd: trustworthy query answering with spatial crowdsourcing. In *Proc. of the 21st SIGSPATIAL Intl. Conf. on Advances in Geographic Information Systems*, pages 314–323. ACM, 2013.

[59] Dingxiong Deng, Cyrus Shahabi, and Linhong Zhu. Task matching and scheduling for multiple workers in spatial crowdsourcing. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '15, pages 21:1–21:10, New York, NY, USA, 2015. ACM.

[60] Hien To, Cyrus Shahabi, and Leyla Kazemi. A server-assigned spatial crowdsourcing framework. *ACM Transactions on Spatial Algorithms and Systems*, 1(1):2:1–2:28, July 2015.

[61] Peng Cheng et al. Reliable diversity-based spatial crowdsourcing by moving workers. *Proc. of the VLDB Endowment*, 8(10):1022–1033, 2015.

[62] Zongjian He et al. High quality participant recruitment in vehicle-based crowdsourcing using predictable mobility. In *IEEE Conference on Computer Communications*, pages 2542–2550. IEEE, 2015.

[63] Yan Huang et al. Large scale real-time ridesharing with service guarantee on road networks. *Proc. of the VLDB Endowment*, 7(14):2017–2028, 2014.

[64] Blerim Cici, Athina Markopoulou, and Nikolaos Laoutaris. Designing an online ride-sharing system. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '15, pages 60:1–60:4, New York, NY, USA, 2015. ACM.

[65] Masayo Ota et al. A scalable approach for data-driven taxi ride-sharing simulation. In *IEEE Intl. Conf. on Big Data*, pages 888–897. IEEE, 2015.

[66] Niels AH Agatz et al. Dynamic ride-sharing: A simulation study in metro atlanta. *Transportation Research Part B: Methodological*, 45(9):1450–1464, 2011.

[67] Pedro M d'Orey and Michel Ferreira. Can ride-sharing become attractive? a case study of taxi-sharing employing a simulation modelling approach. *IET Intelligent Transport Systems*, 9(2):210–220, 2014.

[68] Monirehalsadat Mahmoudi and Xuesong Zhou. Finding optimal solutions for vehicle routing problem with pickup and delivery services with time windows: A dynamic programming approach based on state–space–time network representations. *Transportation Research Part B: Methodological*, 89:19–42, 2016.

[69] Raj K Jain. *Art of Computer Systems Performance Analysis: Techniques for Experimental Design Measurements, Simulation, and Modeling.* John Wiley, 1991.

[70] Saif Benjaafar et al. Peer-to-Peer Product Sharing: Implications for Ownership, Usage and Social Welfare in the Sharing Economy. *Social Science Research Network Working Paper Series. Available at http://ssrn.com/abstract=2669823*, October 2015.

[71] Samuel P. Fraiberger and Arun Sundararajan. Peer-to-Peer Rental Markets in the Sharing Economy. *Social Science Research Network Working Paper Series. Available at http://ssrn.com/abstract=2574337*, March 2015.

[72] Ming Hu and Yun Zhou. Dynamic Type Matching. *Social Science Research Network Working Paper Series. Available at http://ssrn.com/abstract=2592622*, April 2016.

[73] Minnesota mndot interactive gis basemap. `http://www.dot.state.mn.us/maps/gdma/gis-data.html`. Minnesota Department of Transportation, 2016.

[74] Fitz Tepper. TechCrunch. Uber Releases Hourly Ride Numbers In New York City To Fight De Blasio. `https://techcrunch.com/2015/07/22/uber-releases-hourly-ride-numbers-in-new-york-city-to-fight-de-blasio/`, July 2015.

[75] United Nations. Sustainable Development Goals: 11 Sustainable Cities and Communities. `http://www.un.org/sustainabledevelopment/wp-content/uploads/2016/08/16-00055K_Why-it-Matters_Goal-11_Cities_2p.pdf`, September 2015.

[76] World Economic Forum. 5 reasons why we need to reduce global inequality. `https://www.weforum.org/agenda/2015/09/5-reasons-why-we-need-to-reduce-globalinequality/`, September 2015.

[77] Center for Disease Control (CDC). CDC Health Disparities & Inequalities Report (CHDIR). `https://www.cdc.gov/minorityhealth/chdireport.html/`, 2013.

[78] M. Frank and S. Nowak. Who's Participating and Who's Not? The Unintended Consequences of Untargeted Programs. ACEEE Summer Study on Energy Efficiency in Buildings. `http://aceee.org/files/proceedings/2016/data/papers/2_542.pdf`, 2016.

[79] US Department of Transportation. Smart City Challenge. `https://www.transportation.gov/smartcity`, January 2017.

[80] CECILIA KANG. The New York Times. Pittsburgh Welcomed Uber's Driverless Car Experiment. Not Anymore. `https://www.nytimes.com/2017/05/21/technology/pittsburgh-ubers-driverless-car-experiment.html`, May 2017.

[81] Evangelos Pournaras, Mark Yao, and Dirk Helbing. Self-regulating supply–demand systems. *Future Generation Computer Systems*, 76:73–91, 2017.

[82] Pedram Samadi, Amir-Hamed Mohsenian-Rad, Robert Schober, Vincent WS Wong, and Juri Jatskevich. Optimal real-time pricing algorithm based on utility maximization for smart grid. In *First IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pages 415–420. IEEE, 2010.

[83] Amir-Hamed Mohsenian-Rad and Alberto Leon-Garcia. Optimal residential load control with price prediction in real-time electricity pricing environments. *IEEE transactions on Smart Grid*, 1(2):120–133, 2010.

[84] Amazon Prime Now. https://primenow.amazon.com/.

[85] Instacart. https://www.instacart.com/.

[86] AmazonFresh. https://www.amazon.com/AmazonFresh/b?node=10329849011.

[87] Roadie. https://www.roadie.com/.

[88] Business Wire. 11 million u.s. homeowners experienced package theft within the last year, august home study reveals. https://www.businesswire.com/news/home/20161025005648/en/11-Million-U.S.-Homeowners-Experienced-Package-Theft, 2016.

[89] August Home Inc. Package theft report: Outsmarting criminals at your front door. http://august.com/wp-content/uploads/2016/10/August-Package-Theft-Report-FINAL-102516.pdf, 2016.

[90] Sarah Perez. Walmart partners with smart lock maker august to test in-home delivery of packages and groceries. https://techcrunch.com/2017/09/21/walmart-partners-with-smart-lock-maker-august-to-test-in-home-delivery-of-packages-and-groceries/, sep 2017.

[91] Elyse Betters. What are amazon key and in-car delivery and how do they work? https://www.pocket-lint.com/smart-home/news/amazon/142667-what-are-amazon-key-and-in-car-delivery-and-how-do-they-work, apr 2018.

[92] Damián Reyes, Martin Savelsbergh, and Alejandro Toriello. Vehicle routing with roaming delivery locations. *Transportation Research Part C: Emerging Technologies*, 80:71–91, 2017.

[93] Gizem Ozbaygin, Oya Ekin Karasan, Martin Savelsbergh, and Hande Yaman. A branch-and-price algorithm for the vehicle routing problem with roaming delivery locations. *Transportation Research Part B: Methodological*, 100:115–137, 2017.

[94] Augustin Lombard, Simon Tamayo-Giraldo, and Frédéric Fontane. Vehicle routing problem with roaming delivery locations and stochastic travel times (vrprdl-s). *Transportation research procedia*, 30:167–177, 2018.

[95] Alexandre M Florio, Dominique Feillet, and Richard F Hartl. The delivery problem: Optimizing hit rates in e-commerce deliveries. *Transportation Research Part B: Methodological*, 117:455–472, 2018.

[96] Johan Los, Matthijs TJ Spaan, and Rudy R Negenborn. Fleet management for pickup and delivery problems with multiple locations and preferences. In *International Conference on Dynamics in Logistics*, pages 86–94. Springer, 2018.

[97] Anke Fabri and Peter Recht. On dynamic pickup and delivery vehicle routing with several time windows and waiting times. *Transportation Research Part B: Methodological*, 40(4):335–350, 2006.

[98] Snežana Mitrović-Minić and Gilbert Laporte. Waiting strategies for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, 38(7):635–655, 2004.

[99] Snežana Mitrović-Minić, Ramesh Krishnamurti, and Gilbert Laporte. Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, 38(8):669–685, 2004.

[100] Michel Gendreau, Francois Guertin, Jean-Yves Potvin, and René Séguin. Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. *Transportation Research Part C: Emerging Technologies*, 14(3):157–174, 2006.

[101] Andrea Attanasio, Jean-François Cordeau, Gianpaolo Ghiani, and Gilbert Laporte. Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. *Parallel Computing*, 30(3):377–387, 2004.

[102] Alberto Colorni and Giovanni Righini. Modeling and optimizing dynamic dial-a-ride problems. *International transactions in operational research*, 8(2):155–166, 2001.

[103] Mark ET Horn. Fleet scheduling and dispatching for demand-responsive passenger services. *Transportation Research Part C: Emerging Technologies*, 10(1):35–63, 2002.

[104] Shuo Ma, Yu Zheng, and Ouri Wolfson. T-share: A large-scale dynamic taxi ridesharing service. In *IEEE 29th International Conference on Data Engineering (ICDE)*, pages 410–421. IEEE, 2013.

[105] Shuo Ma, Yu Zheng, Ouri Wolfson, et al. Real-time city-scale taxi ridesharing. *IEEE Transactions on Knowledge and Data Engineering*, 27(7):1782–1795, 2015.

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

[107] Marlin W Ulmer and Sebastian Streng. Same-day delivery with pickup stations and autonomous vehicles. *Computers & Operations Research*, 108:1–19, 2019.

[108] Ido Orenstein, Tal Raviv, and Elad Sadan. Flexible parcel delivery to automated parcel lockers: Models, solution methods and analysis. *Available through researchgate.net*, 2019.

[109] Marius M Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2):254–265, 1987.

[110] Metropolitan Council. `https://metrocouncil.org/Data-and-Maps/Data.aspx`, December 2018.

[111] Reem Y Ali, Yan Li, Shashi Shekhar, Shounak Athavale, and Eric Marsman. Supply and demand aware synthetic data generation for on-demand traffic with real-world characteristics. In *Proceedings of the 10th ACM SIGSPATIAL Workshop on Computational Transportation Science*, pages 36–41. ACM, 2017.

[112] Data Source: Didi Chuxing. Ride-hailing Data for Chengdu, China in November 2016. `https://gaia.didichuxing.com`.

Appendix A

Experimental Results for *pkgRendezvous* on Didi Chuxing Ride-Hailing Data

This appendix presents additional experimental results for running the *pkgRendezvous* algorithm proposed in Chapter 4 for solving the FLOPDMC problem on ride hailing requests from the Didi Chuxing mobile transportation platform. The data used represents ride requests for a single day (November 1st, 2016) in Chengdu, China [112]. The dataset includes a ride request file describing the order id, ride start and stop times, and the GPS coordinates of the pickup and drop-off locations. In addition, the dataset includes a route data file which describes the routes of the vehicles matched to serve the different orders. Each line in this file includes the driver id, order id, the timestamp, and the latitude and longitude coordinates of the vehicle at this timestamp.

A.1 Experimental Design

Experiments assumed 16 hours of operation for the day starting at 8:00 am, and a fixed number of candidate pickup and candidate delivery locations per request. Service Locations (i.e., used for pickup) were selected based on the most frequent pickup nodes in the whole dataset for the month of November. A total of 179 locations were identified covering 50% of the ride requests pickup locations. The service rates for these service locations were generated using a random number that was uniformly distributed over an input range [$minR$, $maxR$]. The candidate pickup nodes of each request were randomly sampled from the set of available service locations. The first delivery location for each request is assumed to be the drop-off location associated with the ride request. Other delivery locations were randomly sampled based on the drop-off nodes distribution in the input ride request data. The time intervals associated with each delivery location were generated in the same way as described in the Minneapolis city simulation in Section 4.4.2. Finally, the initial location of each delivery vehicle was selected based on the vehicle's GPS location at its earliest occurrence in the route data file. The default parameter values were set as follows: $|V| = 2000$ vehicles, $n_P = n_D = 3$ locations, $minL = 1$ hour, $maxL = 3$ hours, $\alpha = 0.95$, $l_G = 1000$ m, $t_{pickup} = t_{rendezvous} = 5$ min, $IL_{CellVehicleDistTable} = IL_{MaxPathTimeTable} = 1$ min, $minR = maxR = 30$ requests/hr, and the last request start time was set to 3 hours from the simulation start time, unless stated otherwise. All experiments were run on a machine with an Intel Xeon Quad Core 3.00 GHz processor with 64 GB RAM.

A.2 Experimental Results

Effect of number of consumer requests ($|R|$): Figure A.1 shows the effect of the number of requests $|R|$ on the average query response time. To allow simulating different number of requests, the last request start time was varied from 1 hour to 5 hours from the start of the simulation time. This resulted in an increasing number of requests which varied from 10,453 requests at 1 hour to 60,167 requests at 5 hours, with an average of 3 requests arriving per second. As can be seen, increasing$|R|$ increases the average query response time since vehicle schedules become increasingly longer which also increases the scheduling overhead of new requests. At 60,167 requests, a decrease is observed in the average query response time due to many scheduling options becoming infeasible. At all request levels, the *pkgRendezvous* algorithm with all its variations consistently outperform the baseline method BL. The combination of both filters (*PR-ALL*) resulted in the highest computational savings (up to 9.7x as fast as BL).

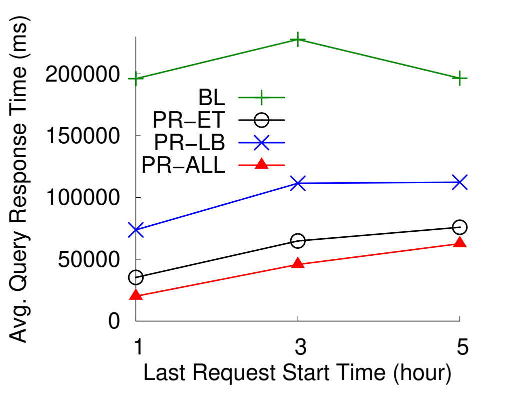


Figure A.1: $|R|$ versus avg. query response time (best viewed in color)

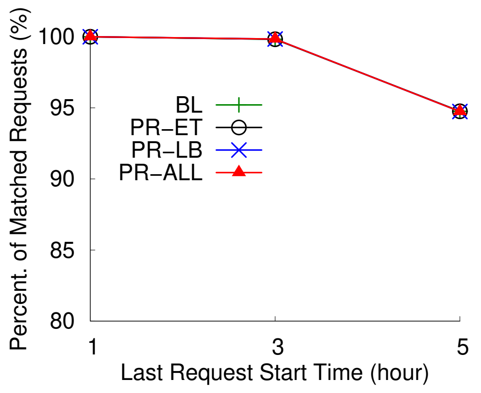


Figure A.2: $|R|$ versus percent. matched requests (best viewed in color)

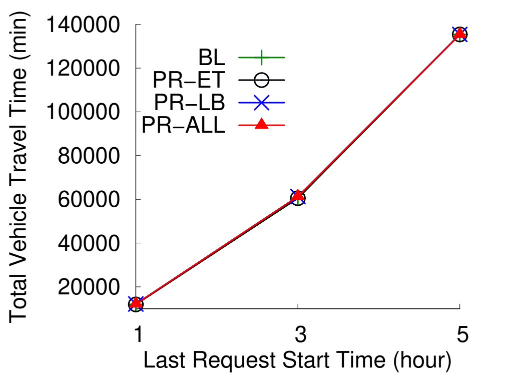


Figure A.3: $|R|$ versus total travel time (best viewed in color)

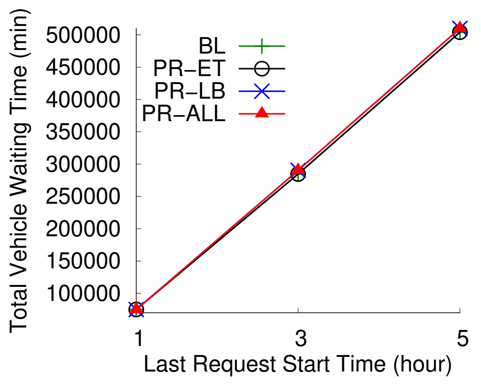


Figure A.4: $|R|$ versus total waiting time (best viewed in color)

Figure A.2 shows the effect of $|R|$ on the percentage of matched requests. As $|R|$ increases, a smaller percentage of requests can be matched by the broker. Figure A.3 and Figure A.4 show the effect of $|R|$ on the total vehicle travel and waiting times respectively. As $|R|$ increases, the vehicles travel time increases to accommodate the new requests. Similarly, the total vehicle waiting time increases as more requests are inserted into the vehicles' schedules, resulting in vehicles waiting at pickup and/or delivery locations. From Figures A.2, A.3 and A.4, it can be clearly observed that the *pkgRendezvous* algorithm does not sacrifice solution quality when reducing the computational cost.

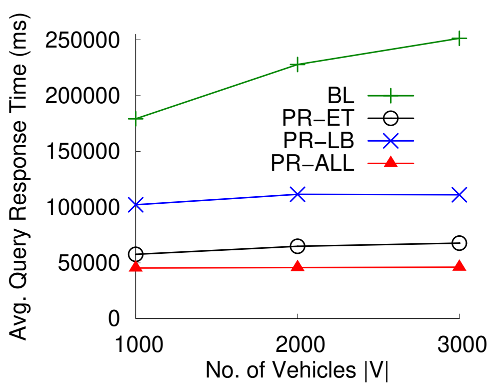


Figure A.5: $|V|$ versus avg. query response time (best viewed in color)

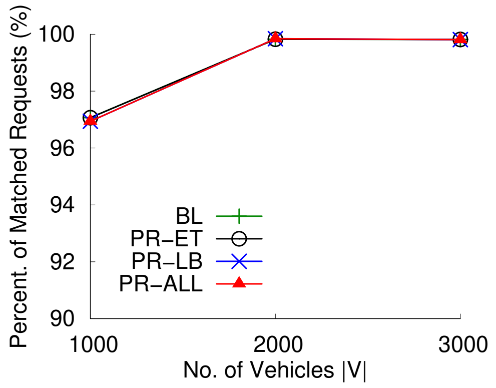


Figure A.6: $|V|$ versus percent. matched requests (best viewed in color)

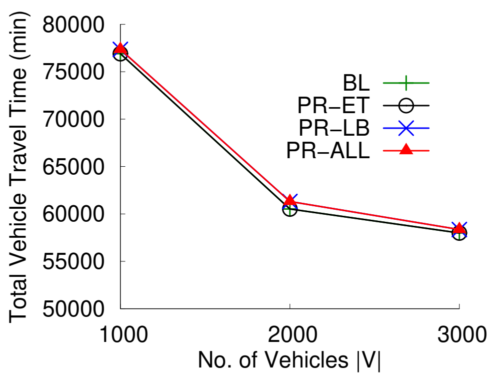


Figure A.7: $|V|$ versus total travel time (best viewed in color)

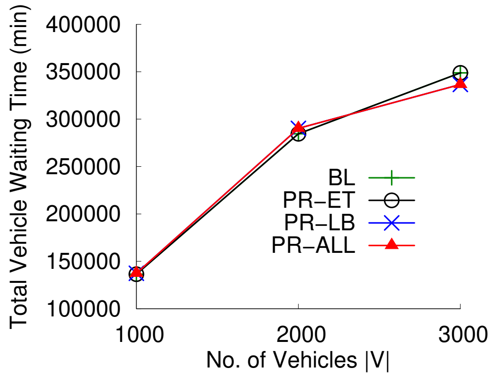


Figure A.8: $|V|$ versus total waiting time (best viewed in color)

Effect of number of delivery vehicles ($|V|$): Figure A.5 shows the effect of the number of vehicles $|V|$ on the average query response times. As $|V|$ increases, the average query response time also increases due to the larger number of candidate vehicles returned from the vehicle searching phase. We can observe that all variations of the *pkgRendezvous* algorithm consistently outperform the baseline method. Figure A.6 shows the effect of $|V|$ on the percentage of matched requests. As $|V|$ increases, the percentage of matched requests also increases up to 100%. However, the total vehicle travel time decreases (as shown in Figure A.7) since a larger number of vehicles provides more possibilities for optimizing vehicle schedules, resulting in a lower total vehicle travel time. Figure A.8 shows that the total vehicle waiting time increases with the increase in V. The reason is that as the travel time is reduced, the vehicles spend more idle time at pickup/delivery locations. Again, from Figures A.6, A.7 and A.8, we can see that *pkgRendezvous* returns very comparable solutions to the baseline method. The total travel time of *PR-All* was only slightly higher than *BL* (only up to 27 secs per vehicle on average). These results agree with the results obtained from our simulation on the Minneapolis, MN sytnthetic dataset in Section 4.4.3.