Real-World Data Driven Characterization of Urban Human Mobility Patterns

Submitted in partial fulfillment of the requirements for
the degree of
Doctor of Philosophy
in
Department of Electrical & Computer Engineering

Abhinav Jauhri

B.Tech. Computer Engineering, Narsee Monjee University
M.S. Software Engineering, Carnegie Mellon University

Carnegie Mellon University
Pittsburgh, PA

May, 2021

Keywords: power law, synthetic data, graph generators, adverserial networks, densification, city planning, graph mining, online algorithms, network evolution

To my grand-parents, parents, parents-in-law, my brother, and my wife.

Abstract

Human movement in urban areas is a complex phenomenon to analyze and understand. The geographical spread of human movement, described by locations between which individuals tend to travel, varies from city to city. The geographical spread of such locations within a city would also vary with time. This thesis explores how to formally analyze and understand human mobility using large sets of real-world data from ride-sharing services for more than a dozen cities and to derive succinct characterizations for large urban areas which account for both geographical and temporal changes.

A wide range of machine learning problems require immense amounts of data. To overcome this issue for human mobility, we propose a framework which includes a stochastic graph model, and adversarial networks to generate synthetic human mobility data which conforms with the geographical and temporal characteristics observed in real data.

Deriving interesting insights from large sets of data and applying those to real-world applications can be challenging. Here we also highlight how formal characterizations can be applied to applications like ride pooling and vehicle placement. We also derive performance bounds for online algorithms using city level characterizations.

Finally, we provide an open-source framework to understand properties of human movement, generate synthetic human mobility data, and apply it to different what-if scenarios for real-world applications. Using such a framework would help public entities and researchers alike to thoroughly understand a complex and highly relevant phenomenon.

Throughout, we link our problem formulations with other domains like social network graphs and online algorithms along with extensive empirical evaluations to increase our understanding of not just human mobility but established techniques in the linked domains.

Acknowledgements

I would like to thank first of all my advisor John Paul Shen for his encouragement, and his good sense of always highlighting worthwhile research directions to explore. His ideas, and suggestions coupled with his experience in both industry and academia helped to shape my research, and other projects undertaken throughtout my graduate student years. John made my time spent at Carnegie Mellon more enjoyable than I could have ever imagined. I will miss having our meetings at some hole in the wall restaurant or some popular diner in Beechview neighorhood.

The depth of knowledge which I have gained over the years at Carnegie Mellon would not have been possible without the faculty members of the different departments who advised me on my research, and under whom I took numerous classes. I would like to thank my thesis committee members Anupam Datta, Jason Hong, and Sean Qian for their time, and invaluable feedback towards my research. Also, I would like to thank Avrim Blum, Anupam Gupta, Ruslan Salakhutdinov, and Carlee Joe-Wang for their insightful lectures which helped me to broaden my thinking.

Internships have been critical to make this thesis possible; I am grateful to my co-authors and mentors at Uber- Radek Grzeszczuk, Vasu Parameswaran, Jerome Berclaz, Bryan Foo; and at Intel- Koichi Yamada, and Jian Hui Li for providing me with the opportunity to collaborate and build my research ideas which subsequently helped to lay the foundational work for many chapters in my thesis.

The many students at Carnegie Mellon at both Pittsburgh and Silicon Valley campuses have made my time as a graduate student particularly enjoyable. I would like to especially thank Erik Reed, Sumeet Kumar, Ming Zeng, Akshay Chandrashekaran, Ervin Teng, Chih Chi Hu, Ritchie Lee, Yuan Tian, Min Hao Chen, Madhumita Harishankar, Tyler Nuanes, Mike Weber, Amit Datta, and Brad Stocks–good friends and researchers who have helped me in many ways including through productive discussions, co-authors on papers & class projects. Also, I would like to thank Aniruddha Basak, Chaitanya Poolla, Vadim Zaliva, and Harideep Nair, my officemates for teaching me about their areas of expertise, and connecting my research with other domains of computer science.

Finally, I would like to thank my family: my grandparents Mahendra and Pushpa for all that they taught me as a young kid; my parents Akshay and Raka for their excitement towards my research, and unmatched support & patience of letting me away for years from home; my brother Avinash for asking the most tough questions about my work; my wife Juhi for her care, and encouragement to help me reach the finish line of my graduate studies; and my dear friends Anugrah, and Divyanshu for making all my vacations to India cherishable.

This research was sponsored by the Mobility21 University Transportation Center (Contract #69A3551747111); and Ericsson (matching fund): *Neuromorphic Systems for Large-Scale Human Mobility Data Analytics* (2018-2020).

Contents

1 Introduction **1**
1.1 Scope and Overview of the Thesis . . . 3
1.2 Summary of contributions . . . 4
1.2.1 Part 1 – Temporal and Spatial Properties: What patterns govern human mobility patterns? . . . 4
1.2.2 Part 2 – Synthetic Generation of Datasets: Scalable and efficient approach for synthetic data generation and validation . . . 5
1.2.3 Part 3 – Example Applications and Toolkit Framework: How can we use city-scale data for different what-if scenarios? . . . 5

2 Temporal Evolution of Human Mobility **7**
2.1 Related Work . . . 8
2.2 Introduction . . . 8
2.2.1 Human Mobility Dataset . . . 9
2.3 Temporal Patterns of Human Mobility . . . 9
2.4 Graph Model . . . 9
2.4.1 Ride Request Graph Generation . . . 10
2.5 Ride Request Graph Densification . . . 12
2.5.1 Characteristic Attributes of RRGs . . . 13
2.6 Degree Distribution Over Time . . . 16
2.6.1 Constant degree exponent over time . . . 18
2.6.2 Measurement from real data . . . 19
2.6.3 Community Effect . . . 19
2.7 Proposed Model . . . 22
2.8 Discussion . . . 24

3 Spatial Distribution & Evolution of Human Mobility **25**
3.1 Introduction . . . 25
3.2 Past work on spatial distribution of human mobility . . . 26
3.3 Introduction to fractals . . . 26
3.4 Fractals and Ride Requests . . . 28
3.4.1 Fractal Dimension . . . 28
3.4.2 Neighbor Approximation Using Fractal Dimension . . . 29
3.4.3 Fractal Dimension of Ride Requests . . . 30

3.5 Discussion 32

4 Scalable Synthetic Generation of Human Mobility Datasets 37
4.1 Why GANs? 38
4.2 Data Sets from Ride Sharing Services 38
4.3 Generating Ride Requests Using GANs 39
4.3.1 Image generating using GANs 39
4.3.2 Using GANs for ride request generation 39
4.3.3 Training Process & Architecture 41
4.4 Experimental Results 42
4.4.1 Running GANs on AWS 42
4.4.2 Training Time 43
4.5 Validation of Synthetic Data Sets 43
4.5.1 Temporal Variation 44
4.5.2 Spatial Variation 47
4.6 An Alternate Method for Synthetic Generation 49
4.6.1 Temporal Variation 50
4.7 Modulating Volume of Ride Requests Using GANs 51
4.8 Discussion 51

5 Real-world Applications of Human Mobility 53
5.1 Dynamic Ride Poooling 54
5.1.1 Travel Distance Distribution 54
5.1.2 Ride Pooling 54
5.1.3 Pooling Methods 57
5.1.4 Pooling Metrics 59
5.1.5 Experimental Details 61
5.1.6 Experimental Goals 62
5.1.7 Experimental Results & Analysis 62
5.1.8 Best Pooling Configuration 65
5.1.9 Temporal Patterns 68
5.1.10 Trip Distance & Duration Distributions 69
5.1.11 Societal Benefits of Pooling 69
5.2 Vehicle Placement 69
5.2.1 Problem Definition 70
5.2.2 Proposed Algorithms 71
5.2.3 Analysis & Results 72
5.2.4 Experimental Results with Real Data 74
5.2.5 Experimental Results with Synthetic Data 74
5.3 Discussion 74

6 Toolkit for Studying Human Mobility **79**
6.1 Temporal and Spatial Modeling . . . 79
6.2 Synthetic Data Generation . . . 80
6.2.1 Extraction of Blocks . . . 80
6.2.2 Trainer . . . 80
6.2.3 Generate Distribution . . . 80
6.2.4 Sample . . . 81
6.2.5 Stitch . . . 81
6.2.6 Graph Generator . . . 81
6.3 Dynamic Ride Pooling . . . 82
6.3.1 GraphHopper . . . 82
6.4 Vehicle Placement . . . 83
6.4.1 Sleeping Experts . . . 83
6.4.2 Follow the Perturbed Leader with Complete History . . . 83
6.4.3 Exp3 . . . 85
6.4.4 Most Recently Used . . . 85
6.5 Discussion and Limitations . . . 85

7 Conclusions and Future Directions **89**
7.1 Implications of Fractal Theory on Human Mobility . . . 89
7.2 Sampling from GANs . . . 89
7.3 Application of Differential Privacy to Human Mobility . . . 90
7.4 Roaming Fleets of Vehicular Computing Systems . . . 90
7.4.1 Applications . . . 91

Bibliography **93**

Chapter 1

Introduction

Traditional human mobility modeling approaches use datasets to make predictions and extrapolations about carbon emissions, traffic congestion on highways, adaptive traffic signal control, real-time scheduling of buses, route planning for peak travel times and many other applications. Though these approaches are successful to provide tools and models to draw inferences, it is difficult to succinctly characterize the phenomenon of city scale human mobility and its evolution over time. Human mobility has been studied from aggregating individual GPS traces; (Song et al., 2010) highlights high predictability of each individual's trajectory due to significant regularity (Gonzalez et al., 2008). Although such aggregation of patterns would be applicable to a majority of cities, it is difficult to infer whether commute pattern varies across cities, and if so which factors cause the variation, and whether or not we can qualitatively measure the difference in mobility patterns. To this end, we propose tools and models to characterize human mobility across different cities and applications which can benefit from such models.

The main goal of our research is to understand patterns and structural properties of human mobility applicable for cities or urban areas across the globe. Such insights would help us understand what does normal human movement in cities look like? How will it evolve with time? For a small time interval, what does the geographical spread of human commute in a city look like? How can such time and geographically varying characteristics help with finding techniques for improving transportation services in cities?

When making critical decisions related to transportation management, researchers and government bodies rely on large datasets. But, how do we know if the data is representative of any typical day of week within a particular city? Using the techniques proposed here to understand patterns across an urban region, we provide statistical tools to quantify patterns and verify how precisely does a dataset capture the properties observed in the real world. To the best of our knowledge, no prior work has looked at multiple cities to draw such characterizations. We also provide generative models for city-wide synthetic data generation capable of amplification or contraction of the volume of ride requests across regions. We believe such models could act as a guide for intelligent transportation management, and formulation of public policies.

The basic premise of understanding large scale mobility patterns is to analyze actual real-world data in order to identify complex collective behavior and how the local interactions of the complex phenomenon affects metrics of real-world applications. We leverage actual real-world data to study three such cases to understand and quantify complex collective behavior:

Thesis Structure		
Part 1: Temporal and Spatial Properties	Part 2: Synthetic Generation of New Datasets	Part 3: Example Applications & Useful Tools
Chapter 2 Chapter 3	Chapter 4	Chapter 5 Chapter 6

Table 1.1: Overall summary of structure of the thesis

1. **Temporal and Spatial Properties:** The study of statistical properties and models that govern the generation and evolution of human mobility patterns. Evolution of human movement in a city is viewed as a graph over time to study the collective behavior of movements within a city. The static properties of the graph and its variation over time help to design models and understand the temporal and static patterns of real-world human mobility.

   Graphs over time, do not capture the spatial information of mobility. Even for a static graph, the spatial distribution is not encoded in it. We examine the spatial distribution using *Fractal Dimensionality*. The structural information gathered with fractal dimension is devised to understand how efficiently can algorithms perform.

2. **Synthetic Generation of New Datasets:** The study of generating city-scale human mobility data which closely matched properties of real-world data. We look at real-world data sets of human mobility from fifteen cities across the globe. Millions of movements captured every week in each city are important to consider potential scalability issues in data generation, and to discover patterns which may not otherwise be visible in smaller and less granular data sets. We demonstrate the value of large data by quantifying temporal and spatial structure where most of the existing work focused on small networks of several hundreds of mobility data points. Qualitative analysis of the generative models using temporal and spatial properties validates and provides basis to look at real-world applications based on large datasets of human mobility.

3. **Example Applications and Useful Tools:** The study of multiple *what-if* scenarios related to human mobility and making a framework available for the wider research community. We consider two problems – ride pooling and vehicle placement for considering how synthetic data generated can be used to evaluate different *what-if* scenarios. In the vehicle placement case, we also highlight the benefits of using the properties discovered in human mobility patterns. Using the two applications and synthetic generation, we provide a customizable framework capable of learning patterns and providing metrics for quick feedback.

1.1 Scope and Overview of the Thesis

In this work we propose methods to exploit large scale human mobility data to facilitate valuable services for societal good. For the scope of this thesis, *human mobility* refers to movement of masses of humans within cities; movement here could be for any purpose and not restricted to home and office. We study human mobility using real-world data from ride-sharing services. On a broader scale, human mobility may include other forms of movement data from trains, public buses, personal cars, and cell towers. Since some of these modes of transportation (or data collection methods) may be restricted in terms of their market penetration and used by certain demographics, in our work we focus on using data from ride-sharing services since they have enabled an economical and convenient option of transportation for consumers of various demographics. Curiously enough they have disrupted major players in the transportation market around the world.

Emergence of ride-sharing services are transforming human mobility patterns in major cities of the world, and is representative of the mobility phenomena due to its mass adoption. In 2017, Didi reported over seven million ride requests in China. Uber, Lyft, and Ola have invested heavily to penetrate markets in North America, Asia, and other parts of the world. With high penetration rate, there is huge potential for such services to transform urban transportation, public policies, and also provide researchers the infrastructure to derive new insights which would not have been possible before their advent.

Human mobility patterns presents interesting problems for researchers in the areas of computer science, civil engineering, public policy, and machine learning. The primary problem we hope to address in this work is – *whether or not it is possible to model the dynamic spatial and temporal properties of human mobility across cities? And, if so, what societal benefits can be extracted from such models?* In prior work, probability distributions are used to model temporal properties like wait times or spatial properties like distances traveled in the underlying datasets. By formally characterizing temporal properties in this work, we want to understand how they evolve in a city over time and do not consider the waiting or travel times since we are interested to understand patterns at a city-level. Also, by formally characterizing spatial properties we need an aggregate qualitative measure of the distribution of human traces and not an aggregation by finding a singular distribution of distances traveled to derive city-level properties. The foremost challenge is to understand succinct ways to capture patterns observed in large mobility data, and to characterize cities using these patterns. Such patterns should inherently capture mobility dynamics which may be governed by the geographical shape, or cultural trends of a city. From a learning perspective, human mobility also poses challenges due to its highly dynamic nature. We leverage popular generative approaches in machine learning literature along with the properties of human mobility to synthesize comprehensive framework for generation and study of large scale human mobility data.

In this work we consider temporal gaps of less than a second, spatial granularity of less than a meter, characterizations formed by aggregating over fifty million samples covering fifteen cities for a week to model human mobility patterns. None of the prior work summarized in (Alessandretti, Sapiezynski, et al., 2017) has covered similar fine granularity in time and space. By going to such granular levels and vasts amount of data, we have been able to unmask the variability and also capture it across all times of a day including weekdays and weekends. The variability across times and regions is important to capture as it allows the modelling approaches proposed to be applicable to any situation like rainy, or holiday seasons, or anomalies like road closures; the modelling approaches would in not ordinary scenarios be able to capture dynamics of human mobility pattern. Prior works which are restricted in terms of dataset not capturing entire regions or not continuous in time may not easily be generalizable. More details about our dataset are provided in chapter 2. The temporal and spatial characterizations proposed can be applied to any dataset provided the dataset represents a decent percentage of the overall mobility pattern of the city or urban area.

1.2 Summary of contributions

1.2.1 Part 1 – Temporal and Spatial Properties: What patterns govern human mobility patterns?

How can we capture and model the temporal evolution of human mobility? (Chapter 2)

Examining large datasets, its synthesis, and formalizing patterns are critical to characterize dynamic patterns of human mobility for different cities. Even though human mobility patterns are dynamic, e.g. number of people moving from one part of the city to another changes rapidly over time, current instruments (models, metrics) are mainly drawn from static or limited snapshots of data which fail to capture the evolution of phenomena and their fine-grained dynamical properties. This work is based on extensive real world data from ride-sharing and yellow cab services. We look at about fifty million ride requests around the globe. To capture the variation of ride requests we introduced a graph model that captures the temporal attributes of ride requests. We discover that ride requests graphs (RRGs) from this model exhibit the well known *densification power law* property (Jauhri, Foo, et al., 2017) often found in real graphs modeling human behaviors.

How can we understand the geographical spread of human mobility? (Chapter 3)

Here we try to answer how does the spatial distribution of human mobility look like? We treat the distribution of mobility locations as a point-set; representing both pickup and drop-off locations. With this point-set we find a way to characterize their deviation from uniformity. Using the concept of *fractal dimension*, we show how human mobility patterns exhibit *fractal behavior*, referring to the fact that sub-sets of the point-set representing locations of human movement are statistically similar to the whole set (Jauhri, Joe-Wong, et al., 2017). These insights provide a way to characterize spatial structure of the mobility patterns apart from the graph modeling which depicts the temporal characterization.

1.2.2 Part 2 – Synthetic Generation of Datasets: Scalable and efficient approach for synthetic data generation and validation

How can we generate city-scale synthetic realistic datasets of human mobility? (Chapter 4)

Using *Generative Adversarial Networks* and *Graph Generators* we provide a novel model for generating synthetic human mobility data. GANs which are known for modeling natural image distribution by forcing generated samples to be similar to natural images help to capture spatial distribution of ride requests without requiring to embed geographical information about an urban area like water bodies, and parks where ride requests would never originate or end. GANs also allow to simplify the process of running simulations or experimentation by not providing access to real data which would otherwise be a privacy concern but rather allow us to obfuscate the locations of origin, destination, and also time of ride requests. The approach for GANs here involves viewing human mobility as a (temporal) sequence of (spatial) images of origin and destination locations of ride requests used to train the GANs with discretized time buckets as labels. By dealing with large amount of data for many cities and long training times for GANs, we also propose an effective way to parallelize and scale the training runs using cluster of machines on AWS. With such scaling, we highlight how real-time inputs of the mobility patterns phenomenon can be used to predict future city-wide mobility patterns.

How do we validate real and synthetic datasets? (Chapter 4)

For fifteen cities around the globe, we provide comparative analysis of real and synthetic data sets for a week of mobility data based on *densification power law* and *correlation fractal dimension* properties. These statistical properties provide insights for both high and low granularity of time and space human mobility patterns.

1.2.3 Part 3 – Example Applications and Toolkit Framework: How can we use city-scale data for different what-if scenarios?

What are some real-world applications which can benefit from human mobility data? (Chapter 5)

We introduce a rigorous formulation of the dynamic ride pooling problem and suggest several approaches, or methods, for dynamic ride pooling. We explore the design space for different pooling methods, and different configuration of these methods based on trade offs involving several key parameters like maximum number of pooled riders, temporal and distance limits within which a ride should be eligible for pooling. We show dynamic ride pooling can produce societal benefits, in reducing total travel distance and the total vehicle count for a city. Human mobility and transportation is a major issue for many large urban areas in the world. In this work we take a data-driven approach to investigate the subject of real-time dynamic ride pooling.

Reducing wait times to below two minutes is a challenging problem that requires real-time vehicle placement. The real-time nature of this problem introduces two challenges: first, to minimize wait times, drivers should proactively predict where future pickup requests will be located and go to these locations in anticipation of future pickup requests, thus eliminating any travel time to the request location. Yet to accurately predict future pickup requests, drivers must account for not just overall patterns in ride requests, but also real-time temporal and geo-spatial request fluctuations, which are influenced by human mobility dynamics. Second, these vehicle placement decisions must be made quickly and cannot require any significant coordination, e.g., drivers can view this information in their mobile application. We highlight a distributed placement algorithms that allow each driver to independently decide where to go after dropping off a passenger.

How can we use a parameterized framework and associated toolkit to study different what-if scenarios? (Chapter 6)

We provide a framework and associated toolkit to generate city scale urban mobility data for a large number of cities around the globe. The framework is based on spatial and temporal insights without compromising the identity and privacy of both the riders and drivers. Apart from generation of datasets, the proposed framework provides tools to determine whether the generated data depicts normal patterns of urban mobility or not. This validation is done by analyzing the temporal and spatial characterizations of the city from both real and synthetic datasets. Since these patterns vary for each city we also depict how our framework can be extended for any other city not covered in this paper.

Our parameterized framework and toolkit use different generative techniques which provide insights into how and why mobility patterns develop between different areas of a city. Such models are also useful to reason about the structure and its evolution with time. The framework allows to customize geographical space, density, and time for data generation.

With models for space and time characterization and algorithms for pooling and driver placement, ours is a holistic framework to aid intelligent transportation studies and decision making for city planning.

Chapter 2

Temporal Evolution of Human Mobility

How does city traffic evolve with time? How can mobility movement at a macroscopic temporal level be captured and used to summarize city level statistics or properties of human mobility? What does the city level statistics tell us about how human movement? Many prior studies have looked at datasets constrained geographically to regions within a city like downtown, or for time spanning a few hours to derive human mobility behaviors. Given the lack of access of data in continuum for both space and time, it is difficult to derive from such findings concrete characteristics over time across large urban settlements.

In this chapter we analyze human mobility using ride sharing services' data from fifteen cities around the world and observe a surprising phenomenon. For all fifteen cities the mobility data encompasses each city's geographical area for an entire week. To derive city level characteristics, we detail a graph based model constructed from human movements grouped by time, and observe the graph to *densify* over time i.e. the growth in the number of edges to be super-linear in the number of nodes. Even more surprising is the discovery that for any city the densification pattern repeats each day of the week with constant growth factor. We also looked at human mobility data from New York's popular yellow cabs to avoid bias in data towards ride sharing services. In the case of yellow cab data too we observe the densification phenomenon in graphs constructed in a similar fashion as with ride sharing services.

We do not consider data from subways, or city buses in our human mobility study since such services would capture only popular mobility routes which would remain very similar for the purposes of our analysis; bus or subway routes do not change drastically over time. Privately owned vehicle data is also not considered in this work which would have given a much more precise characterizations of every city although we think the insights would not differ by a lot but only be more comprehensive if we had had access to such data. Pedestrian walking data would have also been beneficial for completeness.

Given the patterns observed in the proposed graph model, we connect human mobility properties to social networks where graph models have been studied extensively. We provide a graph generator, that has a simple, intuitive justification of producing graphs exhibiting full range of temporal properties of human mobility observed in cities like densification, and *rich getting richer* (Leskovec, Kleinberg, et al., 2007).

2.1 Related Work

Studies about human mobility patterns have been governed by a variety of data sets. Call detail records, with its temporal sparseness, was used to derive insights about carbon footprints and traffic congestion for metropolitan areas for three major cities in the United States (Becker et al., 2013). Social media data platforms like Instagram and Twitter provide much more fine grained data both spatially and temporally also provide real-time insights about human mobility beneficial to policy makers, advertisers, and city denizens (Rashidi et al., 2017; Tasse and Hong, no date). These works provide evidence on how data sets can be used to correlate or infer about human movement related phenomenons. Our work is focused around providing universal and general laws which under certain constraints hold true for any city for any time.

This work models human movement as a graph by converting group of ride-requests, either pickups or drop-offs, within a geographical area into nodes, and then connecting the nodes with directed edges from pickup node to drop-off node. Multiple ride-requests with same pickup and drop-off nodes will have edge weight of greater than one. Some important prior works on graphs have helped us to model and discover properties for ride requests. There is a class of graphs that models real networks, e.g. social network graphs and publication citation graphs, that evolve over time. These graphs become more and more dense as they evolve in time, i.e. the edge count grows super-linearly relative to the node count growth. This *densification* of the graph can be modeled concisely by a power law relationship between the edge count and the node count. Graphs for many real networks all seem to exhibit this densification power law (DPL) (M. E. Newman, 2005). For this class of graphs it is possible to automatically generate synthetic graphs that exhibit the same densification power law without needing the original data (Leskovec, Kleinberg, et al., 2005). These graphs also exhibit the attribute of having strongly connected subgraphs or communities (Fortunato, 2010). In this work, we have discovered that our graph model of human mobility for a city belongs to this class of graphs. This fact allows us to discover interesting attributes and insights about ride requests and the potential of ride pooling for ride-sharing services.

2.2 Introduction

Recent emergence of ride-sharing services is transforming human mobility and transportation in major cities of the world (Buzzfeed, 2016) with New York completing over seventeen million trips using ride sharing services in January, 2018, double the number of medallion trips (Parrott and Reich, 2018). The growth of ride-sharing has made it easier to aggregate data from multiple cities around the world to understand patterns in human movement.

Many of the human mobility studies are motivated by the desire to improve infrastructure in urban areas like construction of roads, highways, and broadening of lanes. Studies also aim to provide insights on how to improve experience of daily riders by providing optimal and efficient routes, trip planning, or techniques to perform rider pooling and other ways to reduce rider travel time while also minimizing cost. Our study here aims to not just formalize the properties and dynamics of human mobility but also to aid prior work and models to easily analyze costs and benefits of a scheme for improving rider experience. More importantly, we also provide tools by which synthetic data exhibiting real-world properties can be generated for a city while preserving privacy.

We introduce a new graph model for human mobility to capture the temporal evolution of mobility patterns. In the next chapter we discuss the spatial evolution properties, and then in Chapter 4 provide a complete model for synthetic data generation which obeys both temporal and spatial properties. We first start our discussion around human mobility graphs exhibiting the well known *densification power law* property often found in real graphs modeling human behavior and many other naturally occurring social phenomenons.

2.2.1 Human Mobility Dataset

The dataset used in all analysis consists of ride request data taken from services which offer on-demand ride-sharing facilities around the world. We study datasets from fourteen cities from a single source, and for one city, New York, from two sources. Both sources of data span over a week in April, 2016, starting from Friday midnight (GMT). Each ride request in our dataset contains the following attributes:

- Pick-up (or source) & drop-off (or destination) latitudes, and longitudes.
- Pickup & drop-off timestamps.

For example, the following represents one of the million ride requests we looked at:

$$1460478841, 40.674758385, -73.95501347, 1460679891, 40.6413914034, -73.9412089714$$

The first comma separated value is the pickup time, represented in unix time; second value is pickup latitude coordinate followed by pickup longitude coordinate; drop-off unix time; and finally drop-off latitude and longitude.

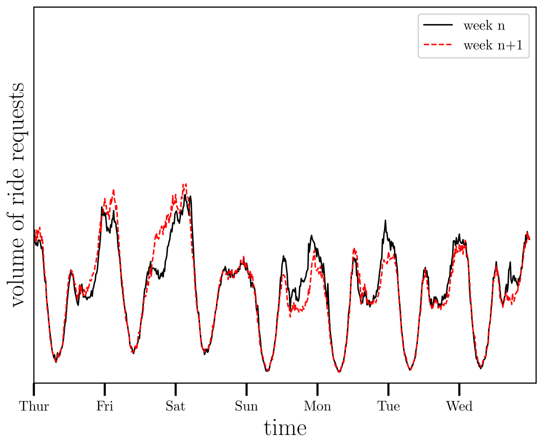
2.3 Temporal Patterns of Human Mobility

Human mobility patterns exhibit significant variability from city to city. Furthermore, with each city, mobility also vary across regions of the city, on different days of the week, and at different times of the day. However, we also observed that, for most cities there is a strong pattern that tends to repeat on a weekly basis. This is shown in Figure 2.1 for two consecutive weeks from two mutually exclusive datasets; both datasets belong to the city of New York.

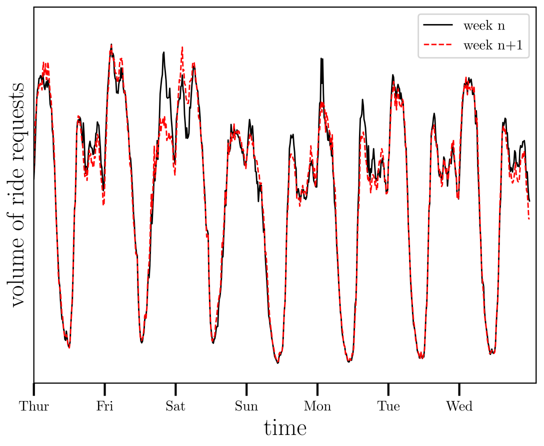
At a macro level temporally, both datasets show a similar pattern in terms of volume. Weekends on average, excluding Sunday, are more busy in comparison to weekdays. The rise and fall of the number of rides temporally overlap for all days in a week in both the datasets. The temporal evolution of the requests is the unknown factor here which we examine in the next section.

2.4 Graph Model

To understand the temporal characterization of a city we introduce a method to construct directed graphs from ride requests. The graph representing ride requests for a specific time period,



(a) Ride-sharing dataset



(b) Yellow cab dataset

Figure 2.1: Temporal distribution of total volume of ride requests for two consecutive weeks from two different data sources in New York City.

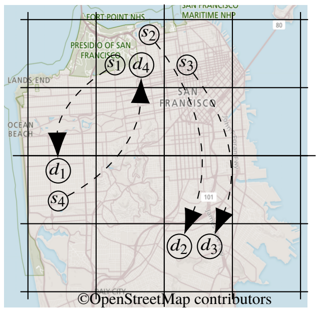
spanning over a few minutes, is referred to as the *Ride Request Graph (RRG)*. Each RRG captures the spatial connections of ride requests across a city within that time period without encoding the geographical distances. The RRG evolves from one time period to the next, to account for new ride requests initiated in the next time period. This evolution of the RRG and the resultant sequence of RRGs capture the temporal aspect of ride requests over many time intervals. For any time snapshot, the RRG does not account for ride requests from the previous time snapshot.

2.4.1 Ride Request Graph Generation

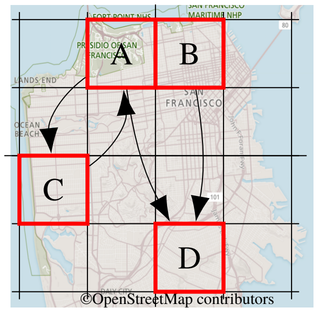
For a given time interval we can generate a RRG representing all the ride requests in that interval. We divide the map of a city into equal sized cells of length ϵ meters. Each cell is considered as a node in the graph only if a source or destination of a ride request falls within that cell. A directed edge connects the source and destination cells of a ride. A directed graph can then be generated to model all the ride requests in that time interval for a given city.

For any ride i, s_i represents the pick-up location and d_i represents the drop-off location. Now consider the ride requests in Figure 2.2a belong to a specific time snapshot. The four ride requests are shown on a gridded map (not drawn to scale). The corresponding RRG in Figure 2.2b is formed by four nodes with node A subsuming s_1, s_2, d_4; node B subsuming s_3; node C subsuming s_4, d_1; and node D subsuming d_2, d_3. All edges in Figure 2.2a have at least unit weights representing single ride requests from source to destination node. Edge weights represent the number of ride requests from the same source and destination nodes, and may not necessarily be the same geographical locations. The term *node* is always used in the context of the RRG graph. We use the term *point* to refer to a specific location defined by its latitude and longitude, which could be the source or destination of a ride request, and is associated with a node in the RRG.

[1]Map data of Figures 2, 3, and 5 are available from OpenStreetMap under the Open Database License and the

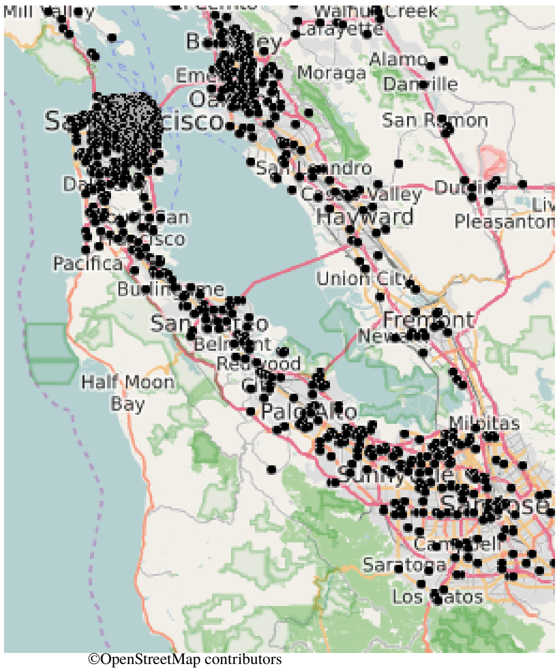


(a) Four ride requests distributed spatially over a map

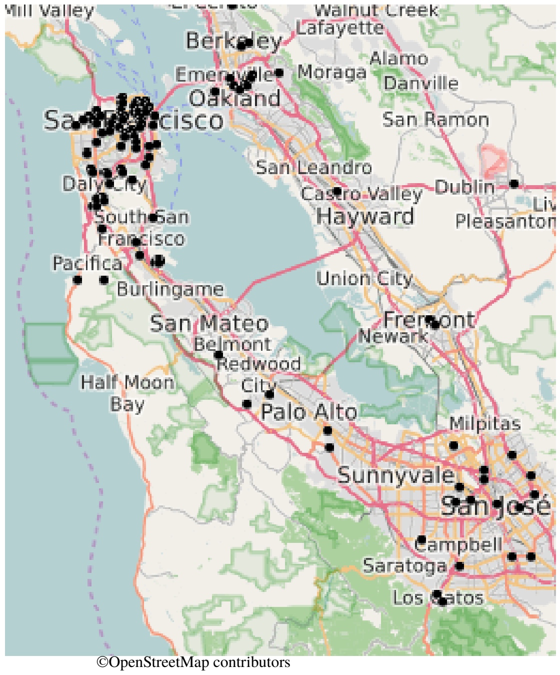


(b) Corresponding Ride Request Graph with four nodes (marked by red boxes) and directed edges.

Figure 2.2: Transformation of ride requests, in a particular time interval, into a directed ride-request graph (RRG).



(a) Distribution of ride requests at rush-hour time, 7pm



(b) Distribution of ride requests at non-rush-hour time, 5am

Figure 2.3: Space-time variability: Each dot represents a source or destination for a ride request in San Francisco[1]

Each ride request is considered independent of any other ride request. In constructing the graph the length of the actual navigation path taken from s to d for a ride request is omitted.

Based on examining the ride request data from the fifteen cities gathered over multiple weeks of Spring 2016, we can make two key observations. For each city, the temporal ride request

cartography is licensed under the Creative Commons Attribution-ShareAlike 2.0. http://www.openstreetmap.org/copyright patterns tend to repeat from week to week, and a typical week of data is sufficient to understand patterns of a city. On the other hand, there is significant variability of ride request patterns from city to city. Furthermore, in each city there is variability across different days of the week, at different times of the day, and across different regions of the city.

Variability in ride request patterns in San Francisco for two snapshots taken at two different times of the day is shown in Figure 2.3. The figure shows the spatial distribution of ride request density over the Bay Area. San Francisco downtown (top left cluster of points) is clearly denser in ride requests. The two figures illustrate two snapshots of ride request density for two different 5-minute intervals one at 7:00pm and the other at 5:00am. The temporal and spatial variations of ride requests can be clearly seen.

2.5 Ride Request Graph Densification

A Ride Request Graph is different from conventional graphs: (1) each node has a geographical area associated with it; (2) RRG is fixed in time but does not evolve with time meaning for every time snapshot a new RRG is generated. Each RRG involves a spatial quantization (like $100m \times 100m$ cells [2]) of the geographical area of the city, and a temporal quantization into sequence of time intervals. In our analysis we use 5-minute intervals for temporal quantization. As a RRG evolves in time, it produces a sequence of RRGs that capture the temporal behavior of ride requests.

As we analyze the RRGs extracted from historical data of ride requests from all the cities, we make an interesting observation about human mobility. *Densification* refers to graphs that evolve in time, and how the edge count grows relative to the growth of the node count. Many graphs modeling aspects of human behaviors, such as social network graphs and publication citation graphs, among others, exhibit densification over time that follows a power law, i.e. the number of edges grows as a power of the number of nodes (Leskovec, Kleinberg, et al., 2007). We have discovered that the RRGs for ride requests exhibit the same power law densification behavior and belongs to this class of graphs.

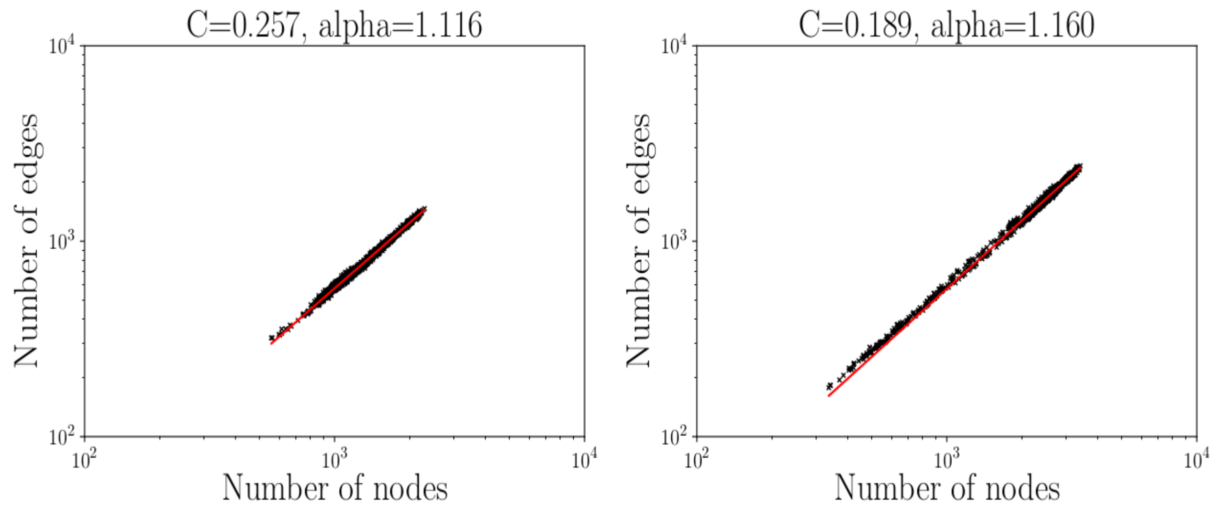
We observe RRGs at different snapshots of time, with each spanning five minutes. For each snapshot, we study the *Densification Power Law* plot (DPL plot) (Leskovec, Kleinberg, et al., 2005) i.e. log-log plot of the number of edges $e(t)$ versus number of nodes $n(t)$.

Figure 2.4 shows the Densification Power Law (DPL) plot for New York for a typical week in 2016. It is observed that for every time interval t:

$$\begin{aligned} e(t) &\propto n(t)^{\alpha} \\ &= Cn(t)^{\alpha} \end{aligned}$$

where $e(t)$ and $n(t)$ are the number of edges and number of nodes respectively, formed by all ride requests occurring in the time interval t. C and α are constants. Number of edges is a good approximation of the number of ride requests since we do not consider edge weights. All cities follow the densification power law but the parameters of the power law vary from city to city. It is important to note that we do not consider the complete graph when plotting any point in

[2]The cell size can be varied based on the dynamics of the city and dataset used.



(a) Ride-hailing dataset for New York City (b) Yellow Cab Dataset for New York City

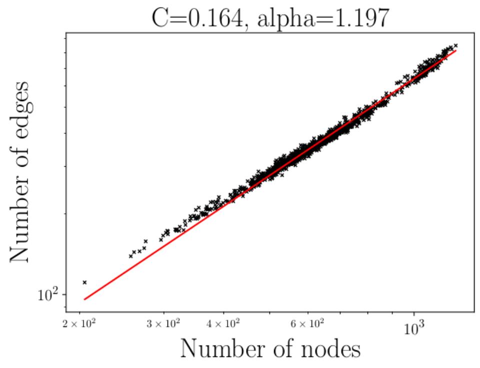
Figure 2.4: DPL plots from real data for New York City from two different sources of data. The red line is the least square fit of the form $y = Cx^{\alpha}$, where y and x are number of edges and nodes respectively. $R^2 \approx 1.00$ for all of them.

Figure 2.4. For every time snapshot t, no historical i.e. ride requests originating before t are aggregated. Only ride requests in five minute interval are used to generate the RRG. This is a distinction from majority of other academic works on graphs and their evolution where all nodes and edges from start till t are captured in the graph. In Figure 2.5 we provide plots for fourteen other cities using one week of data divided over five-minute discreet time snapshots.

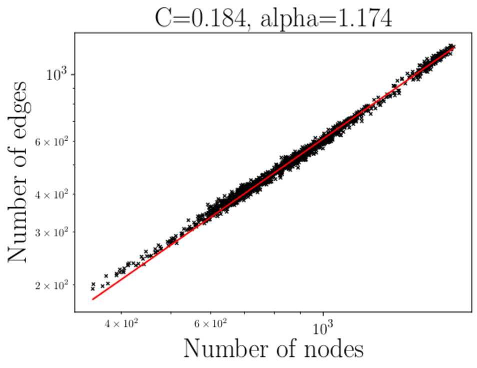
2.5.1 Characteristic Attributes of RRGs

Based on the previous observation it appears that the pattern of ride requests for a city can be characterized concisely by C, and α derived from the power law of RRGs for that city. The exponential α depicts the *densification* of ride requests within a city. This densification factor α can range in value from 1.0 to 2.0. If $\alpha = 1.0$, this means the number of edges is growing linearly with respect to the number of nodes; if $\alpha = 2.0$, then the RRGs become fully connected.

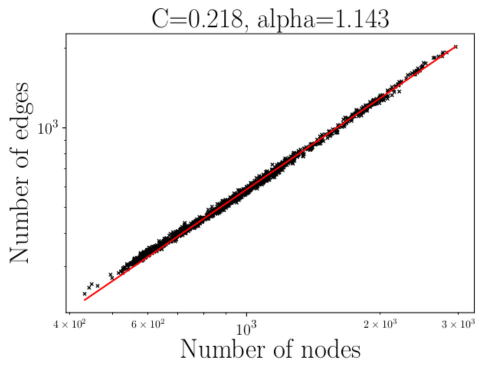
It is interesting to note that both datasets from yellow cab and ride-hailing service exhibit densification factors greater than 1.0. This means the edge count is growing super-linearly to the node count. We speculate this demonstrates the human tendency towards the creation of clustered/connected communities, perhaps reflecting the *small world* effect (Watts and Strogatz, 1998). When compared across cities, a city with higher densification factor would mean higher rate of growth in the number of edges in comparison to number of nodes; implying small set of regions or communities are present which tend to experience higher volume of ride requests. Prior work (Gonzalez et al., 2008) had highlighted repetition of patterns in human movement but with densification factor we not only confirm there are certain geographical pockets within



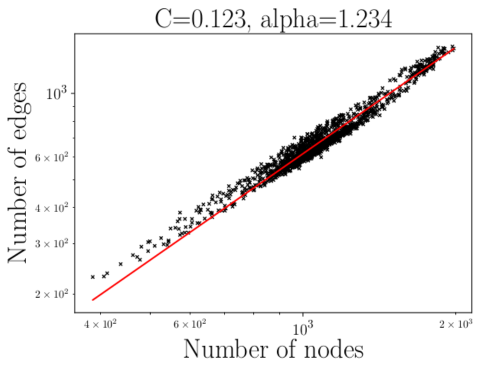
(a) Boston



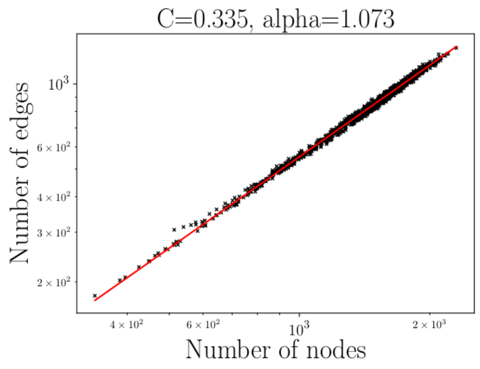
(b) Chicago



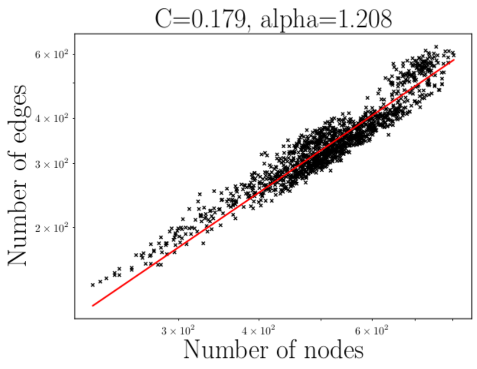
(c) London



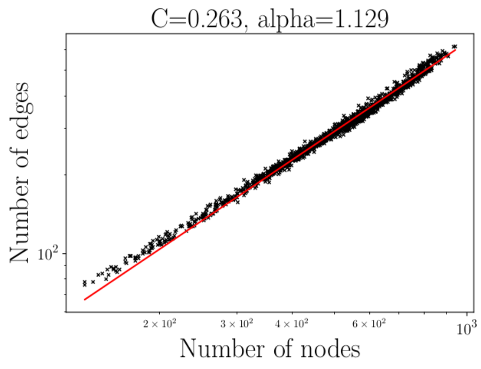
(d) Los Angeles



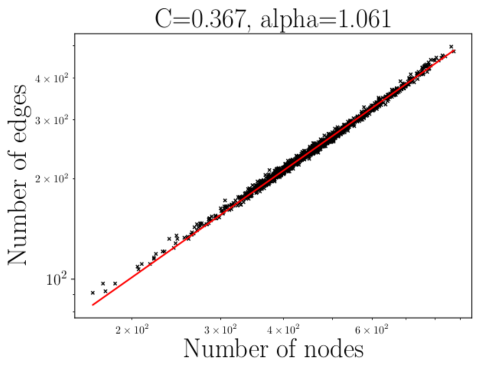
(e) Mexico City



(f) Miami

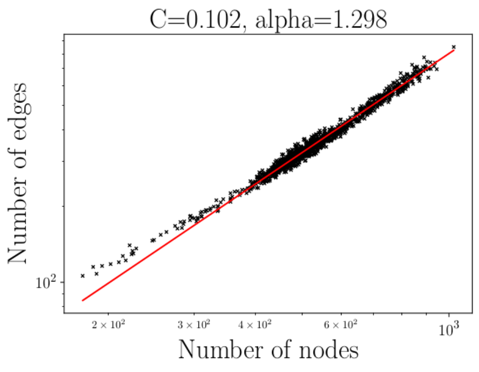


(g) New Delhi

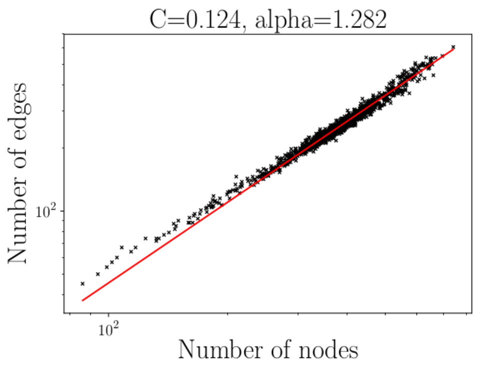


(h) New Jersey

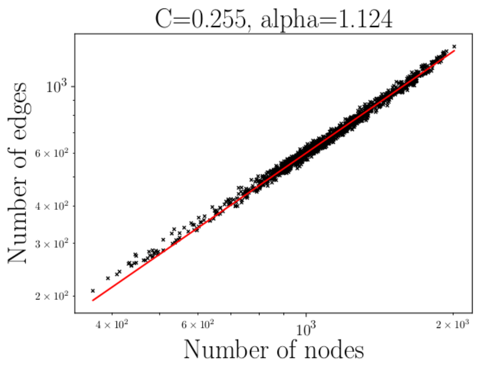
Figure 2.5: DPL plots from real data for multiple cities. The red line is the least square fit of the form $y = Cx^{\alpha}$, where y and x are number of edges and nodes respectively. $R^2 \approx 1.00$ for all of them.



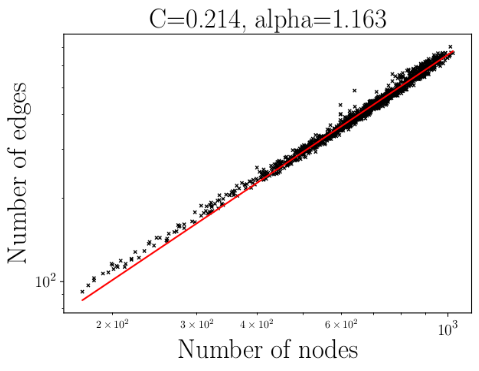
(i) Paris



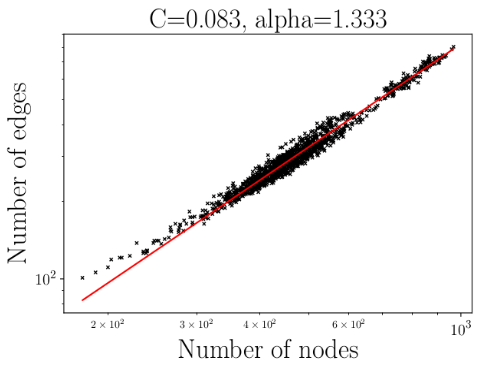
(j) Rio De Janeiro



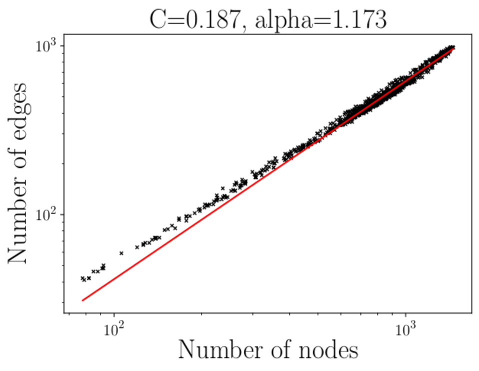
(k) San Francisco



(l) Sao Paulo



(m) Toronto



(n) Washington

Figure 2.5: DPL plots from real data for multiple cities. The red line is the least square fit of the form $y = Cx^{\alpha}$, where y and x are number of edges and nodes respectively. $R^2 \approx 1.00$ for all of them.

which people tend to travel more often but we are also able to distinguish qualitatively how these patterns differ from city to city.

In Figure 2.4, the difference in the values of densification factor, α, for the two datasets varies from 1.116 to 1.16; proportional to the popularity of each of the two services at the time of collection of dataset. This α could look different if computed exclusively using data from privately owned vehicles in New York city.

DPL graphs exhibit a fascinating attribute. (Leskovec, Kleinberg, et al., 2005) and (Chakrabarti and Faloutsos, 2012) have shown that for graphs that evolve according to the densification power law, it is possible to automatically generate these graphs that exhibit densification. This implies that we can automatically generate RRGs that exhibit very similar densification factor as that of RRGs extracted from real data for any city.

2.6 Degree Distribution Over Time

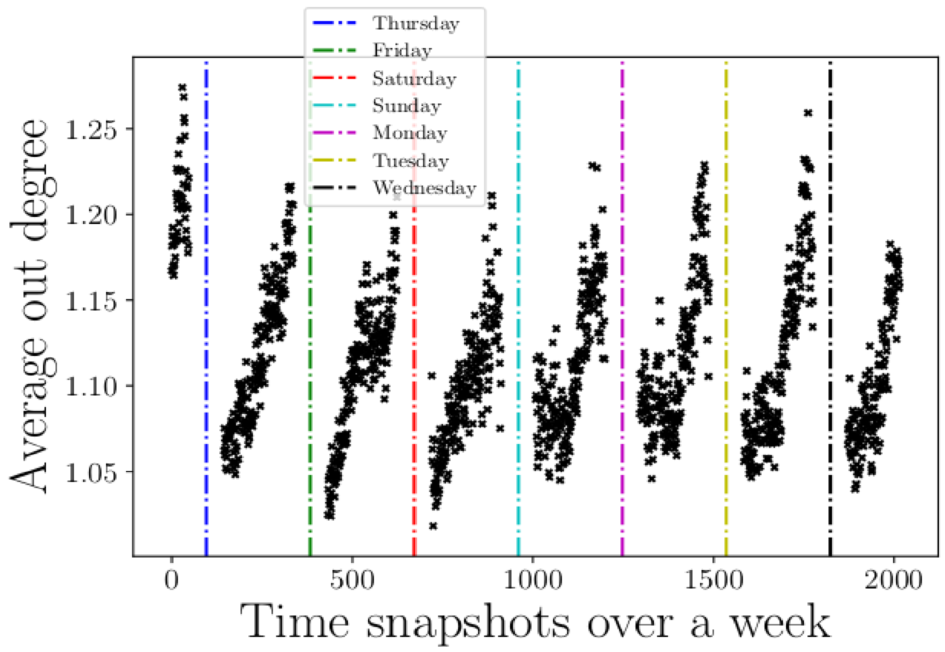
Given the observation around the densification of human mobility patterns, we now discuss the distribution of node degree over time to understand the reason for the densification. It is important to reiterate here that the DPL property observed in Figure 2.4 exists when every graph generated uses mobility data only with the defined time snapshot interval of five minutes and not continuously aggregating over multiple time snapshots.

Based on previous observations around *densification* the hypothesis is that urban human mobility patterns tend to create specific communities which lead to densification. Although, the prior analysis does not capture the reason for the densification. Previous studies have highlighted that *effective diameter*, defined as the minimum number of hops in which 90% of all connected pairs of nodes can reach each other, reduces with densification of graph. The reduction of effective diameter, a concept more commonly known as *shrinking diameters*, is attributed to how degree sequence evolves over time, and not related to how edges connect with nodes over time (Leskovec, 2008). For this reason we leave out any discussion around effective diameter here but we did observe its decline over time in the generated ride requests graphs.

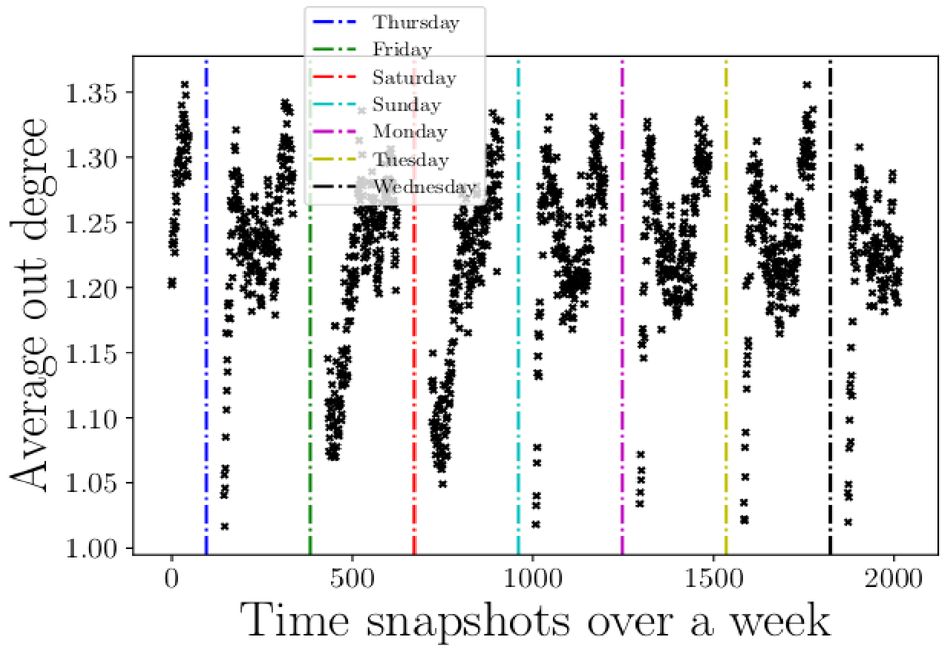
We analyze the case where the degree distribution of graphs over time exhibits a power law, and it maintains constant power law exponent γ over time, then we show that for $1 < \gamma < 2$ we obtain the *Densification Power Law* exponent:

$$\alpha = 2/\gamma \tag{2.1}$$

In this case the Densification Power Law is the consequence of the fact that a power law distribution with exponent $\gamma < 2$ has no finite expectation (M. E. Newman, 2005). This implies that the mean on a sample may be small, although it would at times vary due to very large values. Thus the large fluctuations in the value of mean lead to the conclusion of no finite expectation. In our case, we observe average degree grows with the number of samples i.e. with more nodes or equivalently increase in human mobility activity; this is highlighted in Figure 2.6. On the other hand, the power law degree exponent is constant over time.



(a) Ride-sharing dataset



(b) Yellow cab dataset

Figure 2.6: Average node degree plots using New York's ride-hailing and yellow cab datasets for a week. For most days there is a rise in the mean with increase in human activity. For yellow-cab dataset there is a slight decrease in mean during mid-day; we suspect this is due to an inherent property of its ridership.

2.6.1 Constant degree exponent over time

Here we discuss the case where the RRG over time snapshots maintains a power law degree distribution with a constant exponent γ. Power law distribution $p(x) = cx^{-\gamma}$ with exponent $\gamma < 2$ has infinite expectation i.e., as the number of samples increases, the average also increases. Assume the exponent (slope) of the degree distribution does not change over time, then the relation between the *Densification Power Law* exponent and the *Degree Distribution* exponent over time is as follows:

Theorem. *In a temporarily evolving graph with a power law degree distribution having constant degree exponent γ over time, the DPL exponent α is:*

$$\begin{aligned} A &= 1 \quad \textit{if} \quad \gamma > 2 \\ &= \frac{2}{\gamma} \quad \textit{if} \quad 1 \leq \gamma \leq 2 \\ &= 2 \quad \textit{if} \quad \gamma < 1 \end{aligned}$$

Proof (taken from Leskovec, 2008): Assume that at any point time t the degree distribution of an undirected graph G follows a power law. This means the number of nodes N_d with degree d is $N_d = cd^{-\gamma}$, where c is a constant. Now assume that at some point in time as the graph grows the maximum degree of the graph is d_{max}. As the graph grows we will let $d_{max} \to \infty$. Using the previous power law relation, we can calculate the number of nodes N and the number of edges E in the graph:

$$N = \sum_{d=1}^{d_{max}} cd^{-\gamma} = \int_{d=1}^{d_{max}} d^{-\gamma} = c\frac{d_{max}^{1-\gamma} - 1}{1-\gamma}$$

$$E = \frac{1}{2}\sum_{d=1}^{d_{max}} cd^{-\gamma}d = \int_{d=1}^{d_{max}} d^{1-\gamma} = c\frac{d_{max}^{2-\gamma} - 1}{2-\gamma}$$

We let the graph grow so $d_{max} \to \infty$. Then the DPL exponent α is:

$$\alpha = \lim_{d_{max}\to\infty} \frac{\log E}{\log N} = \frac{\gamma \log d_{max} + \log \|d_{max}^{2-\gamma} - 1\| - \log \|2-\gamma\|}{\gamma \log d_{max} + \log \|d_{max}^{1-\gamma} - 1\| - \log \|1-\gamma\|}$$

Note that the degree distribution exponent is γ so we also have the relation $\log c = \gamma \log d_{max}$. Now we have three cases:

1. Case 1: $\gamma \geq 2$; no densification:

$$a = \frac{\gamma \log d_{max} + o(1)}{\gamma \log d_{max} + o(1)} = 1$$

2. Case 2: $1 < \gamma < 2$ is the interesting case where densification arises:

$$a = \frac{\gamma \log d_{max} + (2-\gamma)\log d_{max} + o(1)}{\gamma \log d_{max} + o(1)} = \frac{2}{\gamma}$$

3. Case 3: $\gamma \leq 1$; maximum densification – the graph is basically a clique and the number of edges grows quadratically with the number of ndoes:

$$a = \frac{\gamma \log d_{max} + (2-\gamma)\log d_{max} + o(1)}{\gamma \log_{d_{max}} + (1-\gamma)\log d_{max} + o(1)} = 2$$

This highlights that for cases when $\gamma < 2$ we observe densification. This densifications means that the number of edges grows faster than the number of nodes. For degree distribution exponent the densification means the tail of the degree distribution has to grow i.e. hash to accumulate more mass with high human mobility activity in busier time snapshots. Here this is the case since power law distributions with exponent $\gamma < 2$ has no finite expectation. In the case of degree distribution this means that the expected node degree grows as the graph accumulates more nodes.

Graphs can also show densification with degree distribution $\gamma > 2$.

Theorem. *In a time evolving graph of N nodes that changes according to Densification Power Law with $\alpha > 1$, and has a power law degree distribution with exponent $\gamma(N) > 2$, the degree exponent changes as*

$$\gamma(N) = \frac{4N^{\alpha-1} - 1}{2N^{\alpha-1} - 1}$$

Proof: Refer to (Leskovec, 2008).

2.6.2 Measurement from real data

For collecting degree distribution from real data, we take a RRG for every time snapshot and compute its *Degree Distribution* exponent using maximum likelihood estimation. Since the densification factor α is computed once after analyzing the entire dataset for a week (or 2015 or five minute time snapshots), we compare the analytical degree exponent computed using Eq 2.1 with average degree distribution exponent observed using real data over all time snapshots. In Table 2.1 we observe the analytical model closely approximate the degree distribution exponent from experimental data. In Table 2.1 we provide the node length for which the experimental degree exponent was closest to theoretical degree exponent. As one would vary the node length (ϵ), the node degree exponent (γ), and DPL exponent (α) would also change. The highest variation between theoretical and experimental degree exponent was when we considered $\epsilon >= 300$ meters. This variation could be attributed to a slightly varying α because with nodes of larger area in densely populated cities may not necessarily result in constant increase in number of edges. Cities like Miami, New Jersey, and Los Angeles have a very thick mass in the middle of their DPL plots (Figure 2.5) implying there exists uniform distribution of geographical pockets with high human mobility activity.

2.6.3 Community Effect

To further concretize the rich getting richer concept and the surprising phenomenon of constant densification for a week, we introduce a new term *community coefficient* defined as follows:

Definition 2.1. *Community Coefficient (ζ): is the ratio of the average out degree and average in degree of a graph.*

Degree Distribution Exponent				
City	ϵ (m.)	Degree exponent from real data (mean)	Theoretical degree exponent	α
Boston	100	2.047	2.164	1.197
Chicago	100	1.860	1.703	1.174
London	200	2.042	2.231	1.143
Los Angeles	250	1.960	1.693	1.181
Mexico City	100	1.849	1.864	1.073
Miami	300	1.822	1.656	1.208
New Delhi	250	1.959	1.772	1.129
New Jersey	100	2.178	2.371	1.101
New York (ride-hailing)	100	1.853	1.792	1.116
New York (yellow cab)	50	1.719	1.724	1.16
Paris	300	2.037	2.084	1.298
Rio De Janeiro	250	2.002	2.105	1.282
San Francisco	100	1.819	1.780	1.124
Sao Paulo	200	2.013	2.211	1.163
Toronto	250	2.083	2.069	1.333
Washington	100	1.928	1.704	1.173

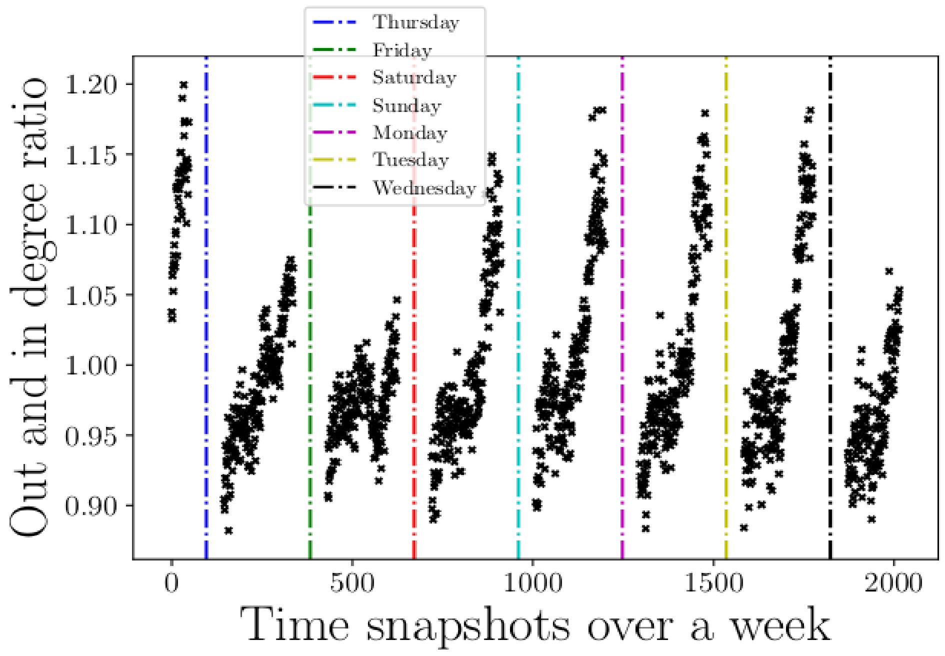
Table 2.1: Degree exponent comparison for fifteen cities using real-data and theoretical method.

Given, there exists DPL property between the number of edges and nodes, ζ depicts two scenarios of community effect:

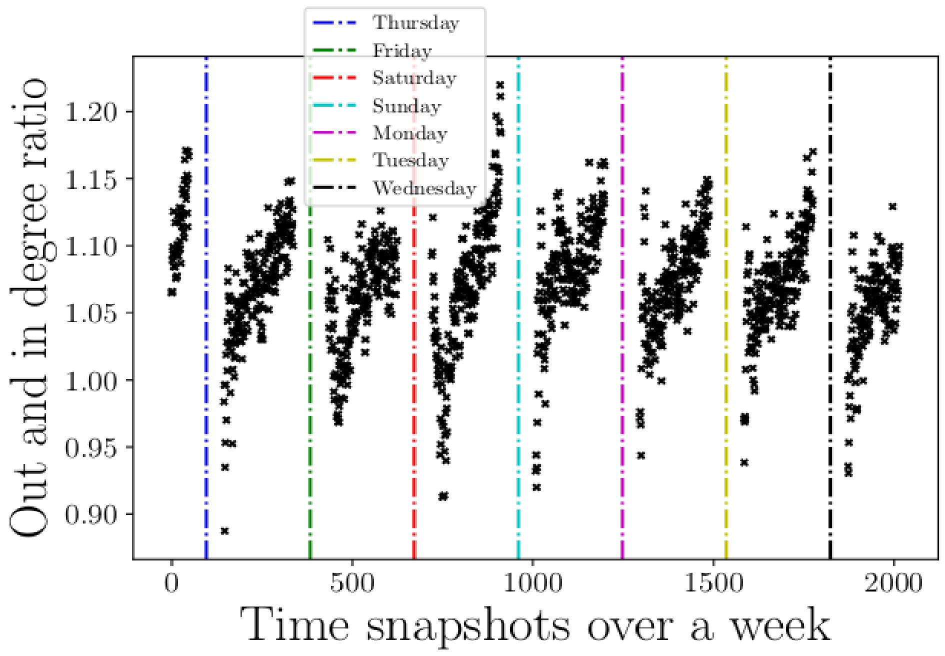
- $\zeta >= 1$ *signifies that a small set of nodes have higher number of out-going edges directed towards a larger set of nodes.*
- $\zeta < 1$ *signifies that a small set of nodes have higher number of incoming edges from a larger set of nodes.*

Intuitively, when $\zeta \geq 1$ it signifies that the mobility patters are from congested to less congested areas. Alternatively, this can be explained by the fact that for offices and downtown areas there will be an influx of people. When $\zeta < 1$ there will be an out-flux; mobility in the morning is more likely to see patterns from sparse areas (residential) to denser areas leading to $\zeta < 1$. This metric provides a good measure of the sort of patterns which we had anticipated – densification and the movement pattern repeats itself.

For every time snapshot we compute the *community coefficient*. The rich getting richer attribute is observed in the form of $\zeta \geq 1$ with time or as human mobility increases. This is shown in Figure 2.7 where mostly in the evenings $\zeta \geq 1$; signifying a larger movement from downtown areas, offices, or other small communities to residential localities which are more spread out geographically in comparison to denser areas of high human activity. Every gap in the graph indicates night hours which skipped from midnight to seven in the morning (local time).



(a) Ride-sharing dataset



(b) Yellow cab dataset

Figure 2.7: Community Coefficient variation over a week in New York.

2.7 Proposed Model

We have now seen the densification properties and how they develop over time. Assuming we have the points distributed over space, our proposed model would connect these points such that they become concrete ride requests. Our graph generator model is inspired from the configuration model (M. Newman, 2018) and forest fire model (Leskovec, 2008) which also aim to generate graphs having DPL property. Algorithm densPropGen allows us to capture the *densification* property observed in RRGs from real data. Algorithm 1 requires three parameters: (1) number of points to generate m; (2) the probability of choosing a point which has not been visited before p; (3) number of outlinks n_{edges} from a source point is defined by geometrically distributed random number with mean $1/q$. $1 - p$ is the probability of choosing a previously visited source point as destination, variation of the preferential attachment technique described in (Chakrabarti and Faloutsos, 2012), which captures the idea of *rich getting richer*.

Algorithm 1 To capture densification properties. Inputs: $m \in \mathbb{N}$, p & $q \in [0, 1]$

```
procedure densPropGen(m,p,q)
    let $M = [0, \ldots, m-1]$
    let $R = []$
    let $rides = []$
    while length(M) > 1 do
        $s = uniformRandomChoice(M)$
        remove$(M, s)$
        $n_{edges} = geometricRandom(q)$
        for each edge $e \in [0, n_{edges})$ do
            if $uniformRandom() < p$ then
                $d = uniformRandomChoice(M)$
                remove$(M, d)$
            else
                $d = uniformRandomChoice(R)$
            end if
            $r =$ connect$(s, d)$
            add$(rides, r)$
        end for
        add$(R, s)$
    end while
    return $rides$
end procedure
```

In Algorithm 1, *uniformRandomChoice* uniformly at random selects a point from the set points. *geometricRandom* generates values from a geometrically distributed random variable with success probability q. *uniformRandom* generates a uniformly random value $\in [0, 1)$. In Figure 2.8 the DPL plot is shown with random nodes without any repetition; this is important because by having the underlying graph node data to exist with repeats can lead to a higher probability of their being DPL with exponent > 1 than without repeats. Hence, we remove this bias from the underlying data to highlight that the proposed graph generator can still generate graphs with the required α.

In Figure 2.9 we show the degree distribution of Algorithm 1 generated wherein few nodes have a high node degree but majority of them have low node degree. To summarize, the proposed

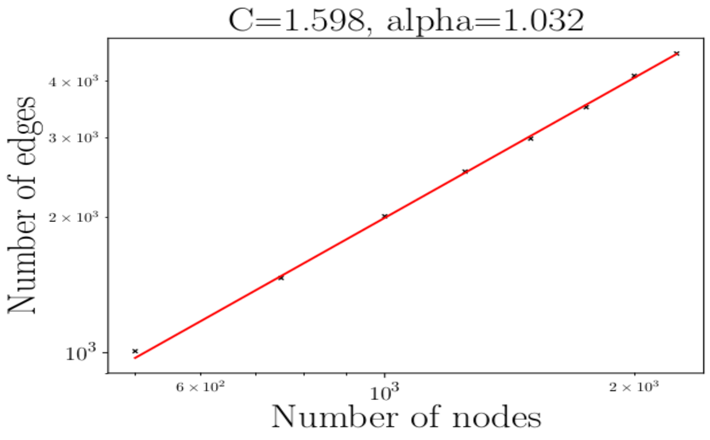


Figure 2.8: Degree Distribution using *densePropGen*

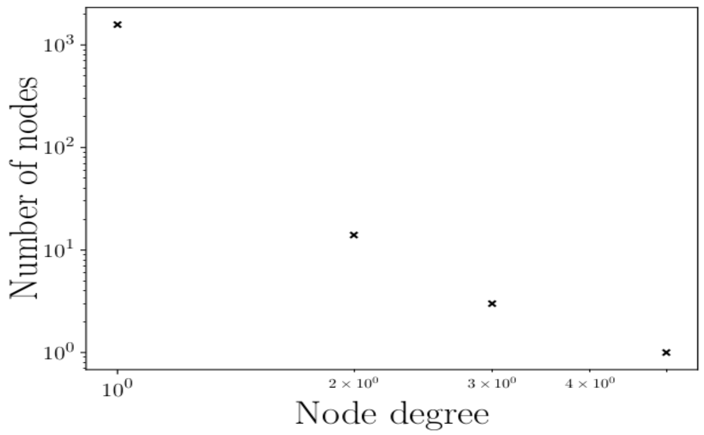


Figure 2.9: DPL with *densePropGen* with $q = 0.5$, $p = 0.99$

model is able to produce graphs qualitatively matching their properties. We establish this by simulation and provide intuition for why these properties arise.

- Heavy-tailed in-degrees: Our model has a rich getting richer flavor; the nodes within the set R have a high likelihood to be chosen with in-degree edges from a new node.
- Communities: The model also has a copying flavor; a new node has higher likelihood of connecting with richer nodes in R than other nodes.
- Heavy-tailed out-degrees: The geometric distribution guarantees (Leskovec and Faloutsos, 2006) that there exists a heavy tailed distribution of out-degrees.
- Densification Power-law: A new node will have a lot of links with community nodes; and a few links with nodes outside of community set of nodes. This is analogous to the community guided attachment although without an explicit set of communities.

2.8 Discussion

Our analysis of human mobility highlights that the phenomenon of rich getting richer is observed in almost all cities around the world. The characterization techniques proposed here can be used to understand real-time trends for different applications. The main findings and contributions are as follows:

- Densification Power Law exists in graphs constructed without any history. Surprisingly the growth in these Ride Request Graphs is constant and varies along the slope i.e. α for an entire week. This implies we can easily generate synthetic RRGs which can closely imitate properties of real RRGs and use those to qualitatively assess structural differences across cities like how dense are communities in a city.
- We prove that the primary reason for DPL is node degree distribution exponent (γ). For any city, γ remains quite stable throughout the week and can be analytically computed from α.
- *Community Coefficient* metric to capture the directionality of flow of community pattern is proposed. Apart from validating the small world phenomenon, it helps to understand whether human mobility directionality is from sparse areas towards dense communities or in the other direction.
- Our proposed model for graph generation uses only a couple of parameters for generating RRGs with properties similar to those observed with real-data. It selects a destination node without any measure of distance. This is because the generator is only concerned with modeling how the connections between the nodes have to be made, the modeling of distribution of nodes (pickup and drop-off points) will be addressed in chapter 4.

Chapter 3

Spatial Distribution & Evolution of Human Mobility

How do we formally interpret the spatial distribution of human mobility data? Given the dynamic nature of the human mobility phenomenon, is there a way to characterize the distribution of points, like locations between which individuals tend to travel, for any urban area? Also, how does the spatial distribution vary with time? With increase in human mobility during certain times of a day, how does the spatial distribution of geographical coordinates representing drop-off locations differ from pickup locations? In the previous chapter we looked at the temporal evolution of human mobility in the form of graphs. Here we shift focus to the spatial distribution and its variations with respect to time. In RRGs, the temporal variations was with respect to how the connections between locations were evolving; here it is with respect to spatial distribution.

3.1 Introduction

We use the proposed *Ride Request Graph* to study the dynamics of human movement with respect to time. The dynamics included nodes signifying geographic locations and how nodes would densify with increase in edge degree relative to time. In the graph itself, we do not address how separated or dispersed are the nodes relative to each other; how does the spread of nodes change from one time snapshot to the subsequent ones. Note the edge weight in the RRG represented the number of ride requests having the same origination and destination nodes; the underlying metric of how far are two nodes was not being captured in the graph or the high level statistic, *Densification Power Law*.

The spatial properties proposed in this chapter provide a statistical model to understand the evolution of human mobility across geographical space. By studying *how the spatial connections* are made, using RRGs, combined with understanding of *spatial distribution of origin and destination points* for any time snapshot and its evolution with time, we would have most of the tools to formally characterize the pattern of human mobility in both temporal and spatial dimensions for any urban region.

3.2 Past work on spatial distribution of human mobility

Studies such as (Gonzalez et al., 2008) and (Lu et al., 2013) aim to derive spatial probability distribution of human trajectory using mobile phone traces, and conclude that human mobility patterns show significant reproducible patterns. Power law distribution are observed from analyzing trip lengths (Hasan et al., 2013) suggest a lot of trips tend to be short. Analysis using six million geotagged Twitter data (Jurdak et al., 2015) find the gyration radius to be approximated by double power-law with sub-linear growth. Later studies like (Alessandretti, Aslak, et al., 2020) provide a model to generate synthetic traces using a probabilistic model.

In most prior literature, the focus has been spatial traces of individuals and combining those to infer traces of human mobility as a whole. These analysis are relevant for understanding human mobility pattern. The missing piece here is that at any given point in time what is the spatial distribution of human movement as a whole for the entire city or geographical area of interest. This distinction here is critical to make collective decisions by public authorities or private entities in the transportation domain. As a concrete example, if we know that the spatial distribution of rides originating or terminating in any particular area of a city would look like at any given point in time, we can:

1. Make a better assessment around how to divert traffic in a dense area.
2. Schedule public transport more frequently in some area at certain times and less frequent in other areas.
3. Incentivize drivers with guarantees to pool riders.
4. Anticipate and intervene to avoid pile-up of people wanting to move out of an area.

These are just some of the many real-world examples which can be done with ease if such information or signals are available at macroscopic and microscopic levels in geographical and temporal dimensions. We illustrate some of these with two specific example applications in Chapter 5, i.e. strategic ride pooling and intelligent vehicle placement.

3.3 Introduction to fractals

Here again we analyze real data from multiple cities; we look at pick-ups and drop-offs independently. Since we are only concerned with the spatial distribution and not how the pick-up points connect with the drop-off points, we treat them independent of each other but would find surprising correlation between them using *fractal dimension*.

The term *fractal* was coined by Benoit B. Mandelbrot to describe objects with anomalous dimension. Fractals are linked with self-similar structures which originate by repetition of a given operation over and over again – on ever smaller scales. Let us take the classical example of such a repetitive construction called the *Koch curve*. Start with a segment of straight line and raise an equilateral triangle over its middle third. The result is called the *generator*. Note that the length of the generator is four-thirds the length of the initiator. Repeating once more the process of erecting equilateral triangles over the middle thirds of straight line segments results in Figure 3.1. The length of the fractured line is now $\frac{4}{3}^2$. Iterating the process infinitely many times results in a

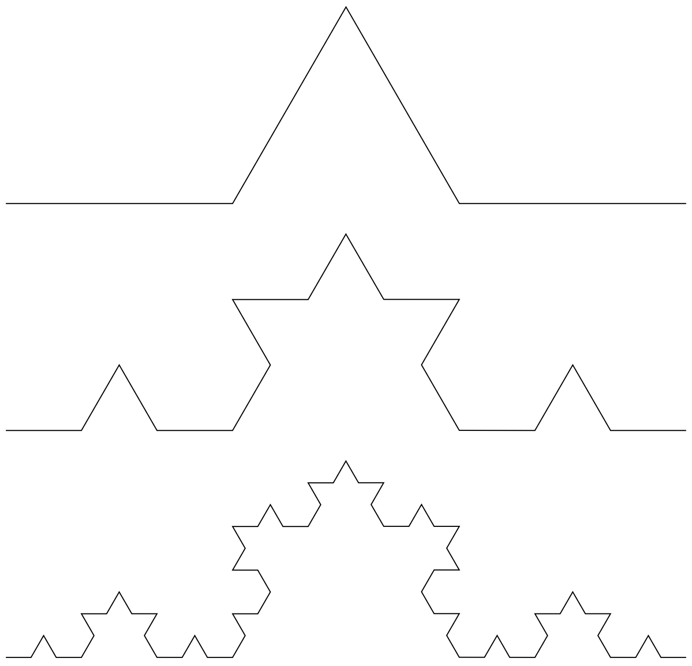


Figure 3.1: Koch Curve

curve of infinite length, which although everywhere continuous is nowhere differentiable.

Similarly, applying the Koch generator to an equilateral triangle results in a Koch snowflake. How long is its perimeter? After n iterations it has increased $\frac{4}{3}^n$ -fold over the perimeter of the initial triangle. Thus, as n approaches infinity, the perimeter becomes infinitely long. To characterize the perimeter's size, we can therefore no longer use its length.

For a smooth curve, an approximate length $L(\epsilon)$ is given by the product of the number N of straight line segments of length ϵ needed to step along the curve from one to the other and the length $\epsilon : L(\epsilon) = N \times \epsilon$. As the step size ϵ goes to zero, $L(\epsilon)$ approaches a finite limit, the length L of the curve. That is not true for fractals. The product $N \times \epsilon$ diverges to infinity because as ϵ goes to zero, we enter finer and finer wiggles of the fractal. However, asymptotically, this divergence behaves according to a well-defined homogeneous power law of ϵ. In other words, there is some critical exponent $D_H > 1$ such that the product $N \times \epsilon^{D_H}$ stays finite. For exponents smaller than D_H the product diverges to infinity while for larger exponents the product will tend to zero. This critical exponent is called the *Hausdorff dimension* after the German mathematician Felix Hausdorff.

Another example of a fractal is a *Sierpinski Triangle* as shown in Figure 3.2. The first triangle in Figure 3.2 is an equilateral triangle; to construct the second triangle, an upside-down equilateral triangle with half the side length of the first triangle is removed. The result are three half-size triangles; the second figure. Repeating the process on the remaining triangles leaves, after n iterations, $N = 3^n$ triangles of side length $r = r_0(2^{-n})$. The result are smaller triangles which look like replicas of the whole triangle. The characteristic property of fractals is this self-similarity property: smaller parts of fractal are similar to the whole fractal. After numerous recursive iterations of a Sierpinski triangle, it becomes evident that it is not a 1-dimensional Euclidean object since it has infinite-length perimeter; it is also not a 2-dimensional Euclidean

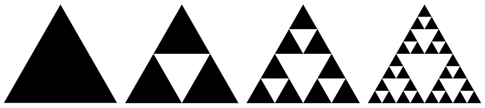


Figure 3.2: Sierpinski triangle shown for four recursive steps.

object since it has zero area (Belussi and Faloutsos, 1995). Instead, we characterize these shapes with *fractal dimensions*. The fractal dimension also indicates whether point-sets exhibit self-similarity.

3.4 Fractals and Ride Requests

Intuitively, a set of points is a fractal if it exhibits self-similarity within a range of scales. It has been observed that real datasets of point-sets representing road intersections of Montgomery county, and Long Beach county in California exhibit properties of self-similarity (Proietti and Faloutsos, 1999). In our problem we are also dealing with a natural phenomena involving human behaviors, i.e. human mobility in urban environments. Characterizing human mobility patterns using fractals provides a succinct description which could be leveraged in a variety of applications and their analyses.

3.4.1 Fractal Dimension

The *Hausdorff fractal dimension* can be defined as follows – divide the $n-$dimensional space into (hyper-)cubic grid cells of side ϵ. Let $N(\epsilon)$ denote the number of cells within which one or more points exist. Then the *Hausdorff fractal dimension* or box-counting dimension D_0 is defined as:
Definition 3.1. *Hausdorff Fractal Dimension For a given point-set in an $n-$dimensional space which has fractal property in the range of (ϵ_1, ϵ_2), its Hausdorff Fractal Dimension D_0 for this range is measured as:*

$$D_0 = -\frac{\partial \log N(\epsilon)}{\partial \log \epsilon} = constant \quad \epsilon_1 < \epsilon < \epsilon_2$$

Definition 3.2. *Generalized Fractal Dimension For a point-set that has self-similarity in the range of (ϵ_1, ϵ_2), the generalized fractal dimension D_q is defined as:*

$$D_q = \frac{1}{q-1} \frac{\partial \log \sum_i p_i^q}{\partial \log r} = constant \quad q \neq 1, \quad for \quad \epsilon_1 < \epsilon < \epsilon_2$$

where p_i denotes the probability of a point occupying $i-$th cell.

Given the definition of generalized fractal dimension, notice the following:

1. for $q = 0$ we have the *Hausdorff fractal dimension* D_0.
2. for a set of points in some high dimensional space to be strictly self-similar like the Sierpinski triangle, it's important to have $\forall q, D_q = D_0$ (Schroeder, 2009) over all or a range of scales.

Fractal dimension can be measured in multiple ways using generalized fractal dimension (Belussi and Faloutsos, 1995). Point-sets for which D_q is not a constant are referred to as *multifractals*. It provides a quantitative measure of self-similarity for a point-set. We shall focus on one specific measure of fractal dimension which relates to the estimation of spatial queries; specifically queries like how many points are present in a cell. Points in our context represent pickup or drop-off locations. Again, consider a two dimensional space divided into square grid cells of side ϵ, similar to nodes of the *Ride Request Graph*,

Definition 3.3. ***Correlation fractal dimension:*** *For a point-set that has self-similarity in the fractal range (ϵ_1, ϵ_2) the correlation fractal dimension D_2 is defined as:*

$$D_2 := \frac{\partial \log \sum_i p_i^2}{\partial \log \epsilon} = constant \qquad \epsilon \in (\epsilon_1, \epsilon_2) \tag{3.1}$$

For a point-set to be self-similar, in the range (ϵ_1, ϵ_2), we observe the plot of $\log \sum_i p_i^2$ versus $\log \epsilon$ must be a straight line, implying self-similarity in that range, i.e. the fractal range (Belussi and Faloutsos, 1995). The slope of the line is equal to the correlation fractal dimension D_2.

3.4.2 Neighbor Approximation Using Fractal Dimension

We are primarily interested in approximating queries such as what is the average number of neighboring source or drop-off points which fall within some neighborhood geographically constrained by a shape, like square or circle, of a certain length. Let $\overline{nb}(\epsilon')$ denote the average number of points within an enclosed square $\square$ of radius ϵ' where $\epsilon' \in (\epsilon_1, \epsilon_2)$ for which we observe the constant fractal dimension as in 3.1. Using Equation 3.1, we can say the sum of squared $S_2(r)$ occupancies for ϵ sized cell follows the relation:

$$S_2(\epsilon) \propto \epsilon^{D_2} \tag{3.2}$$

Also, proportion of pairs within an Euclidean distance or more commonly known as *Correlation Integral* can be defined as:

$$C(\epsilon) \equiv \frac{\sum \text{unique pairs of points within } \epsilon \text{ distance}}{N \times (N-1)/2} \tag{3.3}$$

Using equations 3.2 and 3.3 we provide the follow Lemma:

Lemma 1. *Given a point set $\mathcal{P}$ and sum of the squared occupancies $S_2(\epsilon)$ on a grid with cell of side ϵ we have:*

$$C(\epsilon) \propto S_2(\epsilon) \tag{3.4}$$

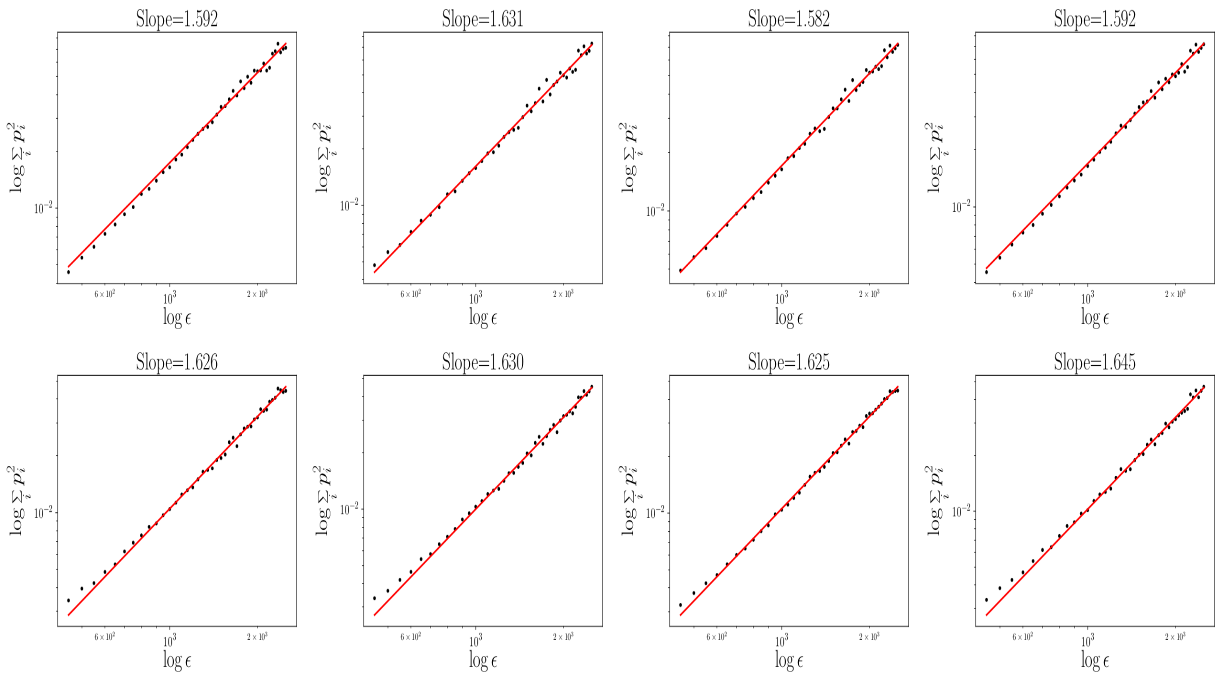
Proof: See (Schuster and Just, 2006).

An important consequence of this is the power law estimation of the neighbors:

Lemma 2. *Given a set of points $\mathcal{P}$ with finite cardinality and its Correlation Dimension D_2, the average number of points within the shape $\square$ with radius of ϵ' follows the power law: $\overline{nb}(\epsilon') \propto \epsilon'^{D_2}$.*

Proof: Using Equations 3.2 and 3.4 (Belussi and Faloutsos, 1998).

There are plenty of significant implications of the above lemma:



(a) Between 20:30-20:35 (b) Between 20:35-20:40 (c) Between 20:40-20:45 (d) Between 20:45-20:50

Figure 3.3: Correlation Fractal Dimension using ride-sharing dataset for New York for four consecutive time snapshots each spanning $m = 300$ seconds. Top row constructed with pick-up points, and bottom row using drop-off points.

1. We can predict the exact number (Belussi and Faloutsos, 1995) of drop-off or pickup points for any geographical space within the constraints of the fractal range.
2. It provides a formal technique to justify validity of any dataset whether real or synthetically generated since this characterization provides a high-level metric by analyzing data across the geographical space of a city.
3. Finally, in time snapshots when there are unexpected events like congestion or any accidents, such a characterization can help evaluate and flag anomalies.

3.4.3 Fractal Dimension of Ride Requests

Based on the our Lemma 2, we now experiment to determine the fractal range and D_2 with our point-set which is a collection of real ride requests in a city for a duration of one week where each point either represents the pickup location (latitude & longitude) or the drop-off location (latitude & longitude). To compute the correlation fractal dimension of human mobility point-set, we discretize time into short snapshots of m minute durations; we divide the geographical space into square cells each with side ϵ meters. Figure 3.3 shows D_2 for four consecutive time snapshots on a typical Friday evening. In all four plots there is a super-linear relation between number of points covered versus the range, ϵ.

When D_2 is measured for yellow cab data from New York in Figure 3.4, we observe almost

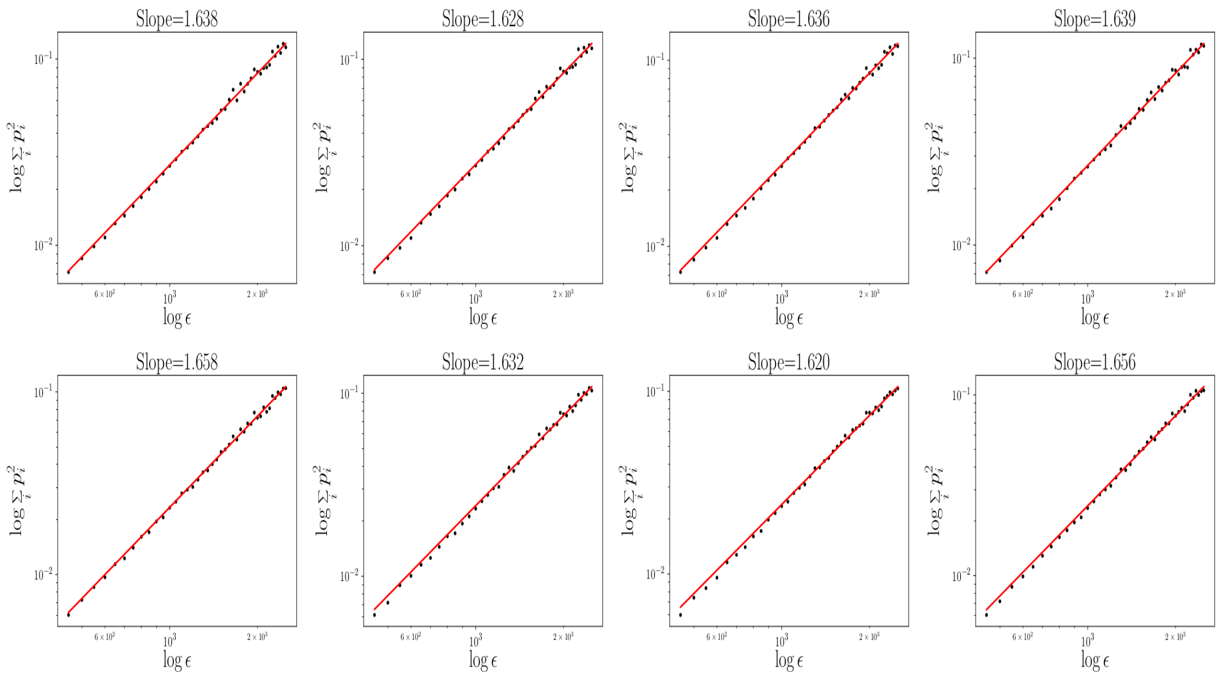
Correlation Fractal Dimension				
City	D_2 min.	D_2 max.	D_2 mean	fractal range (m.)
Boston	0.783	1.571	1.284	(600, 2500)
Chicago	1.003	1.459	1.206	(600, 3000)
London	0.678	1.686	1.419	(600, 2500)
Los Angeles	0.438	1.465	1.135	(1500, 4000)
Mexico City	0.832	1.706	1.529	(600, 2500)
Miami	1.049	1.686	1.343	(600, 2500)
New Delhi	1.049	1.686	1.343	(450, 2500)
New Jersey	1.049	1.686	1.343	(900, 2500)
New York (ride-sharing)	0.950	1.694	1.515	(450, 2500)
New York (yellow cab)	0.958	1.709	1.582	(450, 2500)
Paris	0.669	1.778	1.586	(900, 4000)
Rio De Janeiro	0.352	1.686	1.343	(600, 2500)
San Francisco	1.049	1.686	1.343	(450, 2500)
Sao Paulo	0.763	1.737	1.574	(900, 4000)
Toronto	0.782	1.536	1.292	(500, 2500)
Washington	1.049	1.686	1.343	(450, 2500)

Table 3.1: Summary of measured correlation fractal dimensions for fifteen cities; computed over a week for every 5-minute time snapshot (2016 snapshots for a week). We exclude seven late-night hours of each day when on average the frequency of ride requests is low.

equivalent values for D_2 as in the case of ride-sharing dataset. Although due to high density of ride requests for yellow cab, the lower part of the plot with low values of ϵ has a much better fit in comparison to ride-sharing dataset; in Figure 3.3 the overlap between black dotted points and the red line is not prominent. This indicates that for human mobility dataset when density is below some threshold the nice fractal-like pattern may not be observed and only once the threshold exceeded would the mobility density provide the fractal pattern. This threshold would vary from city to city as it would become clear in the experiments later.

Table 3.1 summarizes the correlation fractal dimension D_2 for the different cities we have studied in this work. For each city, there is a consistent weekly pattern of ride requests and a corresponding variation of its D_2 values. The values of D_2 ranges between minimum, D_2min, and maximum, D_2max. The mean values for the cities fluctuates a lot, indicating the degree of self-similarity of ride request patterns varies from city to city. The fractal range also varies across the cities; however, except for Los Angeles, all other cities exhibit very similar fractal ranges, from 500 meters to 3000 meters.

We notice cities like Mexico City, Paris, and Sao Paulo have $D_2 > 1.5$ which is much higher than the rest of the cities. This implies that such cities are much densely populated leading to higher rate in increase of points of interest where individuals are likely to travel. The bias due to how much ride sharing services have penetrated in a city would show in the data as well. For instance, New Delhi does not have a high D_2 value in comparison to some of the less densely populated cities; this low market penetration is also visible in the D_2 plot for New Delhi in



(a) Between 20:30-20:35 (b) Between 20:35-20:40 (c) Between 20:40-20:45 (d) Between 20:45-20:50

Figure 3.4: Correlation Fractal Dimension using yellow cab dataset for New York for four consecutive time snapshots each spanning 300 seconds. Top row constructed with pick-up points, and bottom row using drop-off points.

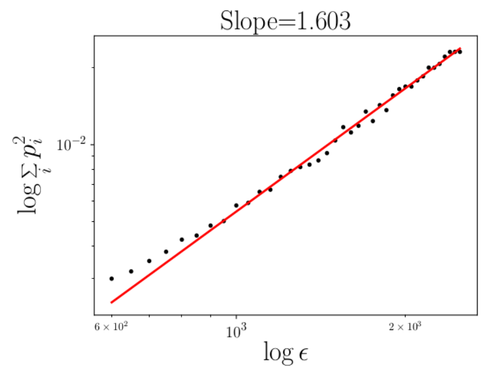
Figure 3.5 where the least square fit does not overlap the points as often the case in plots for other cities.

The fractal dimension and the fractal range succinctly capture the characteristics of ride request patterns in a city. Later in Chapter 5 we will show a real-world application on how the fractal dimensions and associated fractal ranges can be instrumental in revealing effective solutions to the real-time vehicle placement problem and facilitating the assessment of the effectiveness of specific algorithms for this problem.

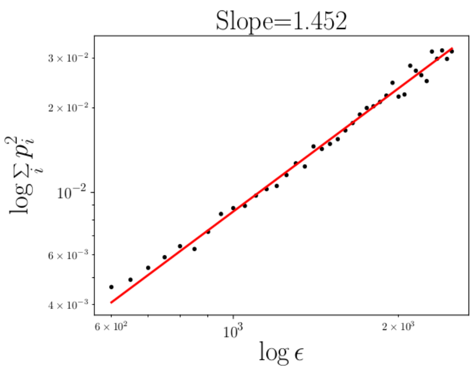
As mentioned in the temporal characteristics of each city – Los Angeles, and New Jersey with maximum variation in degree exponent computed using experimental data and theoretical model; here too we observe fractal range starting from much higher value like 1500, and 900 for the same cities along with Paris, and Sao Paulo. In the context of spatial analysis, this could be due to the sprawling nature of the city. Plots for the remaining 14 cities are provided in Figure 3.5 from a single time snapshot only.

3.5 Discussion

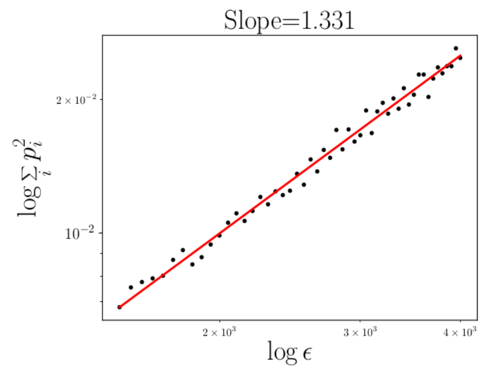
For human mobility, fractal dimension is a concise representation of a complex and random set of points. Other ways of modeling density of points in space can be done using non-parametric estimators like multivariate kernel density estimator although the underlying assumption of a



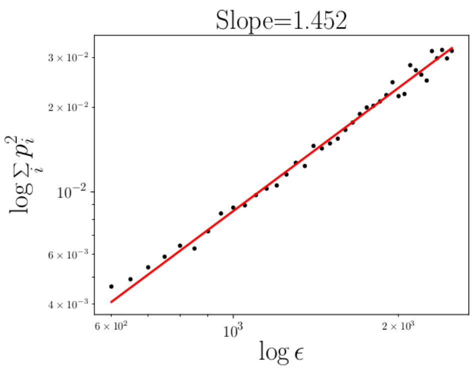
(a) Mexico City



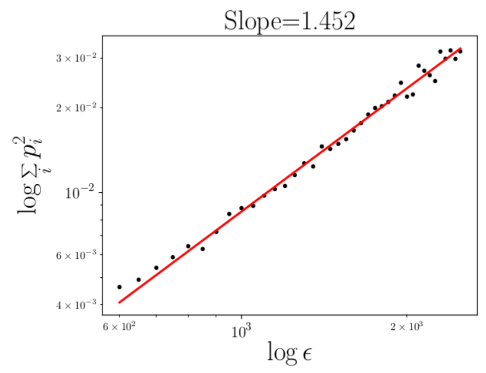
(b) London



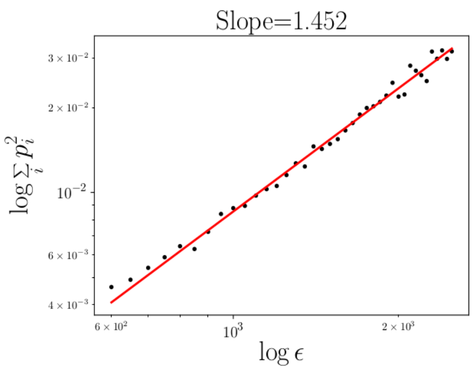
(c) Los Angeles



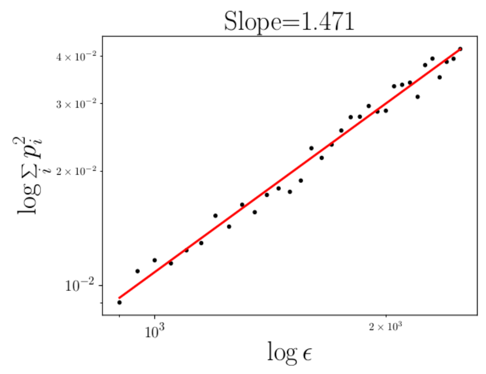
(d) San Francisco



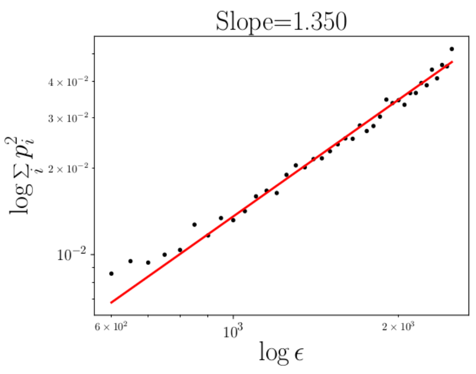
(e) Chicago



(f) Washington

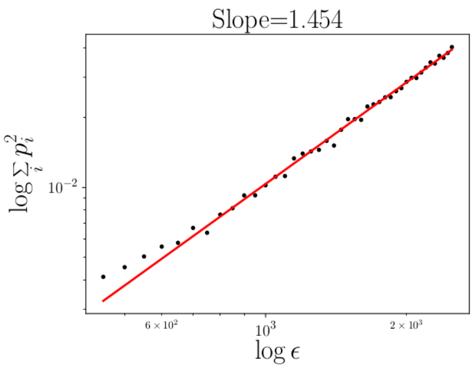


(g) Paris

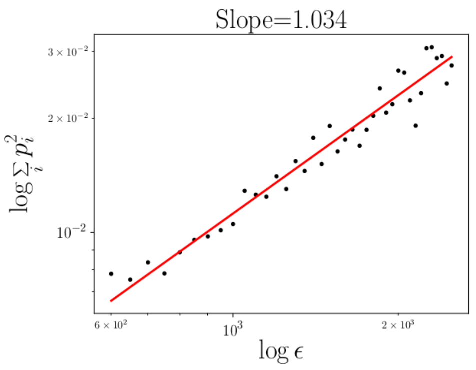


(h) Boston

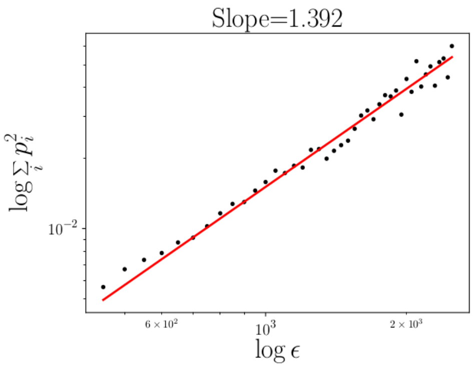
Figure 3.5: Correlation fractal dimension plots from real data for multiple cities. The red line is the least square fit of the form $y = Cx^{\alpha}$. $R^2 \approx 1.00$ for all of them.



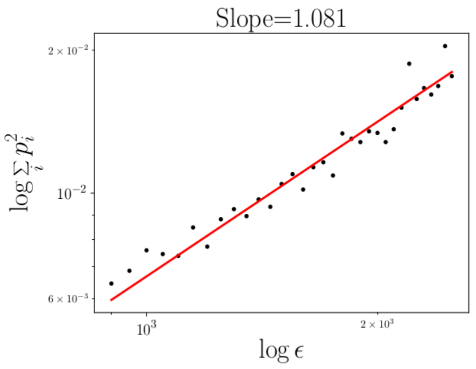
(i) Sao Paulo



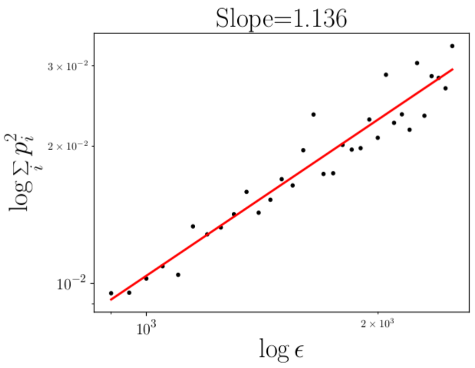
(j) Miami



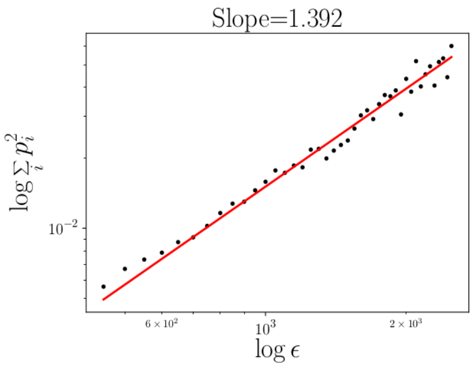
(k) Toronto



(l) New Jersey



(m) New Delhi



(n) Rio De Janeiro

Figure 3.5: Correlation fractal dimension plots from real data for multiple cities. The red line is the least square fit of the form $y = Cx^{\alpha}$. $R^2 \approx 1.00$ for all of them.

gaussian surface may not always capture urban-level dynamics (Chan, Wood, et al., 2004). For instance, gaussian or spatial auto-correlation techniques assume there to be peak in density of ride requests at some geographical region and then gradual reduction as one moves away from the peak. Such a pattern does not hold true across all urban areas since dense areas or areas with a peak in volume of ride requests are spread in a non-uniform manner. For instance, large parts of lower Manhattan in NY city can have high volume ride requests without any gradual reduction. Conversely, parts of San Francisco city can have large geographical area alternating between pockets of regions with high and low number of ride requests with varying spreads of each. Such patterns tend to change from city to city and hence it is difficult to capture it using a distribution or mutliple instances of the same distribution.

Also such an estimator may provide a good estimation of the density function but not a succinct way to characterize distribution of pickup and drop-off points for a city. By aggregating the points in grids/cells we also make this mode of characterization privacy aware such that there does not exist a trivial way to trace back the exact geographical source or destination coordinates.

Fractal dimension describes the dispersion affect of human mobility. When comparing D_2 for different cities, we noticed higher values of D_2 may not necessarily correlate with higher population density but rather with how population density is spread across the geographical area of a city. For instance if one were to look at light pollution map [1] of Toronto and Paris, Toronto has a much wider spread of density across its area with many spots of high human activity. On the other hand, Paris's spots of high human activity are more centralized and not as dispersed as in the case of Toronto, and tend to disappear as we move away from central region of the city resulting in a higher value of D_2.

The possibility of characterizing a point-set as fractals is quite feasible (Belussi and Faloutsos, 1998). In numerous studies like (Mandelbrot and Mandelbrot, 1982) and (Schroeder, 2009), a lot of real datasets look like fractals including coast lines, country borders, the periphery of clouds and rainfall patches. Importantly, one can efficiently compute D_2 (Belussi and Faloutsos, 1998) and then analytically approximate the number of neighbors for both pick-ups and drop-offs. The spatial observations can provide context around the human mobility density pattern in any area for civic authorities to perform planning in terms of new construction, transportation service, and methods to intervene to divert traffic based on density of human mobility traces.

[1] https://lightpollutionmap.info

Chapter 4

Scalable Synthetic Generation of Human Mobility Datasets

Ride request data from ride sharing services can potentially be of great value to understand human mobility patterns. Data gathered from ride sharing services could be used to provide insights about traffic and human mobility patterns which are essential for intelligent transportation systems (Stiglic et al., 2018). Ride requests in major cities with high penetration by such services exhibit spatial and temporal variability. Modeling of such variability is a challenging problem for researchers although there is existing work to predict human mobility patterns at coarser-grained geographical locations (Isaacman et al., 2012). This chapter aims to combine both spatially and temporally finer-grained human mobility patterns to propose a technique to efficiently generate synthetic human mobility data at the city scale. Such granularity of data can help with unresolved challenges related to intelligent transportation, such as: optimal algorithms for dynamic pooling of ride requests (M. H. Chen et al., 2017), real-time pre-placement of vehicles (Phithakkitnukoon et al., 2010), simulations, and city scale traffic congestion prediction (Maystre and Grossglauser, 2016) and avoidance (J. Tang et al., 2018). Access to large amount of actual ride request data is essential to understanding and addressing these challenges.

As noted before, each city exhibits a different pattern of urban mobility – there could be cultural or economical factors governing these patterns. If ride sharing services constitute a significant percentage of the riders in a city, we can build models from ride request data to model urban mobility for the whole city and provide societal benefit without compromising riders' privacy.

Synthetic data can be very useful in multiple applications to train, test, or inform algorithms. It is widely used to train anomaly detection systems, but also in microsimulation models[1] to inform policy intervention, planning, sensor network deployment, and much more. Some general purpose synthetic data generation methods also utilize ways of describing the data to be generated (Hoag, 2008), while others are designed to generate very specific types of data in a rather unstructured way. In this paper, we focus on synthetic data sampling methods that preserve privacy and the correlation between geographical locations in urban cities while incorporating constraints on time for which the data is generated.

[1]https://github.com/citybound/citybound

Our proposed approach involves viewing ride requests as a (temporal) sequence of (spatial) images of ride request locations. The approach uses GANs to match the properties of the synthetic data sets with that of real ride request data sets. Many recent works using neural networks have looked at demand prediction (Yao, Wu, et al., 2018; Zhou et al., 2018) and traffic prediction at intersections (Yao, X. Tang, et al., 2018). In our work, we are looking at generating actual ride requests for both spatially and temporally granular intervals. Also, we compare and validate the spatial and temporal variations of the synthetic data sets with the real data sets.

4.1 Why GANs?

The proposed approach allows to capture the spatial properties of geospatial data. Specifically we would like to compute the likelihood of a node to either possess a source or destination location. Alternate approaches like kernel density estimation are also discussed later in this chapter to generate synthetic ride requests to highlight some issues with alternate methods. Here we would like summarize some of them – 1) we want an approach by which we could modulate the volume of ride requests easily, this is possible with GANs as highlighted in Section 4.7; 2) we would like to give the option to create virtual environments by modifying images of ride requests and hence allow models to be trained on them, and subsequently use the synthetic data to understand different what-if scenarios in the virtual setting; 3) the number of parameters to a model should be minimal and with GANs we are able to generate synthetic data with just one tweakable parameter which determines the volume/intensity of ride requests generated; 4) lastly, we want the ability to learn ride request pattern locally for a region and not require global city-level information to avoid any privacy issues.

With extensions like InfoGAN (X. Chen et al., 2016), we believe there exists numerous ways to extend this work by finding hidden and semantic representations on human mobility image datasets for better approximation of real data.

4.2 Data Sets from Ride Sharing Services

Our real ride request data sets consist of all the ride requests for an entire week for each city. There is a strong repeating pattern from week to week capturing varying patterns for workdays and weekends including Sundays. Hence the week-long data should be quite representative of the general trend. For changes in weather or seasons, our model would still be able to capture the variations as it does not rely on geographical changes like road or highway closures which might be a consequence of weather. As long as the underlying data highlights enough penetration of the mobility share, the seasonal changes would inherently be captured by the model. For most of the cities, the ride sharing services have significant penetration, and hence we believe the ride request data sets also reflect the overall urban mobility patterns for these cities.

In this chapter, our data sets are real ride requests for fifteen cities over one week period. Each ride request in the data set includes: request time and pickup location (latitude & longitude), and drop-off time and location (latitude & longitude). For this work we focus on ride request time and pickup location for generating pickup locations; and ride request time and drop-off location to generate drop-off locations. After training independent GANs models for pickup and drop-off locations, we generate synthetic locations using GANs and leverage graph generator approach, Algorithm densPropGen, proposed in Chapter 2 to pair all pickup and drop-off locations to obtain synthetic ride requests. The trajectory or optimal route for a ride is not within the scope of this work. For the rest of this chapter, we use the term *ride-locations* to refer to both pickup and drop-off locations wherever they can be used interchangeably.

We do temporal and spatial quantization of the raw ride request data. We partition the entire week into 2016 time intervals of 5 minutes each, and lump together all the ride requests within each interval. We partition spatially the area of the entire city into small squares with side length, ϵ, of 50 meters, and lump together all the ride-locations occurring within the same square area. Lower values of ϵ can be considered with added computational costs of model training; higher values of ϵ (>= 100 meters) may hide some of variational properties of ride requests in densely populated cities. Each square area is then represented by a single pixel in a 2-D image with the gray scale intensity of the pixel reflecting the number of ride-locations in that square area (in a time interval). Occurrence of no ride-locations in an area is denoted by zero pixel intensity; positive integers $(1, 2, 3, \ldots)$ as pixel intensity denote the number of ride-locations in the square area.

Combining the temporal and spatial quantizations, the real ride request data set for each city becomes a time sequence of images with each image spatially capturing all the ride requests occurring in a particular 5-min interval.

4.3 Generating Ride Requests Using GANs

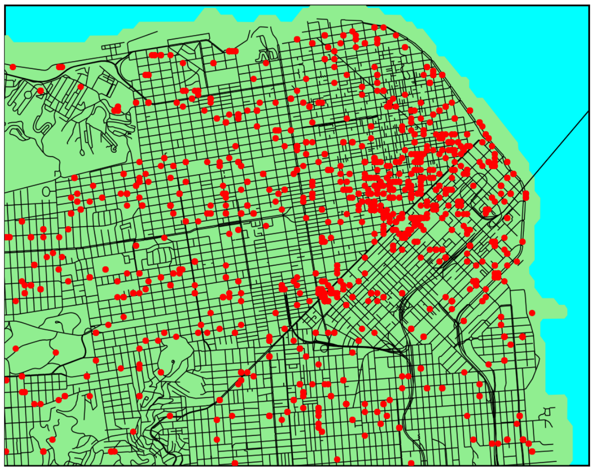
4.3.1 Image generating using GANs

Generative Adversarial Networks learn to generate high quality samples (Goodfellow et al., 2016) i.e. sample from the data distribution $p(x)$. Previous works by (Denton et al., 2015; Ledig et al., 2017) synthesized images of a higher quality using GANs which were hard for humans to distinguish from real images. Conditional GANs are an extension of GANs to sample from a conditional distribution given each image has an associated label which is true for our case of ride requests.

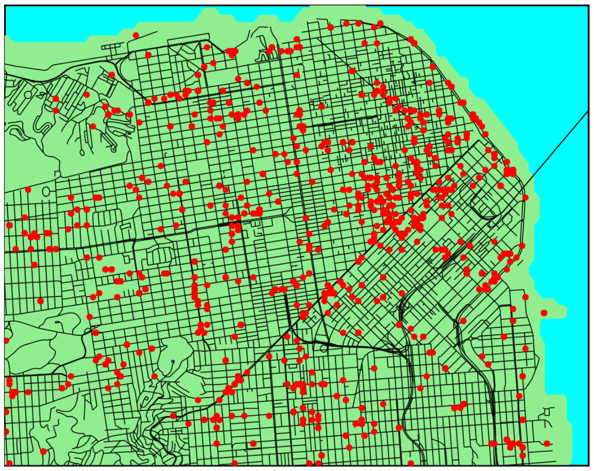
In our framework, we would apply conditional GANs using ride request data in the form of images; similar to as shown in Figure 4.1 but without the base map shown in color. Representing ride requests as images allows us to not add other constraints to our model such as parks, water bodies, mountains, or any other area where ride request origin or end is very unlikely.

4.3.2 Using GANs for ride request generation

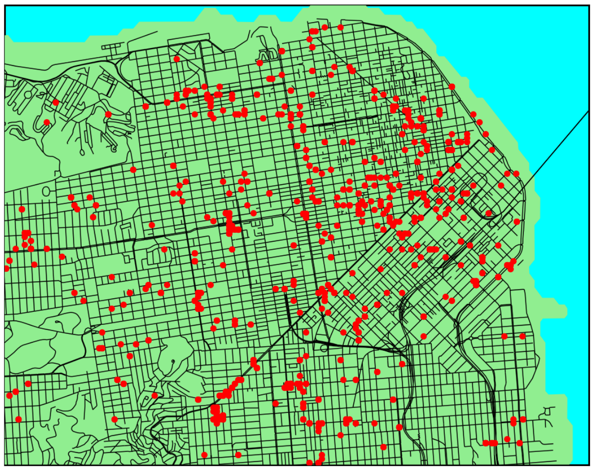
GANs learn a mapping from a random noise vector z to output image x. Conditional GANs learn a mapping from noise vector z and a label y to x (Mirza and Osindero, 2014; Gauthier, no date). The additional variable in the model allows to generate and discriminate samples conditioned on y. The generator accepts noise data z along with y to produce an image. The discriminator accepts an image x and condition y to predict the probability under condition y that x came from



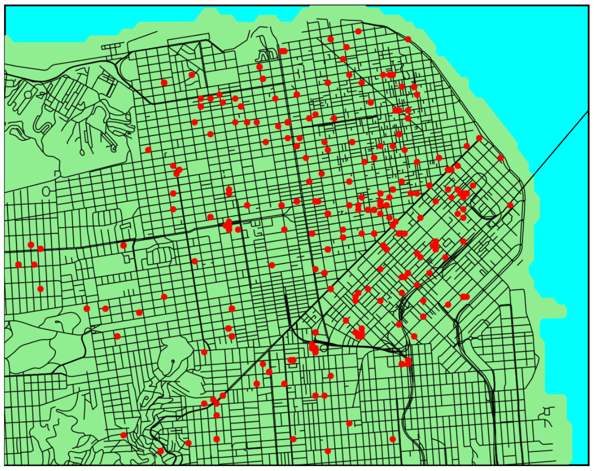
(a) 6pm



(b) 9pm



(c) 12am



(d) 3am

Figure 4.1: Ride requests for a small region of downtown San Francisco for a typical week day. Each figure shows the aggregated ride-locations (red dots) over a period of an hour. Each red dot may represent one or more ride-locations. Ride density varies spatially and temporally.

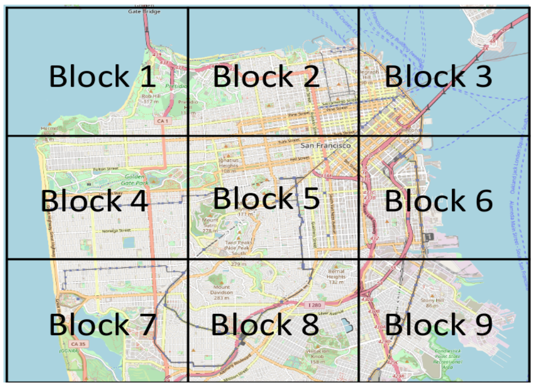


Figure 4.2: An illustration of how the geographical region of each city is divided into smaller blocks of equal size and trained independently. Note: image above is not drawn to scale.

the empirical data distribution rather than from the generative model. The objective function can be expressed as:

$$\mathop{\mathbb{E}}_{x,y\sim p_{data}(x,y)}[log D(x,y)] + \mathop{\mathbb{E}}_{z\sim p(z), y\sim p_y}[log(1 - D(G(z,y),y))]$$

where G tries to minimize to this objective function against an adversarial D that tries to maximize it.

4.3.3 Training Process & Architecture

Every image is assigned a label from the set $\{0, 1, 2, ..., 23\}$ representing the hour of a day. All twelve instances of five minute snapshots within an hour are assigned the same hour label [2]. To accelerate our training using multiple machines, we exploit spatial parallelism by dividing the entire geographical region of a city into an array of blocks. Figure 4.2 illustrates the division of San Francisco downtown into nine blocks. Keeping our image size similar to MNIST (Salimans et al., 2016), each block is set to represent an image of size 24×24 pixels, with each pixel representing one $50m \times 50m$ square area. Hence, each block covers an area of $1200m \times 1200m$.

Each block, representing a grey scale image of 24×24 pixels, depicts all the ride-locations in that block. Separate images are formed for pickup and drop-off locations; models trained are also separate for pickup and drop-off locations. Each image of a block is labeled with a time interval (for our experiments, the hour in a day) which is similar for both images created from pickup and drop-off locations. The synthetically generated images from an array of blocks with the same time interval label are combined by stitching together all the processed blocks of a city.

The generator network takes an input of a 100-dimensional Gaussian noise sample as well as a one-hot vector encoding of the time snapshot to be generated. It has a single, fully-connected hidden layer without any convolution (Goodfellow, 2018) consisting of 128 ReLU-activated

[2]One could easily extend this approach to a label within the set $\{0, 1, ..., 287\}$ if looking at labels associated with any five minute slots of a day or the set $\{0, 1, ..., 2015\}$ if looking at labels associated with any five minutes slots of a week.

neurons which then passes to a sigmoid activated output layer with the same number of output neurons as the total number of pixels in each block.

The discriminator network has a single hidden layer of 128 ReLU-activated neurons with a single sigmoid activated output neuron. We find that small networks are appropriate for the training data and allow for a quick and stable convergence to be achieved between the discriminator and the generator. Using relatively simple network architectures makes it possible to ensure that the discriminator and generator are *evenly matched* such that the loss for either network does not saturate early in the training process.

In addition to the standard GANs architecture of generator and discriminator, an additional network is introduced which is referred to as the classifier (Lee and Seok, 2017); it is pre-trained on the training data with the five minute label of the data serving as the classification target. In this way the time information that is encoded into the synthetic data by the generator network is then decoded by the classifier network. The generator is then trained on a weighted sum of the loss from both the classifier and discriminator networks as shown in the following equation:

$$\beta \log D(G(z, y)) + (1 - \beta) \log C(G(z, y))$$

where β is a tune-able hyper-parameter.

This allows for more explicit loss attribution such that the generator receives two different error signals; one indicating the realism of the synthetic data and the other indicating accuracy relative to the conditioning values. By experiments using MNIST data and (Lee and Seok, 2017), we found adding a classifier increases the efficiency of the training process and results in higher quality synthetic data while incurring considerably less training time than other conditional GANs architectures we have experimented.

4.4 Experimental Results

In this section, we present the cloud infrastructure used for running our experiments. We also present performance results on scaling our GANs workloads on the cloud infrastructure.

4.4.1 Running GANs on AWS

All experiments are conducted on Amazon Web Services (AWS) using c5.18x instances with each instance containing an Intel Xeon Scalable Processor with 72 virtual cores (vCores) running at 3.0GHz and 144 GB of RAM. We use AWS for easier reproducibility and its large adoption by academia and industry for large scale distributed processing.

In this work we set the block size for each of the four cities to be 1200×1200 meters; each block is trained separately. Enlarging the block size will increase the computational time for training; and the complexity of the model can potentially impact scalability. The total number of blocks for each city are shown in Table 4.1. The number of blocks are mainly determined by the size of the greater metropolitan area of each city.

To help enhance the scalability of our GANs workload across multiple nodes we make use of Ray (Moritz et al., 2017) from Berkeley, a distributed framework for AI Applications,

City	*Number of Blocks*
San Francisco	1402
Los Angeles	1978
New York	765
Chicago	1155

Table 4.1: Training Workload for four cities. Each block is trained independently. Doubling the value provided in the table for each city, would give the approximate number of models trained because we have pickup and drop-off locations trained and generated separately.

to efficiently parallelize our workload across cluster of CPU nodes on AWS. Ray provides a convenient API in Python to scale deep learning workloads for numerous libraries, and support for heterogeneous resources like CPUs and GPUs. We also make use of Intel's Math Kernel Library (Intel, 2018b) (MKL) which provides machine learning libraries for supporting operations like activation (ReLU), inner product, and other useful functions (Intel, 2018a).

4.4.2 Training Time

Using Ray we scale our training runs by using from 2 to 8 c5.18x instances (containing from 144 cores to 576 cores) on AWS. The scalability results are shown in Figure 4.3. As can be seen increasing the number of c5.18X Xeon CPU instances can significantly reduce the GANs training time up to 8 c5.18x instances. For the city of Los Angeles, the training time can be reduced from over one hour to less than 20 minutes. For New York City the training time can be reduced to just minutes. Running times for sampling ride requests from the trained models and stitching the images of all the blocks together are significantly less than the training times, and are not included in these results.

We also conduct our GANs scaling experiments using GPU instances on AWS. In our initial experiments we observe no real performance improvements using GPUs. Training time using GPUs on AWS was observed to be 5.93 hours on a p3.8xlarge instance using NVIDIA's Multi-Process Service (MPS) (Nvidia, 2018). With MPS, the GPU utilization is close to maximum by running multiple of our small GANs training jobs in parallel on a single GPU. Although, the number of jobs which could be executed in parallel on a GPU are not that many in comparison to Xeons. Scaling on GPUs requires more investigation. In this work, we show that it is possible to achieve very nice scalability of our GANs workload using only CPU cores supported by Intel's MKL library and Berkeley's Ray framework.

4.5 Validation of Synthetic Data Sets

We use our trained models to generate synthetic data for a day and validate it using real data by computing metrics for temporal & spatial variations.

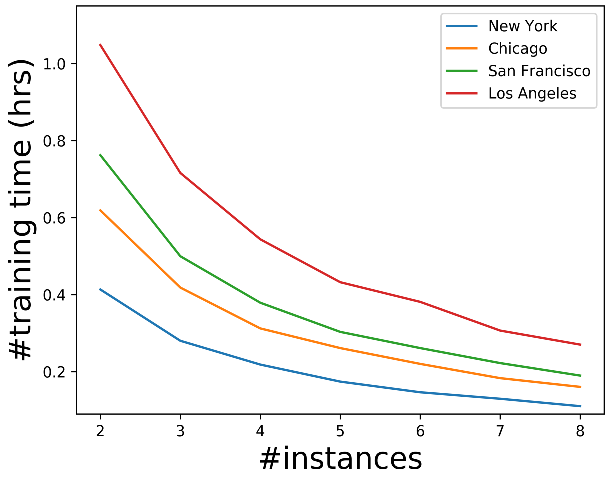


Figure 4.3: Training Time Performance Results for training our GANs for pickup locations on AWS with c5.18xlarge Xeon CPU instances. Results for training using drop-off locations show a similar trend.

4.5.1 Temporal Variation

DPL provides a characterization of the temporal evolution of ride requests. In Table 4.2 we observe the plot of the DPL exponents α (slop of the line) based on the temporal patterns of the real and synthetic data sets. For the ride request graph to obey DPL properties, we use graph generator proposed in Chapter 2 to connect source and destination locations for synthetic dataset. We clearly see that the DPL exponent values α correlated quite nicely with that from the real data sets most of the cities. For Los Angeles, the synthetic exponent is higher than the real observed value[3]; the geographical region for LA is much larger and due to many prominent regions of high request density, the model may likely suffer from bias towards generating more requests in prominent regions leading to a faster increase of the number of edges connecting nodes present in high density regions.

Another validation of our GANs approach is provided in Figure 4.4. Here we observe temporal variation of ride requests in terms of the volume of ride requests generated for each hour of a typical weekday. We see that for all cities, the temporal variation of the synthetic data sets match quite well the temporal variation exhibited by the actual data set. Our model for Los Angeles captures the total volume of ride requests over time intervals correctly but it may not necessarily imply that the model also captures geographical spread of requests within each time interval. For this reason we observe a high α for synthetic data generated in Table 4.2; it is critical to have multiple metrics for a model to capture human mobility patterns.

[3]An increase of 0.1 in exponent translates to $\approx$ 10% decrease in number of nodes.

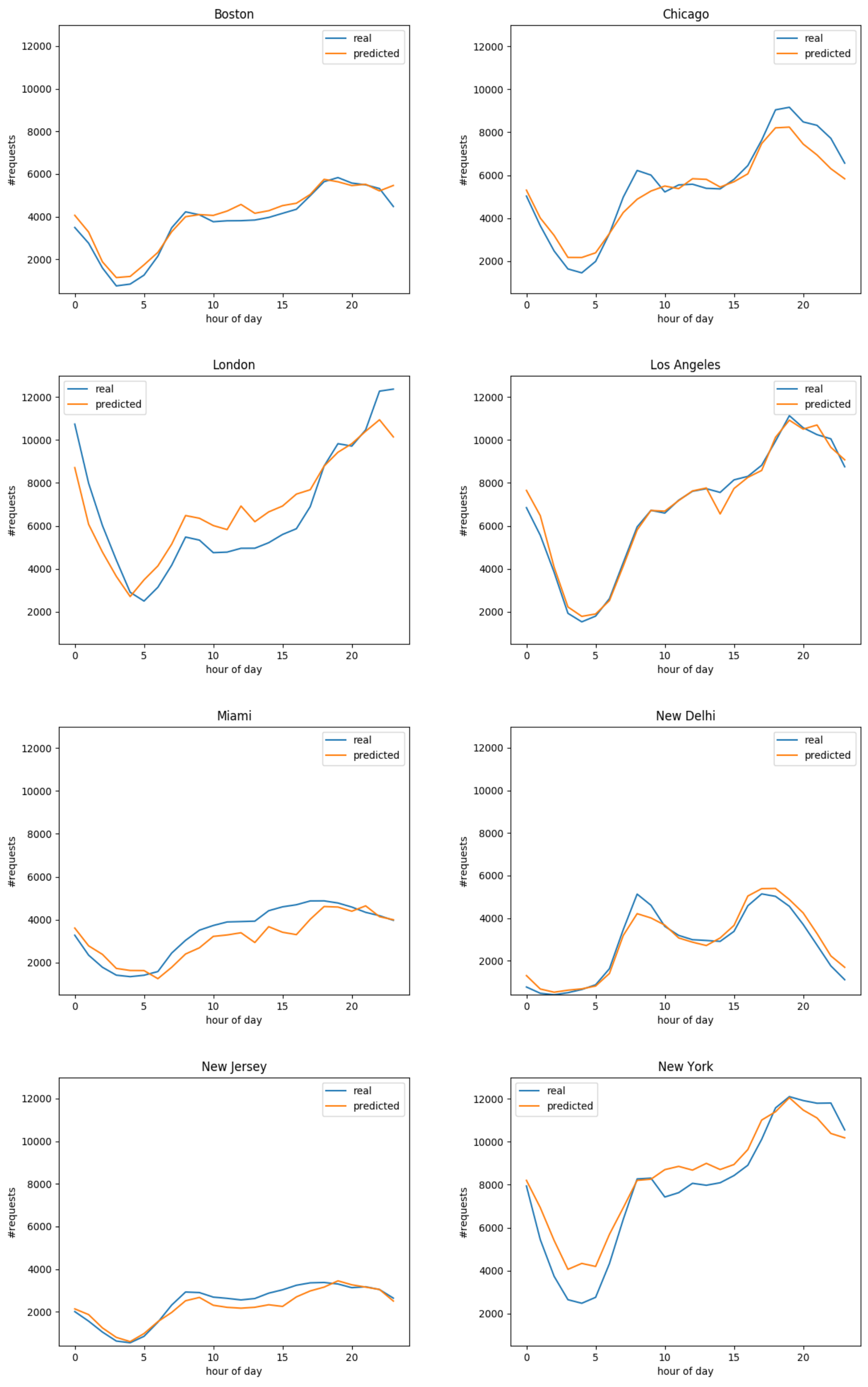


Figure 4.4: Plots for all cities highlighting the temporal variability of ride requests visible in both real and our model (predicted) for ride request generation. The pattern is representative of any typical day of a week.

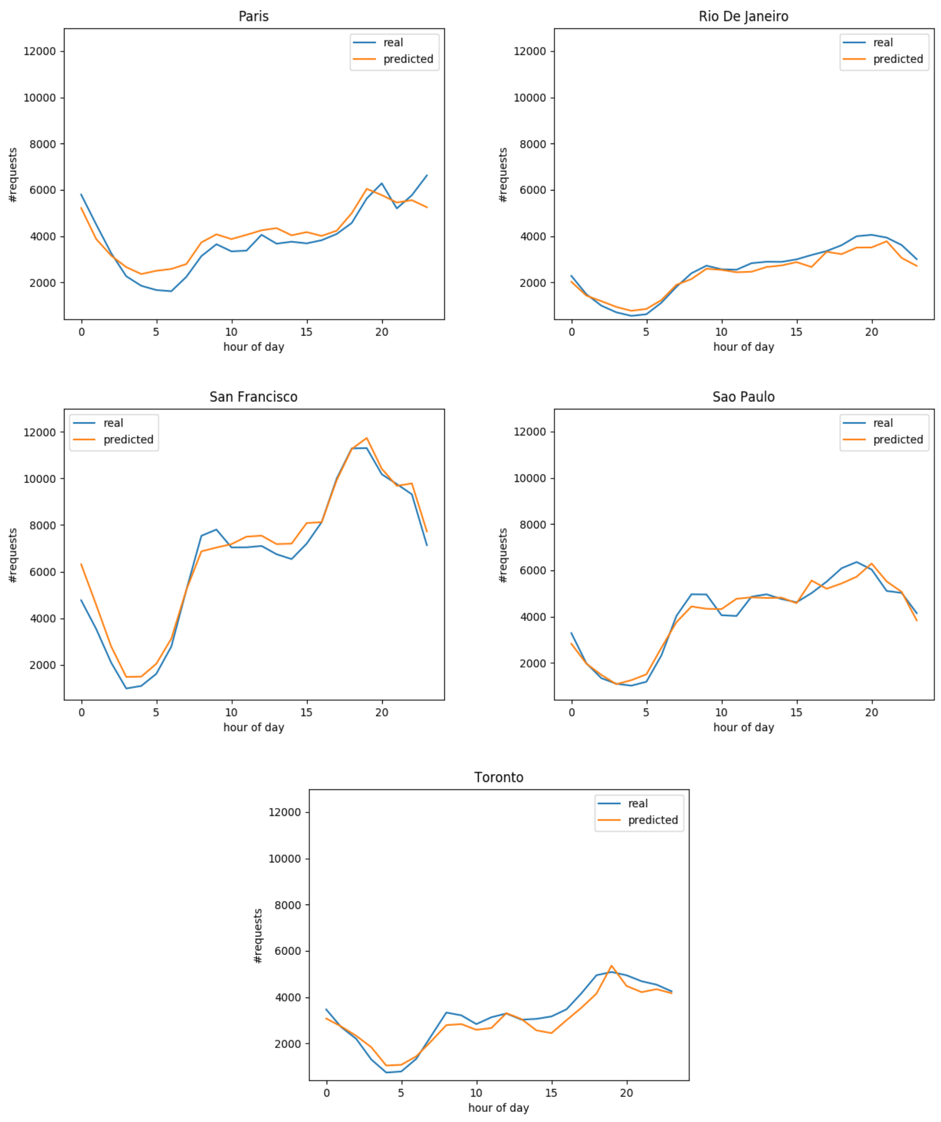


Figure 4.4: Plots for all cities highlighting the temporal variability of ride requests visible in both real and our model (predicted) for ride request generation. The pattern is representative of any typical day of a week. All plots start at midnight local time.

Temporal Properties: Densification Power Law		
City	α (real data)	α (syn. data)
Boston	1.417	1.341
Chicago	1.434	1.486
London	1.302	1.352
Los Angeles	1.053	1.614
Mexico City	1.189	1.274
Miami	0.985	1.185
New Delhi	1.195	1.403
New Jersey	1.063	1.099
New York (ride-sharing)	1.299	1.361
Paris	1.433	1.373
Rio De Janeiro	1.357	1.401
San Francisco	1.250	1.341
Sao Paulo	1.154	1.372
Toronto	1.354	1.571
Washington	1.331	1.429

Table 4.2: Comparison of densification factor α between real and synthetic datasets.

4.5.2 Spatial Variation

The correlation fractal dimension (D_2) gives a bound on the number of ride requests within a geographical region. This is an essential characteristic to match for the data set we are generating using GANs. In Tables 4.3 & 4.4, we provide the fractal range (in meters) for each city within which the fractal dimension remains constant. The fractal dimension is computed for every fractal range increment of 100 meters. It is important to highlight again that the fractal range for each city differs. The fractal range provides the ϵ range for which the data exhibits statistical self-similarity (Belussi and Faloutsos, 1998). The variation in the fractal ranges for the different cities can be attributed to the geographical shape of the city for which the ride requests are generated. We hypothesize that due to Los Angeles's sprawling nature, a larger ϵ is needed to observe self-similar patterns in comparison to the other three cities, which have a more corridor-like geographical region.

One may also interpret D_2 as a way to measure the fidelity of generated images to that from real data. Comparison of the ranges of values of D_2, in terms of min, max, and mean values, for the real and the synthetic data sets are fairly close although not identical. In most instances the mean value for D_2 for the synthetic data sets is quite close to the one calculated using real data sets. In some cases where there is discrepancy between the synthetic and real values of D_2, it is due to the weakness of the model to capture geographical areas with low frequency of ride requests. Recent works to improve capture learning of high-resolution details of an image (Karras et al., 2017) can potentially benefit the learning for our ride request images.

City	Fractal Range (m.)	Real Data Sets			Synthetic Data Sets		
		D_2 min.	D_2 max.	D_2 mean	D_2 min.	D_2 max.	D_2 mean
Boston	(600, 2500)	0.990	1.709	**1.497**	1.169	1.625	**1.464**
Chicago	(600, 3000)	1.143	1.555	**1.384**	1.188	1.567	**1.435**
Los Angeles	(1500, 4000)	0.955	1.625	**1.352**	1.047	1.488	**1.314**
Miami	(600, 2500)	0.799	1.389	**1.094**	0.628	1.210	**1.021**
New Delhi	(450, 2500)	0.657	1.620	**1.207**	0.353	1.242	**0.918**
New Jersey	(900, 2500)	0.686	1.512	**1.183**	0.853	1.331	**1.081**
New York	(450, 2500)	1.482	1.750	**1.648**	1.415	1.662	**1.540**
Paris	(900, 4000)	1.463	1.829	**1.763**	1.539	1.785	**1.697**
Rio De Janeiro	(600, 2500)	0.302	1.509	**1.314**	1.061	1.496	**1.309**
San Francisco	(450, 2500)	1.277	1.730	**1.548**	1.207	1.678	**1.442**
Sao Paulo	(900, 4000)	1.095	1.747	**1.613**	1.451	1.766	**1.622**
Toronto	(500, 2500)	1.318	1.655	**1.564**	1.296	1.588	**1.477**

Table 4.3: Summary of measured correlation fractal dimensions (D_2) for four cities; computed over a day for every hour using **pickup** locations of real and synthetic data sets.

City	Fractal Range (m.)	Real Data Sets			Synthetic Data Sets		
		D_2 min.	D_2 max.	D_2 mean	D_2 min.	D_2 max.	D_2 mean
Boston	(600, 2500)	0.614	1.639	**1.283**	0.604	1.573	**1.284**
Chicago	(600, 3000)	0.328	1.467	**1.214**	0.388	1.5	**1.224**
Los Angeles	(1500, 4000)	0.118	1.503	**1.033**	0.214	1.456	**1.042**
Miami	(600, 2500)	0.271	1.277	**1.004**	0.312	1.261	**0.923**
New Delhi	(450, 2500)	0.134	1.475	**1.048**	0.055	1.160	**0.827**
New Jersey	(900, 2500)	0.465	1.523	**1.125**	0.549	1.476	**1.04**
New York	(450, 2500)	0.617	1.716	**1.497**	0.914	1.594	**1.413**
Paris	(900, 4000)	0.672	1.831	**1.617**	0.831	1.779	**1.591**
Rio De Janeiro	(600, 2500)	0.383	1.470	**1.287**	0.417	1.428	**1.183**
San Francisco	(450, 2500)	0.496	1.696	**1.361**	0.570	1.684	**1.334**
Sao Paulo	(900, 4000)	0.360	1.695	**1.511**	0.599	1.655	**1.519**
Toronto	(500, 2500)	0.799	1.609	**1.426**	0.530	1.581	**1.339**

Table 4.4: Summary of measured correlation fractal dimensions (D_2) for four cities; computed over a day for every hour using **drop-off** locations of real and synthetic data sets.

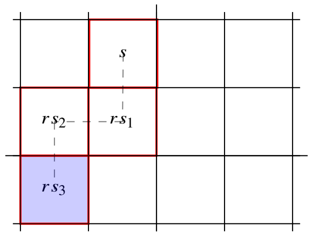


Figure 4.5: Random walk starting at node s with three random steps to reach node rs_3. Final point is selected uniformly random within the blue colored grid.

4.6 An Alternate Method for Synthetic Generation

As mentioned before to capture spatial properties, we compute the likelihood of a node to either possess a source or destination location by using geospatial information from OSM's public data (OpenStreetMap, 2016) on node density[4]. To avoid confusion with a node of the Ride Request Graph, we shall refer to an OSM node[5] as *Point of Interest* (PoI). A PoI is defined by a tuple of latitude and longitude. PoIs are used to define standalone features such as traffic signals, businesses, schools, hospitals, and many others. Alternatively, any other dataset providing a quantitative measure over geographical space which is correlated with the likelihood of ride requests can also be used.

Algorithm 2 performs node selection using vector of probabilities $pr \in \mathbb{R}^n, n = |S|$, where S is a subset of nodes of interest. pr is computed by aggregating all PoIs present at a RRG node, and then normalizing to get the probability mass function across nodes in S. Algorithm 2 takes as input the number of synthetic points m to be generated, and associates each synthetic point to an initial node; this is determined from prior probability vector pr. Once the initial node is chosen, Algorithm 3 performs a random walk starting from initial node centroid such that the synthetic points are spread out in the geographical area (Figure 4.5). Since PoI data could be sparse, we use a kernel density estimation function K[6] over the geographical space to guide the random walk.

In Algorithm 2 method *randomChoice* generates m points with replacement using the prior probabilities vector pr; *geoCoords* returns the latitude and longitude associated with the node label; *perturb* performs a uniform random selection within the final node (blue area in Figure 4.5) to determine the final location of the newly generated point.

In Algorithm 3, method *randomStep* chooses a node amongst the neighbouring eight nodes (or less) of the current node, *curr*, by normalizing the probabilities returned by the kernel density estimates; it returns the new node *new*, and a reward r. We kept reward equal to the probability estimate for current node, $r = K(curr)$. Note that every node is defined by latitude, longitude coordinates of its centroid.

[4]OSM data is publicly available at http://download.geofabrik.de/

[5]http://wiki.openstreetmap.org/wiki/Node

[6]We used Gaussian Kernel Density Estimation library http://statsmodels.sourceforge.net/devel/generated/statsmodels.nonparametric.kernel_density.KDEMultivariate.html

Algorithm 2 To capture spatial properties. Inputs: Kernel density estimation function K, number of synthetic points $m \in \mathbb{N}$, prior probability vector $pr \in \mathbb{R}^n$

```
1:  procedure SPATIALPROPGEN($K, m, pr$)
2:      $labels = randomChoice([0, \ldots, \mathrm{len}(pr)], m, pr)$
3:      $pts = []$
4:      for each point $i \in [0, m)$ do
5:          $l = labels[i]$
6:          $s = geoCoords(l)$
7:          $p = $ RANDOMWALK($K, s, max_r, max_s$)
8:          add($pts, perturb(p)$)
9:      end for
10:     return $pts$
11: end procedure
```

Algorithm 3 Random Walk. Inputs: Kernel density estimation function K, start location s, maximum reward, and maximum number of steps max_r & max_s

```
1:  procedure RANDOMWALK($K, s, max_r, max_s$)
2:      let $tot_r = 0$
3:      let $n_{steps} = 0$
4:      let $curr = s$
5:      while $tot_r \leq max_r$ and $n_{steps} \leq max_s$ do
6:          $curr$ , $r = randomStep\ (curr, K)$
7:          $n_{steps} = n_{steps} + 1$
8:          $tot_r = tot_r + r$
9:      end while
10:     return $curr$
11: end procedure
```

4.6.1 Temporal Variation

In Table 4.5 we note very similar set of values for densification factor for both real and synthetic datasets for the three cities. We used Algorithm DENSPROPGEN to pair pickup and drop-off locations as done after generating synthetic data from GANs as well. Note that numerous parameters in Algorithm 3 had to be hand tuned apart from picking volume of ride requests from real data for each time snapshot for which synthetic data had to generated and the exact location of each ride request for selection of PoIs. The obfuscation of real-data is possible but it is a parameterized approach. For reasons stated in Section 4.1, we believe GANs would still be our preferred approach.

Temporal Properties: Densification Power Law		
City	α (real data)	α (syn. data)
New York (ride-sharing)	1.098	1.110
Paris	1.054	1.045
San Francisco	1.104	1.087

Table 4.5: Comparison of densification factor α between real and synthetic datasets.

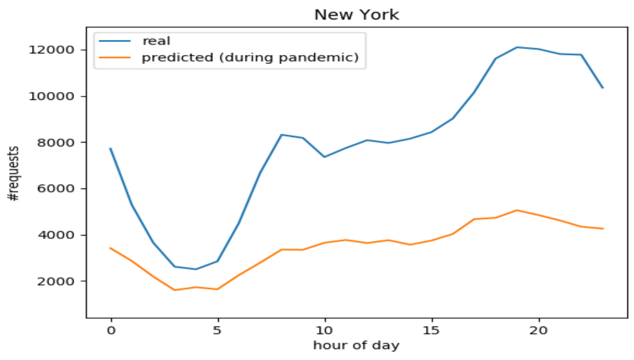


Figure 4.6: Temporal variability of ride requests for a typical day in New York using real data and a pandemic-hit day during Covid-19 when there was a lockdown imposed.

4.7 Modulating Volume of Ride Requests Using GANs

Covid-19 has affected ride-sharing services immensely in terms of reduction in volume of ride-requests. Looking at publically available New York yellow cab data for April 2020[7] when New York city had imposed a lockdown, we noticed more than fifty percent reduction in ride-requests in comparison to what we observed in a typical week of 2016, which was used for most of our analysis and training. We used our GANs model to generate ride requests by tweaking the ride intensity parameter with the goal that the model would closely imitate the hypothesis that there was greater than fifty percent reduction in volume of ride requests, and also to understand how the pattern of ride requests over a day would look like.

In figure 4.6 we compare the volume of requests for a typical day using real data and a day when lockdown was imposed in New York using synthetic data. Notice the high peaks of requests are flattened during busy hours and rate of increase of requests is much less during the pandemic/lockdown. The observed densification factor (α) during the pandemic was still similar to as observed during a non-pandemic day which was also our observation using real data for a DPL plot; it is expected for the edges versus node growth to oscillate on the same straight line defined by α in the DPL plot. Despite the fact that we do not have real pandemic data to verify the temporal pattern but intuitively looking at Figure 4.6 makes sense since busy hours would not experience a similar surge in requests during a lockdown as on a typical day. This example further highlights a *what-if* scenario which can easily be exploited by applying a GANs modeling approach.

4.8 Discussion

The emergence of ride sharing services and the availability of extensive data sets from such services are creating unprecedented opportunities for: 1) doing city-scale data analytics on urban transportation for supporting Intelligent Transportation Systems (ITS); 2) improving the

[7]https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

efficiency of ride sharing services; 3) facilitating real-time traffic congestion prediction; and 4) providing new public services for societal benefit. Moreover, the power of neural networks for machine learning has allowed the creation of useful models which can capture human behavior and dynamic real-world scenarios.

Simulations are a very useful tool to understand city level dynamics. Simulating the city or urban dynamics means implementing it step by step, in order to produce something which looks similar to a real setting. And, doing large number of simulations gives a good estimate of the evolution and dynamics of the human mobility pattern.

The model used to generate synthetic data generated here is not constrained in the geographical or temporal space. Although it may be useful to define specific regions and times for which data should be generated based on the application. The model here can also act as a feature vector for a particular geographic region for other applications.

The key contributions here include:

- We map the ride requests of ride sharing services into a time sequence of images that capture both the temporal and spatial attributes of ride request patterns for a city.
- Based on extensive real world ride request data, we introduce a GANs based workflow for modeling and generating synthetic and realistic ride request data sets for a city.
- We further show that our GANs workload can be effectively scaled using Xeon CPU clusters on AWS, in reducing training times from hours to minutes for each city.
- Using previous work on modelling urban mobility patterns, we validate our GANs generated data sets for ride requests for four major US cities, by comparing the spatial and temporal properties of the GANs generated data sets against that of the real data sets.
- We apply publically available PoI data to generate synthetic ride request data to highlight that it is possible to generate mobility data even if real data was not accessible for any new city.

There are other promising avenues for further research. Some open research topics include:

- Using the GANs generated data sets for experiments on new algorithms for dynamic ride pooling, and real-time traffic congestion prediction.
- Using the GANs generated data sets for conducting experiments on *what-if* scenarios related to traffic congestion prediction and mitigation, and planning for future development of transportation infrastructures.

Chapter 5

Real-world Applications of Human Mobility

Here we formulate and propose models to handle some real-world applications related to human mobility. The two problems discussed here require large scale human mobility data to understand different *what-if* scenarios. With significant increase in urbanization and penetration of ride-sharing platforms, it has become essential for public authorities and private entities to focus on ways to better utilize their resources, i.e. drivers and vehicles. In particular the two problems discussed here we do a limit study on their effectiveness to tackle growing human mobility demand:

1. *Dynamic Ride Pooling*: Increase in ride pooling can potentially benefit society. Human mobility and transportation is a major issue for many large urban areas in the world. As ride sharing services become ubiquitous, other than providing convenient and affordable transportation for the mobile population, there is the potential for such services to also contribute societal benefits. In this problem, we take a data-driven approach to investigate the subject of dynamic ride pooling. Dynamic ride pooling in real time without advanced knowledge of ride requests is a challenging problem. The pooling decisions must be made in real time for a large number of vehicles. Ride requests occur dynamically and continuously and can be widely distributed spatially in terms of pickup and drop-off locations. The pooling decisions must take into account both temporal and spatial proximity of the ride requests. Increasing the proximity scope for harvesting poolable ride requests can improve the amount of pooling and overall pooling benefits. However this can also lead to increase of travel time penalty for riders. Good benefit-cost trade off is essential.
2. *Vehicle Placement*: Ride-sharing companies specifically have turned their attention towards the problem of optimizing rider experience. In particular, they have begun to examine ways to minimize rider wait times. Future vehicular technologies like autonomous cars can also benefit from algorithms to reduce wait times. The real-time nature of placing vehicles introduces two challenges: first, drivers should proactively predict where future pickup requests will be located and go to these locations in anticipation of future pickup requests, eliminating passenger waiting due to vehicle travel time. Yet to accurately predict future requests in the next few minutes, drivers must account for not just overall patterns in ride requests, but also real-time temporal and geo-spatial request fluctuations, which are influenced by human mobility dynamics. Second, these vehicle (driver) placement

decisions must be made quickly, which precludes any significant coordination between different vehicles.

5.1 Dynamic Ride Poooling

In chapters 2 and 3 we have established that human mobility patterns for a city vary both temporally and spatially, and there is strong spatial clustering in densely populated areas even for very short time intervals (Jauhri, Foo, et al., 2017). We make use of this key observation in formulating our dynamic ride pooling methods. We take advantage of the temporal similarity of spatial patterns of ride requests in formulating the pooling methods discussed in Section 5.1.3.

5.1.1 Travel Distance Distribution

The distribution of travel distances provides insight about when should a vehicle actively look for pooling opportunities. If the distribution of trip distances is uniformly distributed, then it would make sense to pool riders at any point after initial pick up as the penalty of extra travel time would amortize over riders. However, if the density of travel distance distribution is skewed, e.g. towards shorter distance trips, then the pooling methods must be more judicious.

We fit the travel distribution to a doubly Pareto log-normal (DPLN) distribution (Reed and Jorgensen, 2004) which yields a good fit with parameters. The major insight drawn from this distribution is the extremely long tail. This is shown for all ride requests (San Francisco) received over an hour in Figure 5.1a. Given the long tail, the benefits of pooling as a vehicle moves away from the initial pick up point diminishes, and plateaus after a certain point. A small peak is visible around 23km which also corroborates prior work (Matsubara et al., 2013). In log-log scale, Figure 5.1b, we observe that the DPLN distribution may not be the best fit. The distinguishing feature of a DPLN plot are two linear plots, and a hyperbolic middle (Seshadri et al., 2008) which are difficult to infer from the plot. Spatial clustering (many dense communities) of ride requests (Jauhri, Foo, et al., 2017) along with long tail distribution (shown in Figure 5.1a) of travel distances provide statistical support for our proposed pooling methods described later.

5.1.2 Ride Pooling

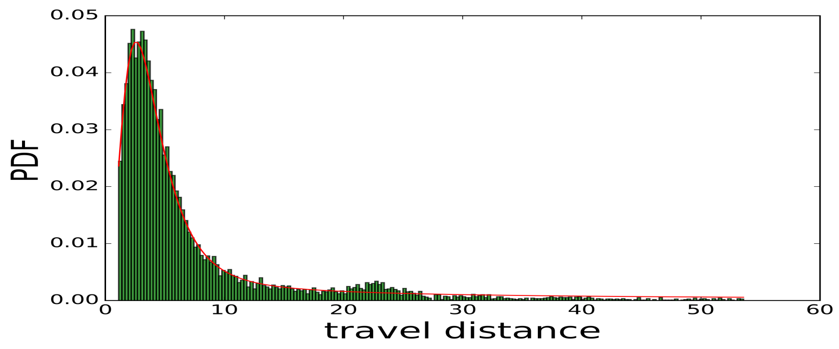
Refer to Table 5.1 for terms and symbols used in this chapter.

The *dynamic* ride pooling problem involves identifying which ride requests and how many ride requests should be bundled into a single vehicle. This involves making decisions on the fly, in real time, in response to spontaneously occurring ride requests. This is done by first identifying a *primary* ride request and use it as the reference for further pooling of other additional *secondary* ride requests. The pooling of multiple ride requests into a single vehicle is referred to as a plan $\mathcal{P}$.

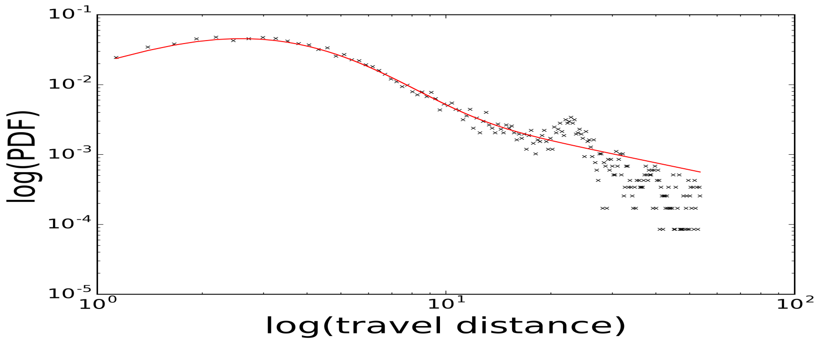
Each pooled vehicle is associated with a plan $\mathcal{P}$, and each plan has its associated benefits and costs. The potential benefits include service efficiency and profitability. With ride pooling, the average cost for servicing a ride request can be reduced, and fewer vehicles are needed to service the same amount of total ride requests. The potential costs for ride pooling include inconvenience

Symbol	**Definition**
m	Count of all ride requests.
$time(\text{‘event’}, i)$	Timestamp of ride request i associated with an event. ‘event’ here could be any of the four values: ‘src’: source or pick up; ‘des’: destination or drop off; ‘src_p’: pick up when pooled’ ‘des_p’: drop off when pooled.
$loc(\text{‘event’}, i)$	Location of ride request i associated with an event. ‘event’ could either be ‘src’ or ‘des’ i.e. pick up or drop off locations of ride request i.
$\theta(i, j)$	Angle between the trajectories of ride requests i & j.
$\mathcal{S}_t$	All the ride requests received in the t-th time window.
$\mathcal{P}$	Plan is a group of pooled ride requests in one vehicle.
p	Primary ride request i.e. the first request in any plan. Ride request time of p is always earlier than any secondary request in plan $\mathcal{P}$.
k	Maximum number of requests allowed in a plan.
ϵ_t	Length of a time window (minutes).
ϵ_{sr}	Radius of the circular source region, that centers at $loc(\text{‘src’}, p)$ (primary request’s pick up location).
ϵ_{dr}	Radius of the circular destination region, that centers at $loc(\text{‘des’}, p)$ (primary request’s drop off location).
ϵ_θ	Maximum angular difference (in degrees) between the trajectory of the primary ride request and the trajectory of any secondary ride request.
$rect(\epsilon_w, \epsilon_l)$	Rectangular region with width ϵ_w, and length ϵ_l.

Table 5.1: Terms and Symbols



(a) PDF versus trip distances (in kilometers) of ride requests.



(b) Log-log plot of PDF and trip distance distributions.

Figure 5.1: Double Pareto lognormal fitting results with parameters $\alpha = 0.033, \beta = 2.54, \nu = 1.572, \tau = 0.494$ for travel distance distribution of ride requests received for a period of an hour in San Francisco. DPLN fit is depicted by the red line.

to the riders in terms of longer wait time for pick up and/or longer travel time for drop off. Also, to accommodate large number of pooled ride requests in one vehicle, larger sized vehicles may be needed. Hence, dynamic ride pooling requires performing trade offs between the benefits and the costs. The goal is to maximize the benefits while minimizing the costs. Given the set of all ride requests, the task is to find mutually exclusive subsets, or plans $\mathcal{P}$, of ride requests, such that each subset can be pooled into one vehicle.

We do not focus on the problem of finding optimal routes for each plan. Given travel times for different road segments, there are plenty of known navigation algorithms (Bast et al., 2016) which work well in practice. The optimal route problem is strongly dependent on the quality of the pooling plans. A sub-optimal plan of requests would yield sub-optimal overall solution regardless how optimal the route finding algorithm is. Therefore, our focus is on solving the problem of finding groups of ride requests which can be effectively pooled.

Effective pooling must take into account both temporal and spatial proximity of ride requests. This means that in searching for requests which can be pooled, one should only consider ride requests that occur very close in time, and with pick up locations that are very near each other. Also, the drop off locations should not be too far apart. In order to perform real-time and on-demand ride pooling, we need to impose a very small time window as the *temporal proximity constraint*. This means that pooling can only occur for rides that are requested within a small time window of size ϵ_t, typically on the order of few minutes. All rides requested within the same time window t is denoted $\mathcal{S}_t$. The next time window $t + 1$ immediately follows t with same duration of ϵ_t with its ride requests denoted as $\mathcal{S}_{t+1}$. Here we fix ϵ_t at 5 minutes and only consider pooling of ride requests occurring within each 5 minute interval. Next subsection presents several options for the *spatial proximity constraints*.

5.1.3 Pooling Methods

For any ride pooling plan $\mathcal{P}$, all ride requests in the plan must satisfy certain specified proximity constraints. We have already specified the temporal proximity constraint of ϵ_t. Here we present three options for specifying the *spatial proximity constraints*. There are two aspects to the spatial proximity constraints, one for specifying proximity of all the requesting or pick up locations, i.e. between pick up locations of the primary request $loc(\text{‘src’}, p)$ and any secondary request j given by $loc(\text{‘src’}, j)$. The other aspect for specifying proximity of all the destination or drop off locations, i.e. between $loc(\text{‘des’}, p)$ and $loc(\text{‘des’}, j)$. We now introduce three specific pooling methods based on how each defines the size and shape of the *source* (pick up) *region* and the *destination* (drop off) *region* as its spatial proximity constraints for pooling. Each pooling method starts with a reference or primary ride request p and specify its source and destination regions relative to the pick up and drop off locations of p.

Restricted (RES)

The first pooling method, called *Restricted* (RES) pooling, is illustrated in Figure 5.2. The source region is a circle centered at the source of p with radius ϵ_{sr}, and the destination region is another circle centered at the destination of p with radius ϵ_{dr}. Therefore, any ride request that starts in the source region of p and ends in the destination region of p satisfies the spatial proximity constraints and becomes a candidate for pooling with p. Hence, given a primary ride request p, and other ride requests within the same time window given by the set $\mathcal{S}_t - \{p\}$, any request $j \in \mathcal{S}_t - \{p\}$ is added to the plan $\mathcal{P}$ if:

1. $dist(loc(\text{‘src’}, p) - loc(\text{‘src’}, j)) < \epsilon_{sr}$
2. $dist(loc(\text{‘des’}, p) - loc(\text{‘des’}, j)) < \epsilon_{dr}$

where ϵ_{sr} and ϵ_{dr} define the circular source and destination regions centered around pick up and drop off locations of the primary ride request, respectively; $dist()$ returns the straight line between the two points. Keeping ϵ_{sr} and ϵ_{dr} small will result in smaller source and destination regions, which can ensure that all requests in each $\mathcal{P}$ are *highly proximal*. This means all the ride requests in $\mathcal{P}$ can be serviced without incurring too much time penalty for riders. However, this can unnecessarily limit the potential benefits of ride pooling. In Section 5.1.7 we explore more in depth the various possible configurations (different values of ϵ_{sr} and ϵ_{dr}) for the RES pooling method.

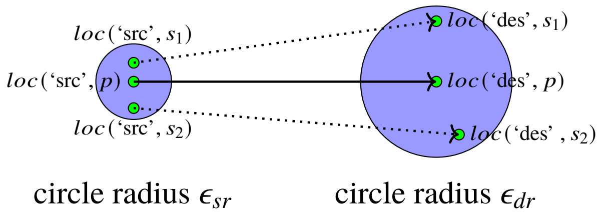


Figure 5.2: *Restricted Pooling*: The source region is defined by a circle with radius ϵ_{sr} and centers at the pick up point of the primary request p. The destination region is defined by another circle with radius ϵ_{dr} and centers at the drop off point of p. Secondary ride requests can be pooled with p, if their pick up and drop off points are located inside the source and destination regions of p, respectively. Both secondary ride requests shown (s_1, s_2) meet this requirement.

Directional (DIR)

The RES method is quite restrictive and misses many additional pooling opportunities. By simply enlarging the RES circular regions is not the most effective way to increase ride pooling. We introduce a more efficient way to enlarge the source and destination regions by incorporating directionality. The *Directional* (DIR) pooling method is illustrated in Figure 5.3. The source region for DIR is defined by a rectangle, containing the source location of p (primary ride request), with width ϵ_w and length ϵ_l, and a companion trapezoidal destination region, containing the destination location of p, and defined by an angular parameter ϵ_θ. By imposing this angular parameter as a constraint, more potential ride-pooling candidates can be identified along the direction that matches the trajectory of the primary ride request p. As shown in Figure 5.3, the trapezium has an open side; this means a secondary ride request does not have to end near the destination of the primary ride request, as long as the angle of the secondary request relative to the primary is less than ϵ_θ, it can be included in the plan. This facilitates much larger destination regions without incurring significant travel time overhead to pick up and drop off additional riders.

Therefore, for the DIR pooling method, any ride request that starts in the source region of p and ends in the destination region of p satisfies the spatial proximity constraints and becomes a candidate for pooling with p. Hence, for a given primary ride request p, and any other request $j \in \mathcal{S}_t - \{p\}$ in the same time window t, j is added to the plan $\mathcal{P}$ if:

1. $loc(\text{'src'}, j) \in rect(\epsilon_w, \epsilon_l)$
2. $|\theta(j, p)| < \epsilon_\theta$

where $rect(\epsilon_w, \epsilon_l)$ defines a rectangular source region with width ϵ_w, and length ϵ_l, $|\theta(j, p)|$ defines the angular difference between the trajectories of ride requests j and p, and ϵ_θ defines the maximum angular difference allowed between any j and p.

Hybrid (HYB)

The *Hybrid* (HYB) pooling method is a combination of RES and DIR. HYB first pools using RES then pools additional candidates using DIR. The intuition is that RES first takes care of pooling all *spatio-temporally proximal* ride requests. Then, DIR efficiently captures the additional *directionally proximal* ride requests. DIR is only applied to the requests that did not get pooled

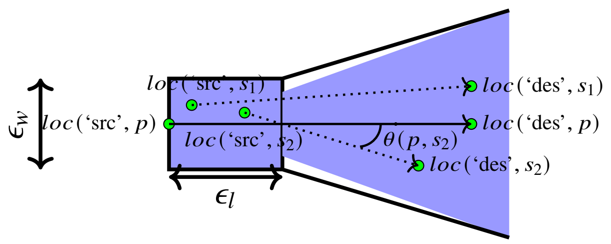


Figure 5.3: *Directional Pooling*: The source region is defined by a rectangle with width ϵ_w and length ϵ_l that starts at the pick up point of p and stretches its length along the direction towards the drop off point of p. The destination region is an adjacent trapezium (with an open side) that contains the drop off location of p. The slant of the trapezium is defined by the angular parameter of ϵ_θ. Secondary ride requests can be pooled with p if their pick up and drop off points are inside the source and destination regions and their trajectory angles towards their drop off points are withing ϵ_θ relative to that of p. The two pooled ride requests s_1 and s_2 meet this requirement. Note that the pick up points of both s_1 and s_2 are within the rectangle. The angle between the trajectory of p and each pooled ride s_i satisfies $\theta(p, s_i) < \epsilon_\theta$.

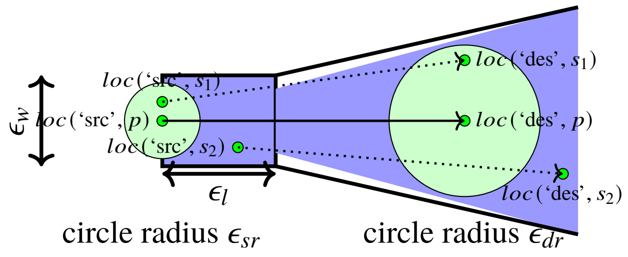


Figure 5.4: *Hybrid Pooling*: The two circular regions are used for the RES pooling phase. The rectangular and open trapezium regions are then used for the DIR pooling phase. Two pooled rides are shown with one satisfying conditions for RES pooling, the other for DIR pooling.

with RES, because doing otherwise can incur additional costs to the RES plans. Figure 5.4 shows the HYB pooling regions.

5.1.4 Pooling Metrics

To quantify the benefits and costs of a plan, we define the following four metrics. The first two metrics measure the benefits achievable by a pooling method. The third metric concisely captures the potential for pooling. The fourth metric is a per-rider cost metric measuring the inconvenience experienced by each rider participating in pooling.

Total Travel Distance Reduction

The total travel distance saved by a plan is calculated by taking the difference between the sum of individual travel distances if no pooling occurred and the total travel distance of the pooled plan (accounting for picking up and dropping off all the riders):

$$dist\_red(\mathcal{P}) = \sum_{i \in \mathcal{P}} travel\_dist(i) - travel\_dist(\mathcal{P})$$

The total travel distance reduced by pooling as a percentage over all the ride requests for an entire week is defined as:

$$total\_dist\_red\_\% = \frac{\sum_{\mathcal{P}} dist\_red(\mathcal{P})}{\sum_{j=1}^{m} travel\_dist(j)} \tag{5.1}$$

Vehicle Count Reduction

Number of vehicles reduced by a plan is:

$$veh\_red(\mathcal{P}) = |\mathcal{P}| - 1$$

The total number of vehicle count reduced as a percentage over all the ride requests for an entire week is defined as:

$$total\_veh\_red\_\% = \frac{\sum_{\mathcal{P}} veh\_red(\mathcal{P})}{m} \tag{5.2}$$

Average Poolability

Poolability is defined as the percentage of ride requests that are *poolable* over all ride requests (Jauhri, Foo, et al., 2017). If a ride is *poolable*, it will belong to a plan $\mathcal{P}$ along with the primary ride request. Real-time pooling is possible if poolability can be calculated within the small time window, ϵ_t (next Section 5.1.5, explains time windows with more details). Poolability for a given time window can be calculated as:

$$poolability(\mathcal{P}_t, \mathcal{S}_t) = \frac{\sum_{\mathcal{P}_t} |\mathcal{P}_t|}{|\mathcal{S}_t|}$$

where $\mathcal{P}_t$ is the set of all the plans for the time window t.

Average poolability over all time windows is given by:

$$avg\_poolability = mean(\sum_{t} poolability(\mathcal{P}_t, \mathcal{S}_t)) \tag{5.3}$$

Equation. 5.3 provides the average poolability over all time windows. This is a more relevant metric to highlight the benefits of on-demand real-time pooling in comparison to aggregate poolability across time windows i.e. fraction of all ride requests poolable over all ride requests received across all time windows.

Average Trip Time Penalty

We model the average trip time penalty to be the delayed *arrival time at destination* for a rider when participating in a pooling plan. This penalty for any request i can be calculated as:

$$arrival\_penalty(i) = time(\text{`des\_p'}, i) - time(\text{`des'}, i) \tag{5.4}$$

where 'des' denotes the event of arriving at destination if i were *not* pooled and 'des_p' denotes the same but pooled.

It is important to note that $arrival\_penalty(i)$ may have a negative value. This usually happens when a plan is able to pick up a request earlier than its original pick up time, that is $time(\text{`src\_p'}, i) < time(\text{`src'}, i)$. Also, Equation 5.4 only makes sense when i is poolable; non-poolable rides are not considered when calculating average trip time penalty.

The overall average trip time penalty is computed by taking the mean across all *pooled* riders:

$$avg\_penalty = \frac{\sum_{\mathcal{P}} \sum_{i \in \mathcal{P}} arrival\_penalty(i)}{\sum_{\mathcal{P}} |\mathcal{P}|} \tag{5.5}$$

5.1.5 Experimental Details

Following are some important details regarding our experimental setup. First, in order to meet the *temporal proximity constraint* in real-time pooling, pooling methods have to divide the entire week into small time windows with $\epsilon_t = 5$ minutes. Pooling would only occur among ride requests occurring in the same time window; this ensures that ride requests get pooled together only if they occur *near* the same time. Note that the division into short time windows is also helpful in capturing the dynamic temporal patterns of the pooling metrics over an entire week.

After all the ride requests in our (week long) data set are separated by time windows, pooling methods are applied under specific rules. In each experiment, requests from the same time window are processed one at a time in chronological order. Each request is viewed as the primary request seeking for other pooling candidates. Once a plan, $\mathcal{P}$, is established, the decision of the order of pick up and drop off for each passenger and the exact routes to take in between these locations are as follows:

1. The order of pick up follows the order of request times.
2. The order of drop off is based on the distances away from the last pick up location, from nearest to farthest.
3. Routes to take in between any two geographical points are queried from GraphHopper, an open-source route navigation and planning tool (Karich and Schröder, 2014) that runs with maps and speeds using PBF[1] files by OpenStreetMap (OpenStreetMap contributors, 2017).

GraphHopper is also useful in calculating the travel distance and arrival time of each request if no pooling occurred. This is necessary for computing both the distance saved and extra distance incurred when a request participates in a pooling plan.

The above steps are repeated for different combinations of pooling methods, cities, maximum pooling degrees (maximum number of riders in a plan), and shapes and sizes of source and

[1]PBF files are available at http://download.geofabrik.de/

destination regions. We refer to a specific combination of methods and parameters as a *pooling configuration*. A pooling configuration consists of a pooling method and all its associated parameters. The short form notation METHOD $< \epsilon_{sr}, \epsilon_w, k >$ is used through out this chapter to facilitate comparison between different pooling configurations.

In our parameter space (ϵ_{sr}, ϵ_{dr}, ϵ_w, ϵ_l, ϵ_θ, k, and METHOD), two out of the seven parameters are kept to constant values for all experiments. The radius of the circular destination region, ϵ_{dr}, used in RES, is fixed at 1000 meters. This is because increasing ϵ_{dr} in RES does not provide enough benefits and at the same time incurs additional time penalty. The benefits of expanding the area of pooling regions to search for more pooling candidates can be achieved with increasing ϵ_{sr} for RES pooling, and in DIR pooling by using open ended trapezium (shown in Figure 5.3). Also, the maximum angular difference, ϵ_θ, is fixed at 20 degrees which gives a reasonable size to explore in the destination region in DIR pooling. The length of the rectangular source region in DIR, ϵ_l, varies for each pooling configuration, is explained in Section 5.1.7.

5.1.6 Experimental Goals

The goal of running such experiments on a real world data set is to find answers to the following questions,

- What are the best values to use for ϵ_{sr} and ϵ_w? ϵ's should ideally be kept small to exploit spatio-temporal proximity, but expanding these search regions can capture more pooling candidates.
- What is the best value of k? What is the effect of limiting k to small values since a typical vehicle may only be able to pool up to 3 or 4 passengers in the real-world setting? What if k is unrestricted, how much pooling is there?
- When selecting a pooling method for a particular city, how does one know which pooling method and configuration might be the best choice?
- How do all these factors correlate and vary in different cities, in different areas of a city, on different days of the week, at different times of the day?
- What is the best combination of these parameters that gives the best trade-off between benefits and costs?

These are all important questions in order to achieve effective dynamic ride pooling and we tackle them in this chapter.

5.1.7 Experimental Results & Analysis

Pooling Parameters & Configurations

We first characterize how each of the parameters ϵ_{sr}, ϵ_w, k behave individually. This can be done by sweeping a range of values for each parameter while holding all other parameters constant. Below are independent sweeps of the three parameters that are of interest, ϵ_{sr} (radius of RES circular source region), ϵ_w (width of DIR rectangular source region), and k (maximum degree of pooling) for the city of San Francisco.

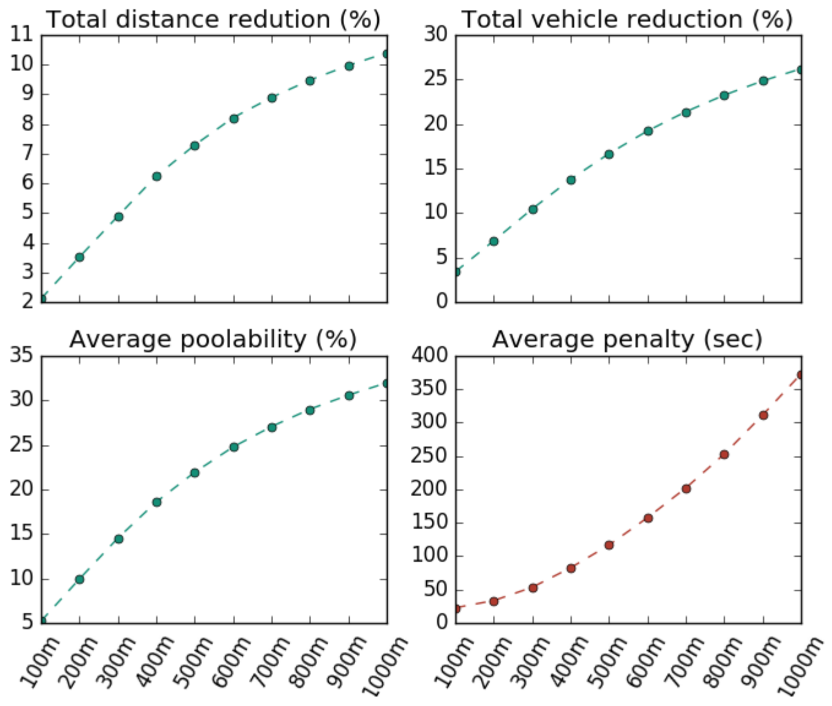


Figure 5.5: RES$< *, NR, 10 >$ for San Francisco

Figure 5.5 shows the results, in terms of the four pooling metrics (Equations 5.1, 5.2, 5.3, and 5.5), for the RES pooling method with the sweep of ϵ_{sr} from 100 to 1000 meters. The other parameters are fixed at $\epsilon_{dr} = 1000m$, and $k = 10$. ϵ_w, width of rectangle is not relevant (NR) for RES. This set of pooling configurations is denoted RES$< *, NR, 10 >$.

Figure 5.5 shows some very interesting trends as we increase the radius of the source region ϵ_{sr} from $100m$ to $1000m$. As the size of the source region increases, there is significant increase in the benefits of RES ride pooling. All three benefit metrics rise sharply but start to encounter some diminishing returns around $\epsilon_{sr} = 500m$. Unfortunately the cost in average travel time penalty also increases, but at a slower rate. The rate seems to accelerate starting at, interestingly, also around $\epsilon_{sr} = 500m$. These results suggest that for RES pooling, the choice of $\epsilon_{sr} = 500m$ provides the best benefit-cost trade off.

We next examine the pooling parameter ϵ_w, the width of the rectangular source region for DIR pooling. In this work, the length of the rectangle ϵ_l is computed based on the primary ride request by the following equation:

$$\epsilon_l = \left(\frac{\epsilon_t}{time(\text{‘des’}, p) - time(\text{‘src’}, p)}\right) \times dist(loc(\text{‘des’}, p) - loc(\text{‘src’}, p)) \tag{5.6}$$

Essentially ϵ_l is set as a *portion* of the straight-line distance between the drop off and pick up points of p. This limits the size of the rectangle so as to ensure all pick ups of pooled ride requests can be done within the time window of ϵ_t. Figure 5.6 shows the results for DIR pooling as we vary ϵ_w from $200m$ to $2000m$, while keeping $k = 10$. Value of ϵ_{sr} is not relevant (NR) for DIR pooling. We denote this set of pooling configurations as DIR $< NR, *, 10 >$.

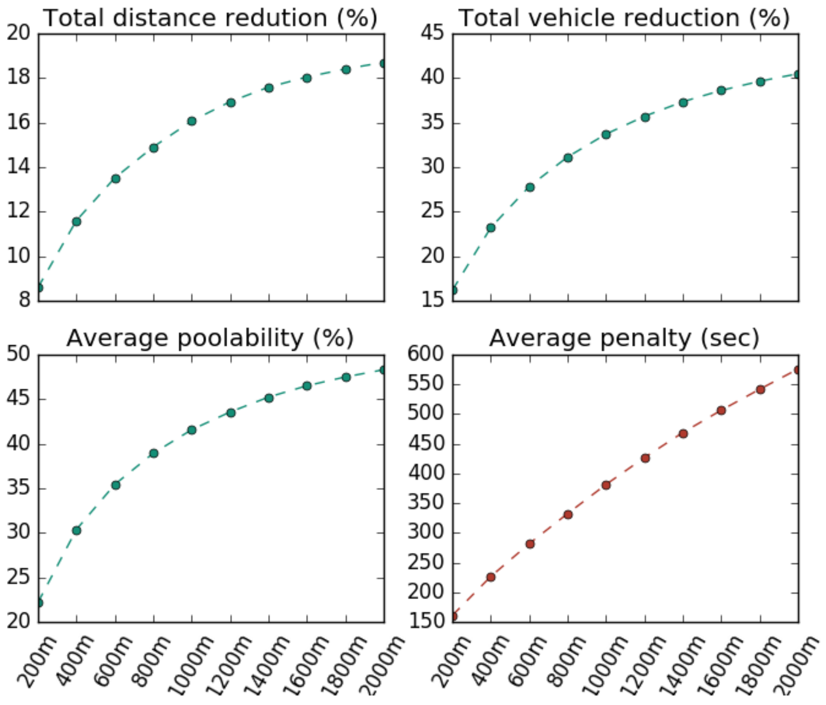


Figure 5.6: DIR $< NR, *, 10 >$ for San Francisco

Figure 5.6 exhibits very similar trending for the three benefit metrics as Figure 5.5. However the benefit curves for DIR are significantly higher than that for RES, although there seems to be slightly stronger diminishing returns at large values of ϵ_w. Corresponding with higher benefits, the trip time penalty is also higher for DIR, due to the overhead for picking up and dropping off more riders. However the penalty curve for DIR seems to stay quite linear missing the steeper rise of RES. These results clearly indicate there is significantly greater pooling opportunities and benefits with the DIR pooling method as compared to RES. The down side is the greater trip time penalty that will need to be addressed.

Finally we examine the parameter k, the maximum pooling degree allowed per vehicle, by varying k from two to nine. Figure 5.7 shows the results for the set of pooling configurations of DIR $< NR, 2000, * >$. We see very similar trending of the benefit and cost metrics in Figure 5.7 as in Figure 5.6. Similar to increasing pooling region size, increasing the maximum pooling degree also increases the benefits. There is substantial gain in allowing more ride requests to be pooled in each vehicle. However, there is a clear "knee" in the curves for all three benefit metrics at around $k = [4, 5]$. The cost curve is still mostly linear. These results indicate that there is significant benefit even at low values of $k = [2, 3]$. On the other hand the benefits continue to increase, with some diminishing returns, at high values of $k = [8, 9]$.

Pooling Method Comparison

We next focus on comparing the different pooling methods and determining the best pooling configurations, in terms of achieving the best benefit-cost trade off. Figure 5.8 compares the

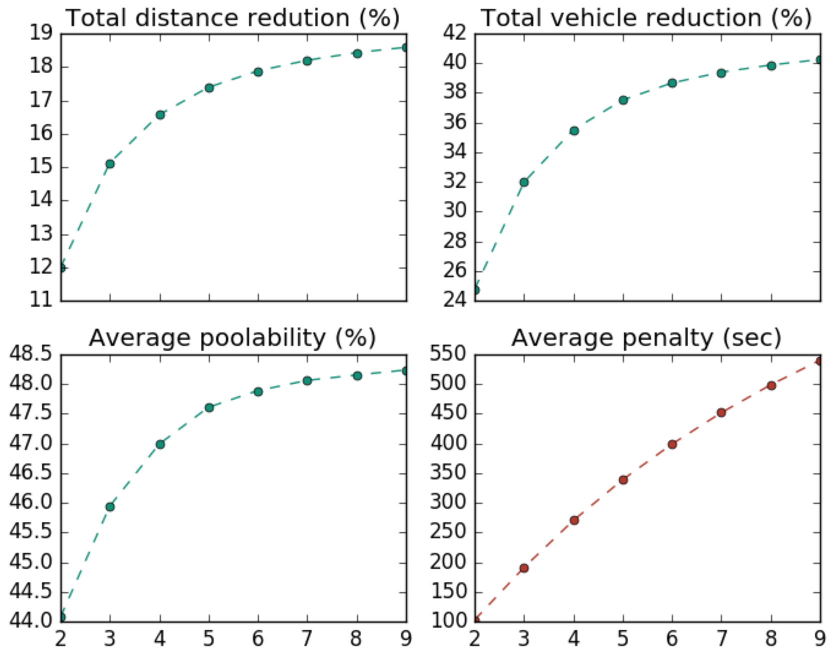


Figure 5.7: DIR $< NR, 2000, * >$ for San Francisco

benefits and cost of the three pooling methods RES, DIR, and HYB, using their respective configurations of : RES $< 100, NR, 3 >$, DIR $< NR, 2000, 3 >$, and HYB $< 100, 2000, 3 >$ in San Francisco. Comparing to RES, DIR achieves roughly 7x improvement on total distance saved and 9x improvement on number of vehicles reduced. These improvements come with a 9x increase in average travel time penalty. Although the roughly 3-minute average travel time penalty is quite tolerable. Going from RES to DIR essentially relaxes the spatial proximity constraints and exposes more pooling opportunities. However, the increased benefit comes with greater penalty.

Comparing to DIR, HYB achieves very comparable benefits. However the average travel time penalty for HYB is approximately 10% lower than that of DIR. Using RES, HYB first captures all the spatio-temporally proximal requests at much lower travel time penalty. HYB then uses DIR to significantly increase poolable ride requests beyond that of RES by pooling many directionally proximal ride requests.

With fixed values of $\epsilon_{sr} = 100m$, $\epsilon_w = 2000m$, and $k = 3$, Fig 5.8 clearly shows that among the three pooling methods, HYB is the preferred choice. The next subsection further explores the HYB pooling method by sweeping the parameters to search for best HYB pooling configurations.

5.1.8 Best Pooling Configuration

In this search for the best HYB pooling configurations, we vary the values of ϵ_{sr} from 100m to 1000m, and k from 2 to 10, but fix $\epsilon_w = 2000m$. Using $\epsilon_w = 2000m$ maximizes the size of the

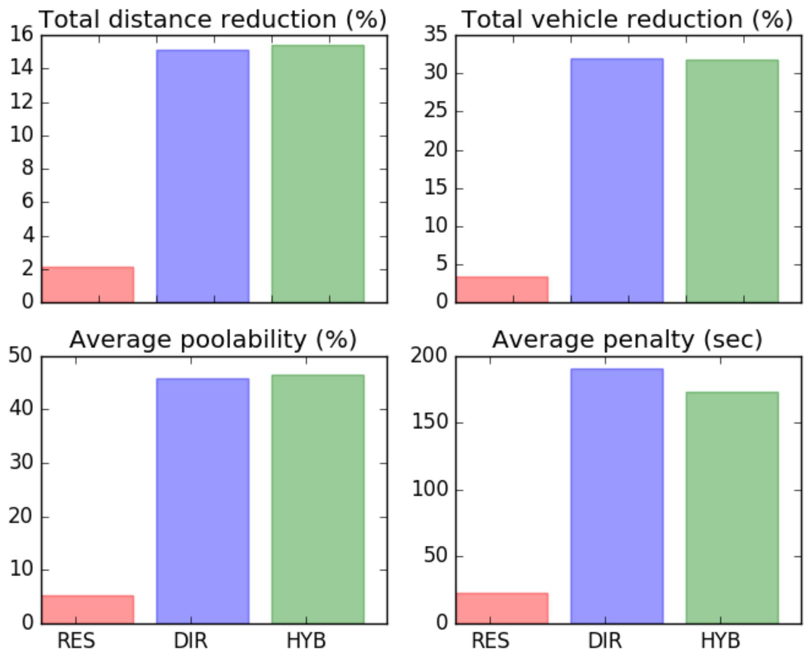
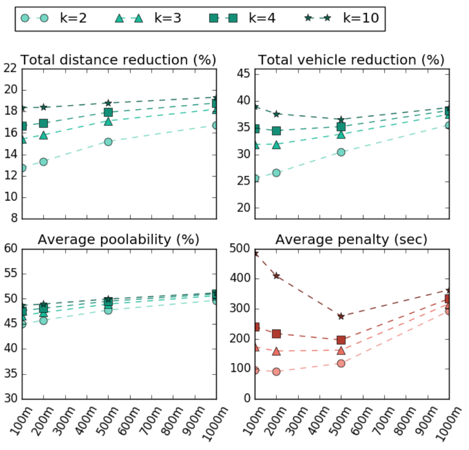


Figure 5.8: Metric comparison between RES, DIR, and HYB

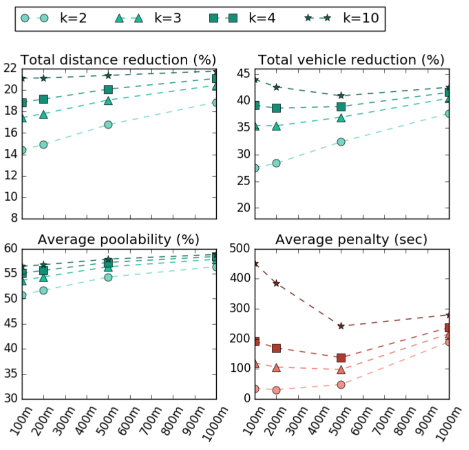
DIR pooling regions. Smaller values unnecessarily limit opportunities for pooling. The resultant set of pooling configurations is denoted HYB $< *, 2000, * >$. Figures 5.9a, 5.9b, and 5.9c, show the results of HYB $< *, 2000, * >$ for San Francisco, New York, and Los Angeles, respectively.

Looking at the results, we see both similarities and differences across the three cities. All the benefit metrics increase with increasing ϵ_{sr} (size of source region) and k (maximum pooling degree). For k there is a strong diminishing return beyond $k = 3$. There is also a slight diminishing return beyond $\epsilon_{sr} = 500m$. The travel time penalty curves for all three cities also exhibit very similar patterns, with an interesting inflection point around $\epsilon_{sr} = 500m$. The "dip" is more apparent as k increases. This has to do with the balancing of pooling attributed to RES v.s. DIR. When ϵ_{sr} is small (100m) most of the pooling comes from DIR which gives great benefits but also has heavy penalty (shown in the previous subsection). As ϵ_{sr} increases, RES can finally start to capture all the spatiotemporal proximal requests while bringing overall average penalty down. This indicates that $\epsilon_{sr} = 500m$ yields the best balance between RES and DIR where strengths of both methods are leveraged. It seems quite clear the sweet spots of $k = 3$ and $\epsilon_{sr} = 500m$ would be the best pooling configuration.

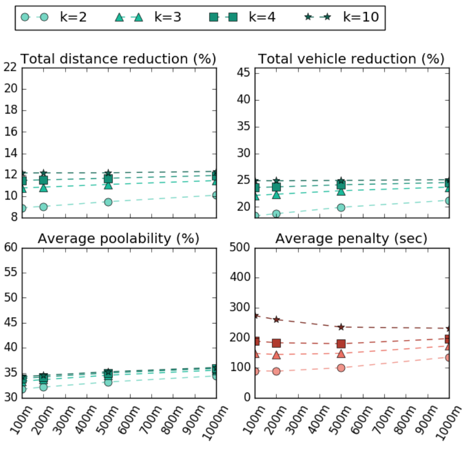
We also observe significant differences across the three cities. The three cities exhibit quite different levels of poolability. In New York, 50-60% of all ride requests are poolable, while in Los Angeles only 30-35%. San Francisco is between these two extremes. We see very strong correlation between poolability and the two benefits of total travel distance reduction and total vehicle count reduction. Again, New York has much higher levels for both benefits as compared to Los Angeles. What makes New York so ride pooling friendly and why is it so much more



(a) HYB $< *, 2000, * >$ for San Francisco.

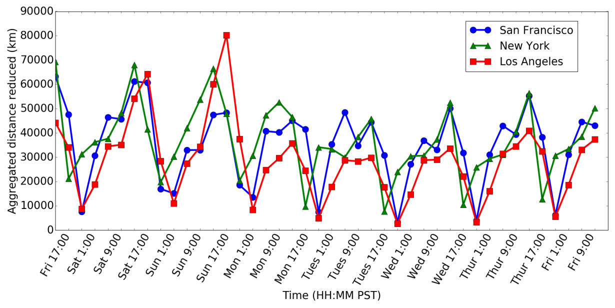


(b) HYB $< *, 2000, * >$ for New York.

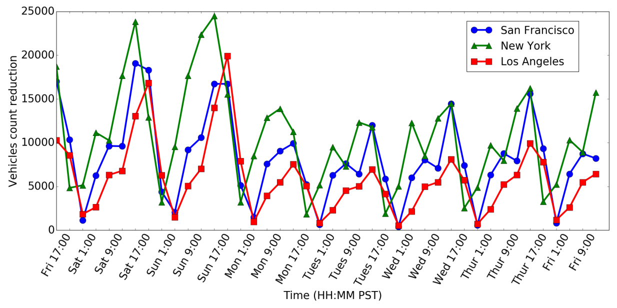


(c) HYB $< *, 2000, * >$ for Los Angeles.

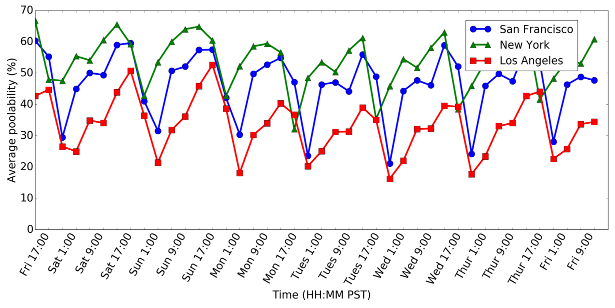
Figure 5.9: Sensitivity analysis by sweeping across ϵ_{src} and k.



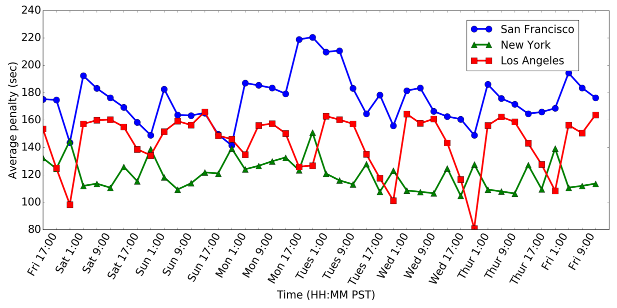
(a) Travel distance reduction.



(b) Vehicle count reduction.



(c) Average poolability (%).



(d) Average penalty (sec).

Figure 5.10: Temporal pattern of the four pooling metrics over a week for three cities.

difficult to do ride pooling in Los Angeles? These are good questions for future research.

We conclude that the best pooling configuration is **HYB $< 500, 2000, 3 >$**. Across the three cities, HYB $< 500, 2000, 3 >$ yields on average 15.76% of total travel distance saved and 31.22% of number of vehicles reduced, while incurring an average trip time penalty of only 135.84 seconds. This translates to roughly one out of every three vehicles can be removed from the roads if riders are willing to ride pool and accept a travel time penalty of approximately two minutes! These results are quite encouraging and imply that dynamic ride pooling is a must for densely populated urban areas.

5.1.9 Temporal Patterns

There are interesting weekly temporal patterns for all four pooling metrics. Figures 5.10a, 5.10b, 5.10c, and 5.10d, plot the weekly temporal patterns of the four metrics for the three cities, using the pooling configuration of HYB $< 500, 2000, 3 >$. The plots start at 5:00 P.M. PST (12 A.M. UTC) on a Friday and each data point is an aggregated value over a 4-hour period (hence a total of 42 data points for a week). All of these plots share similar weekly patterns and correlate well with the weekly pattern shown for ride requests. It makes sense that during times of higher ride request volumes there is greater poolability and potential for pooling benefits. For instance, poolability can reach up to 60% on weekend afternoon hours. Figure 5.10d depicts that the average penalty is higher during low ride request volumes. The temporal pattern should be quite useful in guiding the implementation of ride pooling algorithms that can potentially be temporally adaptive.

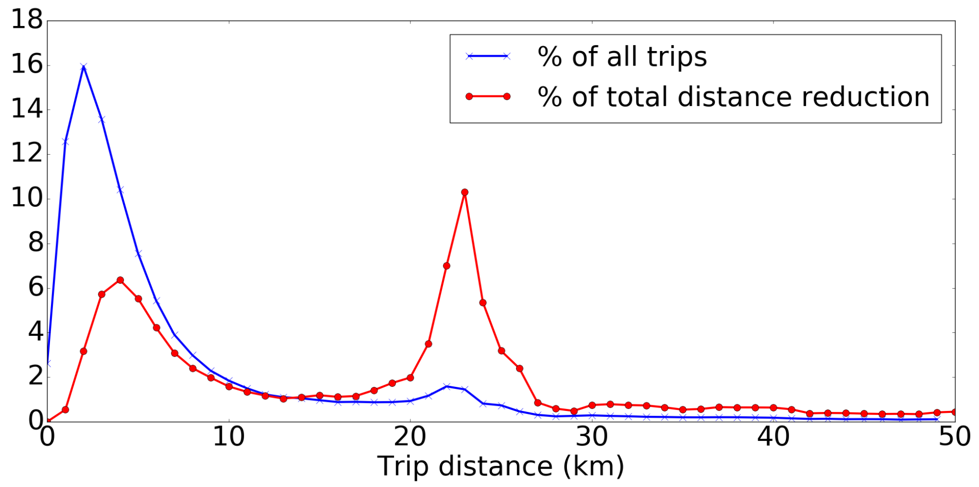


Figure 5.11: Distribution of trip distances and travel distance savings.

5.1.10 Trip Distance & Duration Distributions

Another interesting perspective to look at the pooling results is creating distributions of requests (trips) by their route distances and savings. Figure 5.11 shows the distribution of all requests by their route distances in San Francisco over a week. Notice the bump around trip distances of 23 to 25 kilometers; we believe this could be due to large volume of trips between the San Francisco International Airport (SFO) and the downtown areas. Pairing this with the distribution of distance saved binned by trip distances (Figure 5.11, with HYB $< 500, 2000, 3 >$), another insightful observation can be made; approximately 5% of all requests are responsible for over 30% of the total travel distance savings! This could be very useful information in the implementation of pooling algorithms that can maximize the benefit/cost ratio.

5.1.11 Societal Benefits of Pooling

Dynamic ride pooling has significant potential for societal benefits. For example, reducing total travel distance by 15% leads to proportional reduction in fuel consumption and CO_2 emission. Taking 31% of vehicles off the roads can reduce traffic congestion in a city. Notice our results indicate that as the ride request volume increases there is greater potential benefit for ride pooling. Ride sharing services can reduce their overall operation cost and improve their efficiency. This then leads to lower cost for the riders. There is the potential for a simultaneous three-way win that benefits ride sharing companies, mobile population, and densely populated cities.

5.2 Vehicle Placement

In this section, we formulate the fine-grained (at a timescale of less than a few minutes and spatial scale of a few hundred meters) real time vehicle placement problem, and propose a distributed online learning approach to this problem. An online learning algorithm gives us the flexibility to consider real-time ride request patterns in making our placement decisions. Historical information can help inform these decisions, but the utility of such historical information may vary a lot geo-spatially.

5.2.1 Problem Definition

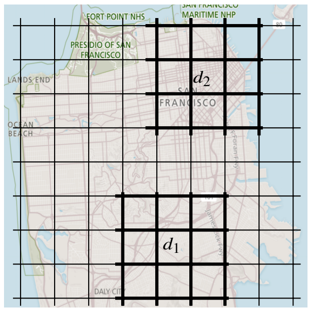
We consider a geographical surface that is divided into equally sized cells of side length ϵ each, as shown in Figure 5.12a, to get an aggregate of $a \times b$ cells. We suppose a consecutive set of time snapshots, taken at intervals τ_ϵ, where each time snapshot t specifies a set of drop-offs at different cells, represented by the matrix $\boldsymbol{D}_t \in \mathbb{Z}^{a\times b}$, and pickups $\boldsymbol{P}_t \in \mathbb{Z}^{a\times b}$ of ride requests. Each (i, j) entry in $\boldsymbol{D}_t$ or $\boldsymbol{P}_t$ is the sum of all drop-offs or pickups that occurred at t, in cell (i, j). In this work, we have kept the length of each time snapshot interval to be a few minutes, reflecting our goal of keeping rider wait times under two minutes. The *real time vehicle placement* problem is to decide how to move vehicles from drop-off cells at time snapshot t to neighboring cells such that pickups happen at those cells at $t + 1$. Formally, our goal is to place vehicles in the "best" neighboring cells, i.e., so as to maximize a reward at time snapshot t that is defined as:

$$R_t(\boldsymbol{P}_t, \boldsymbol{\Gamma}_t) = \frac{1}{n_t}\sum_{i,j} min(\boldsymbol{P}_t[i,j], \boldsymbol{\Gamma}_t[i,j]) \tag{5.7}$$

where: $\boldsymbol{\Gamma}_t \in \mathbb{Z}^{a\times b}$; each entry represents the number of vehicles placed at each cell, or zero otherwise. $n_t = \sum_i^a \sum_j^b \boldsymbol{D}_{t-1}[i,j] = \sum_i^a \sum_j^b \boldsymbol{\Gamma}_t[i,j]$; the number of drop-offs at $t-1$ or equivalently, the number of placements at t. For any t, $\boldsymbol{P}_t$ is known and the goal is to find $\boldsymbol{\Gamma}_t$ using $\boldsymbol{D}_{t-1}$ such that R_t is maximized. The reward R_t thus represents the average number of successful pick-ups per cell.

The algorithms decide the placement of each vehicle without knowing where future pick-ups will occur. Moreover, for every drop-off by a vehicle at time $t-1$ at cell (i, j), its corresponding placement at t may be either (i, j) itself or one of the neighbouring cells. This constraint is shown by the bolded cell outlines in Figure 5.12a. The number of neighbouring cells at which the vehicle can be placed is determined by the vehicle's ability to travel to the neighbouring cell within the time snapshot interval; it is thus determined by the length of the snapshot interval (τ_ϵ) and ϵ. This constraint also ensures that vehicles incur negligible cost (e.g., gasoline used) in moving to the exact pick-up location, as they only move to neighbouring cells. We denote the radius of the permitted neighbourhood by ϵ'. The goal of our online algorithm is then to choose the vehicle placements within any time interval $[t-1, t)$ so as to maximize the reward (5.7), subject to the constraint that vehicles cannot move too far from the cell where they dropped off passengers at time $t-1$. Any excess placements for t are not considered in the reward function for pickups after t; for $t+1$, placements are computed using drop-offs at t only.

The real time vehicle placement problem generalizes other known problems in computer science. For instance, in the k-server problem one must place k servers within a metric space so as to fulfill requests that can come in at any point in the space (Manasse et al., 1990). Viewing the servers as vehicles, our problem is a dynamic version of the k-server problem: we must adjust the vehicle placements in real time as new requests arrive, subject to constraints on the travel time of the vehicles.



(a) Geographical space discretized into cells; each cell of side ϵ. For each drop-off d_i, there are 9 possible cell placements highlighted by a thick border.

Figure 5.12: Illustration of vehicle pick-ups and drop-offs.

5.2.2 Proposed Algorithms

In this section, we examine in detail three potential algorithms for the real-time vehicle placement problem. We make use of the fractal analysis in the previous section to analyze their effectiveness. For each drop-off, the possible set of neighboring cells to which the vehicle can potentially be moved are restricted; see Figure 5.12a. If the drop-off location is in the (i, j) cell, then the set of neighboring cells within a square-shaped neighborhood centered at (i, j) and containing c^2 cells is denoted as $\eta(\square, (i, j), c)$. Figure 5.12a illustrates neighborhoods with c^2 = 3x3 cells.

Baseline Algorithm: Our simplest baseline (Algorithm 4) assumes a uniform weight across all neighbouring cells without considering any past history on the number of requests in those cells.

Algorithm 4 URand-NH (Uniform Random with No History)

```
Require: Search radius $\epsilon'$; cell length $\epsilon$; $\boldsymbol{D}_i, \boldsymbol{P}_i \ \forall i \in \{1, \ldots, k\}$.
1: for each time snapshot $t = 1, 2, ..., k$ do
2:     $\boldsymbol{\Gamma}_{t+1} = 0, \in \mathbb{R}^{a \times b}$
3:     for each non-zero entry $(i, j) \in \boldsymbol{D}_t$ do
4:         Pick a set of cells $N = \eta(\square, (i, j), \frac{2\epsilon'}{\epsilon})$.
5:         Choose a cell $\in N$ uniformly at random with coordinates $(i', j')$.
6:         $\boldsymbol{\Gamma}_{t+1}[i', j'] = \boldsymbol{\Gamma}_{t+1}[i', j'] + 1$
7:     end for
8:     Obtain rewards for $R_{t+1}(\boldsymbol{P}_{t+1}, \boldsymbol{\Gamma}_{t+1})$
9: end for
```

Poisson Process Based Algorithm: To capture the temporal pattern of ride requests, we consider a placement approach that models ride requests for every grid as a Poisson Process, as used in (Zhang et al., 2014) (Algorithm 5). Numerous works (Oliveira and Barabási, 2005; Castellano et al., 2009) have recommended against modeling human behavior as a Poisson process, due to the assumption made in the Poisson Process that consecutive events happen at regular time intervals. Contrarily, human activity tends to be bursty for short time intervals accompanied by long intervals of inactivity (Barabasi, 2005; Alfi et al., 2007). For this reason, our maximum likelihood estimator (MLE) estimator is based on limited, and most recent history of inter arrival times of ride requests. Formally, let $N(t), t \geq 0$ be the number of ride requests (events) that occurred by time t.

Lemma 3. *The probability of at least one event occurring in the time snapshot t is given by:*

$$\Pr\{N(t) > 0|\lambda\} = 1 - e^{-\lambda t} \tag{5.8}$$

Proof: See (Harchol-Balter, 2013).

Algorithm 5 PP-LH (Poisson Process with Limited History)

```
Require: Search radius $\epsilon'$; cell length $\epsilon$; $\boldsymbol{D}_i, \boldsymbol{P}_i \forall i \in \{1, ..., k\}$; history length $m$; min. samples $u$.
 1: for each time snapshot $t = (m+1), (m+2), ..., k$ do
 2:     $\boldsymbol{M} = 0 \in \mathbb{R}^{a \times b}$
 3:     $\boldsymbol{\Gamma}_{t+1} = 0, \in \mathbb{R}^{a \times b}$
 4:     $\boldsymbol{M} = \sum_{i=t-m}^{m} \boldsymbol{D}_i + \boldsymbol{P}_i$
 5:     Estimate $\lambda$ for each cell entry having a value $> u$ in $\boldsymbol{M}$ by computing MLE using inter-arrival times.
 6:     for each non-zero entry $(i, j) \in \boldsymbol{D}_t$ do
 7:         Pick a set of cells $N = \eta(\square, (i, j), \frac{2\epsilon'}{\epsilon})$.
 8:         Choose a cell $(i', j') = \text{argmax}_{(i,j) \in N} \Pr\{N(1) > 0|\lambda[i, j]\}$ (see Equation 5.8).
 9:         $\boldsymbol{\Gamma}_{t+1}[i', j'] = \boldsymbol{\Gamma}_{t+1}[i', j'] + 1$
10:         $\boldsymbol{M}[i', j'] = 0$        ▷ Not discarding $(i', j')$, degrades performance since it might be considered again.
11:     end for
12:     Obtain rewards for $R_{t+1}(\boldsymbol{P}_{t+1}, \boldsymbol{\Gamma}_{t+1})$
13: end for
```

For any instance when none of the neighbourhood cells have an estimator $\in \boldsymbol{\lambda}$, the algorithm falls back to uniform random strategy (like for URand-NH) from within the set of neighbors.

Follow The Leader (FTL) Algorithm: Follow The Leader is a sequential prediction strategy, which always puts all the weight on the cell with the highest reward so far. It has been applied to multiple online problems for time series predictions (De Rooij et al., 2014). If there is a tie among the leaders, i.e., there are multiple leaders in the neighbourhood with equal reward values, a leader is chosen uniformly at random. If all cells are zero, FTL-CH also falls back to a uniform random strategy.

Algorithm 6 FTL-CH (Follow The Leader with Complete History)

```
Require: Search radius $\epsilon'$; cell length $\epsilon$; $\boldsymbol{D}, \boldsymbol{P}$; initial history length $m$.
 1: for each time snapshot $t = (m+1), (m+2), ..., k$ do
 2:     $\boldsymbol{M} = 0 \in \mathbb{R}^{a \times b}$
 3:     $\boldsymbol{\Gamma}_{t+1} = 0, \in \mathbb{R}^{a \times b}$
 4:     $\boldsymbol{M} = \sum_{i=1}^{t-1} \boldsymbol{D}_i + \boldsymbol{P}_i$
 5:     for each non-zero entry $(i, j) \in \boldsymbol{D}_t$ do
 6:         Pick a set of cells $N = \eta(\square, (i, j), \frac{2\epsilon'}{\epsilon})$.
 7:         Choose a cell $(i', j') = \text{argmax}_{(i,j) \in N} \boldsymbol{M}[i, j]$.
 8:         $\boldsymbol{\Gamma}_{t+1}[i', j'] = \boldsymbol{\Gamma}_{t+1}[i', j'] + 1$
 9:     end for
10:     Obtain rewards for $R_{t+1}(\boldsymbol{P}_{t+1}, \boldsymbol{\Gamma}_{t+1})$
11: end for
```

5.2.3 Analysis & Results

In this section we analyze the performance of our proposed algorithms. We use the fact that our ride request data exhibits self-similarity to provide an average bound on the algorithms' reward functions.

Reward Analysis: Given the observation of self-similarity, we are primarily interested in approximating queries such as: what is the average number of neighboring cells which are intersected by some neighborhood shape (like square or circle) of a certain length. Let $\overline{nb}(\epsilon')$ denote the average number of points within an enclosed square $\square$ of radius ϵ' where $\epsilon' \in (\epsilon_1, \epsilon_2)$ for which self-similarity is observed. Note $\epsilon' > \epsilon$. An important consequence of this is:

We now evaluate the performance of our algorithms under this self-similarity assumption. We assume throughout that our performance is determined by the placement strategy, i.e., the number of drop-offs is smaller than the number of pick-up requests in a given neighbourhood. Otherwise, if the number of drop-offs is too large, any strategy is likely to perform well. We show in our numerical analysis (Section 5.2) that this assumption holds in our dataset.

Theorem. *Suppose that the total number of drop-offs in a given cell of radius ϵ' is no more than the expected number of pick-up requests in a cell. Then the expected performance of FTL-CH is strictly better than URand-NH for any pickup matrix $\boldsymbol{P}_t$ with $1 < D_2 < 2$: $\mathbb{E}_{FTL\text{-}CH}[R_t] > \mathbb{E}_{URand\text{-}NH}[R_t]$, and the expected number of fulfilled pick-ups per drop-off with FTL-CH exceeds that for URand-NH.*

Proof: In a given grid with radius ϵ', the expected reward for the Follow The Leader with Complete History (FTL-CH) algorithm at time t is given by the minimum of the number of drop-offs in that grid at time t, which we denote by D, and the number of pickups at time $t + 1$ within the cell (i, j) chosen by the FTL-CH algorithm: all drop-offs at time t are placed in cell (i, j) at time $t + 1$. The expected number of pickups in this cell is then proportional to ϵ^{D_2} from Lemma 2, and the expected reward is $\min\left\{D, C\epsilon^{D_2}\right\}$ for some constant $C > 0$.

If $D \leq C\epsilon^{D_2}$, then the expected reward with FTL-CH is D, which is also the maximum possible reward. Thus, the uniform random strategy cannot perform any better.

To show the second part of the theorem, we note that for the uniform random strategy, the expected fraction of total pick-ups in the grid that can be fulfilled by one drop-off is ϵ^2/ϵ'^2, i.e., the reciprocal of the number of cells in a grid. Then the expected number of pickups fulfilled is the expected total number of pickups multiplied by this fraction, i.e., $C\epsilon'^{D_2-2}\epsilon^2 = C\epsilon^{D_2}\left(\epsilon/\epsilon'\right)^{2-D_2} \leq C\epsilon^{D_2}$, the expected number of pickups under FTL-CH, since $1 < D_2 < 2$ and $\epsilon \leq \epsilon'$.

Theorem. *Suppose that the expected number of future pickup requests in each cell at time t is given by the number of requests experienced in the past interval of $[t-1, t)$, and the interarrival times of a sequence of pickups follow an Exponential distribution. Then the expected performance of FTL-CH is equivalent to that of PP-LH: $\mathbb{E}_{FTL\text{-}CH}[R_t] \equiv \mathbb{E}_{PP\text{-}LH}[R_t]$.*

Proof: Using Lemma 3 with $t = 1$, we claim that the placement cell chosen by PP-LH will on average have experienced higher pick-ups than any of its neighbouring cells for a fixed time interval. This is true because with the interval $[t-1, t)$, the maximum likelihood estimator for the Exponential distribution is the inverse of the mean inter-arrival times. The resulting λ estimate will be large for cells which have high rate of events, or equivalently cells which observed more events in the given time interval. From (5.8), we then see that PP-LH will choose the cell with the largest λ estimate, i.e., the cell that experienced the most number of events. Thus, the average reward obtained by PP-LH will be equivalent to that of FTL-CH [2].

[2]For a more detailed proof refer to Lemma 1 in https://classes.soe.ucsc.edu/cmps290c/Spring09/lect/10/wmkalai-rewrite.pdf.

5.2.4 Experimental Results with Real Data

We apply the three algorithms to the ride request data sets and find the achieved rewards in each, indicating the effectiveness of the algorithms. Figure 5.13 plots the reward functions for the three algorithms for each 3-minute (τ_ϵ) time interval over a week. $\epsilon' = 500$ meters which effectively means there are 10^2 possible cells (total cells = $(\frac{2\epsilon'}{\epsilon})^2$) for any vehicle to be placed in. In Algorithms 5 and 6, $m = 20$ and 3 respectively.

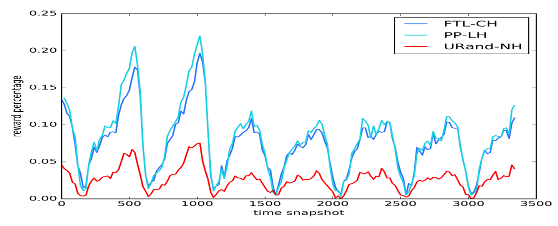
The weekly 7-day pattern of these functions can be clearly seen. All three algorithms tend to be more effective when there are large numbers of ride requests. Supporting the claims in Theorems 5.2.3 and 5.2.3, PP-LH and FTL-CH perform similarly for all cities. On average, the reward percentage for PP-LH is 0.75% better than FTL-CH. In San Francisco, PP-LH guarantees that 11.7% of pickups occur almost instantaneously, roughly within thirty seconds of the ride requests, since our cells are of length $\epsilon = 100$ meters. The average rewards using PP-LH for Chicago, Los Angeles, and New York yield similar results of 11.1%, 7.2%, & 10.2% respectively. Los Angeles has the lowest expected reward as well as the lowest average correlation fractal dimension D_2 (Figure 3.1).

5.2.5 Experimental Results with Synthetic Data

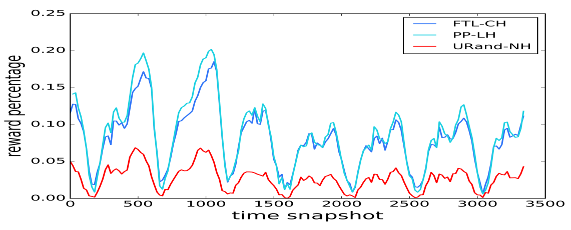
Using synthetic data generated using the GAN approach in Chapter 4, we conducted experiments for the vehicle placement for day of data and compared it with real data. In Figure 5.15, synthetic data tends to mostly predict higher than real data but the trend over a typical data is captured for most of the time snapshots.

5.3 Discussion

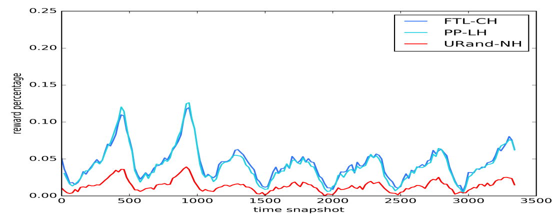
There are several important insights from this work. 1) History of past pickups and drop-offs helps; although when accompanied with some sort of randomization does not necessarily help to boost the performance of the algorithm. For instance, uniform random selection of the leader as in the case of our FTL-CH, does as good as selecting a leader with the lowest index (Blum and Monsour, 2007). 2) We believe randomization is essential but some weighting mechanism needs to be considered which can be characterized with the help of self-similarity. 3) Having limited most recent history is almost as good as having complete history. One obvious extension is to consider vehicle placement actions not just in the next time snapshot but on a longer time scale.



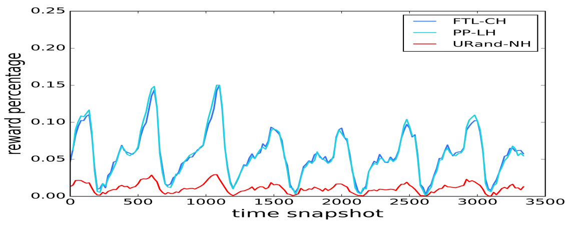
(a) Boston



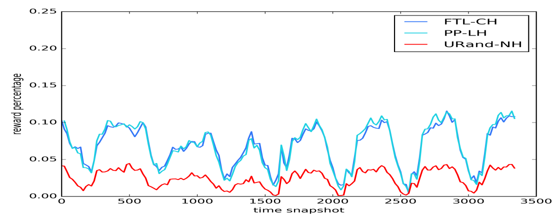
(b) Chicago



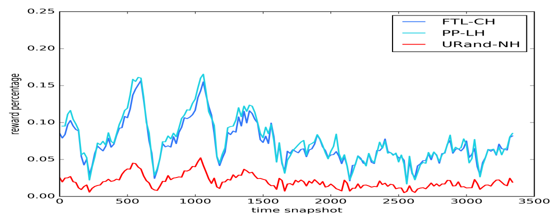
(c) London



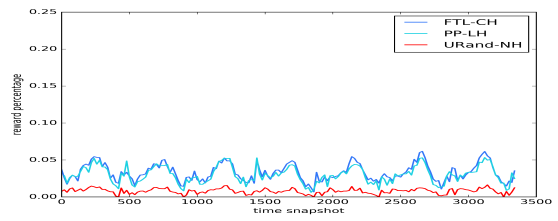
(d) Los Angeles



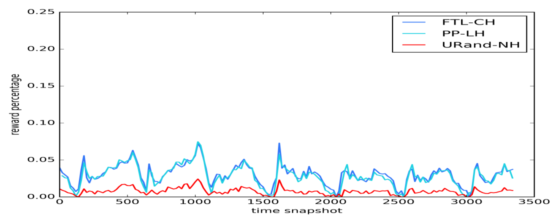
(e) Mexico City



(f) Miami

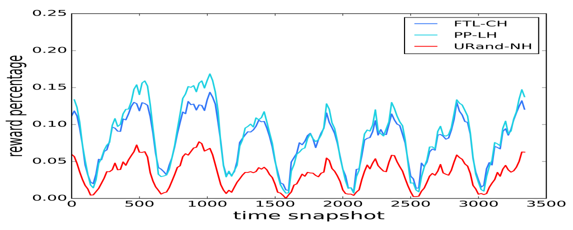


(g) New Delhi

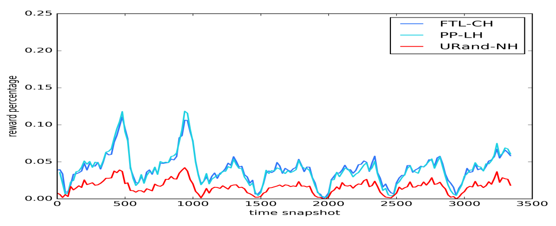


(h) New Jersey

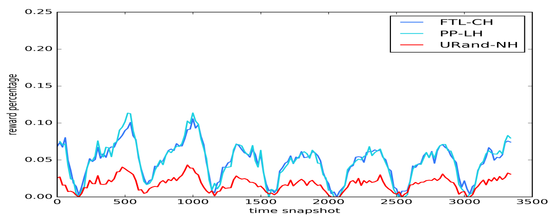
Figure 5.13: The PP-LH algorithm out-performs FTL-CH slightly and URand-NH significantly across all cities in terms of the reward (Eq. 5.7) for a week with three minute time snapshots.



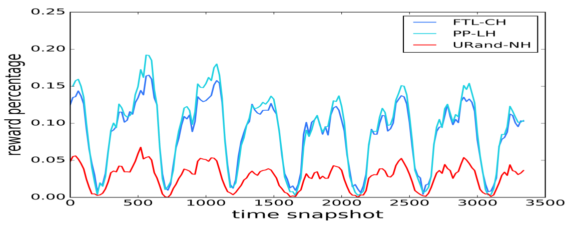
(i) New York



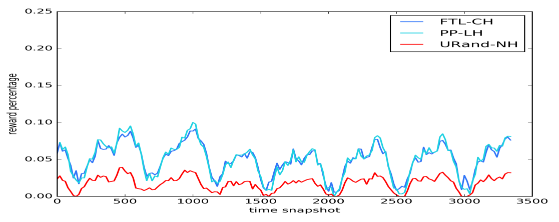
(j) Paris



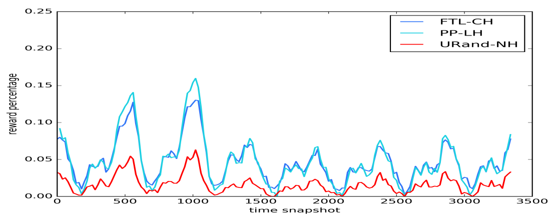
(k) Rio De Janeiro



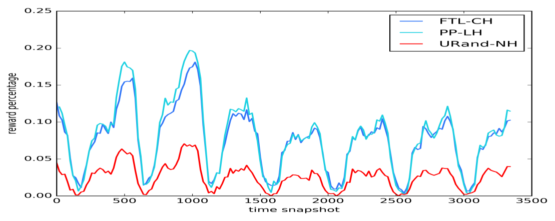
(l) San Francisco



(m) Sao Paulo

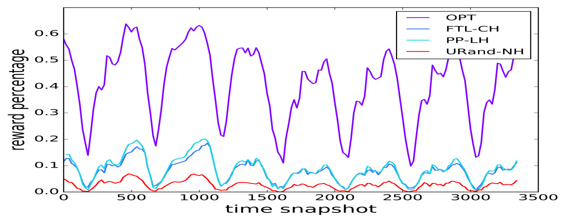


(n) Toronto

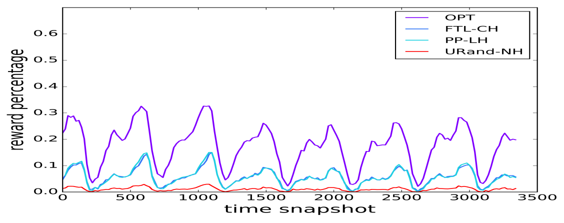


(o) Washington DC

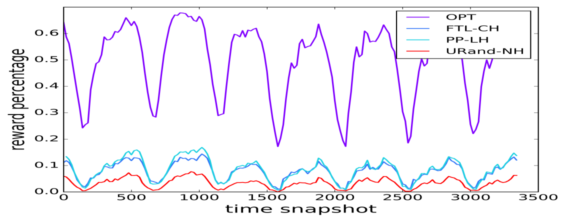
Figure 5.13: The PP-LH algorithm out-performs FTL-CH slightly and URand-NH significantly across all cities in terms of the reward (Eq. 5.7) for a week with three minute time snapshots.



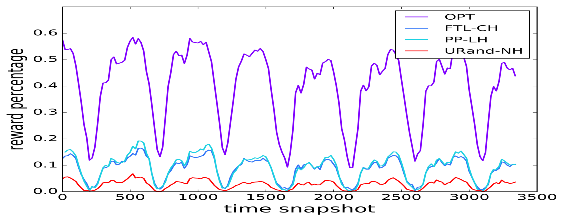
(a) Chicago



(b) Los Angeles

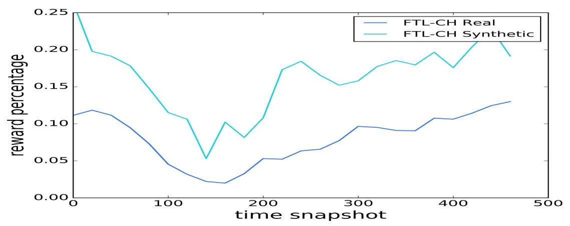


(c) New York

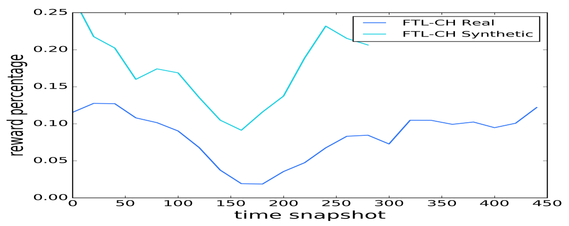


(d) San Francisco

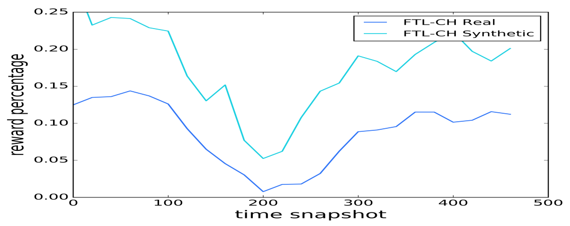
Figure 5.14: Comparison of reward percentage plots for 3 algorithms along with optimal (OPT) reward.



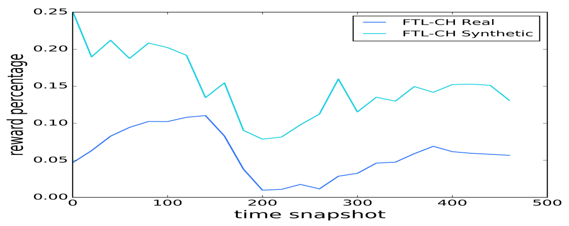
(a) New York



(b) Chicago



(c) San Francisco



(d) Los Angeles

Figure 5.15: FTL-CH algorithm performance on real and synthetic data sets across four cities for a day with three minute time snapshot (aggregate of 480 time snapshots). Certain time snapshots observe no good placements in Chicago using synthetic data; reward percentage is not shown for these time snapshots.

Chapter 6

Toolkit for Studying Human Mobility

Based on our observations, we provide an open source toolkit for generation of synthetic data using pre-trained models for fifteen cities analyzed in this work, along with algorithms used for dynamic ride pooling and real-time vehicle placement problem for experimentation and analyzing different what-if scenarios.

Given societal issues of privacy, the General Data Protection Regulation (GDPR), and the impact of Covid-19 pandemic, the case for availability of synthetic data is even stronger. The data provided within the toolkit is both temporally and spatially vague to reverse engineer and compromise the privacy of a real riders or drivers.

In the sections below we shall provide code pointers and tweakable parameters for temporal and spatial analysis done in Chapters 2 and 3; different components of the GAN training and generation of data; pooling approaches and tweakable knobs associated with it; an entire suite of online placement algorithms apart from the three highlighted in Chapter 5. Apart from generating synthetic data, any researcher can also try to use the different components of the toolkit with their own datasets as well. All datasets and source code are available at `https://github.com/ajauhri/mobility-modeling`.

6.1 Temporal and Spatial Modeling

For temporal modeling there are primarily two parameters which can be changed. The first is the node length or ϵ which determines the cell length which is used to grid a city. Each cell becomes a node in the *Ride Request Graph* so if one goes to very high values of node length, the densification power law property may not hold true. With square grids of very large area one can practically cover huge geographical chunks of a city including downtown and residential areas, and if all such points of interest become part of one node, the densification over time is less likely to be captured in the proposed ride request graph models.

The other modifiable parameter is the width of the time snapshot. We mostly tried five minutes time intervals but these too can be changed. With greater percentage of human mobility captured for a city, we strongly believe both spatial and temporal patterns will be observed for smaller time snapshots and node lengths. Spatial modeling also has the same modifiable parameters as for the temporal modeling. Relevant files here are:

- Temporal modeling – `mobility-modeling/src/modelling/temporal.py`
- Spatial modeling – `mobility-modeling/src/modeling/fractals.py`

For spatial modeling, we did not analyze much about the *Hausdorff Fractal Dimension*. In cases where it is equivalent to correlation fractal dimension, it would imply strict self-similarity. Methods to measure both fractal dimensions are provided in `fractals.py`.

6.2 Synthetic Data Generation

This section focuses on the synthetic data generation model described in chapter 4 along with how data is pre-processed before training, and post-processed after.

6.2.1 Extraction of Blocks

Corresponding file – `mobility-modeling/src/synthetic/extract.py`

We start by dividing the city into n blocks. For each block i we create a separate image (2-dimensional matrix) for each time snapshot t, B_t^i. Each image of the block is further divided into smaller blocks or nodes (as used in RRGs) of length ϵ, $B_t^i(a, b)$ where a and b are the unique horizontal and vertical coordinates of the node within the block. Each $B_t^i(a, b) \in \mathbb{Z}$ represents the number of requests or drop-offs occurred within the node located at coordinates (a, b) within $[t - 1, t)$ time snapshot.

After creating images for each block for each time snapshot, we normalize all images using the block and time snapshot which had the maximum number of rides:

$$\max_{t,i} \sum_{a,b} B_t^i(a, b)$$

6.2.2 Trainer

Corresponding file – `mobility-modeling/src/synthetic/train.py`

Training is separate for each block and can be parallelized on CPUs as mentioned in chapter 4. The *discriminator* takes a batch of training examples in the form of images with each image flattened to a single dimension equivalent to the number of nodes within the block, and returns a single floating point value. *Generator* takes as input the noise vector of configurable dimensionality concatenated with one hot encoding of desired label of time snapshot for which the data is to be generated; the output dimension is equivalent to the flattened image used as input to the discriminator. The *Classifier* also takes flattened image as input and generates a vector equivalent in dimension to the one used by generator for one hot encoding the time snapshot.

We pre-train the classifier for 5k iterations, and subsequently the GAN for 10k iterations.

6.2.3 Generate Distribution

Corresponding file – `mobility-modeling/src/synthetic/gendist.py`

Based on the models trained in the previous step, we now generate data for each time snapshot used as the label for *generator*'s input apart from the noise vector. Due to restricted training dataset, we sampled data multiple times to get the aggregated image/matrix for a single time snapshot i.e. one hour.

6.2.4 Sample

Corresponding file – `mobility-modeling/src/synthetic/sample.py`

After generation of data for each time snapshot we sample ride requests by pixel values for each node within a block into potential number of rides originating or ending at that node. We compute the maximum activation using:

$$\omega_i = \max_{t,a,b} \beta_t^i(a, b)$$

where $\beta_t^i(a, b)$ is the value of node located at coordinate (a, b) of the ith block at time snapshot t using the synthetic data from the trained model for the block. Using the maximum activation, we apply:

$$\beta_t^i(a, b) \times \omega_i \times \phi$$

The multiplication is just to amplify the signal from places where there is very less likelihood of ride request originating or ending. We can modify ϕ as per the requirement to increase or decrease the overall traffic volume of the underlying data for any block within a city as shown in Section 4.7. We have kept ϕ constant for all blocks within a city but it would be interesting to understand if and when should it be a function of region or time as oppose to being a constant value.

6.2.5 Stitch

Corresponding file – `mobility-modeling/src/synthetic/stitch.py`

Final step is a sanity check which is to combine all the blocks generated by the sampling process such that they resemble the city. Stitched results of synthetic data generated for a single time snapshot for San Francisco and New York are given in Figure 6.1.

6.2.6 Graph Generator

Corresponding file – `mobility-modeling/src/synthetic/dpls.py`

After generation of spatially distributed points, we mark them as pickup or drop-off points and connect them using our proposed graph generator. This generator takes as arguments two parameters to determine the number of edges coming out of a node, and probability that a destination node should be chosen from *rich set* or not. Rich set here signifies the small set of nodes which are more likely to be busier in terms of ride requests than majority of other nodes to capture the *small world effect*.

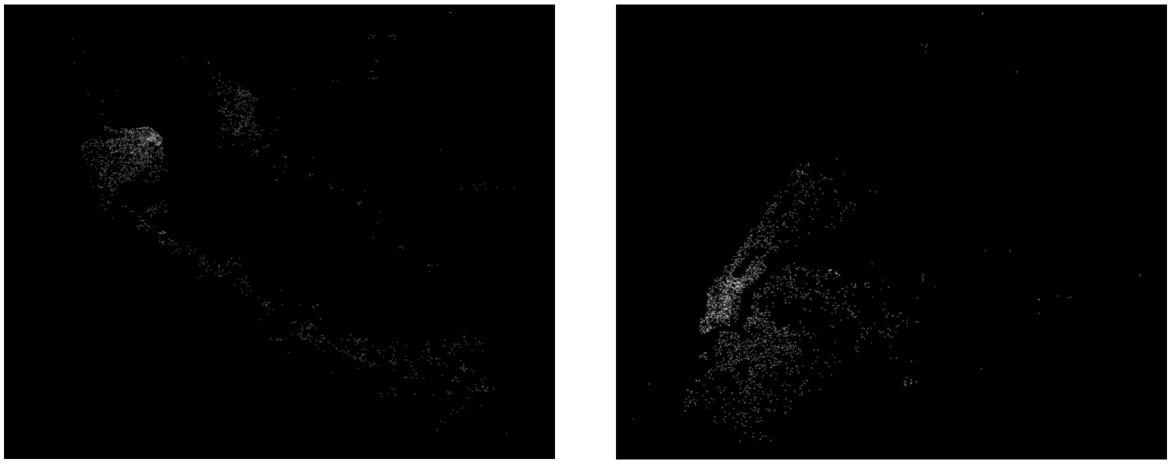

(a) Stitched synthetic requests in SF (b) Stitched synthetic requests in NY

Figure 6.1: Stitched results of synthetic data generated.

6.3 Dynamic Ride Pooling

Ride pooling component of the toolkit includes *Restricted, Directed, and Hybrid* pooling techniques along with implementations to compute relevant metrics like total distance reduction, total vehicle reduction, average poolability, and average penalty.

These metrics have been added to aid different *what-if* scenarios. For instance, using the GANs based approach one can amplify the number of ride requests for any city, and then measure the savings. All savings can be computed for each of the pooling techniques. The following is the list of modifiable parameters for pooling:

1. `--src_radius` specifies the source search region (in meters) in the restricted or hybrid approach.
2. `--dst_radius` specifies the destination search region (in meters) in the restricted or hybrid approach.
3. `--directional_pool` boolean flag to enable directional pooling.
4. `--tunnel_width` specifies ϵ_w (in meters) for the directional or hybrid pooling approach.
5. `--pool_degree` the maximum number of rides which can pooled together.

As noted earlier ϵ_l is analytically computed using Equation 5.6 and ϵ_θ is set to 20 degrees but can be modified. The relevant files for pooling are:

- `mobility-modeling/src/applications/pooling/compute_poolability.py`
- `mobility-modeling/src/applications/pooling/grouping.py`

6.3.1 GraphHopper

To run any of the pooling experiments, one would need to run the *GraphHopper* with the required PBF files. For instance, if running experiments for the city of Los Angeles, GraphHopper's server can be started using:

```
./graphhopper.sh web california-latest.osm.pbf
```

This would allow the pooling component of the toolkit to query approximate travel times and distances for computing metrics like distance reduction, and rider penalty.

`mobility-modeling/src/applications/pooling/graphhopper.py` has the relevant utility functions to process the output from GraphHopper.

6.4 Vehicle Placement

For vehicle placement problem we provide an entire set of algorithms in the toolkit. We believe this can help the community for further analysis and potential improvement of online non-stochastic algorithms.

As stated in the original problem, the vehicle placement tries to anticipate where a ride request may originate in the next few minutes from a drop-off location such that it can start to move towards that direction. For this problem both the time and the distance which can be moved by the vehicle can be changed as desired. Apart from the three best performing algorithms highlighted in the previous chapter, in the toolkit we provide a dozen other algorithms to experiment with.

6.4.1 Sleeping Experts

In the sleeping experts problem setting, there exists a set of experts; at any given time step each expert may be awake (making a prediction) or asleep (not predicting) (Blum, 1997). In the vehicle placement setting, we start with ω weights associated with each expert for each placement node; for each placement decision one of the experts would be chosen based on the weights associated with it to decide the drop-off location. The decision making process continues for all drop-offs within the time snapshot without adjustment to weights. Subsequently after the end of a time snapshot, we adjust the weights. To do so we record all experts' predictions within the time snapshot such that we can penalize accordingly. Only when a correct pickup node is in the predictions from one of the experts, will other experts be penalized if the pickup was not present in their predictions. No penalty is exercised if none of the experts predict a good pickup node.

6.4.2 Follow the Perturbed Leader with Complete History

Follow the perturbed leader (Van Erven et al., 2014) adds a random perturbation to the reward for each of the potential placement nodes and then chooses one with the maximal value. The perturbation is configured by the constant ρ. In the conventional FTL settings, the node with highest reward based on prior history amongst the potential placement nodes is chosen as the prediction node for placement. In experimentation with vehicle placement, we found this approach to vary in performance from city to city when compared with Follow the Leader with Complete History. For instance in Rio de Janeiro the perturbed leader performs much better in comparison to perturbed leader in San Francisco as shown in Figure 6.2. Also, there are instances when perturbed leader does as well as non-perturbed leader with complete history in Rio de Janeiro.

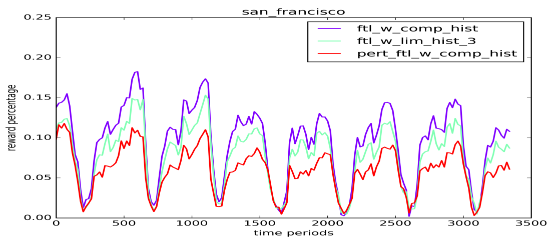
Another interesting insight which can be drawn from Figure 6.2 is the performance of follow the leader with limited history of three previous time snapshots. The intention to highlight this

Algorithm 7 SE-VH (Sleeping Experts with Variable History)

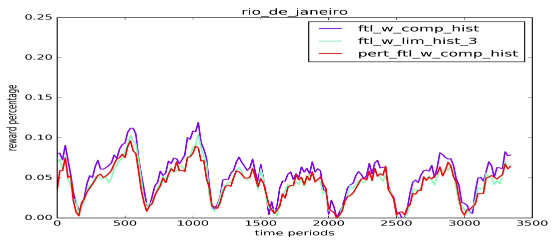
```
Require: Search radius $\epsilon'$; cell length $\epsilon$; $\boldsymbol{D}, \boldsymbol{P}$; initial history length $m$; initial weight $\omega$; penalty $\rho$ s.t. $\rho < \omega$; weight reset period $r$; history size for each of the $n$ experts $H = [h_1, h_2 \ldots h_n]$.
 1: for each time snapshot $t = (m+1), (m+2), ..., k$ do
 2:     $\boldsymbol{\Gamma}_{t+1} = 0, \in \mathbb{R}^{a \times b}$
 3:     if $t\%r == 0$ then
 4:         $\boldsymbol{E}_e = \omega, \in \mathbb{R}^{a \times b}$ for each expert $e \in |E|$.
 5:     end if
 6:     for each expert $e \in |E|$ do
 7:         $\boldsymbol{O}_e = \{a : \{b : \emptyset\}\}$                    ▷ double dictionary in python.
 8:         $\boldsymbol{M}_e = 0 \in \mathbb{R}^{a \times b}$
 9:         $\boldsymbol{M}_e = \sum_{i=t-H[e]}^{t-1} \boldsymbol{D}_i + \boldsymbol{P}_i$
10:     end for
11:     for each non-zero entry $(i, j) \in \boldsymbol{D}_t$ do
12:         Pick random expert $e$ from $|E|$ with each expert being picked with probability $\frac{\boldsymbol{E}_e[i,j]}{\sum_l E_l[i,j]}$.
13:         Pick a set of cells $N = \eta(\square, (i, j), \frac{2\epsilon'}{\epsilon})$.
14:         Choose a cell $(i', j') = \text{argmax}_{(i,j) \in N} \boldsymbol{M}_e[i, j]$.
15:         $\boldsymbol{\Gamma}_{t+1}[i', j'] = \boldsymbol{\Gamma}_{t+1}[i', j'] + 1$
16:         for each expert $e \in \boldsymbol{E}$ do
17:             Find expert's opinion $(i', j') = \text{argmax}_{(m,n) \in N} \boldsymbol{M}_e[m, n]$.
18:             $\boldsymbol{O}_e[i'][j'].add((i, j))$
19:         end for
20:     end for
21:     for each non-zero entry $(i, j) \in \boldsymbol{P}_{t+1}$ do
22:         if $\boldsymbol{\Gamma}_{t+1}[i, j] > 0$ then
23:             for each expert $e \in |E|$ do
24:                 if $\boldsymbol{O}_e[i][j] = \emptyset$ then
25:                     $\boldsymbol{E}_e[O_e[i][j]] = E_e[O_e[i][j]] - \rho$
26:                 end if
27:             end for
28:         end if
29:     end for
30:     Obtain rewards for $R_{t+1}(\boldsymbol{P}_{t+1}, \boldsymbol{\Gamma}_{t+1})$
31: end for
```

Algorithm 8 PFTL-CH (Perturbed Follow The Leader with Complete History)

```
Require: Search radius $\epsilon'$; cell length $\epsilon$; $\boldsymbol{D}, \boldsymbol{P}$; initial history length $m$; perturbation constant $\rho < 1$.
 1: for each time snapshot $t = (m+1), (m+2), ..., k$ do
 2:     $\boldsymbol{M} = 0 \in \mathbb{R}^{a \times b}$
 3:     $\boldsymbol{\Gamma}_{t+1} = 0, \in \mathbb{R}^{a \times b}$
 4:     $\boldsymbol{M} = \sum_{i=1}^{t-1} \boldsymbol{D}_i + \boldsymbol{P}_i$
 5:     for each non-zero entry $(i, j) \in \boldsymbol{D}_t$ do
 6:         Pick a set of cells $N = \eta(\square, (i, j), \frac{2\epsilon'}{\epsilon})$.
 7:         Choose a cell $(i', j') = \text{argmax}_{(i,j) \in N} \boldsymbol{M}[i, j] + uniform(0, 1 + \frac{1}{\rho})$.
 8:         $\boldsymbol{\Gamma}_{t+1}[i', j'] = \boldsymbol{\Gamma}_{t+1}[i', j'] + 1$
 9:     end for
10:     Obtain rewards for $R_{t+1}(\boldsymbol{P}_{t+1}, \boldsymbol{\Gamma}_{t+1})$
11: end for
```



(a) San Francisco



(b) Rio de Janerio

Figure 6.2: Comparison of three variations of Follow the Leader online algorithms; with complete history, with limited history from previous three time snapshots, with complete history but with perturbed leader.

algorithm was to depict the importance of recent history and how important can it be to make decisions in the near future. Also, in scenarios where one experiences sudden anomalies like rain, or snow, follow the leader with limited history can potentially be more relevant in comparison to follow the leader with complete history since people may tend to request a ride and wait at more convenient spots which may not be captured in complete history of the past.

6.4.3 Exp3

With Multi-Arm Bandit setting we aim to balance exploration and exploitation for the placement problem. Exp3 (Seldin et al., 2013) is one way to achieve this. The distribution in $\boldsymbol{P}$ is a mixture of the uniform distribution and a distribution which assigns to each placement action a probability mass exponential in the estimated cumulative reward for that action. The uniform distribution allows for exploration using γ as a knob to control the amount of exploration, and the other probability distribution allows for exploitation. $\frac{X}{P}$ is to compensate the reward of actions.

6.4.4 Most Recently Used

As a variation from choosing the most frequent leader, we also looked at most recent variant for the placement problem. For every entry in the two-dimensional array $\boldsymbol{M}$ we store the highest or most recent time snapshot for where there was a pick-up request or zero if no request happened. For placement we choose the neighboring cell with a most recent pickup.

6.5 Discussion and Limitations

In this chapter we have tried to provide technical details around the toolkit which we hope can be of great value for the research community. The toolkit serves as another medium to convey information about some of the intricate details of the characterizations and models used in this work.

Algorithm 9 MAB-Exp3 (Multi-Arm Bandit Exp3)

```
Require: Search radius $\epsilon'$; cell length $\epsilon$; $\boldsymbol{D}, \boldsymbol{P}$; area of neighboring region $K$; probability of choosing an arm $\gamma$.
1: for each time snapshot $t = (1), (2), \ldots, k$ do
2:     $\boldsymbol{W} = 1 \in \mathbb{R}^{a \times b}$
3:     $\boldsymbol{X} = 0 \in \mathbb{R}^{a \times b}$
4:     $\boldsymbol{P} = (1-\gamma) \times \frac{\boldsymbol{W}}{K} + \frac{\gamma}{K}$.
5:     $\boldsymbol{\Gamma}_{t+1} = 0, \in \mathbb{R}^{a \times b}$
6:     for each non-zero entry $(i, j) \in \boldsymbol{D}_t$ do
7:         Pick a set of cells $N = \eta(\square, (i, j), \frac{2\epsilon'}{\epsilon})$.
8:         Get boundary coordinates $x_s, x_e, y_s, y_e = Boundary(\square, (i, j), \frac{2\epsilon'}{\epsilon})$.
9:         $\boldsymbol{W}_t = \boldsymbol{W}[x_s : x_e, y_s, y_e]$.
10:        $n_{bandits} = (x_e - x_s) \times (y_e - y_s)$
11:        for each $(m, n) \in \boldsymbol{N}$ do
12:            $\boldsymbol{P}[m, n] = (1-\gamma) \times \frac{W[m,n]}{sum(W_t)} + \frac{\gamma}{n_{bandits}}$.
13:        end for
14:        Choose a cell $(i', j')$ randomly according to the probabilities in $\boldsymbol{P}$ constrained to $N$ cells only.
15:        $\boldsymbol{\Gamma}_{t+1}[i', j'] = \boldsymbol{\Gamma}_{t+1}[i', j'] + 1$
16:    end for
17:    Obtain rewards for $R_{t+1}(\boldsymbol{P}_{t+1}, \boldsymbol{\Gamma}_{t+1})$
18:    For each correct placement $(i', j')$, $\boldsymbol{X}[i', j'] = \boldsymbol{X}[i', j'] + 1$
19:    $\boldsymbol{W} = \boldsymbol{W} \times \exp(\gamma \times \frac{\frac{X}{P}}{K})$
20: end for
```

Algorithm 10 MRU-LH (Most Recently Used with Limited History)

```
Require: Search radius $\epsilon'$; cell length $\epsilon$; $\boldsymbol{D}, \boldsymbol{P}$; initial history length $m$.
1: for each time snapshot $t = (m+1), (m+2), \ldots, k$ do
2:     $\boldsymbol{M} = 0 \in \mathbb{R}^{a \times b}$
3:     $\boldsymbol{\Gamma}_{t+1} = 0, \in \mathbb{R}^{a \times b}$
4:     for $x \in \{0, 1, \ldots a\}$ do
5:         for $y \in \{0, 1, \ldots b\}$ do
6:             $\boldsymbol{M}[x, y] = \text{argmax}_{i \in [t-1-m, t-1]} \boldsymbol{P}_i[x, y] \neq 0$
7:         end for
8:     end for
9:     for each non-zero entry $(i, j) \in \boldsymbol{D}_t$ do
10:        Pick a set of cells $N = \eta(\square, (i, j), \frac{2\epsilon'}{\epsilon})$.
11:        Choose a cell $(i', j') = \text{argmax}_{(i,j) \in N} \boldsymbol{M}[i, j]$.
12:        $\boldsymbol{\Gamma}_{t+1}[i', j'] = \boldsymbol{\Gamma}_{t+1}[i', j'] + 1$
13:    end for
14:    Obtain rewards for $R_{t+1}(\boldsymbol{P}_{t+1}, \boldsymbol{\Gamma}_{t+1})$
15: end for
```

Although the toolkit can be applied to any dataset, one needs to be cognizant about the fact that if used with sparse datasets which does not capture the temporal and spatial variability of ride requests as likely to be observed in any urban area using a representative dataset, the toolkit may not provide appropriate qualitative measures. Reasonable penetration in terms of percentage of all journeys within a city is necessary for the models proposed here to clearly characterize the variations.

When trying to analyze how accurate is the synthetic data, it is important to look at volume of journeys over time, DPL plots, and fractal dimension plots. Looking at only one or a subset will not highlight issues in the synthetically generated dataset which could be due to not fine-tuned hyper-parameters in either of the generators or just how well the proposed GANs approach is able to capture the distribution of ride requests.

Chapter 7

Conclusions and Future Directions

Our overall thesis statement was based on analysis of real-world datasets from ride-sharing services, to understand and characterize human mobility patterns at the city scale, and gain insights on how to improve ridership experience, and overall system efficiency. We achieved those characterizations in chapters 2 and 3 along with highlighting the intuition on how they reflect city level characteristic of mobility patterns. In chapter 4 we propose a scalable efficient method for synthetic data generation and validate synthetic data using temporal and spatial characterizations. More importantly, these characterizations are able to identify certain deficiencies in our modeling approach. In chapter 5 we extended our characterizations and applied those to create techniques to improve rider and driver experiences using real-world applications. Finally, in chapter 6 we highlight how the toolkit can be used with any dataset to understand its properties, and application to different what-if scenarios related to human mobility.

The remainder of this chapter discusses future areas to explore.

7.1 Implications of Fractal Theory on Human Mobility

In chapter 3 we only looked at correlation fractal dimension. It would be interesting to understand the implications of Hausdorff fractal dimension D_0, or more aspects of fractal theory on human mobility pattern. From our initial analysis, we observed Hausdorff fractal dimension to be not equivalent to correlation fractal dimension leading to not strictly self-similar points in the 2D geographical space. Fractal dimension theory has been applied to other domains such as dynamical systems, diagnosis of diseases, earthquakes, and analysis of the human retina, just to name a few. Hence, a more comprehensive application of fractal theory to human mobility may uncover more interesting properties similar to the one highlighted for vehicle placement in chapter 5.

7.2 Sampling from GANs

Generative Adversarial Networks are used in different what-if scenarios like vehicle placement and understanding temporal patterns of ride requests. Capturing fine details of an image is more critical for the applications proposed here and also for capturing both temporal and spatial characterizations at the city-level. Some interesting work like in (Bhattarai et al., 2020) does provide an additional learning objective to better capture whether or not certain images correctly represent the underlying data or not. As seen from results in chapter 4 where synthetic images tend to be biased towards certain geographical areas, we believe sampling can be an interesting research to pursue given that these synthetically generated images in our scenario should not only look aesthetically pleasing but also valid as per the spatial and temporal characterizations which capture very minute details of these images.

7.3 Application of Differential Privacy to Human Mobility

As important and useful it is to analyze the ride sharing data to derive insights about human mobility patterns, it is all the more important to ensure correct handling of data and to ensure that it does not reveal any user habits or location-identifiable information. Even though the data used here for generating synthetic dataset, does not directly include user information but it still exposes critical location specific information within small cells, and local characteristics of a place. Application of differential privacy to ride request data to make it resistant to privacy attacks (Mir et al., 2013) but still ensuring it depicts the city level characterizations for both temporal and spatial patterns can be a non-trivial problem to tackle.

Differential privacy is a method to ensure that the outcome of data analysis for a dataset (R1) is almost similar to analysis on a dataset (R2) even though these two data datasets differ by only one user's features. This means looking at the outcome of the dataset should not lead to determining absence or presence of individual user's features. This should hold even if the user has unique behaviour.

Differential privacy can be achieved by addition of noise to a dataset through Laplace mechanism; it has been well studied that the addition of noise is a reliable way to conserve privacy of data. A dataset is said to be ϵ-differentially private when the noise added to it depends on ϵ and $S(f)$, where $S(f)$ is defined as the sensitivity of a function or the change needed to make a user's data indifferentiable, and an appropriate value of ϵ which determines the level of privacy and therefore the accuracy of the data.

7.4 Roaming Fleets of Vehicular Computing Systems

The potential convergence of Cloud-Edge Computing, Internet-of-Things, and Connected Vehicles, is creating a fascinating research opportunity. The explosive proliferation of mobile devices that rely heavily on cloud services will necessitate highly distributed cloud edge servers to avoid saturating the core network bandwidth and to reduce service delivery latency. The emergence of diverse sensing platforms for supporting the Internet-of-Things will necessitate highly distributed sensor data processing and analytics in real time and at the cloud edge. We anticipate the emergence of powerful general-purpose Vehicular Computing Systems (VCS) that can be deployed in 5G connected vehicles for supporting both mobile cloud-edge computing and distributed in-situ data analytics.

This research focuses on the system architecture and potential applications of VCS and roaming fleets of VCS-equipped connected vehicles (VCS Fleets). We plan to explore VCS as mobile cloud-edge servers and edge sensing/analytics platforms. VCS in connected vehicles can become the new mass market for mobile computers. We plan to explore the integration of VCS with 5G picocell base stations in connected vehicles. This will enable connected vehicles to become part of the 5G connectivity infrastructure with adaptive roaming capability. We also plan to research the potential applications of VCS Fleets in urban environments. One potential application of VCS Fleets is facilitating large scale federated and continuous learning for diverse intelligent services targeting the vehicular environment. We believe VCS Fleets can potentially become a highly distributed, mobile, and adaptive connectivity and computing infrastructure for supporting human mobility services and societal benefits.

7.4.1 Applications

- Learning applications which require constant communication of real-time information spread over geographical space shall also be explored (Hull et al., 2006). For instance, the emergence of ride-sharing services and the availability of extensive data from such services has created unprecedented opportunities for conducting large scale data analytics on urban transportation, gaining new insights on human mobility, and facilitating new public services for societal benefit. Moreover, applications for which online learning algorithms are considered require multiple streams of data. Network latency for many such algorithms is a bottleneck. To this end, we plan to deploy communication intensive applications with quantifiable benefits without need to access the cloud.
- Computational and network overhead can lead to issues with analyzing real time video content. We plan to add textual information to live geo-tagged videos collected and processed with deep neural networks deployed in a VCS Fleet. Tagged with textual descriptions, these real-time videos of urban streets collected by vehicles can be used by civic authorities for analysis; these descriptions would help authorities quickly target urban areas which are of concern and/or require maintenance. The video stream would be updated continuously, and so would the descriptions associated with it. Apart from civic authorities, security personnel, people with disability, and the general public can gain insights about live events in any urban area without having to manually analyze single frames of a constrained area. This is a novel application which combines two areas of learning: video and linguistics, and can nicely leverage VCS Fleets.

Bibliography

Alessandretti, Laura, Ulf Aslak, and Sune Lehmann (2020). “The scales of human mobility”. In: *Nature* 587.7834, pages 402–407 (page 26).

Alessandretti, Laura, Piotr Sapiezynski, Sune Lehmann, and Andrea Baronchelli (2017). “Multi-scale spatio-temporal analysis of human mobility”. In: *PloS one* 12.2, e0171686 (page 3).

Alfi, Valentina, Giorgio Parisi, and Luciano Pietronero (2007). “Conference registration: how people react to a deadline”. In: *Nature Physics* 3.11, pages 746–746 (page 71).

Barabasi, Albert-Laszlo (2005). “The origin of bursts and heavy tails in human dynamics”. In: *Nature* 435.7039, pages 207–211 (page 71).

Bast, Hannah, Daniel Delling, Andrew Goldberg, et al. (2016). “Route planning in transportation networks”. In: *Algorithm Engineering*. Springer, pages 19–80 (page 56).

Becker, Richard, Ramón Cáceres, Karrie Hanson, et al. (2013). “Human mobility characterization from cellular network data”. In: *Communications of the ACM* 56.1, pages 74–82 (page 8).

Belussi, Alberto and Christos Faloutsos (1995). “Estimating the Selectivity of Spatial Queries Using the ’Correlation’ Fractal Dimension”. In: *VLDB* (pages 28–30).

— (1998). *Estimating the selectivity of spatial queries using thecorrelation’fractal dimension*. Technical report (pages 29, 35, 47).

Bhattarai, Binod, Seungryul Baek, Rumeysa Bodur, and Tae-Kyun Kim (2020). “Sampling strategies for GAN synthetic data”. In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pages 2303–2307 (page 90).

Blum, Avrim (1997). “Empirical support for winnow and weighted-majority algorithms: Results on a calendar scheduling domain”. In: *Machine Learning* 26.1, pages 5–23 (page 83).

Blum, Avrim and Yishay Monsour (2007). “Learning, regret minimization, and equilibria”. In: (page 74).

Buzzfeed (2016). *People in Los Angeles Are Getting Rid of Their Cars*. `https://www.buzzfeed.com/priya/people-in-los-angeles-are-getting-rid-of-their-cars`. [Online; accessed 3-February-2020] (page 8).

Castellano, Claudio, Santo Fortunato, and Vittorio Loreto (2009). “Statistical physics of social dynamics”. In: *Reviews of modern physics* 81.2, page 591 (page 71).

Chakrabarti, Deepayan and Christos Faloutsos (2012). “Graph mining: laws, tools, and case studies”. In: *Synthesis Lectures on Data Mining and Knowledge Discovery* 7.1, pages 1–207 (pages 16, 22).

Chan, Grace, Andrew TA Wood, et al. (2004). “Estimation of fractal dimension for a class of non-Gaussian stationary processes and fields”. In: *The Annals of Statistics* 32.3, pages 1222–1260 (page 35).

Chen, Min Hao, Abhinav Jauhri, and John Paul Shen (2017). “Data driven analysis of the potentials of dynamic ride pooling”. In: *Proceedings of the 10th ACM SIGSPATIAL Workshop on Computational Transportation Science*, pages 7–12 (page 37).

Chen, Xi, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel (2016). “Infogan: Interpretable representation learning by information maximizing generative adversarial nets”. In: *arXiv preprint arXiv:1606.03657* (page 38).

De Rooij, Steven, Tim Van Erven, Peter D Grünwald, and Wouter M Koolen (2014). “Follow the leader if you can, hedge if you must.” In: *Journal of Machine Learning Research* 15.1, pages 1281–1316 (page 72).

Denton, Emily L, Soumith Chintala, Rob Fergus, et al. (2015). “Deep generative image models using a laplacian pyramid of adversarial networks”. In: *Advances in neural information processing systems*, pages 1486–1494 (page 39).

Fortunato, Santo (2010). “Community detection in graphs”. In: *Physics reports* 486.3, pages 75–174 (page 8).

Gauthier, Jon (no date). “Conditional generative adversarial nets for convolutional face generation”. In: () (page 39).

Gonzalez, Marta C, Cesar A Hidalgo, and Albert-Laszlo Barabasi (2008). “Understanding individual human mobility patterns”. In: *nature* 453.7196, pages 779–782 (pages 1, 13, 26).

Goodfellow, Ian (2018). *Introduction to GANs*. `http://efrosgans.eecs.berkeley.edu/CVPR18_slides/Introduction_by_Goodfellow.pdf`. [Online; accessed 31-August-2018] (page 41).

Goodfellow, Ian, Yoshua Bengio, Aaron Courville, and Yoshua Bengio (2016). *Deep learning*. Volume 1. MIT Press (page 39).

Harchol-Balter, Mor (2013). *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press (page 72).

Hasan, Samiul, Christian M Schneider, Satish V Ukkusuri, and Marta C González (2013). “Spatiotemporal patterns of urban human mobility”. In: *Journal of Statistical Physics* 151.1, pages 304–318 (page 26).

Hoag, Joseph E (2008). *Synthetic data generation: Theory, techniques and applications*. University of Arkansas (page 37).

Hull, Bret, Vladimir Bychkovsky, Yang Zhang, et al. (2006). “Cartel: a distributed mobile sensor computing system”. In: *Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 125–138 (page 91).

Intel (2018a). *Intel MKL-DNN: Primitive Operations*. `https://intel.github.io/mkl-dnn/group__c__api__primitive.html`. [Online; accessed 31-August-2018] (page 43).

— (2018b). *Introducing DNN primitives in Intel Math Kernel Library*. `https://software.intel.com/en-us/articles/introducing-dnn-primitives-in-intelr-mkl`. [Online; accessed 31-August-2018] (page 43).

Isaacman, Sibren, Richard Becker, Ramón Cáceres, Margaret Martonosi, James Rowland, Alexander Varshavsky, and Walter Willinger (2012). “Human mobility modeling at metropolitan scales”. In: *Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM, pages 239–252 (page 37).

Jauhri, Abhinav, Brian Foo, Jérôme Berclaz, Chih Chi Hu, Radek Grzeszczuk, Vasu Parameswaran, and John Paul Shen (2017). “Space-Time Graph Modeling of Ride Requests Based on Real-World Data”. In: *ArXiv* abs/1701.06635 (pages 4, 54, 60).

Jauhri, Abhinav, Carlee Joe-Wong, and John Paul Shen (2017). “On the Real-time Vehicle Placement Problem”. In: *ArXiv* abs/1712.01235 (page 4).

Jurdak, Raja, Kun Zhao, Jiajun Liu, Maurice AbouJaoude, Mark Cameron, and David Newth (2015). “Understanding human mobility from Twitter”. In: *PloS one* 10.7, e0131469 (page 26).

Karich, Peter and S Schröder (2014). “Graphhopper”. In: *http://www. graphhopper. com, last accessed* 4.2, page 15 (page 61).

Karras, Tero, Timo Aila, Samuli Laine, and Jaakko Lehtinen (2017). “Progressive growing of gans for improved quality, stability, and variation”. In: *arXiv preprint arXiv:1710.10196* (page 47).

Ledig, Christian, Lucas Theis, Ferenc Huszár, et al. (2017). “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network.” In: *CVPR*. Volume 2. 3, page 4 (page 39).

Lee, Minhyeok and Junhee Seok (2017). “Controllable Generative Adversarial Network”. In: *arXiv preprint arXiv:1708.00598* (page 42).

Leskovec, Jure (2008). “Dynamics of large networks”. PhD thesis. Carnegie Mellon University, School of Computer Science, Machine Learning (pages 16, 18, 19, 22).

Leskovec, Jure and Christos Faloutsos (2006). “Sampling from large graphs”. In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 631–636 (page 23).

Leskovec, Jure, Jon Kleinberg, and Christos Faloutsos (2005). “Graphs over time: densification laws, shrinking diameters and possible explanations”. In: *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM, pages 177–187 (pages 8, 12, 16).

— (2007). “Graph evolution: Densification and shrinking diameters”. In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 1.1, page 2 (pages 7, 12).

Lu, Xin, Erik Wetter, Nita Bharti, Andrew J Tatem, and Linus Bengtsson (2013). “Approaching the limit of predictability in human mobility”. In: *Scientific reports* 3.1, pages 1–9 (page 26).

Manasse, Mark S, Lyle A McGeoch, and Daniel D Sleator (1990). “Competitive algorithms for server problems”. In: *Journal of Algorithms* 11.2, pages 208–230 (page 70).

Mandelbrot, Benoit B and Benoit B Mandelbrot (1982). *The fractal geometry of nature*. Volume 1. WH freeman New York (page 35).

Matsubara, Yasuko, Lei Li, Evangelos Papalexakis, David Lo, Yasushi Sakurai, and Christos Faloutsos (2013). “F-trail: Finding patterns in taxi trajectories”. In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, pages 86–98 (page 54).

Maystre, Lucas and Matthias Grossglauser (2016). “ChoiceRank: Identifying Preferences from Node Traffic in Networks”. In: *arXiv preprint arXiv:1610.06525* (page 37).

Mir, Darakhshan J, Sibren Isaacman, Ramón Cáceres, Margaret Martonosi, and Rebecca N Wright (2013). “Dp-where: Differentially private modeling of human mobility”. In: *2013 IEEE international conference on big data*. IEEE, pages 580–588 (page 90).

Mirza, Mehdi and Simon Osindero (2014). “Conditional generative adversarial nets”. In: *arXiv preprint arXiv:1411.1784* (page 39).

Moritz, Philipp, Robert Nishihara, Stephanie Wang, et al. (2017). “Ray: A Distributed Framework for Emerging AI Applications”. In: *arXiv preprint arXiv:1712.05889* (page 42).

Newman, Mark (2018). *Networks*. Oxford university press (page 22).

Newman, Mark EJ (2005). “Power laws, Pareto distributions and Zipf’s law”. In: *Contemporary physics* 46.5, pages 323–351 (pages 8, 16).

Nvidia (2018). *Multi-Process Service*. https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf. [Online; accessed 31-August-2018] (page 43).

Oliveira, Joao Gama and Albert-László Barabási (2005). “Human dynamics: Darwin and Einstein correspondence patterns”. In: *Nature* 437.7063, pages 1251–1251 (page 71).

OpenStreetMap (2016). *OpenStreetMap contributors. (2016) Planet dump*. http://http://wiki.openstreetmap.org/wiki/Main_Page. [Online; Data file accessed till 12-May-2021] (page 49).

OpenStreetMap contributors (2017). *Planet dump retrieved from https://planet.osm.org*. https://www.openstreetmap.org (page 61).

Parrott, James and Michael Reich (2018). “An earnings standard for New York City’s app-based drivers: Economic analysis and policy assessment”. In: *Report for the New York City Taxi and Limousine Commission* 5 (page 8).

Phithakkitnukoon, Santi, Marco Veloso, Carlos Bento, Assaf Biderman, and Carlo Ratti (2010). “Taxi-aware map: Identifying and predicting vacant taxis in the city”. In: *International Joint Conference on Ambient Intelligence*. Springer, pages 86–95 (page 37).

Proietti, Guido and Christos Faloutsos (1999). “I/O complexity for range queries on region data stored using an R-tree”. In: *Data Engineering, 1999. Proceedings., 15th International Conference on*. IEEE, pages 628–635 (page 28).

Rashidi, Taha H, Alireza Abbasi, Mojtaba Maghrebi, Samiul Hasan, and Travis S Waller (2017). “Exploring the capacity of social media data for modelling travel behaviour: Opportunities and challenges”. In: *Transportation Research Part C: Emerging Technologies* 75, pages 197–211 (page 8).

Reed, William J and Murray Jorgensen (2004). “The double Pareto-lognormal distribution—a new parametric model for size distributions”. In: *Communications in Statistics-Theory and Methods* 33.8, pages 1733–1753 (page 54).

Salimans, Tim, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen (2016). “Improved techniques for training gans”. In: *Advances in Neural Information Processing Systems*, pages 2234–2242 (page 41).

Schroeder, Manfred (2009). *Fractals, chaos, power laws: Minutes from an infinite paradise*. Courier Corporation (pages 28, 35).

Schuster, Heinz Georg and Wolfram Just (2006). *Deterministic chaos: an introduction*. John Wiley & Sons (page 29).

Seldin, Yevgeny, Csaba Szepesvári, Peter Auer, and Yasin Abbasi-Yadkori (2013). “Evaluation and analysis of the performance of the EXP3 algorithm in stochastic environments”. In: *European Workshop on Reinforcement Learning*. PMLR, pages 103–116 (page 85).

Seshadri, Mukund, Sridhar Machiraju, Ashwin Sridharan, Jean Bolot, Christos Faloutsos, and Jure Leskove (2008). “Mobile call graphs: beyond power-law and lognormal distributions”. In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pages 596–604 (page 54).

Song, Chaoming, Zehui Qu, Nicholas Blumm, and Albert-László Barabási (2010). “Limits of predictability in human mobility”. In: *Science* 327.5968, pages 1018–1021 (page 1).

Stiglic, Mitja, Niels Agatz, Martin Savelsbergh, and Mirko Gradisar (2018). “Enhancing urban mobility: Integrating ride-sharing and public transit”. In: *Computers & Operations Research* 90, pages 12–21 (page 37).

Tang, Jinjun, Jian Liang, Shen Zhang, Helai Huang, and Fang Liu (2018). “Inferring driving trajectories based on probabilistic model from large scale taxi GPS data”. In: *Physica A: Statistical Mechanics and its Applications* 506, pages 566–577 (page 37).

Tasse, Dan and Jason I Hong (no date). “Using social media data to understand cities”. In: (page 8).

Van Erven, Tim, Wojciech Kotłowski, and Manfred K Warmuth (2014). “Follow the leader with dropout perturbations”. In: *Conference on Learning Theory*. PMLR, pages 949–974 (page 83).

Watts, Duncan J and Steven H Strogatz (1998). “Collective dynamics of ‘small-world’networks”. In: *nature* 393.6684, pages 440–442 (page 13).

Yao, Huaxiu, Xianfeng Tang, Hua Wei, Guanjie Zheng, Yanwei Yu, and Zhenhui Li (2018). “Modeling Spatial-Temporal Dynamics for Traffic Prediction”. In: *arXiv preprint arXiv:1803.01254* (page 38).

Yao, Huaxiu, Fei Wu, Jintao Ke, et al. (2018). “Deep multi-view spatial-temporal network for taxi demand prediction”. In: *arXiv preprint arXiv:1802.08714* (page 38).

Zhang, Desheng, Tian He, Shan Lin, Sirajum Munir, and John A Stankovic (2014). “Dmodel: Online taxicab demand model from big sensor data in a roving sensor network”. In: *Big Data (BigData Congress), 2014 IEEE International Congress on*. IEEE, pages 152–159 (page 71).

Zhou, Xian, Yanyan Shen, Yanmin Zhu, and Linpeng Huang (2018). “Predicting multi-step citywide passenger demands using attention-based neural networks”. In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, pages 736–744 (page 38).