Spatial Big Data Analytics for Urban Informatics

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

Michael Robert Evans

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

Advisor: Professor Shashi Shekhar

August, 2013

Acknowledgements

I want to thank my advisor, Professor Shashi Shekhar, for his incredible support throughout my Ph.D. One phone call from him out of the blue back in 2008 truly changed my life. His dedication and patience has been invaluable over the past five years. I also want to thank all of the professors who helped me over the years in classes and especially those who accepted to serve on my committee: Dr. Vipin Kumar, Dr. Mohamed Mokbel, and Dr. Francis Harvey. Each of your unique insights helped shape and craft this work and my overall research interests. Thank you.

I extend thanks to Prof. Shekhar's spatial research group, members both past and present, for all the meetings, proposals and late nights we spent together. I will miss all of you and thank you for all the help you have given me over the years. Lastly I want to thank my lovely wife and parents for all their support.

Abstract

Urban Informatics is the practice of using computer technology to support core city functions: planning, governance and operations. This technology consists of hardware, software, databases, sensors, and communication devices used to develop and sustain more livable and healthy cities. Urban Informatics provides governments with the tools to make data-driven decisions regarding long-term plans, predict and respond to current and upcoming situations, and even help with day-to-day tasks such as monitoring water use and waste processing. New and immense location-aware datasets formally defined in this thesis as Spatial Big Data are emerging from a variety of sources and can be used to find novel and interesting patterns for use in urban informatics. Spatial big data is the key component driving the emerging field of Urban Informatics at the intersection of people, places, and technology. However, spatial big data presents challenges for existing spatial computing systems to *store*, *process*, and *analyze* such large datasets. With these challenges come new opportunities in many fields of computer science research, such as spatial data mining and spatial database systems. This thesis contains original research on two types of spatial big data, each study focusing on a different aspect of handling spatial big data (storage, processing, and analysis). Below we describe each data type through a real-world problem with challenges, related work, novel algorithmic solutions, and experimental analysis.

To address the challenge of *analysis of spatial big data*, we studied the problem of finding primary corridors in bicycle GPS datasets. Given a set of GPS trajectories on a road network, the goal of the All-Pair Network Trajectory Similarity (APNTS) problem is to calculate the similarity between all trajectories using the Network Hausdorff Distance. This problem is important for a variety of societal applications, such as facilitating greener travel via bicycle corridor identification. The APNTS problem is challenging due to the high cost of computing the exact Network Hausdorff Distance between trajectories in spatial big datasets. Previous work on the APNTS problem takes over 16 hours of computation time on a real-world dataset of bicycle GPS trajectories in Minneapolis, MN. In contrast, this work focuses on a scalable method for the APNTS problem using the idea of row-wise computation, resulting in a computation time of

less than 6 minutes on the same datasets. We provide a case study for transportation services using a data-driven approach to identify primary bicycle corridors for public transportation by leveraging emerging GPS trajectory datasets. Experimental results on real-world and synthetic data show a two orders of magnitude improvement over previous work.

To address the challenge of *storage of spatial big data*, we studied the problem of storing spatio-temporal networks in spatial database systems. Given a spatio-temporal network and a set of database query operators, the goal of the Storing Spatio-Temporal Networks (SSTN) problem is to produce an efficient data storage method that minimizes disk I/O access costs. Storing and accessing spatio-temporal networks is increasingly important in many societal applications such as transportation management and emergency planning. This problem is challenging due to strains on traditional adjacency list representations when storing temporal attribute values from the sizable increase in length of the time-series. Current approaches for the SSTN problem focus on orthogonal partitioning (e.g., snapshot, longitudinal, etc.), which may produce excessive I/O costs when performing traversal-based spatio-temporal network queries (e.g., route evaluation, arrival time prediction, etc) due to the desired nodes not being allocated to a common page. We propose a Lagrangian-Connectivity Partitioning (LCP) technique to efficiently store and access spatio-temporal networks that utilizes the interaction between nodes and edges in a network. Experimental evaluation using the Minneapolis, MN road network showed that LCP outperforms traditional orthogonal approaches.

The work in this thesis is the first step toward understanding the immense challenges and novel applications of Spatial Big Data Analytics for Urban Informatics. In this thesis, we define spatial big data and propose novel approaches for *storing* and *analyzing* two popular spatial big data types: GPS trajectories and spatio-temporal networks. We conclude the thesis by exploring future work in the *processing* of spatial big data.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Urban Informatics is a set of ideas and technologies that will transform the lives of everyday citizens by helping understand the urbanized world, knowing and communicating our relation to people and places in that world, and generally enabling more livable and healthier cities. Recent examples of urban informatics in use across the world are listed in Table 1.1. The city of Santander, Spain uses over "12,000 electronic sensors that track everything from traffic and noise to surfing conditions at local beaches [7]". A reoccurring theme in the example cities given is a centralized computer system to collect data and respond to events in real-time.

New and immense location-aware datasets formally defined in this thesis as Spatial Big Data are emerging from a variety of sources and can be used to find novel and interesting patterns for use in urban informatics. This large variety of disparate data sources is fostering the new field of Urban Informatics which has the potential to transform metropolitan services such as public health, transportation, and urban utilities. In public health, spatial aspects, e.g., neighborhood context [8], are critical in understanding many contributors to disease process including environmental toxicant exposure as well as human behavior and lifestyle choices. This exposome [9], a characterization of a person's lifetime exposures, is becoming an increasingly popular subject of research for public health [10].

Urban Informatics requires the ability to *store, process, and analyze* spatial big data, something that challenges traditional spatial computing systems [11]. With these challenges come new opportunities in many fields of computer science research, such as

spatial data mining and spatial database systems. This thesis contains original research on two types of spatial big data, each study focusing on a different aspect of handling spatial big data (storage, processing, and analysis).

Table 1.1: Examples of Current Urban Informatics Projects

City	**Country**	**Urban Informatics and Use-Cases**
Songdo	South Korea	"Computers will be built into the houses, streets and offices as part of a "ubiquitous" network linking everyone in a sort of digital commune [12]."
SmartSantander	Spain	"Buried under the streets of Santander, Spain - or discreetly affixed to buses, utility poles, and dumpsters - are some 12,000 electronic sensors that track everything from traffic to noise to surfing conditions at local beaches [7]."
PlanIT Valley	Portugal	"using a centralized computer brain to control functions like water use, waste processing... from a network of sensors much like a nervous system to collect data and control the city [13]."
Masdar City	Abu Dhabi	"Everything is connected through a cloud to an Urban Operating System, which acts as the city's brain [13]."

1.1 Urban Informatics

Urban Informatics is the practice of using computer technology to support core city functions: planning, governance and operations. This technology consists of hardware, software, databases, sensors and communication devices used to develop and sustain more livable and healthy cities [14]. Urban Informatics provides governments with the tools to make data-driven decisions regarding long-term plans, predict and respond to current and upcoming situations, and even help with day-to-day tasks such as monitoring water use and waste processing.

Urban Informatics is an interdisciplinary field across many related disciplines (e.g., Citizen Science [15], Urban Computing [16], Ubiquitous Computing [17]) and consist of a variety of academic fields. Recently defined in the *Handbook of Urban Informatics* [18],

urban informatics can be defined as:

> "the study, design, and practice of urban experiences across different urban contexts that are created by new opportunities of real-time, ubiquitous technology and the augmentation that mediates the physical and digital layers of people, networks, and urban infrastructures." (Forth, Choi, & Satchell, 2011, p.4).

Urban informatics and related technologies are beginning to quantify these toxicant exposures through the use of wide-spread sensing devices. For example, Accra, Ghana used air quality measuring devices attached to smartphones to record spatio-temporal air quality information and compare it to the static daily reports given out by the government [15]. This data-driven approach found a huge variation of air quality across a single city and a single day, questioning single-source measurement for quantifying public health risks [15]. These case-studies provide ample motivation to push the development of Urban Informatics and the technologies behind it, such as Spatial Big Data.

1.2 Spatial Big Data

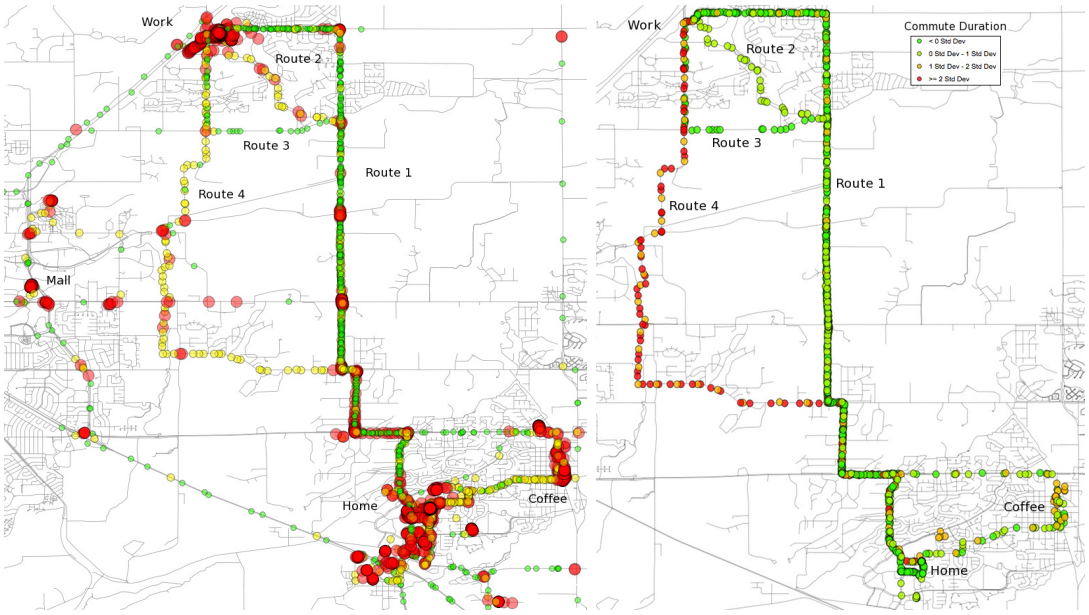
Increasingly, the size, variety, and update rate of spatial datasets exceed the capacity of commonly used spatial computing technologies to learn, manage, and process the data with reasonable effort. We refer to these datasets as Spatial Big Data (SBD). A 2011 McKinsey Global Institute report defines traditional big data as data featuring one or more of the 3 "V's": Volume, Velocity, and Variety [19]. Examples of emerging SBD include temporally detailed roadmaps that provide traffic speed values every minute for every road in a city, GPS trajectory data from cell-phones, and engine measurements of fuel consumption, greenhouse gas emissions, etc. Temporally-detailed roadmaps are providing more accurate travel time estimates for commuters depending on the time of day. Location-based services are allowing cities to examine usage patterns of bike lanes and park trails and make data-driven decisions about placing new corridors. Spatial big data is the key component driving the emerging field of Urban Informatics at the intersection of people, places, and technology. However, spatial big data presents challenges for existing spatial computing systems to *store*, *process*, and *analyze* such large

datasets. With these challenges come new opportunities in many fields of computer science research, such as spatial data mining and spatial database systems. Below we describe the societal applications, related work, and challenges of working with these two example spatial big datasets.

1.2.1 Analysis of GPS Trajectories

GPS trajectories are quickly becoming available for a larger collection of people due to rapid proliferation of cell-phones, in-vehicle navigation devices, and other GPS data-logging devices [20] such as those distributed by insurance companies [21]. Such GPS traces allow analysis of people for a number of urban informatics use-cases. For example, indirect estimation of fuel efficiency and GHG emissions is possible via estimation of vehicle-speed, idling and congestion. They also make it possible to make personalized route suggestions to users to reduce fuel consumption and GHG emissions. For example, Figure 1.1 shows 3 months of GPS trace data from a commuter with each point representing a GPS record taken at 1 minute intervals, 24 hours a day, 7 days a week. As can be seen, 3 alternative commute routes are identified between home and work from this dataset. These routes may be compared for idling which are represented by darker (red) circles. Assuming the availability of a model to estimate fuel consumption from speed profile, one may even rank alternative routes for fuel efficiency. In recent years, consumer GPS products [20, 22] are evaluating the potential of this approach. A key hurdle is the dataset size, which can reach 10^{13} items per year given constant minute-resolution measurements for all 100 million US vehicles.

Trajectory pattern mining is a popular field with a number of interesting problems both in geometric (Euclidean) spaces [24] and networks (graphs) [25]. A key component to traditional data mining in these domains is the notion of a similarity metric, the measure of sameness or closeness between a pair of objects. A variety of trajectory similarity metrics, both geometric and network, have been proposed in the literature [26]. One popular metric is Hausdorff distance, a commonly used measure to compare similarity between two geometric objects (e.g., polygons, lines, sets of points) [27]. A number of methods have focused on applying Hausdorff distance to trajectories in geometric space [28, 29, 30, 31].



(a) GPS Trace Data. Color indicates speed. (b) Routes 1, 2, 3, & 4 [23].

Figure 1.1: A commuter's GPS tracks over three months reveal preferred routes. (Best viewed in color)

Hausdorff distance has been shown to be a useful tool in geometric space for measuring similarity between trajectories, but applying Hausdorff distance to network-based trajectories is non-trivial. A number of papers have proposed approximation heuristics to compute Hausdorff distance on networks [32, 33, 34, 35, 36]. This is due to the expensive graph-distance computations needed when dealing with trajectories on networks. These approximations allow for interesting and useful pattern discovery, but do not compute exact similarities between trajectories and may alter results. We propose fast and correct algorithms for computing these similarities for a variety of use cases. We go on to provide a case study on real-world GPS data from cyclists in Minneapolis, MN.

1.2.2 Storage of Spatio-Temporal Networks

A Spatio-Temporal Network (STN) can be defined as a graph G = (N, E, T), where N is a set of nodes, E is a set of edges connecting two nodes, and every node and edge is associated with temporal information T (e.g., travel time). STN datasets are becoming

indispensable in societal applications, such as surface and air transportation management systems. One of the most appealing properties of these datasets is their ability to capture network attributes that vary over time. Consequently, STN datasets are usually massive in size and are accessed based on spatio-temporal movement patterns, making I/O efficient storage and access methods a significant challenge. Analyzing movement in spatio-temporal networks is important in many societal applications such as transportation, distribution of electricity and gas, and evacuation route planning. The ability to efficiently store, process and analyze spatio-temporal networks with large time series data would provide benefit to a wide variety of applications.

The Federal Highway Administration [3] is recording traffic data of major roads and highways using sensors such as loop detectors, among others, across the United States. Depending on the type of sensor, traffic levels are recorded every minute throughout a day, as shown in Figure 1.2. The Mobility Monitoring Program (MMP), started in 2000 by the Texas Transportation Institute, aimed to evaluate the use of sensors for traffic information around the United States. By 2003, MMP was receiving traffic sensor data from over 30 cities and 3,000 miles of highway, with sensor readings occurring roughly every 30 seconds. This data is then recorded 24 hours a day, 365 days a year, resulting in millions of time steps per year for each sensor. MMP published a report citing the need for processing and storage of historical traffic data, and how it may benefit traffic management [37].

Spatio-temporal networks (STN) are used to represent temporally-detailed road networks, where analysis can be done on fine-grained traffic speed measurements. However, these datasets are significantly larger than their atemporal brethren, and pose a number of challenges for existing analytical, processing, and underlying infrastructure technology. Over the last decade, considerable work on STNs has focused on pre-computation techniques and speed-up algorithms for time-dependent route planning [38, 39, 40, 41, 42, 43]. By comparison, there has been relatively little work on the design and evaluation of storage and access methods for STNs. Early work employed geometric space indexes for both space and space-time [44]. Orthogonal partitioning methods, such as the longitudinal or snapshot method [43], are able to capture network connectivity based on either space or time orthogonally. The longitudinal method stores temporally consecutive properties of a node (or edge) into the same data page

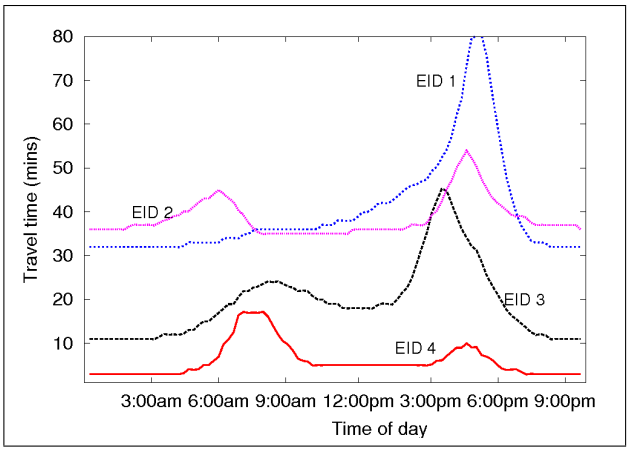


Figure 1.2: Traffic speed measurements averaged over 30 days by time of day. Courtesy: [3]

whereas the snapshot method stores a topologically connected sub-graph for a given time instance into the same data page. Current related work for storing and accessing STN data have relied on these orthogonal approaches [43]. In contrast with these methods, our approach focuses on non-orthogonal partitioning based on movement-connectivity [45, 46].

1.3 Thesis Contributions

The main contributions of this thesis address the challenges of defining, storing, processing, and analyzing spatial big data. First, we define spatial big data and discuss the challenges and opportunities SBD brings to spatial computing, elaborating on work we did in [11]. Second, we address the challenge of analyzing spatial big data using GPS trajectories as a case-study, motivated by the societal application of finding bicycle corridors from urban cyclists [47, 48]. Third, we address the challenge of storing spatial big data in spatial database systems, specifically storing spatio-temporal networks for use in advanced routing services [45, 46]. We lastly present our preliminary work on the challenge of processing spatial big data in the future work section of the thesis.

Table 1.2: Thesis Framework: Spatial Big Data Analytics for Urban Informatics

Spatial Big Data	**Urban Informatics**		
	Strategic	Tactical	Operational
Long-term Forecasts (climate, demographics, economy)	SBD for Urban Informatics (Ch. 2)		
Location Traces (GPS)		Bicycle Corridor Selection (Ch. 3)	
Spatio-Temporal Networks (STN)			Commuter Information Systems (Ch. 4)

1.4 Thesis Overview

Each layer of urban management (planning, governance, and operations) can be mapped to a temporal range consisting of *strategic*, *tactical*, and *operational* (STO) plans. This STO model is traditionally used to describe short and long term planning and management in businesses and government. Strategic plans provide long-term directions for a city, such as 20 year traffic and demographic projections. Tactical plans are on the time scale of 2-5 years, corresponding to the term of a mayor who may fund new bike lanes and other eco-friendly transportation options. Finally, operational tools provide day-to-day tools and information for city employees to help keep the city functioning smoothly and efficiently.

Table 1.2 illustrates how spatial big data may facilitate different scales of management and analysis for urban informatics. The left column labeled Spatial Big Data contains three different example datasets. The right three columns, strategic, tactical, and operational, labeled under the umbrella of Urban Informatics, correspond to different temporal scales of urban informatics. Each cell intersecting an urban informatics scale and example spatial big dataset illustrate a potential application. For example, the cell intersecting Location Traces and Tactical, Bicycle Corridor Selection, refers to a real-world application described in Chapter 3 of using GPS trajectory data to identify bicycle corridors for urban commuters. The rest of the cells in Table 1.2 highlight the other work in this thesis, such as an introduction to spatial big data for Urban

Informatics, and work on spatio-temporal networks to illustrate operational analysis for day-to-day driving information for travelers.

1.5 Outline

We begin in Chapter 2 by defining Spatial Big Data (SBD) and the underlying challenges and opportunities it presents for Urban Informatics. In Chapter 3, we describe GPS trajectory datasets and provide a case-study and contributions from our work on identifying primary corridors for urban cyclists [47]. Chapter 4 introduces spatio-temporal networks, and illustrates our contributions of storage and access of these networks from [45, 46] along with the potential uses for urban informatics. Finally, we conclude the thesis with our planned future work.

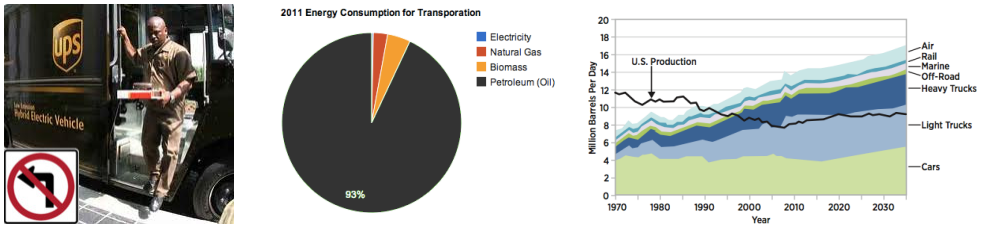
Chapter 2

Enabling Urban Informatics with Spatial Big Data

Increasingly, the size, variety, and update rate of spatial datasets exceed the capacity of commonly used spatial computing and spatial database technologies to learn, manage, and process the data with reasonable effort. We refer to these datasets as Spatial Big Data, processed via spatial computing. In this chapter, we present published work in which we were first to define spatial big data and the challenges and opportunities it presents to the spatial computing community [11]. Spatial computing is a set of ideas and technologies that transform lives by understanding the geo-physical world, knowing and communicating relations to places in that world, and navigating through those places. The transformational potential of spatial computing technologies is already evident. From Google Maps to consumer Global Positioning System (GPS) devices, society has benefited immensely from routing services and technology. Scientists use GPS to track endangered species to better understand behavior, and farmers use GPS for precision agriculture to increase crop yields while reducing costs. We've reached the point where a hiker in Yellowstone, a biker in Minneapolis, and a taxi driver in Manhattan know precisely where they are, their nearby points of interest, and how to reach their destinations.

We believe that harnessing Spatial Big Data (SBD) will enable the growing field of Urban Informatics. Examples of emerging SBD datasets include temporally detailed

(TD) roadmaps that provide speeds every minute for every road-segment, GPS trace data from cell-phones, and engine measurements of fuel consumption, greenhouse gas (GHG) emissions, etc.



(a) UPS avoids left-turns to save fuel [49].

(b) Petroleum is dominant energy source for US Transportation [50].

(c) Gap between US petroleum consumption and production is large and growing [51, 52].

Figure 2.1: Eco-routing supports sustainability and energy independence. (Best in color)

Eco-Routing: A 2011 McKinsey Global Institute report estimates savings of "about $600 billion annually by 2020" in terms of fuel and time saved [19] by helping vehicles avoid congestion and reduce idling at red lights or left turns. Preliminary evidence for the transformative potential includes the experience of UPS, which saves millions of gallons of fuel by simply avoiding left turns (Figure 2.1(a)) and associated engine-idling when selecting routes [49]. Immense savings in fuel-cost and GHG emission are possible in the future if other fleet owners and consumers avoided left-turns and other hot spots of idling, low fuel-efficiency, and congestion. 'Eco-routing' may help identify routes which reduce fuel consumption and GHG emissions, as compared to traditional route services reducing distance traveled or travel-time. Eco-routing has the potential to significantly reduce US consumption of petroleum, the dominant source of energy for transportation (Figure 2.1(b)). It may even reduce the gap between domestic petroleum consumption and production (Figure 2.1(c)), helping bring the nation closer to the goal of energy independence [53].

Public Safety: Evacuation planning is a crucial task for managing public safety. Whether natural (flood, hurricanes, etc.) or man-made (release of chemical or toxic substances, etc.) disasters require that emergency personnel be able to move affected populations to safety in as short a time as possible. Despite the increased threat of disasters posed by global climate change and the rise of terrorism, however, current

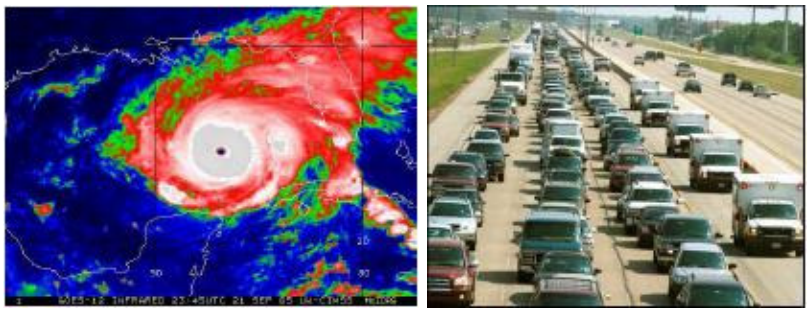


Figure 2.2: Hurricane Rita and Evacuation Traffic. Source: National Weather Services and FEMA.

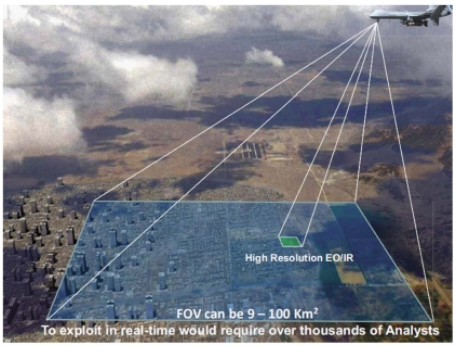
evacuation planning tools have serious limitations. Evacuation conducted during Hurricane Katrina and Rita in 2005 were a stark reminder of how much can go wrong despite intensive emergency preparation. For example, Figure 2.2 shows the miles of backed-up traffic that occurred as Houston residents followed orders to flee the path of Hurricane Rita. Therefore, efficient tools are needed to produce plans that identify optimal evacuation routes and schedules, given a transportation network, node/edge capacity constraints, source/destination nodes and evacuee population.

Even before cable news outlets began reporting the tornadoes that ripped through Texas on Tuesday, a map of the state began blinking red on a screen in the Red Cross' new social media monitoring center [54], alerting weather watchers that something was happening in the hard-hit area.

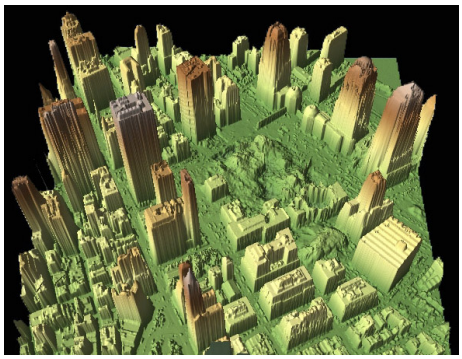
UN Global Challenges: Real-time or near real-time water quality monitoring is gaining traction in the field of environmental engineering as a way to better respond to water quality problems. Nevertheless, our ability to *use* the data generated in such systems is limited. These systems generate so much data (#/day for a single sensor) that analyzing the information generated by networks of sensors quickly becomes impossible. In addition, although we turn to computational methods to help us understand events, current computational thinking in environmental engineering emphasizes simulation. Thus, we are currently limited to explaining observations based on our knowledge of biological/chemical/physical phenomena. Indeed, we do not rely on computational methods to help identify new phenomena and lead beyond what is already known. Data mining techniques will allow this to occur, however, and drive discovery in the environmental field.

2.1 Defining Spatial Big Data

Spatial datasets are discrete representations of typically continuous phenomena. Discretization of continuous space is necessitated by the nature of digital representation. There are three basic models to represent spatial data, namely, raster (grid), vector and graph. Satellite images are good examples of raster data. On the other hand, vector data consists of points, lines, polygons and their aggregate (or multi-) counter parts. Graphs consisting of spatial networks are another important data type. Spatial Big Data can also be represented via these basic models, only the datasets have become much richer and more sophisticated.



(a) Wide-area persistent surveillance. FOV: Field of view. (Photo courtesy of the Defense Advanced Research Projects Agency.) EO: Electro-optical. [55]



(b) LIDAR images of ground zero rendered Sept. 27, 2001 by the U.S. Army Joint Precision Strike Demonstration from data collected by NOAA flights. Thanks to NOAA/U.S. Army JPSD.

Raster data, such as geo-images (Google Earth), are frequently used for remote sensing and land classification. New Spatial Big Raster Datasets are emerging from a number of sources.

UAV Data: Wide area motion imagery (WAMI) sensors are increasingly being used for persistent surveillance of large areas, including densely populated urban areas. The wide-area video cover-age and 24/7 persistence of these sensor systems allow for new and interesting patterns to be found via temporal aggregation of information. However, there are several challenges associated with using UAVs in gathering and managing raster datasets. First, UAV has a small footprint due to the relatively low flying height, therefore, it captures a large amount of images in a very short period of time to achieve the spatial coverage for many applications. This poses a significant challenge to store

rapidly increasing digital images. Image processing is another challenging because it would be too time consuming and costly to rectify and mosaic the UAV photography for large areas. The large quantity of data far exceeds the capacity of the available pool of human analysts [56]. It is essential to develop automated, efficient, and accurate technique to handle these spatial big data.

LiDAR: Lidar (Light Detection and Ranging or Laser Imaging Detection and Ranging) data is generated by timing laser pulses from an aerial position (plane or satellite) over a selected area to produce a surface mapping [57]. Lidar data are very rich to analyze surface or extract features. However, these data sets contain irrelevant data for spatial analysis and sometimes miss critical information. These large volumes of data from multiple sources poses a big challenge on management, analysis, and timely accessibility. Particularly, Lidar points and their attributes have tremendous sizes making it difficult to categorize these datasets for end-users. Data integration from multiple spatial sources is another challenge due to the massive amounts of Lidar datasets. Therefore, Spatial Big Data is an essential issue for Lidar remote sensing

Vector data over space is a framework to formalize specific relationships among a set of objects. Traditionally vector data consists of points, lines and polygons; and with the rise of Spatial Big Data, corresponding datasets have arisen from a variety of sources.

VGI Data: Volunteered geographic information (VGI) brings a new notion of infrastructure to collect, synthesize, verify, and redistribute geographic data through geolocation technology, mobile devices, and geo-databases. Theses geographic data are provided, modified, and shared based on user interactive online services (e.g., OpenStreetMap, Wikimapia, GoogleMap, GoogleEarth, Microsofts Virtual Earth, Flickr, etc). In recent years, VGI leads an explosive growth in the availability of user-generated geographic information and requires bigger storage model to handle large scale spatial dataset. The challenge for VGI is to enhance data service quality regard to accuracy, credibility, reliability, and overall value [54].

Graph data, in spatial computing, is commonly used to represent road maps for routing queries. While the network structure of the graph may not be changing, the amount of information about the network is rising drastically. New temporally-detailed road maps give minute by minute speed information, along with elevation and engine

measurements to allow for more sophisticated querying of road networks.

Spatio-Temporal Engine Measurement Data: Many modern fleet vehicles include rich instrumentation such as GPS receivers, sensors to periodically measure sub-system properties [58, 59, 60, 61, 62, 63], and auxiliary computing, storage and communication devices to log and transfer accumulated datasets. Engine measurement datasets may be used to study the impacts of the environment (e.g., elevation changes, weather), vehicles (e.g., weight, engine size, energy-source), traffic management systems (e.g., traffic light timing policies), and driver behaviors (e.g., gentle acceleration or braking) on fuel savings and GHG emissions. These datasets may include a time-series of attributes such as vehicle location, fuel levels, vehicle speed, odometer values, engine speed in revolutions per minute (RPM), engine load, emissions of greenhouse gases (e.g., CO2 and NOX), etc. Fuel efficiency can be estimated from fuel levels and distance traveled as well as engine idling from engine RPM. These attributes may be compared with geographic contexts such as elevation changes and traffic signal patterns to improve understanding of fuel efficiency and GHG emission. For example, Figure 2.3 shows heavy truck fuel consumption as a function of elevation from a recent study at Oak Ridge National Laboratory [4]. Notice how fuel consumption changes drastically with elevation slope changes. Fleet owners have studied such datasets to fine-tune routes to reduce unnecessary idling [64, 65]. It is tantalizing to explore the potential of this dataset to help consumers gain similar fuel savings and GHG emission reduction. However, these datasets can grow big. For example, measurements of 10 engine variables, once a minute, over the 100 million US vehicles in existence [66, 67], may have 10^{14} data-items per year.

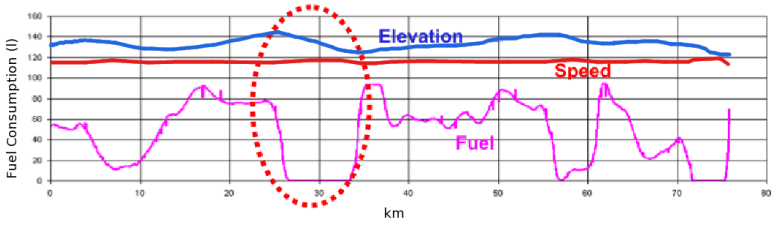
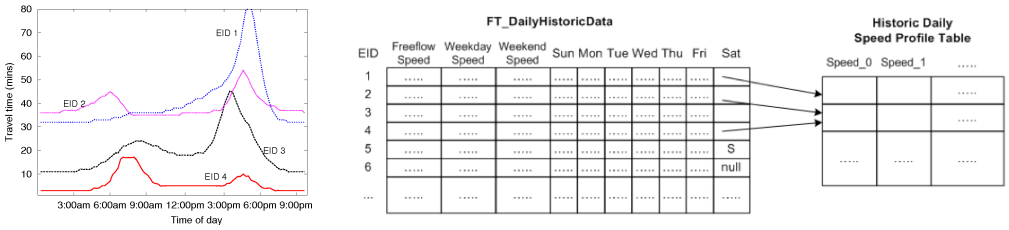


Figure 2.3: Engine measurement data improve understanding of fuel consumption [4]. (Best in color)

Historical Speed Profiles: Traditionally, digital road maps consisted of center lines

and topologies of the road networks [68, 69]. These maps are used by navigation devices and web applications such as Google Maps [23] to suggest routes to users. New datasets from companies such as NAVTEQ [70], use probe vehicles and highway sensors (e.g., loop detectors) to compile travel time information across road segments for all times of the day and week at fine temporal resolutions (seconds or minutes). This data is applied to a profile model, and patterns in the road speeds are identified throughout the day. The profiles have data for every five minutes, which can then be applied to the road segment, building up an accurate picture of speeds based on historical data. Such TD roadmaps contain much more speed information than traditional roadmaps. While traditional roadmaps have only one scalar value of speed for a given road segment (e.g., EID 1), TD roadmaps may potentially list speed/travel time for a road segment (e.g., EID 1) for thousands of time points (Figure 2.4(a)) in a typical week. This allows a commuter to compare alternate start-times in addition to alternative routes. It may even allow comparison of (start-time, route) combinations to select distinct preferred routes and distinct start-times. For example, route ranking may differ across rush hour and non-rush hour and in general across different start times. However, TD roadmaps are big and their size may exceed 10^{13} items per year for the 100 million road-segments in the US when associated with per-minute values for speed or travel-time. Thus, industry is using speed-profiles, a lossy compression based on the idea of a typical day of a week, as illustrated in Figure 2.4(b), where each (road-segment, day of the week) pair is associated with a time-series of speed values for each hour of the day.



(a) Travel time along four road segments over a day.

(b) Schema for Daily Historic Speed Data.

Figure 2.4: Spatial Big Data on Historical Speed Profiles. (Best viewed in color)

2.2 Spatial Big Data Opportunities

Spatial Big Data provides the opportunity to solve some of the long-standing challenges in spatial computing which stemmed from lack of data. Here we describe four novel opportunities: estimating spatial neighbor relationships, supporting place-based ensemble models, simplifying spatial models, and on-line spatio-temporal data analytics.

2.2.1 Estimating Spatial Neighbor Relationships

The data inputs of SDM are complex because they include extended objects such as points, lines, and polygons in vector representation and field data in raster data [71]. During data input, relationships among spatial objects are often implicit (e.g., overlap, intersect, etc.) and are often captured by models or techniques that incorporate spatial information into the SDM process. One such technique is to model the spatial relationship among locations in a spatial framework via a contiguity matrix which may represent a neighborhood relationship defined using adjacency or Euclidean distances. These neighborhood or W matrices are used in many SDM tasks such as spatial outlier detection, co-location pattern discovery, spatial classification and regression modeling, spatial clustering, and spatial hotspot analysis [72].

Table 2.1: Spatial Auto-Regression and the W-matrix

NAME	MODEL
Classical Linear Regression	$y = x\beta + \epsilon$
Spatial Auto-Regression	$y = \rho W y + x\beta + \epsilon$

The W matrix poses a significant burden to end users due to the fact that W is quadratic in the number of locations and reliable estimation of W needs a very large number of data samples. In spatial classification and regression modeling, for example, the logistic spatial autoregressive model (SAR) includes the neighborhood relationship contiguity matrix. Table 2.1 shows a comparison of the classical linear regression model and the spatial auto-regression model where the spatial dependencies of the error term, or the dependent variable, are directly modeled in the regression equation.

SBD Opportunity 1: Post-Markov Assumption. SDB may be large enough

to provide a reliable estimate of W. This may ultimately relieve user burden and may improve model accuracy. Traditional assumptions might not have to be made such as limited interaction length (e.g., the Markov assumption), spatially invariant neighbor relationships (e.g., the eight-neighborhood contiguity matrix), and tele-connections derived from short-distance relationships.

2.2.2 Supporting Place-based Ensemble Models

Spatial heterogeneity (or non-stationarity) is an important concept is not captured in traditional data mining approaches. An important feature of spatial data sets is the variability of observed processes over space. Spatial heterogeneity refers to the inherent variation in measurements of relationships over space. In other words, no two places on Earth are identical. The influence of spatial context on spatial relationships can be seen in the variation of human behavior over space (e.g., differing cultures). Different jurisdictions tend to produce different laws (e.g., speed limit differences between Minnesota and Wisconsin). The term spatial heterogeneity is most often used interchangeably with spatial nonstationarity, which is defined as the change in the parameters of a statistical model or change in the ranking of candidate models over space [73].

Traditional physics-based models have been place-independent for the most part with the notable exception of geographically weighted regression (GWR) [74, 75]. The regression equation for GWR, shown by Eq. 2.1, has the same structure as standard linear regression, with the exception that the parameters are spatially varying, where $\beta(s)$ and $\epsilon(s)$ represent the spatially varying parameters and the errors, respectively. GWR provides an ensemble of linear regression models, one per place of interest.

$$y = X\beta(s) + \epsilon(s) \tag{2.1}$$

Opportunity 2: SBD may support a Place-based ensemble of models beyond GWR. Examples include place-based ensembles of decision trees for land-cover classification and place-based ensembles of spatial auto-regression models. The computational challenge stems from the fact that naive approaches may run a learning algorithm for each place. Reducing the computation cost by exploiting spatial autocorrelation is an interesting possibility that will need to be explored further.

2.2.3 Simplifying Spatial Models

Spatial models are usually computationally more expensive than traditional models. For example, spatial auto-regression requires more computing power due to the fact that W is quadratic in the number of locations (Table 2.1). Geographically weighted regression has the same limitation as opposed to classical linear regression, also due to the inclusion of the W matrix (Eq. 2.1). Colocation pattern mining, which finds the subsets of features frequently located together is more computationally expensive that traditional association rule mining [76] and confidence estimation adds more costs (e.g., M.C.M.C. simulations).

Opportunity 3: Bigger the SBD, Simpler the spatial models. SDB creates an opportunity to simplify spatial models in traditional SDM. It may be the case that some of the complexity from SDM is due to the paucity of data at individual places which in turn forces one to leverage data at nearby places via spatial autocorrelation and spatial joins. SBD may provide a large volume of data at each place which may allow algorithmic techniques such as place-based divide and conquer. Consequently, it may only be necessary to build one model per place using local data and simpler models. There are, however, a few challenges that must be considered when comparing place-based ensembles of simpler models with current spatial models. First, it is unclear when bigger data leads to simpler models. Second, the definition of SBD from an analytics perspective is also unclear (e.g., ratio of samples to number of parameters).

2.2.4 On-line Spatio-Temporal Data Analytics

A fundamental limitation of SDM is off-line batch processing where spatial models are usually not learned in real time (e.g., spatial auto-regression, colocation pattern mining, and hotspot detection). However, SBD includes streaming data such as event reports and sensor measurements. Furthermore, the use cases for SBD include monitoring and surveillance which requires on-line algorithms. Examples of such applications include 1) the timely detection of outbreak of disease, crime, unrest and adverse events, 2) the displacement or spread of a hotspot to neighboring geographies, and 3) abrupt or rapid change detection in land cover, forest-fire, etc. for quick response.

Opportunity 4: On-line Spatio-Temporal Data Analytics. Models that are

purely local may leverage time-series data analytics models but regional and global models are more challenging. For spatial interactions (e.g., colocations and tele-connections) with time-lags, SBD may provide opportunities for precisely computing them in an on-line manner. If precise on-line computation is not possible, SBD might be useful in providing on-line approximations.

2.3 Spatial Big Data Infrastructure

The management of large volumes of spatial data is a big challenge for several applications, such as satellite monitoring systems, intelligent transportation systems, and disaster management systems. These applications need to provide a solution to the increasing data demands and offer a shared, distributed computing infrastructure as well as reliable system. The complexity and nature of spatial datasets makes them ideal for applying parallel processing. Recently, the concept of a cloud environment has been introduced to provide a solution for these requirements. Existing approaches and solutions provide a general framework for distributed file system (e.g., Google file [77] system and HDFS [78]) and process these data sets based on replicas of data blocks (e.g., map-reduce [79] and Hadoop [78]). Column-oriented database systems have also been introduced to support OLAP or join processing (e.g., MongoDB and HBase).

However, it is hard to generalize the infrastructure to handle spatial problems. For instance, cloud computing should divide big data sets and distribute in terms of load balancing and support parallel processing with minimum communication cost [80]. Particularly, the overhead of synchronization for every process is challenging for heterogeneous spatial datasets (e.g., polygons and line-strings, spatio-temporal road networks). Furthermore, fault-tolerance and reliability for big data sets is another challenge on emergency management system (e.g. evacuation route planning).

2.3.1 Parallelization of Spatial Big Data

The assumption that learning samples are independently and identically distributed (IID) has been traditionally made by many classical data mining algorithms such as linear regression. However, the IID assumption is violated when dealing with spatial data due to spatial auto-correlation [81] where models such as classical linear regression

yield both low prediction accuracy and residual error exhibiting spatial dependence [82]. The spatial auto-regression (SAR) model [83, 81] was proposed as a generalization of the linear regression model to account for spatial auto-correlation. It was successfully used to analyze spatial datasets in areas such as regional economics and ecology, and it was shown to yield better classification and prediction accuracy for many spatial datasets exhibiting strong spatial auto-correlation [84, 82]. However, estimating the parameters of SAR was found to be computationally expensive, which relegated its applicability to small problems, despite its promise to improve classification and prediction accuracy [85]. This has created an opportunity for parallel processing to speed-up sequential procedures such as the SAR model.

Map-reduce provides an adequate framework for computing spatial auto-regression where large spatial data sets may be processed in a distributed environment. This works well for applications such as multiscale multigranular image classification into land cover categories.

2.3.2 Difficulties of Parallelization

The GIS-range-query problem has three main components: (i) approximate filtering at the polygon level, (ii) intersection computation, and (iii) polygonization of the result [86]. A search structure is used to filter out many non-interesting polygons from the set of input polygons. The query box is then intersected with each of the remaining polygons, and the output is obtained as a set of polygons by polygonizing the results of the intersection computation.

The GIS-range-query operation can be parallelized either by function-partitioning or by data-partitioning [86]. Function-Partitioning uses specialized data structures and algorithms which may be different from their sequential counterparts. A data-partitioning technique divides the data among different processors and independently executes the sequential algorithm on each processor. Data-partitioning is in turn achieved by declustering the spatial data.

The goal of declustering is to partition the data so that each partition imposes approximately the same load for any range query. Intuitively, the polygons close to each other should be scattered among different processors such that for each range-query, every processor has an equal amount of work. Polygonal maps can be declustered

at the polygonal level or at the sub-polygonal level. Optimal declustering of extended spatial data like polygons is difficult to achieve due to the non-uniform distribution and variable sizes of polygons. In addition, the load imposed by a polygon for each range query depends on the size and location of the query. Since the location of the query is not known a priori, it is hard to develop a declustering strategy that is optimal for all range queries. As the declustering problem is NP-Hard [86], heuristic methods are used for declustering spatial data. Random partitioning, local load-balance and similarity-graph-based methods are three popular, algorithms for declustering spatial data. Intuitively, a local load-balance method tries to balance the work-load at each processor for a given range query. A similarity based declustering method tries to balance the work-load at each processor over a representative set of multiple range queries.

2.3.3 Problems with Current Techniques

Another problem is the difficulty in applying existing iterative algorithms to cloud environments. For instance, because most spatial graph algorithms (e.g., breadth-first search and shortest path) use previous information for the next iteration, it is hard to perform parallel processing. Recently, several interesting and effective solutions and prototype systems have been developed [87, 88, 89, 87] but they have limitations when dealing with spatial data sets. Specifically, spatial networks (e.g., transportation networks) have higher diameters than complex networks (e.g., social networks), and as such the large number of iterations becomes the main bottleneck in processing spatial big data. Although processing one iteration is parallelizable, the synchronization overhead for cloud environment is too enormous to handle large scale datasets. Future work should include non-iterative algorithms or different parallel programming models.

2.4 Conclusion

Spatial Big Data has immense potential to benefit a number of societal applications. By harnessing this increasingly large, varied and changing data, new opportunities to solve worldwide problems are presented. To capitalize on these new datasets, inherent challenges that come with spatial big data need to be addressed. Many spatial operations are iterative by nature, something that parallelization has not yet been able to

handle completely. By expanding our cyber-infrastructure, we can harness the power of these massive spatial datasets. New forms of analytics using simpler models and richer neighborhoods will enable solutions in a variety of disciplines.

Chapter 3

Analysis of GPS Trajectories for Bicycle Corridor Identification

3.1 Introduction

To address the challenge of *analysis of spatial big data*, in this chapter we study the problem of finding primary corridors in bicycle GPS datasets. Given a set of trajectories on a road network, the goal of the All-Pair Network Trajectory Similarity (APNTS) problem is to calculate the Network Hausdorff Distance (NHD) between all pairs of input trajectories. The classical Hausdorff distance is defined as the "maximum distance of a set (of points) to the nearest point in the other set [90]." For example, the Network Hausdorff Distance from Trajectory B to Trajectory A in Figure 3.1 is 4 units (edge traversals), as every node in Trajectory B is at least 4 units or less away from any node in Trajectory A. Note that the Hausdorff distance is not symmetric. It is a commonly used similarity measure in computer vision [91] and computational geometry [92], but recently has been used to define similarity between trajectories, both in Euclidean space [29] and recently on graphs [32]. As the Hausdorff distance is set-based, it has no notion of order and therefore when applied to trajectories, traditionally time is ignored. Depending on the application domain, this may be acceptable as it is the underlying route choices that are of interest in our case study on urban bicycle corridor planning.

Motivation: Trajectory similarity measures are used in a number of important of societal applications, such as summarizing population movement within a city, or to

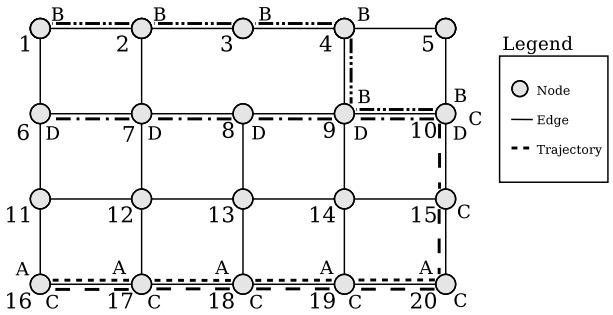


Figure 3.1: Road network represented as an undirected graph with four trajectories illustrated with bold dashed lines.

optimize bus route placement. Using network-based trajectories enforces topological constraints that are inherently present in road networks. Let us consider the problem of determining primary bicycle corridors through a city to facilitate safe and efficient bicycle travel. GPS trajectory data shows us where in the city bicycle commuters travel, allowing for data-driven decisions of bike corridor placement. By selecting representative corridors for a given group of commuters, we can minimize the overall alteration to their routes and encourage use of the bicycle corridors. Facilitating commuter bicycle traffic has been shown in the past to have numerous societal benefits, such as reduced greenhouse gas emissions and healthcare costs [93].

Challenges: The APNTS problem is challenging due to the computational cost of computing the Network Hausdorff Distance (NHD) between all pairs of input trajectories, as it requires a large amount of node-to-node distance comparisons. In our previous work [47], we proposed an exact but expensive baseline algorithm to compute the NHD, requiring multiple invocations of common shortest-path algorithms (e.g., Dijkstra's [94]). For example, given two trajectories consisting of 100 nodes each, a baseline approach to calculate NHD would need to compare the distances for all pairs of nodes within those two trajectories (10^4), which over a large trajectory dataset (e.g., pair-wise comparison of 10,000 trajectories) would require 10^{12} distance comparisons. This quickly becomes computationally prohibitive without faster algorithms.

Related Work: Trajectory pattern mining is a popular field with a number of interesting problems both in geometric (Euclidean) spaces [24] and networks (graphs) [25]. A key component to traditional data mining in these domains is the notion of a similarity metric, the measure of sameness or closeness between a pair of objects. A variety

of trajectory similarity metrics, both geometric and network, have been proposed in the literature [26]. One popular metric is Hausdorff distance, a commonly used measure to compare similarity between two geometric objects (e.g., polygons, lines, sets of points) [27]. A number of methods have focused on applying Hausdorff distance to trajectories in geometric space [28, 29, 30, 31].

Hausdorff distance has been shown to be a useful tool in geometric space for measuring similarity between trajectories for applications that do not use the temporal information of trajectories, but applying Hausdorff distance to network-based trajectories is non-trivial. A number of papers have proposed heuristics to approximate the Hausdorff distance on networks [32, 33, 34, 35, 36]. This is due to the large number of graph-distance computations needed to compute the NHD. These approximations allow for interesting and useful pattern discovery, but do not compute exact similarities between trajectories and may alter results. In our previous work [47], we proposed a correct but computationally expensive algorithm for clustering network trajectories on road networks using Network Hausdorff Distance to identify new bicycle corridors through a city to facilitate safe and efficient bicycle travel. However, while the optimized algorithm was a significant improvement over the baseline, it still remained computationally prohibitive for large trajectory datasets. We illustrate a classification of related work in Figure 3.2, highlighting the various approaches of Hausdorff distance computation for trajectories. In this chapter, we propose a new algorithm that improves performance by over an order of magnitude on synthetic and case study datasets.

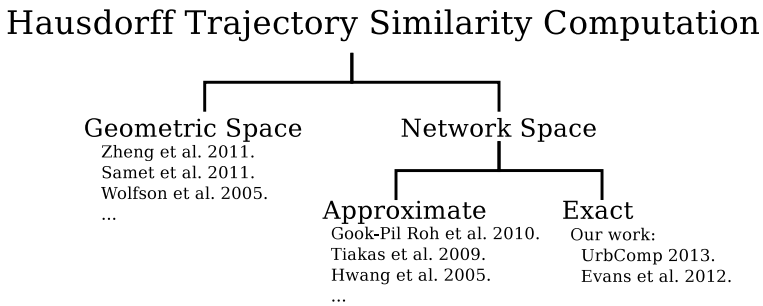


Figure 3.2: Classifications of Hausdorff Trajectory Similarity Algorithms.

Proposed Approach: In this chapter, we formalize the Network Hausdorff Distance for weighted graphs and propose a novel approach that is orders of magnitude

faster than the baseline approach and our previous work, allowing for Network Hausdorff Distance to be computed efficiently on large trajectory datasets. We propose a novel approach that computes the exact NHD while remaining computationally efficient. While the baseline approach computes node-to-node distances, the Network Hausdorff Distance (NHD) essentially requires node-to-trajectory minimum distance values. We take advantage of this insight to compute the NHD between nodes and trajectories directly by modifying the underlying graph, inserting a super-source node [95] and computing from a trajectory to an entire set of trajectories with a single shortest-paths distance computation.

Contributions: This chapter proposes a number of key contributions:

- We formalize the All-Pair Network Trajectory Similarity (APNTS) problem for computing similarities between a set of trajectories.
- We propose a fast and exact algorithm, ROW-TS, to solve the APNTS problem.
- We provide a case study on real-world GPS trajectory data of bicycle commuters in Minneapolis, MN.
- We validate the ROW-TS algorithm with experimental analysis.

Scope This chapter focuses on network trajectory similarity computation, being a crucial step for a number of trajectory pattern mining algorithms (e.g., trajectory clustering algorithms), as illustrated in our previous work which focuses on clustering for primary corridors in transportation [47]. A key component of the Network Hausdorff Distance is shortest-path distance computations and values. In this chapter, we will discuss a Dijkstra-like [94] framework for computing these paths during trajectory similarity computation. This assumes that the underlying network is too large to precompute and store the all-pair shortest-path distance matrix (e.g., Floyd-Warshall [94]). We ignore order-dependent trajectories (chronological ordering, etc.) for simplicity, but the discussed techniques can be transferred to a spatio-temporal network if a relation between space and time is needed.

Outline: The rest of the chapter is organized as follows. A brief description of the basic concepts and problem formulation is presented in Section 3.2. A description of the computational structure of the APNTS problem is presented in Section 3.3. In

Section 3.4 we propose the ROW-TS algorithm. We provide a case-study on a real-world bicycle GPS trajectory dataset in Section 3.5 and experimental analysis on synthetic datasets in Section 4.3. Finally, we conclude in Section 4.5 with a discussion of future work.

3.2 Problem Formulation

In this section, we describe the basic concepts required to describe the All-Pair Network Trajectory Similarity (APNTS) problem. We provide an example dataset and a formal problem statement.

3.2.1 Basic Concepts

We begin with a review of trajectories, networks, and how they are used to calculate the Hausdorff distance.

Definition 1 ***Road Network***

A road network is defined in this chapter as an undirected, weighted graph $G = \{V, E\}$ illustrated in Figure 3.1 where $V = (1, 2, 3, ..., 18, 19, 20)$ are the vertices or nodes in the graph G and $E = (1-2, 1-6, ..., 18-19, 14-19, 19-20...)$ are the edges connecting the vertices (edge weights of 1 unit). This representation is commonly used for transportation networks, where intersections are modeled as nodes. Note that the graph could be directed without a change to the proposed approach.

Definition 2 ***Network Trajectory***

A spatial trajectory traditionally refers to a series of points (a trace) of a moving object in geographic space [24]. In this chapter, we will be focusing on network trajectories, or, trajectories that consist as a set of nodes and edges in a graph. A common operation for GPS trajectory data on road networks involves map-matching the GPS points to the network [24]. This allows describing the trajectory using graph notation (a set of nodes and edges) and perform graph-based calculations, such as shortest-path distance. In this chapter, we define network trajectories as a set of connected vertices: $t_x = [v_1, ..., v_n]$.

Figure 3.1 contains four network trajectories: t_A = [16, 17, 18, 19, 20]; t_B = [1, 2, 3, 4, 9, 10]; t_C = [16, 17, 18, 19, 20, 15, 10]; t_D = [6, 7, 8, 9, 10].

Definition 3 ***Shortest-Path Distance***

To measure shortest-path distance on a network or graph, we sum of the length of edges required to traverse the graph between two specific nodes. A number of well-known algorithms compute shortest-paths (shortest path tree) on graphs [94]. In this chapter, we will use the generic function $dist(n, m)$ to indicate the shortest-path distance between nodes n and m. The choice of algorithm used to calculate this value may vary (Dijkstra's, A*, Floyd-Warshall if network size is small). In our pseudocode and implementation, we use Dijkstra's single-source shortest-paths algorithm [94] taking a source node v and a graph G as input: $distMap[] = Dijkstra(G, v)$. It returns the shortest-path distance to all other nodes in G. In Figure 3.1, an example $dist(3, 8)$ would be a distance of 3, traversing the shortest-path 3-2-7-8 or 3-4-9-8.

Definition 4 ***Network Hausdorff Distance***

As discussed in the related work, there are a number of variations and approximations of Hausdorff distance on networks [32, 33, 34, 35, 36]. Based on these and the original Hausdorff distance in geometric space [96], we formulate Network Hausdorff Distance (NHD) between two trajectories t_x and t_y in Equation 3.1. An example of NHD, and its asymmetric distances is $NHD(t_A, t_B)$ and $NHD(t_B, t_A)$. $NHD(t_A, t_B) = 3$ as the farthest node in t_A from t_B is 3 edges away and tied between a number of pairs: $(dist(16_A, 1_B) = 3)$, $(dist(17_A, 2_B) = 3)$, $(dist(18_A, 9_B) = 3)$. $NHD(t_B, t_A) = 4$ as the farthest node in t_B from t_A is 4 edges away with: $(dist(3_B, 17_A) = 4$ and $dist(3_B, 19_C) = 4$.

$$NHD(t_i, t_j) = \max_{n \in t_i} \min_{m \in t_j} dist(n_{t_i}, m_{t_j}) \tag{3.1}$$

Definition 5 ***Trajectory Similarity Matrix***

A similarity matrix contains the values based on some measure comparing each pairwise combination of objects in a dataset. They are input for a number of data mining

Table 3.1: Output for the All-Pair Network Trajectory Similarity problem: a Trajectory Similarity Matrix for the input data in Figure 3.1 using Network Hausdorff Distance.

$NHD(t_x, t_y)$	Track A	Track B	Track C	Track D
Track A	0	3	0	2
Track B	**4**	0	3	2
Track C	2	3	0	2
Track D	2	1	2	0

problems (e.g., clustering, classification). In this chapter, our focus is to quickly and efficiently produce a similarity matrix of input trajectories for use in popular trajectory pattern mining algorithms [24]. Table 3.1 contains the trajectory similarity matrix for the APNTS problem on the input data in Figure 3.1. Each cell value is the result of the $NHD(t_x, t_y)$ equation given the two corresponding tracks (row, column).

3.2.2 Problem Statement

The All-Pair Network Trajectory Similarity (APNTS) problem can be formulated as follows:

Input:

- Road Network: $G = \{V,E\}$
- Collection of Trajectories: T

Output:

- Trajectory Similarity Matrix: $M : T \times T \rightarrow \mathbb{R}$

Objective:

- Minimize computation time

Constraints:

- M values are correct given Equation 3.1 (Network Hausdorff Distance)
- G is a undirected, weighted graph with nonnegative edge weights
- Trajectories in T are paths in G

Example: Given the input road network and four trajectories in Figure 3.1, the corresponding trajectory similarity matrix is shown in Table 3.1. For each cell in the

matrix, the NHD value was calculated based on the two input trajectories.

3.3 Computational Structure

Network Hausdorff Distance is a recently popular topic in the literature [32, 33, 34, 35, 36, 97]. However, each of these papers cite performance issues with a network-based Hausdorff distance computation, with [32] saying "the baseline algorithm requires a large number of distance computations which significantly degrades performance...". This idea is echoed in the other papers, each proposing interesting and novel algorithms to approximate a network-based Hausdorff distance computation. In this section, we will discuss the underlying computational structure of this baseline algorithm and mention a few of the approximation algorithms and their limitations.

Essentially, the related work provides algorithms to reduce the number of Network Hausdorff Distance computations between trajectories using clustering techniques (e.g., density-based [97] or k-nn [32]) or by converting network trajectories to geometric space [98]. These clustering approaches degrade to the baseline brute-force approach when threshold values (density and neighborhood) are not appropriately set. In addition, as some portion (depending on thresholds) of trajectories are not being compared via the Network Hausdorff Distance, the resulting distance values are not exact and may affect the quality of the results of any algorithm using such values. In this chapter, we focus on an exact approach, ensuring each trajectory pair similarity is computed with the exact Network Hausdorff Distance while retaining computational scalability.

Table 3.2: Network distance between node pairs; required for $NHD(t_B, t_A)$ (Input: Figure 3.1, Full trajectory similarity matrix shown in Table 3.1.

$NHD(t_B, t_A)$	16_{t_A}	17_{t_A}	18_{t_A}	19_{t_A}	20_{t_A}	Min
1_{t_B}	3	4	5	6	7	3
2_{t_B}	4	3	4	5	7	3
3_{t_B}	5	4	5	4	5	4
4_{t_B}	6	5	4	3	4	3
9_{t_B}	5	4	3	2	3	2
10_{t_B}	6	5	4	3	2	2
Max	-	-	-	-	-	**4**

3.3.1 Graph-Node Track Similarity Baseline (GNTS - B)

The baseline algorithm to compute the Network Hausdorff Distance Track Similarity Matrix M computes the shortest-path distance between each pair of nodes within each pair of trajectories, choosing the maximum of the values of this set. Due to this enumeration, the approach is given the name Graph-Node Track Similarity, as it focuses on repeated graph computations between all pairs of nodes within the two trajectories being compared. For example, in Figure 3.1, to compute the similarity between Trajectory t_B and Trajectory t_A, we need to find the minimum distance from each node in Trajectory $t_B = [1_{t_B}, 2_{t_B}, 3_{t_B}, 4_{t_B}, 9_{t_B}, 10_{t_B}]$ to any node in Trajectory $t_A = [16_{t_A}, 17_{t_A}, 18_{t_A}, 19_{t_A}, 20_{t_A}]$, as described in Equation 3.1. These necessary shortest-path distances are shown in Table 3.2. These values can be computed via any number of shortest-path algorithms, but for simplicity we will focus on the popular Dijkstra's single-source shortest-paths algorithm [94]. Computing Table 3.2 would require 6 (or $|t_B|$) invocations of Dijkstra's algorithm to find the shortest-path distance from each node in t_B to each node in t_A with a runtime of $O(|V|log|V| + |E|)$ [94]. That amount of computation is necessary to find $NHD(t_B, t_A)$, one cell in the trajectory similarity matrix (TSM) shown in Table 3.1. Therefore, in the baseline case, a number of shortest-path algorithm invocations are required for each cell in the TSM.

Execution trace of GNTS: This baseline Graph-Node Track Similarity (GNTS) algorithm is shown as pseudocode in Algorithm 1, essentially invoking a shortest-path computation for each combination of nodes within each trajectory pair. To compute the TSM in Table 3.1, GNTS iterates through all pairs of input trajectories in Lines 1 & 2. Lines 3 and 4 test if we are computing the Network Hausdorff Distance from one trajectory to itself, which is always zero. Starting in Line 6, we begin to look for the maximum value of the minimum network distances from every node in Trajectory t_x to any node in Trajectory t_y. We set the current maximum value at negative infinity, and proceed to iterate over each node n in trajectory t_x. From n in Line 8 and 9, we set the current minimum value as infinity and run a undirected shortest path tree algorithm (Dijkstra's [94]) to find the shortest-path distance to all nodes in G from n. In Lines 10-14, we iterate through all nodes m in trajectory t_y, checking the distance map returned by the Dijkstra's computation in Line 9 to see if the distance from n to m is the smallest so far, if so, setting that fact in Line 12. By Line 15, we have found the

minimum distance from node n from trajectory t_x to any node in trajectory t_y. We then test to see if it is greater than our current maximum value for this pair of trajectories, which once done looping over these two sets of nodes in Line 19, is the correct answer for $NHD(t_x, t_y)$, saving it in the trajectory similarity matrix M. This continues for all other pair of trajectories in the dataset.

3.3.2 Graph-Node Track Similarity with Precomputed Distances (GNTS - P)

While our earlier work used the GNTS - Baseline algorithm above for comparison, there was interest in providing the baseline algorithm with all-pair node distance values in a lookup table, as they can be precomputed using various algorithms (e.g., Floyd-Warshall [94]). This is achieved by precomputing the all-pair node distance and replacing line 9 in Algorithm 1 with the precomputed values (constant time lookup). While this becomes prohibitive for large networks, we show in Section 4.3 that GNTS - P has some interesting performance characteristics, but has trouble with trajectories of longer length.

3.4 Proposed Approach

In this section we will describe our proposed approach to efficiently solve the APNTS problem. We begin by proposing our novel row-wise trajectory similarity (ROW-TS) algorithm. The ROW-TS algorithm calculates an entire row of the trajectory similarity matrix described in the APNTS problem with a single shortest-path computation.

3.4.1 Matrix-Element Track Similarity (METS)

While the baseline GNTS approach produces the correct output, a quick examination reveals a major computational bottleneck as summarized in Section 3.3 and directly visible in Algorithm 1, Line 9: multiple invocations of shortest-path algorithms for each element in the trajectory similarity matrix. Our previous work [47] highlighted an approach to reduce the number of shortest-path invocations to one invocation for each cell in the trajectory similarity matrix.

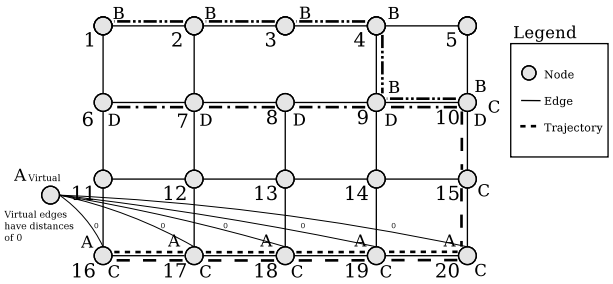


Figure 3.3: Inserting a virtual node ($A_{virtual}$) to represent Track A for efficient Network Hausdorff Distance computation.

To compute the Network Hausdorff Distance $NHD(t_x, t_y)$ between two trajectories, we don't actually need the shortest distance from all pairs of nodes in the two trajectories. We need the minimum distance from each node in t_x to *any* node in t_y. We propose a novel approach to find this distance, not the all-node-pair distance. This significantly reduces the number of distance calculations and node iterations needed to compute the NHD. To do this, we modify the underlying graph. Using Figure 3.3, to calculate $NHD(D, A)$, we begin by inserting a virtual node ($D_{virtual}$) representing Track D into the graph. This node will have edges with weights of 0 connecting it to each other node in Track D. Now, we can run a shortest-path distance computation from the virtual node as a source, with the destination being every node in Track A. We then know the shortest distance from each node in Track A to the virtual node. Since the virtual node is only connected to nodes in Track D, and all the edge weights are 0, we actually have the shortest-path from each node in Track A to the closest node in Track D, exactly what Network Hausdorff Distance requires for computation.

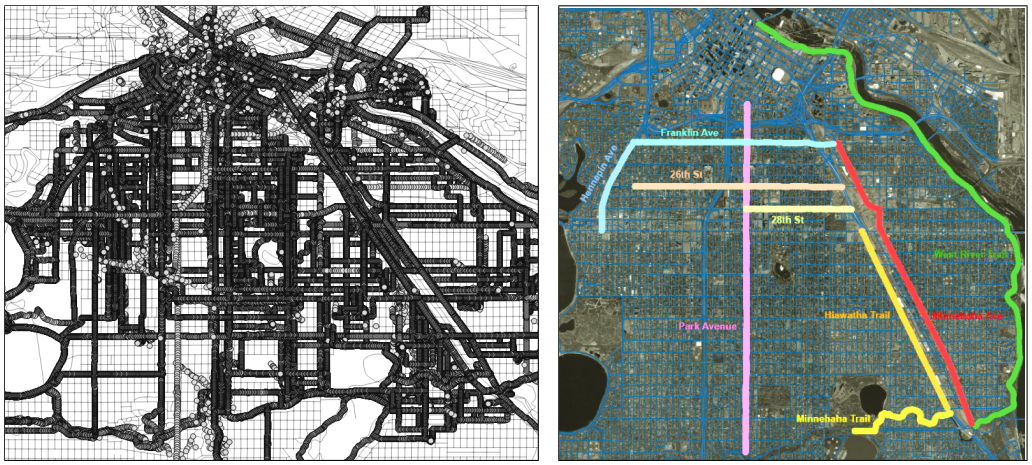
For example, when we want to find the Track Similarity between Track D and Track A, we now start by creating a virtual node connected to all nodes in Track A. We then run Dijkstra from the virtual node to nodes (13, 14, 15, 16, 18, 19), giving us the shortest-paths from those nodes to the virtual node. At this point we have the same distance information that is given in Table 3.2, and can compute the trajectory similarity without multiple invocations of shortest-path algorithms for each element.

Execution Trace of METS: The pseudocode for METS is given in Algorithm 2. To compute the TSM in Table 1, METS iterates through all pairs of input trajectories t_x and t_y in Lines 1 and 2. Lines 3 and 4 test if network Hausdorff Distance is being

computed from one trajectory to itself, which is always zero. In Lines 6 - 10, a virtual node is added to the graph and connected to each node t_x by an edge of length zero. In line 11, METS runs an undirected shortest path tree algorithm (Dijkstras [94]) to find the shortest-path distance between tracks the virtual node and t_y, which effectively finds the distance between t_x and t_y because each node in t_y is connected to v via the nearest node in t_x. At this point, the virtual node is removed from the graph (line 12). In lines 14 18, METS iterates through all nodes m in trajectory t_y, checking the distance map returned by the Dijkstras computation (involving the virtual node) in line 11 to see if the distance from n to m is the largest so far (Network Hausdorff Distance). In line 19, METS updates the track similarity matrix M with the newly calculated Hausdorff distance between t_x and t_y and returns M in line 23.

3.4.2 Row-Wise Track Similarity (ROW-TS)

Computing the Network Hausdorff Distance $NHD(t_x, t_y)$ between two trajectories does not require the shortest distance between all-pairs of nodes in t_x and t_y. Instead, it requires the minimum distance from each node in t_x to the *closest* node in t_y. In Figure 3.3, to calculate $NHD(t_B, t_A)$, we begin by inserting a virtual node ($A_{virtual}$) representing Trajectory t_A into the graph. This node has edges with weights of 0 connecting it to each other node in Trajectory t_A. We then run a shortest-path distance computation from the virtual node as a source, with the destination being every node in Trajectory t_B. The result was the shortest distance from each node in Trajectory t_B to the virtual node $A_{virtual}$. Since the virtual node is only connected to nodes in Trajectory t_A, and all the edge weights are 0, we had the shortest-path from each node in Trajectory t_B to the closest node in Trajectory t_A, exactly what $NHD(t_B, t_A)$ requires for computation. However, this focused on computing a single cell in the trajectory similarity matrix per invocation of a single-source shortest-paths algorithm. That means at least $O(|T|^2)$ shortest-path invocations to compute the TSM for the APNTS problem, still quite expensive. We propose a new approach, ROW-TS, to compute an entire *row* of the TSM with one invocation of a single-source shortest-paths algorithm. Using a row-based approach, we can essentially calculate $NHD(t \in T, t_A)$ with one single-source shortest-paths invocation from $A_{virtual}$. This approach reduces the overall number of shortest-paths invocations to $O(|T|)$ at the cost of additional bookkeeping, as we will



(a) Recorded GPS points from bicyclists in Minneapolis, MN. Intensity indicates number of points.

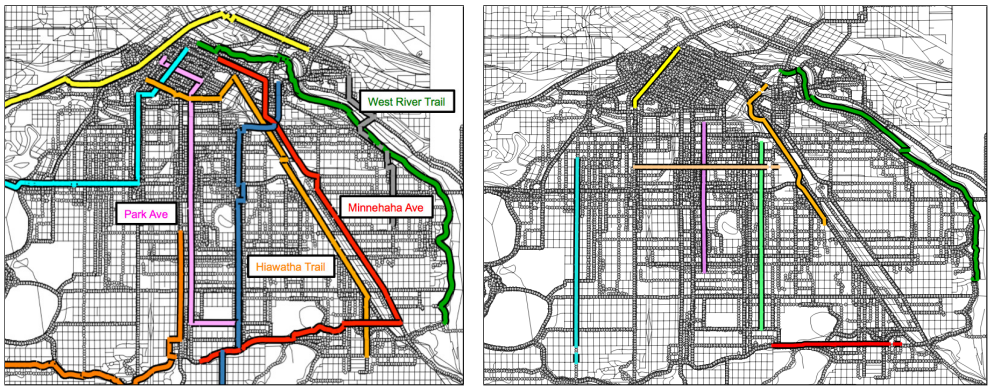
(b) Original, hand-crafted primary corridors identified by our co-author and fellow geographers in [1]

Figure 3.4: Example input and output of the k-Primary Corridor problem.

show below.

The pseudocode for ROW-TS is given in Algorithm 3. To compute the TSM in Table 1, ROW-TS iterates through each input trajectory t_x. In Lines 2 - 6, a virtual node is added to the graph and connected to each node t_x by an edge of length zero. In line 7, ROW-TS runs an undirected shortest path tree algorithm (Dijkstra's [94]) to find the shortest-path distance between all tracks and the virtual node, updating a distance map. In lines 9-10, ROW-TS iterates through all nodes m in trajectory t_y and tests if the network Hausdorff Distance from one trajectory to itself is being computed, which is always zero. In lines 13-19 ROW-TS checks the distance map returned by the shortest-path computation to get the Hausdorff distance. In line 19, ROW-TS updates the track similarity matrix M with the newly calculated Network Hausdorff Distance between t_x and t_y and returns M in line 23.

Execution Trace of ROW-TS: We begin by iterating through each trajectory t_x in T. Let's use Trajectory t_A in Figure 3.1 as an example trajectory through this algorithm. In Line 2 we create a virtual node to represent Trajectory t_A ($A_{virtual}$ in Figure 3.3). Connecting $A_{virtual}$ to each node in Trajectory t_A (16, 17, 18, 19, 20) with 0-length edges, we can compute the single-source shortest-paths distance from $A_{virtual}$ to every other node in the graph G in Line 7. Since the edges connected to $A_{virtual}$ had



(a) 8-Primary Corridors using input GPS trajectories as candidates.

(b) 8-Primary Corridors using single roads as candidates (e.g., Park Ave)

Figure 3.5: Set of *8*-primary corridors identified from bicycle GPS trajectories and candidate corridors with varying restrictions on number of street traversals.

a length (weight) of 0, we know that each node n distance returned in the $distMap[]$ on Line 7 is actually the shortest-path distance from n to some node in A. This set of minimum distances is what we need to calculate the NHD from each trajectory t in T, which we loop through in Line 9. For each node m in trajectory t, we already have the minimum distance from m to any node in trajectory A in the $distMap[]$, and therefore need to find the maximum of those minimums as defined in Definition 4 for the NHD(t,t_A). For an example, lets use Trajectory t_B as t starting in Line 9. We iterate through each node m_B in t_B ($1_{t_B}, 2_{t_B}, 3_{t_B}, 4_{t_B}, 9_{t_B}, 10_{t_B}$) and lookup the distance from $A_{virtual}$ to m_B in $distMap[]$ (1_{t_B}:3, 2_{t_B}:3, 3_{t_B}:4, 4_{t_B}:3, 9_{t_B}:2, 10_{t_B}:2) as shown in Table 3.2. The maximum of those minimum distances is 4, which we set as the value of $NHD(t_B, t_A)$ in Line 19. This repeats for every other trajectory t in T to compute the entire row corresponding to Trajectory t_A in the trajectory similarity matrix M.

3.5 Case Study: k-Primary Corridors for Commuter Bicyclists

In Summer 2006, University of Minnesota researchers collected a variety of data to help gain a better understanding of commuter bicyclist behavior using GPS equipment and personal surveys to record bicyclist movements and behaviors [1]. One possible issue

they looked at was identifying popular transportation corridors for the Minnesota Department of Transportation to focus funds and repairs. At the time, they hand-crafted the primary corridors. Shortly after this study, the U.S. Department of Transportation began a four-year, $100 million pilot project in four communities (including Minneapolis) aimed to determine whether investing in bike and pedestrian infrastructure encouraged significant increases in public use. As a direct result of this project, the US DoT found that biking increased 50%, 7,700 fewer tons of carbon dioxide were emitted, 1.2 fewer gallons of gas was burned, and there was a $6.9 million/year reduction in health care costs [93]. In our previous work [47], we proposed a data-driven approach (the METS algorithm) to identify these primary bike corridors. While this approach identified interesting corridors, it remained computationally expensive as shown in Table 3.4. We re-ran the case-study in [47] on the Minneapolis, MN bicycle GPS dataset using the METS and ROW-TS algorithms in Table 3.4, given the input of the Hennepin County road network (57,644 nodes), 819 trajectories (shown in Figure 3.4(a) and a k of 8. As expected, the ROW-TS algorithm significantly outperformed the METS algorithm. The faster execution allowed a modification to the case-study to help identify more appropriate corridors.

Table 3.3: Descriptive statistics about the case study dataset from [1].

Dataset	Nodes	Trajectories	Average Track Length	Length σ
Bike Traces	57,644	819	72.9 nodes	23.9

Table 3.4: CPU Execution Time on Bicycle GPS trajectories in Minneapolis, MN

Dataset	METS	ROW-TS
57,644 Nodes, 819 Tracks	55,823 sec	353.9 sec

In the previous case-study, the chosen primary corridors where selected from the input trajectories (a k-medoid approach). While this ensured that chosen corridors were routes that had been biked in the real world and shared commonalities with the hand-crafted corridors chosen in Figure 3.4(b), they had problems of being too long or having too many turns and deviations. In Figure 3.5(a), we show the corridors chosen from the input trajectories. For this case-study, we generated a set of candidate

corridors based on input from geographers and had the ROW-TS algorithm compute the similarity between these candidates and the input GPS trajectory data. The candidate corridors were generated with two restrictions: 1) the corridor was a shortest-path between two nodes on the network, and 2) the corridor could only consist of n different streets, defined by their street name (e.g., a corridor traversing Park Ave and Cedar Ave would consist of 2 streets). This approximated the types of corridors our geographer collaborators were looking for while retaining the data-driven approach by comparing them to the actual GPS trajectory data. For this study, we generated 5,000 trajectories using these two constraints (source and destination nodes chosen at random) to be used as candidate corridors. The faster ROW-TS algorithm allowed us to generate a large number of candidate corridors and compute the similarity for the clustering. In Figure 3.5(b), we show the chosen clusters generated from single-road candidates (e.g., a subset of Park Ave would be a candidate).

3.6 Analytical Analysis

We show that the output of the proposed algorithm (ROW-TS) retain correctness compared to the baseline GNTS algorithm output while showing significant computational speedup.

Lemma 1 *The proposed baseline GNTS algorithm is correct and complete.*

PROOF. The GNTS algorithm is baseline in that it explicitly computes exactly what Definition 4 defines, namely the network distance between every pair of nodes in two given trajectories, selecting the minimum distances for each set of nodes. This algorithm uses enumeration of all pairs of trajectories, and for each pair, the enumeration of all pairs of nodes within the trajectories. Finally it sums the minimum distances between each pair of nodes to create the distance matrix. As every possible pairing is examined independently (separate distance calculations for each), and no modifications to the graph are done, each returned similarity is both correct and complete.

Lemma 2 *The proposed Matrix-Element Trajectory Similarity (METS) algorithm gives the same solution as the baseline GNTS algorithm.*

PROOF. Consider trajectories t_i with nodes x_i ... $x_{|t_i|}$ and t_j with nodes y_i ... $y_{|t_j|}$. A naive method calculates the minimum distance d_{naive} between t_i and t_j by selecting the shortest path with minimum weight from the set of paths P_{naive} between x_i and y_j, $\forall x_i \in t_i$, $\forall y_j \in t_j$. Consider further that a virtual node v is connected to all $x_i \in t_i$ by edges of weight 0 and not connected to any other node. METS calculates the minimum distance d_{METS} between t_i and t_j by selecting the shortest path with minimum weight from the set of paths P_{METS} between v and y_j, $\forall y_j \in t_i$. The correctness of d_{METS} may be argued in a manner similar to Dijkstra's algorithm [94] where each node $y_j \in t_j$ is connected to v via the nearest node $x_i \in t_i$. $d_{METS} = d_{naive}$ because the shortest path with minimum weight in P_{naive} will have the same weight as the shortest path with minimum weight in P_{METS}. This is because v is connected to all $x_i \in t_i$ by edges of weight 0 (all x_i's are considered) and all shortest paths between v and y_j, $\forall y_j \in t_j$ are calculated. Thus, METS gives the same solution as GNTS.

The METS algorithm retains its correctness (matching exactly with the GNTS algorithm's solution) while improving computational scalability by taking advantage of the proposed trajectory similarity metric. By definition, the metric is looking for the shortest-path from each node in one trajectory (e.g., Track D) to *any* node in the other trajectory (e.g., Track A). Therefore, the baseline solution computes Dijkstra from each node in Track D to all the nodes in Track A, returning the smallest distance. However, this can be improved by reversing the comparison. We insert a virtual node connected with edges to every node in Track A. We then compute Dijkstra from the virtual node connected to Track A to all the nodes in Track D, giving us, by the definition of Dijkstra algorithm [94], the shortest distances from each node in the destination nodes (each of the nodes in Track D), to the virtual (source) node. Since our source node is connected to each node in Track A with a length of 0, the distance from Track D nodes to any node in Track A is the same as the distance from Track D nodes to the virtual node. This gives us the same distance values used in the baseline GNTS algorithm, retaining correctness and completeness.

Lemma 3 *The proposed Row-Vector Track Similarity (ROW-TS) algorithm is correct.*

PROOF. By Definition 4, $NHD(t_B, t_A)$ is correct and it uses the shortest-path distances from each node in t_B to the respective closest node in t_A. When ROW-TS

groups all nodes in t_A as a super-source node ($A_{virtual}$) with a cost of 0, a label-setting shortest-path algorithm guarantees correctness of the shortest-distance labels from all nodes in t_B to the super-source node $A_{virtual}$ [94].

Table 3.5: Notation used in this chapter.

Notation	**Explanation**
V	Set of all Vertices in the Graph
T	Set of all Trajectories
n	Average number of nodes in a trajectory (V in worst-case)
S or $V \lg V$	Runtime of chosen shortest path algorithm
M	Track Similarity Matrix

3.6.1 Cost Analysis

Using the notation in Table 3.5, we show the costs of each trajectory similarity algorithm in Table 3.6. The baseline algorithm, GNTS, and METS require a polynomial number of distance computations, whereas the ROW-TS algorithm requires a linear amount in the number of input trajectories, resulting in a massive computational savings. It is this *repeated* expense of this network distance computation that we are reducing in this chapter. The algorithm METS results in $O(|T|^2)$ shortest path distance computations to calculate the Track Similarity Matrix, whereas GNTS does a distance calculation for each node in a given trajectory $O(|T|^2 v)$. With our restructured algorithm, we reduce the number of distance calculations to $O(|T|)$ in the ROW-TS algorithm, which significantly reduces computation time, as we will see in the experimental validation section.

Table 3.6: Asymptotic Complexity of Track Similarity Algorithms

	GNTS-B	GNTS-P	METS	ROW-TS
Complexity	T^2n^2S	T^2n^2	$T^2S + T^2n$	$TS + T^2n$

We differentiate between average case complexity and worst-case complexity in Table 3.6. In the average case, we use v to represent the average length (number of nodes) in any given trajectory in the dataset. In most cases, this number is quite small

compared to the total number of vertices V in the graph G. In the worst-case, each trajectory could consist of every vertices (or close to) in the graph, in which case we use V in place of v. In addition, note that the notation in Table 3.5 assumes the shortest-paths distance computation has a overall complexity of $|V^2|$, which we use in the worst-case complexity. While this is algorithm dependent, it provides a guide to the number of times this explicit cost D is invoked as shown in the average-case complexity.

3.7 Experimental Evaluation

In this section, we explain our experimental design for investigating a number of computational questions related to the APNTS problem and our proposed solution. We begin with the overall experimental design, followed by an explanation of the explored parameters, and finally the experimental validation.

3.7.1 Experimental Goals

We explored a number of computational questions related to the scalability of the algorithms we proposed. Our intention was to examine how varying certain key parameters would affect the runtime of the GNTS, METS, and ROW-TS algorithms. Each question was run on a small dataset and a large dataset for each algorithm. The questions we explored were:

- How does the number of *trajectories* affect the computational cost of each algorithm?
- What is the impact of the *number of nodes* in the road network on the computational cost of the various trajectory similarity algorithms?
- Does the *trajectory length* significantly affect the runtime of each algorithm?

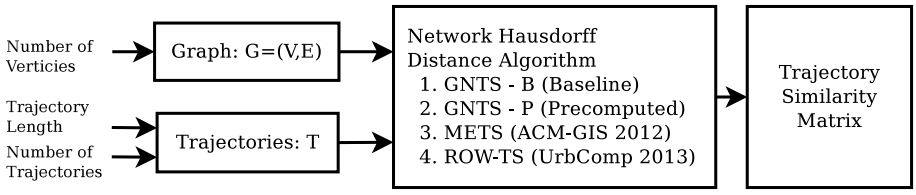


Figure 3.6: Experiment Design

3.7.2 Experimental Design

The experiments in this section were performed on synthetic datasets. We generated the underlying roadmap (graph), along with suitable trajectories. To generate the roadmap, we created a generator that required a number of nodes as input, and then generated a grid-like planar graph, similar to urban road networks. For the trajectories, we chose two random nodes on the graph and found the shortest-path, using that as a trajectory. We did not attempt to simulate how bikers would move about our networks, as we have real data (in the previous section) to test on. Here we are simply testing for scalability. These experiments were carried out on a Quad-core 2.4 Ghz Intel Core Duo 2 Ubuntu Linux machine with 8 GB of RAM. All of the algorithms were coded and run with Java SE 1.6. Each test measured CPU execution time and was run 10 times for each parameter value, then averaged for a final score shown in the plots. Our intention with these experiments was to vary key input parameters to measure how they affect the performance of our proposed work ROW-TS using the experiment design in Figure 3.6. We will be comparing against related work with the following algorithms:

Graph-Node Track Similarity (GNTS - Baseline): The baseline algorithm to compute the Network Hausdorff Distance Track Similarity Matrix M computes the shortest-path distance between each pair of nodes within each pair of trajectories, choosing the maximum of the values of this set.

Matrix-Element Track Similarity (METS [47]): In our previous work [47], we proposed a correct but computationally expensive algorithm for clustering network trajectories on road networks using Network Hausdorff Distance for identifying new bicycle corridors through a city to facilitate safe and efficient bicycle travel. This algorithm used a virtual node to reduce the number of shortest-path invocations to one per cell in the trajectory similarity matrix. It proved to be faster than the baseline, but remained computationally prohibitive for large datasets.

3.7.3 Experimental Results

How does the number of *trajectories* affect the computational cost of each algorithm? In the first experiment, shown in Figure 3.7(a), we created a road network with 500 nodes, an average trajectory length of 50 nodes. The x-axis shows the number

of trajectories given as input, the y-axis shows the runtime in seconds on a **logarithmic** scale. The figure shows all three algorithms varying over the given numbers of trajectories, and as is clear, GNTS and METS quickly become prohibitive even with this small dataset size. ROW-TS grows slowly and appears to run at least two orders of magnitude faster. In Figure 3.7(d) we increased the number of trajectories to be tested, withdrawing GNTS due to prohibitive computation time. METS was also computationally prohibitive after an input of 4,000 trajectories, while ROW-TS continued to be reasonable (under 100 seconds) past 10,000 input trajectories.

Does the *trajectory length* significantly affect the runtime of each algorithm? We ran this experiment with 500 nodes and 500 trajectories at a trajectory length shown on the x axis. METS and ROW-TS are not significantly affected by the smaller dataset. Moving to the larger dataset in Figure 3.7(f) of 1,000 nodes and 500 tracks with longer trajectory lengths on the x axis, METS is not significantly affected, as expected because the main cost is still the repeated shortest-path computations. ROW-TS grows more quickly due to the extra bookkeeping in the algorithm (see the pseudocode in Algorithm 3).

What is the impact of the *number of nodes* in the road network on the computational cost of the various trajectory similarity algorithms? In Figure 3.7(b), it is clear that the size of the underlying graph (number of nodes as plotted on the x axis) is relevant to the runtime of each algorithm as shortest-path algorithms make up a large portion of the computational cost in each. We ran this experiment with 100 trajectories and an average trajectory length of 10. Figure 3.7(e) shows a larger dataset with the number of nodes in the network going up to 10,000. Due to the reduced number of distance calculations in ROW-TS, not only is the overall runtime differing by orders of magnitude, the growth of ROW-TS compared to GNTS is also significantly slower.

3.8 Conclusion

In this chapter, we formalized the All-Pair Network Trajectory Similarity (APNTS) problem. We proposed a novel algorithm (ROW-TS) for quickly solving this problem

and compared it to previous and related work (METS, GNTS). Calculating similarity between network trajectories has shown to be important in a number of societal applications, such as bike corridor planning and transportation management. Related work in the field have repeatedly used the Network Hausdorff Distance (NHD) measure to compute similarities, but have proposed approximation-based approaches to reduce computation time. Our previous work introduced an algorithm to compute NHD exactly and scaled to a medium-sized dataset. Our novel row-wise approach in this chapter allowed us to scale to spatial big data and compute the NHD between tens of thousands of trajectories in a relatively short time. We demonstrated the scalability of ROW-TS with experimental validation.

Algorithm 1 Graph-Node Track Similarity (GNTS - Baseline)

Input:

- Road Network $G = \{V, E\}$
- Tracks T: Set of trajectories

Output:

- Track Similarity Matrix: $M_{T \times T}$

for t_x in T **do**
  for t_y in T **do**
    if $t_x == t_y$ **then**
      $M[t_x][t_y] = 0$
    else
      $max = -\infty$
      for nodes n in t_x **do**
        $min = \infty$
        $distMap[] = Dijkstra(G, n)$
        for nodes m in t_y **do**
          if $distMap[m] \leq min$ **then**
            $min = distMap[m]$
          end if
        end for
        if $max \leq min$ **then**
          $max = min$
        end if
      end for
      $M[t_x][t_y] = max$
    end if
  end for
end for
return M

Algorithm 2 Matrix-Element Track Similarity (METS)

Input:

- Road Network $G = \{V, E\}$
- Tracks T: Set of trajectories

Output:

- Track Similarity Matrix $M_{T \times T}$

for t_x in T **do**
  for t_y in T **do**
    if $t_x == t_y$ **then**
      $M[t_y][t_x] = 0$
    else
      $vTrack = newNode()$
      $G.add(vTrack)$
      for nodes n in t_x **do**
        Create 0-length edge from $vTrack$ to n
      end for
      $distMap[] = Dijkstra(G, vTrack)$
      $G.remove(vTrack)$
      $max = -\infty$
      for nodes m in t_y **do**
        if $distMap[m] \geq max$ **then**
          $max = distMap[m]$
        end if
      end for
      $M[t_y][t_x] = max$
    end if
  end for
end for
return M

Algorithm 3 Row-Wise Track Similarity (ROW-TS)

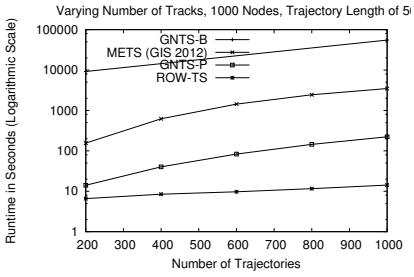
Input:

- Road Network $G = \{V, E\}$
- Tracks T: Set of trajectories

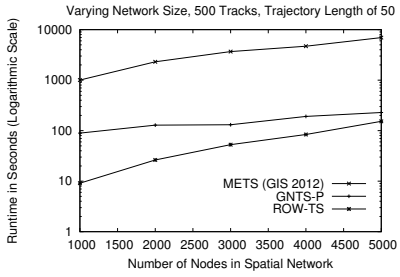
Output:

- Track Similarity Matrix M

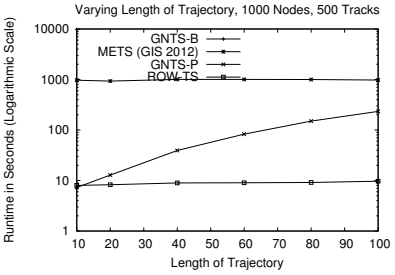
```
1: for t_x in T do
2:    vTrack = newNode()
3:    G.add(vTrack)
4:    for nodes n in t_x do
5:       Create 0-length edge from vTrack to n
6:    end for
7:    distMap[] = Dijkstra(G, vTrack)
8:    G.remove(vTrack)
9:    for t_y in T do
10:      if t_x == t_y then
11:         M[t_y][t_x] = 0
12:      else
13:         max = −∞
14:         for nodes m in t_y do
15:            if distMap[m] ≥ max then
16:               max = distMap[m]
17:            end if
18:         end for
19:         M[t_y][t_x] = max
20:      end if
21:   end for
22: end for
23: return M
```



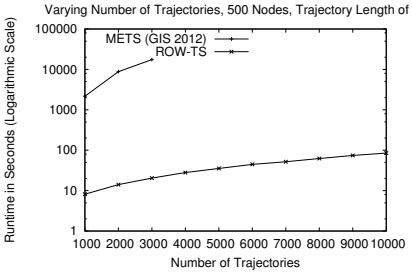
(a) Tracks (Small Dataset)



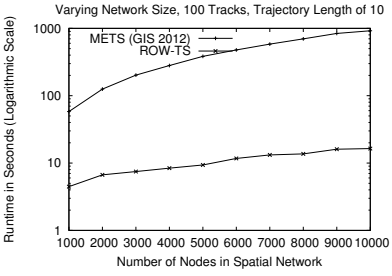
(b) Nodes (Small Dataset)



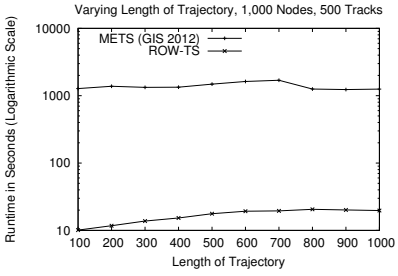
(c) Track Length (Small Dataset)



(d) Tracks (Large Dataset)



(e) Nodes (Large Dataset)



(f) Track Length (Large Dataset)

Figure 3.7: Experimental results on synthetic data. Note the y-axis is in logarithmic scale.

Chapter 4

Storage of Spatio-Temporal Networks for Advanced Routing

4.1 Introduction

To address the challenge of *storage of spatial big data*, we studied the problem of storing spatio-temporal networks in spatial database systems. Given a set of operators and a spatio-temporal network, the goal of the Storing Spatio-Temporal Networks (SSTN) problem is to produce an efficient data storage method that minimizes disk I/O access costs. Storing and accessing spatio-temporal networks is increasingly important in many societal applications such as transportation management and emergency planning. This problem is challenging due to strains on traditional adjacency list representations when storing temporal attribute values from the sizable increase in length of the time-series. Current approaches for the SSTN problem focus on orthogonal partitioning (e.g., snapshot, longitudinal, etc.), which may produce excessive I/O costs when performing traversal-based spatio-temporal network queries (e.g., route evaluation, arrival time prediction, etc) due to the desired nodes not being allocated to a common page. We propose a Lagrangian-Connectivity Partitioning (LCP) technique to efficiently store and access spatio-temporal networks that utilizes the interaction between nodes and edges in a network. Experimental evaluation using the Minneapolis, MN road network showed that LCP outperforms traditional orthogonal approaches.

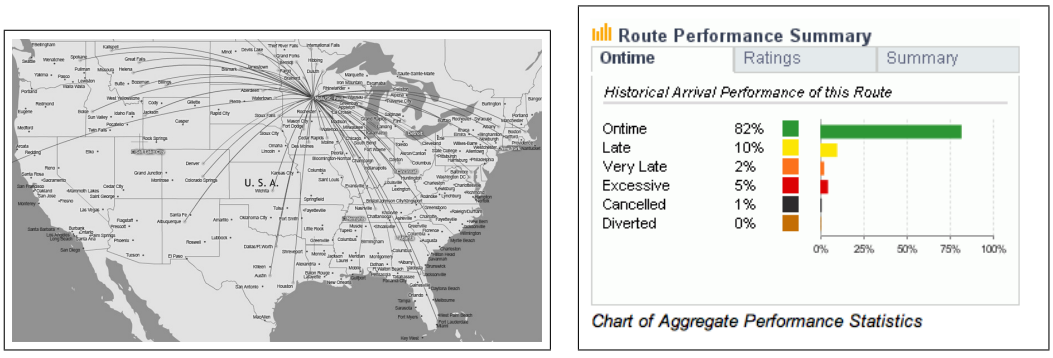
This paper proposes a new data storage method for storing spatio-temporal networks

into data files for use in database systems. Our work is motivated by the growing number and size of real-world spatio-temporal networks.

4.1.1 Motivation

Analyzing movement in spatio-temporal networks is important in many societal applications such as transportation, distribution of electricity and gas, and evacuation route planning. The ability to efficiently store, process and analyze spatio-temporal networks with large time series data would provide benefit to a wide variety of applications.

Airlines connect thousands of destinations across the world through various 'routes' between airports. Maintaining accurate records of route performance is key to evaluating and ensuring timely airline service, along with analyzing potential effects from various delays. Figure 4.1 (a) shows the various routes available from MSP airport to other cities in the United States. In order to measure route characteristics, such as average delay, each flight along the route is recorded with parameters such as flight time, delay, causes, etc. This flight information creates a spatio-temporal network from these airline routes, allowing for historical average queries, such as the one shown in Figure 4.1 (b) to be answered. Other, more complexed queries, such as how delay on a particular route affects connecting flights, can also be analyzed with this data. These large spatio-temporal networks, with a high number of temporal attribute data (flight instances), can benefit from efficient secondary storage techniques.



(a) Delta Airline Routes from MSP. Courtesy: www.airlineroutemaps.com

(b) Example Route Statistics. Courtesy: www.flightstats.com

Figure 4.1: Airline travel information as a spatio-temporal network.

The U.S. natural gas pipeline network, shown in Figure 4.2 is a transmission and distribution grid for transporting natural gas across the continental United States. Underground storage is used for efficient and reliable delivery across this network. The transmission of natural gas through the network and its various storage tanks (edges and nodes with dynamic gas levels) is monitored to ensure adequate support for short-term peaking and volatile swing demands for gas that occur on a daily and even hourly basis. Storing and optimizing the transport of natural gas is a large business and can benefit from the analysis of large spatio-temporal networks [5, 99].

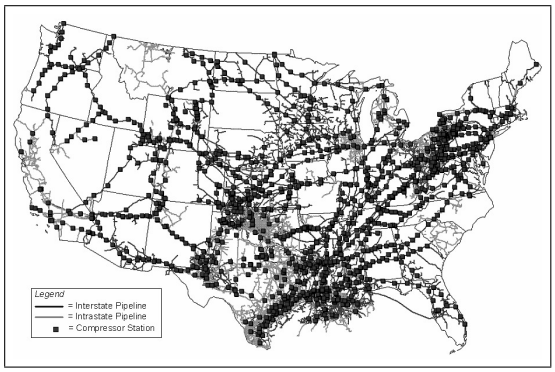


Figure 4.2: The U.S. natural gas pipeline network. [5]

The Federal Highway Administration [3] is recording traffic data of major roads and highways using sensors such as Loop Detectors, among others, across the United States. Depending on the type of sensor, traffic levels are recorded every minute, as shown in Figure 4.3. The Mobility Monitoring Program (MMP), started in 2000 by the Texas Transportation Institute, aimed to evaluate the use of sensors for traffic information around the United States. By 2003, MMP was receiving traffic sensor data from over 30 cities and 3,000 miles of highway, with sensor readings occurring roughly every 30 seconds. This data is then recorded 24 hours a day, 365 days a year, resulting in millions of time steps per year for each sensor. MMP published a report citing the need for processing and storage of historical traffic data, and how it may benefit traffic management [37].

In this paper, we consider storage and access of STN data used by traversal-based road transportation applications. These applications pose queries on STNs such as shortest path and route evaluation based on traffic levels. A route evaluation query

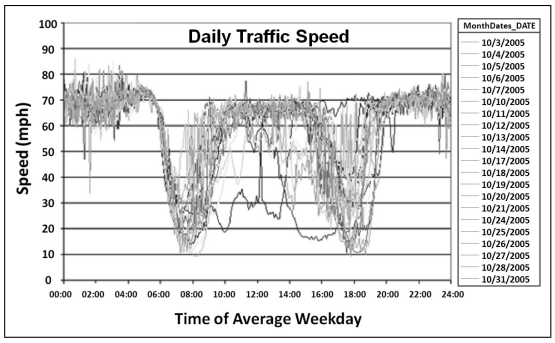


Figure 4.3: Traffic speed measurements over 30 days on a portion of highway. Courtesy: [3]

calculates travel time along a given route and start time, motivated by 'commuter' movement where a person headed to work at 8:00 am has a few different known routes and is looking for the shortest travel time. Companies such as NAVTEQ [70] are beginning to provide traffic information for road networks where travel-time for an edge is specified for all distinct five minute time-intervals of a week. However, industry is relying on an aggregation of these spatio-temporal network datasets, specifically using lossy compression, turning the recorded traffic information into "speed profiles", in order to reduce the magnitude of the data. These profiles then are used to represent a subset of road segments. Interesting events (e.g., interesting or anomalous traffic patterns) may be missed due to this lossy compression and while useful for some applications, it may be desirable to capture and utilize the full granularity of the data as it is recorded.

4.1.2 Spatio-Temporal Networks (STN)

A spatio-temporal network (STN) can be represented as a spatial graph with temporal attributes. Spatial elements of the graph are a finite set of nodes and a finite set of edges connecting nodes. Temporal attributes are represented by discrete time steps, as shown as a snapshot series in Figure 4.4. In the figure, time steps 1, 2, 3, and 4 illustrate the progression of time in the network, and the corresponding effect on edge attribute values. For example, at Time=1, edge AC has a value of 2. In the next time step, Time=2, the edge value decreases to 1, indicating a reduced travel cost for the edge. Consider a car traveling from node A to node D starting at time step 1. As

the car traverses across AC, it takes 2 time steps to reach C. Once time progresses, travel time edge attributes may change, note that the travel time edge attribute of AC changes from time step 1 to 2.

Table 4.1 lists the operators defined for spatio-temporal networks as described in [2]. This paper mainly focuses on Lagrangian movement queries, such as route evaluation.

Table 4.1: Access Operators for Spatio-Temporal Networks from [2]

	atTime	**atAllTime =**
Node	getNode(node,time)	getNode(node)
Edge	getEdge(node1,node2,time)	getEdge(node1,node2)
Route	getRoute(node1,node2,time)	getRoute(node1,node2)
Evaluate Route	evalRoute(route,time)	evalRoute(route)
Graph	getGraph(time)	getGraph()

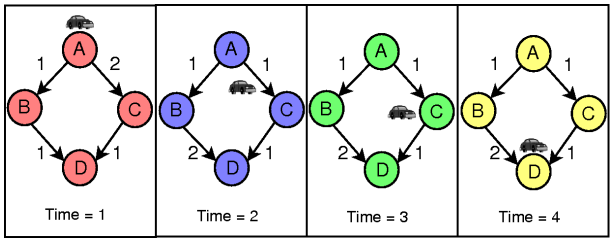


Figure 4.4: Snapshot model of a spatio-temporal network

Node Retrieval: In Figure 4.4, every node in every time step has a boolean attribute representing whether the car is currently at that node. Calling $getNode(C)$ will return a series of 4 booleans indicating whether the car is present at the corresponding time step, e.g., (False, False, True, False). Calling $getNode(C, 1)$, thereby indicating a specific time instant, returning a boolean with the attribute at that moment, in this case, False.

Edge Retrieval: In Figure 4.4, each edge has a series of scalar attribute values representing traversal time corresponding to each time instant. Calling $getEdge(A, C)$ returns the traversal attribute series of edge AC, e.g., (2,1,1,1). Calling $getEdge(A, C, 1)$ returns the traversal attribute of edge AC at time instant 1, e.g., (2).

Route Evaluation Operation: As mentioned above, industry is beginning to offer route evaluation services motivated by commuter's desire to check morning traffic along favorite routes. Given a route $A \rightarrow C \rightarrow D$, starting at time step 1, $evalRoute(ACD, 1)$, this operation will retrieve traversal time based on the temporal edge properties from $A \rightarrow C$ at time 1, and $C \rightarrow D$ at time step 3. If no starting time is given, $evalRoute(ACD)$ returns a series of traversal times for which each time instant is the starting time, e.g., (3,2,2,2).

Since we use the route evaluation query extensively in our experiments, we provide the pseudocode in Algorithm 4. Line (1) sets the original starting time, which is useful when calculating total trip time. Line (2) iterates through each node in the given route, Line (3) gets the desired edge with the *getEdge* operator, defined as the connection between a node and its next connection in route N. Finally, the current time value is updated with the travel time taken when moving across edge E in Line (4) and finally the total travel time is calculated in Line (6).

Algorithm 4 Pseudocode for Route Evaluation

Inputs:
- R: A sequence of nodes and edges
- time: Departure time

Outputs:
- Travel Time

evalRoute(R,time)

```
1: startTime = time
2: for each edge in R do
3:     E = getEdge(edge.srcNode, edge.sinkNode, time)
4:     time = time + travel time of E
5: end for
6: return  time - startTime
```

Route Retrieval: This operation represents the shortest path traversed between nodes, ignoring travel time if no start time is given. However, if a start time is given, a time dependent shortest path will be found. The details of these and other routing queries, which are out of the scope of this paper, can be found in [2].

4.1.3 Problem Statement

The problem of storing spatio-temporal networks (SSTN) can be formalized as follows: Given a spatio-temporal network and a set of spatio-temporal operations– find a storage

scheme that minimizes the I/O costs of operations. The input to this problem is a spatio-temporal network, (e.g., Figure 4.4) and a set of operations, listed in Table 4.1. We formally define the Storage of Spatio-Temporal Network (SSTN) problem as follows:

Input:

- A spatio-temporal network S
- A set of operations O

Output:

- Data file containing S stored across data pages

Objective:

- Minimize data page access for operations in O

Constraints:

- S is too large for storage in main memory.
- Preserve temporal edge attribute information.

Due to the increasing size of spatio-temporal network datasets, potentially containing hundreds of thousands of nodes and millions of time steps, we assume that main memory cannot handle these networks. Therefore we focus on secondary storage techniques of STNs for database systems. A secondary storage method for database systems is composed of a data file (consisting of data pages) and an indexing method. A data file, the output of a storage method, consists of a partitioned STN across a set of data pages. For example, a storage scheme may assign each snapshot in Figure 4.4 to a different data page. A secondary index can be built on the data file. For example, an unclustered B+tree [100] can be used to identify the data record needed for a query given keys like (node-id, time) or (edge-id, time). Once a data page is retrieved, the page may be reused on subsequent data record retrievals if they happen to collocate that data page. An efficient output of the SSTN problem should reduce data page retrieval for the operations in Table 4.1 through this collocation of relevant data.

In a database environment, the I/O cost to answer queries is determined by the number of pages which are transferred between disks and main memory. If topologically related nodes can be stored physically into the same data page, the retrieval of data pages is reduced resulting in lower disk I/O. Thus, data partitioning plays a crucial role

in decreasing the I/O cost of an access method. An important observation is that as the storage space decreases, the I/O cost is also reduced due to fewer numbers of pages with the same data.

Challenges: Storing spatio-temporal network datasets is not a straightforward task due to the complexity of incorporating temporal data into the network and the careful analysis required to reduce the disk I/O. In spatio-temporal network database systems, the accessibility of data records is constrained by network topology as well as temporal access patterns. One of the keys to improving storage methods for spatio-temporal network datasets is understanding and capturing the space-time interaction that occurs between nodes and edges in a network. Developing partitioning strategies that account for this interaction is crucial for improving the efficiency of storing and accessing spatio-temporal networks.

4.1.4 Related Work and Limitations

Current methods for storing spatio-temporal data have focused on orthogonal partitioning, e.g., snapshot and longitudinal partitioning. In essence, the data is segmented based on some temporal aspect. For example, snapshot partitioning stores data based on grouping data of the same time step together, whereas longitudinal partitioning groups data based on the object (e.g., a node) and it's entire time series. Spatio-temporal operators such as route evaluation do not follow orthogonal partitions and therefore may benefit from a different kind of partitioning for storage.

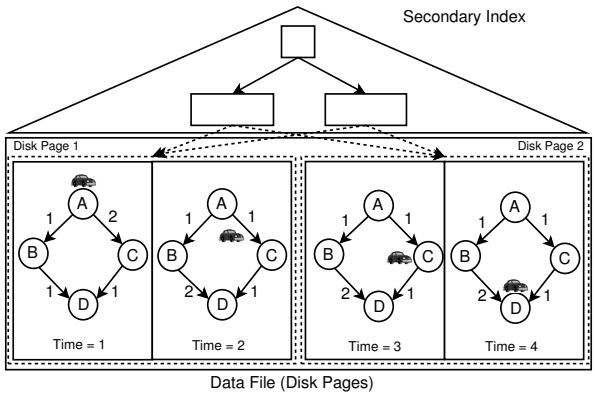


Figure 4.5: Snapshot storage of a STN

Snapshot Partitioning for spatio-temporal network storage can be represented by a snapshot graph, such as Figure 4.4. Snapshot storage techniques partition data into pages using geometry [101, 102] or connectivity [103] methods. Figure 4.5 shows a snapshot storage approximation, with the data page partitioning visualized with dashed lines and page numbering, using a small graph where multiple snapshots fit inside a data page. However producing a time stamped static graph at each time step leads to great I/O cost when executing queries such as $evalRoute(route, time)$ in Table 4.1 due to the need to frequently access data pages as the STN is traversed. In this Snapshot example, calling $evalRoute(ACD, 1)$ requires first accessing the traversal time attribute of edge AC at $t = 1$, stored on Data Page 1. Next, edge CD at $t = 3$ is needed to complete the route evaluation, stored on Data Page 2. Thus, under snapshot partitioning, the route evaluation operation has to access two different data pages.

Longitudinal Partitioning for a spatio-temporal network is based on the adjacency-list main memory storage structure used by [104, 2]. Each node is stored with its attribute information and all outgoing edges and their attribute information. This orthogonal storage solution, as shown in Figure 4.6, also suffers from the increasing disk I/O to evaluate routes in spatio-temporal networks using operators such as $evalRoute(route, time)$ in Table 4.1. This example network has a short time series compared to its graph size, allowing multiple node records (with adjacency list) to fit inside a data page. However, if the time-series length was larger, then the node record may not fit into one data page, and be split into multiple pages. This is due to the long time series being stored with each node, resulting in a small number of nodes to be stored on each data page.

As with Snapshot partitioning, when using the Longitudinal method, calling $evalRoute(ACD, 1)$ requires first accessing the traversal time attribute of edge AC at $t = 1$, stored on Data Page 1 and then accessing edge CD at $t = 3$ is needed to complete the route evaluation, stored on Data Page 2. Again, the route evaluation operation had to access two different data pages due to the node-based orthogonal partitioning. CCAM [103] did not consider STNs, however, it uses node-centric storage techniques. Therefore, if CCAM was applied to a spatio-temporal network, it may end up with a storage method resembling Longitudinal Partitioning due to the long time series characteristic of STNs causing node-centric storage to fill entire data pages with a single node's information.

The limitations of orthogonal approaches such as Snapshot and Longitudinal stem from their inability to capture spatio-temporal movement access patterns. In other sciences, spatio-temporal movement is formalized via a Lagrangian frame of reference [105] attached to a user moving through space over time. For example, evaluation of route ACD at different start times will retrieve the following subsets of edges: (AC at $t = 1$, CD at $t = 3$), (AC at $t = 2$, CD at $t = 3$), (AC at $t = 3$, CD at $t = 4$), etc. Either orthogonal approach will require a new data page access for each edge as they group entirely by either time or space. Our proposed use of a Lagrangian frame of reference in our approach intends to move toward capturing such non-orthogonal access patterns.

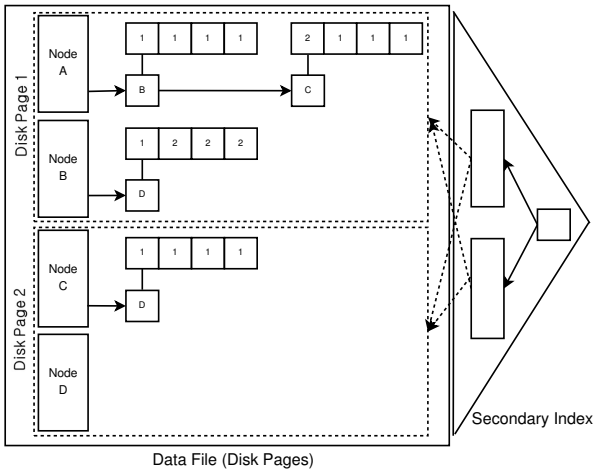


Figure 4.6: Longitudinal storage of a STN

4.1.5 Contribution

In this paper, we propose a novel storage and accessing method called Lagrangian-Connectivity Partitioning (LCP) using the following key concepts: non-orthogonal STN partitioning, a sub-node database record format and Lagrangian use of time-expanded networks for storage. Non-orthogonal STN partitioning groups temporally connected information for storage. The new sub-node database record format allows this temporally connected information, with variable time information, to be stored on a single data page. Lastly, LCP partitions data using a Lagrangian point of reference using a time-expanded graph representation of data, allowing graph partitioning to divide the

network into spatio-temporal groups and therefore attempts to capture "movement" through a spatio-temporal network. The result is a stored STN in a data file that requires less disk I/O needed for the spatio-temporal operators listed in Table 4.1.

In summary, our contributions are as follows:

- Proposed a Lagrangian-based storage and access method for spatio-temporal networks.
- Proposed a cost model to estimate data I/O cost for Lagrangian queries on stored spatio-temporal networks.
- Experimentally evaluated proposed storage method and cost model against traditional approaches.

4.1.6 Scope and Outline

We propose a new method for storing spatio-temporal networks into a data file with efficient data clustering for spatio-temporal operators. Secondary indexing techniques are not considered, as they can be applied on top of the data file.

This paper focuses on route evaluation operations on spatio-temporal networks. For simplicity of discussion, it does not examine compression techniques (e.g., sharing time-series among edges, aggregation of time-series, etc), workload related to shortest-path computations, choice of graph partitioning algorithms, and the infinite nature of time.

Industry is examining and implementing approaches for storing spatio-temporal networks based on orthogonal partitioning and sharing of time-series between edges and other compression techniques. These "speed profiles" are not examined in this paper, as we focus on full data storage as it is recorded from sensors. In addition, the intent of our work is not to evaluate industry choices but to explore conceptual ideas relevant to storage of spatio-temporal networks.

The paper is organized as follows. In Section 4.2 we introduce our proposed approach, followed by our cost model. Section 4.3 gives our experimental evaluation followed by related work in Section 4.4. Our concluding remarks are in Section 4.5.

4.2 Proposed Approach

We use a Lagrangian representation of a STN through a model called a time-expanded network [106] (TEN). A TEN is a spatio-temporal network model that replicates each node along the time set such that a time varying attribute is represented between replicated nodes. Figure 4.7 illustrates the spatio-temporal network displayed in Figure 4.4 as a time-expanded network.

The TEN is used as a representation of spatio-temporal connectivity of the data. It allows for partitioning of the temporal attribute data (travel time values of edges) based on Lagrangian connectivity. It also helps illustrate how orthogonal approaches to partitioning temporal attribute data inefficiently for Lagrangian queries. The time-expanded network used for the partitioning decisions is not stored on disk, only the node, edge and temporal attribute data coming from the input STN are stored.

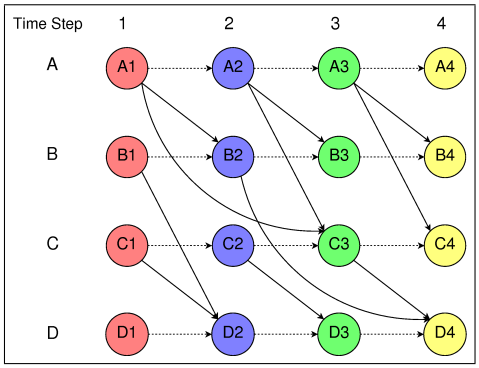
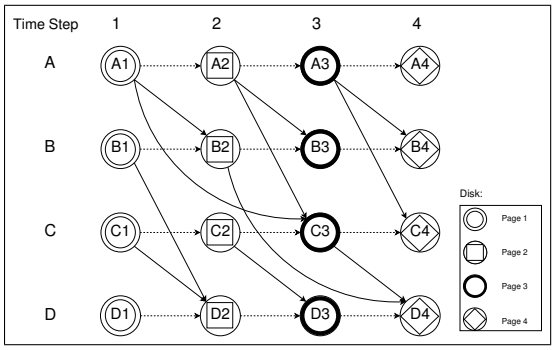


Figure 4.7: STN as a time-expanded network

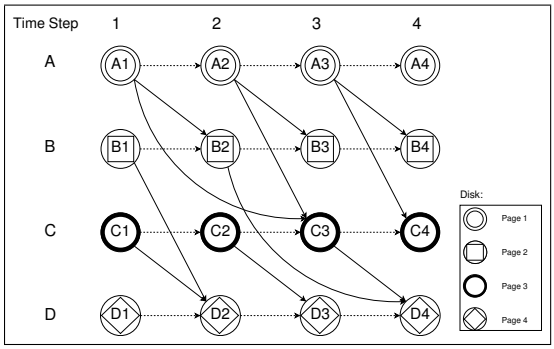
Our approach, Lagrangian-Connectivity Partitioning (LCP), utilizes a time-expanded graph to capture non-orthogonal access patterns of route evaluation operations along with a novel data record based on $sub-nodes$. This section details each of our main contributions and provides a cost model for estimating disk I/O based on STN operators (e.g., Table 4.1).

We propose an asynchronous time series record format for storing our temporal attribute data. Data can be stored based on either synchronous or asynchronous time series. Synchronous time grouping, or clustering data within a set time series, stores some number of nodes and edges with a set time interval, see Figure 4.8. Each page in

both approaches, Snapshot and Longitudinal, represents a synchronous time interval, either one time step or the entire time series in this example. For example, a page in both approaches, Snapshot or Longitudinal, represents a synchronous time interval where the Snapshot method stores its all data relating to a single time step in a record. The Longitudinal storage method stores an entire temporal attribute's time series (broken into multiple records if needed). Asynchronous time data grouping allows for storing data disjoint time intervals and yet be grouped and stored on the same data page. This model, which we refer to as *sub-nodes*, allows for storage of disjoint temporal attribute data in the same record. This is useful as, for route evaluation queries, it is seldom that temporal data is accessed orthogonally.



(a) Snapshot Partitioning



(b) Longitudinal Partitioning

Figure 4.8: Orthogonal partitioning of Spatio-Temporal Networks.

4.2.1 Lagrangian-Connectivity Partitioning

The Lagrangian-Connectivity Partitioning (LCP) method optimizes disk storage based upon traversal across spatio-temporal networks. Data is stored using the sub-node record design, allowing for non-orthogonal temporal information to be stored on a data page.

By representing a spatio-temporal network as a modified time-expanded graph, focusing on Lagrangian connections between nodes (movement edges), a min-cut graph partitioning [107] algorithm creates partitions clustering nodes by minimizing the cuts of these movement edges. This results in LCP collocating connected spatio-temporal nodes together on data pages, stored as sub-node records. We propose LCP will result in more efficient I/O when using STN operators from Table 4.1 or queries composed of them.

To illustrate, we call the route evaluation operation *evalRoute(ACD,1)* on each of the partitioning methods. With orthogonal partitioning, Snapshot or Longitudinal, whenever an edge is traversed, (e.g., $A1$ to $C3$ and $C3$ to $D4$), a disk I/O is needed to retrieve the data page containing the record for the next node. However, with Lagrangian-Connectivity Partitioning, traversing from node $A1$ to $C3$ and then $C3$ to $D4$ requires only one data page as all relevant sub-node records are collocated on the same data page. The pseudocode shown in Algorithm 5.

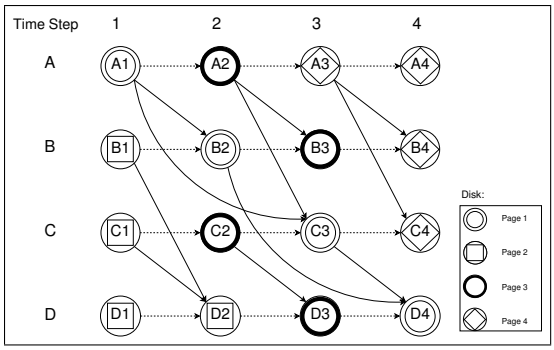


Figure 4.9: Lagrangian-Connectivity Partitioning

The input for the pseudocode is a spatio-temporal network consisting of nodes, edges and time values for each edge along with the physical page size for storage on disk. The output is a data file consisting of the data pages, containing records of node and

Algorithm 5 Pseudocode for the LCP Method

Inputs:

- A set of nodes V
- A set of edges E
- A set of travel times T
- P: size of data page

Outputs:

- Data Pages written to disk

LCP

```
1: numPages = estimate num of pages using V,E,T and P
2: TEN[] = create time-expanded network from V,E,T
3: Part[] = run graph partitioning on edges in TEN using numPages
4: for each partition in Part[] do
5:     SN[] = create sub-nodes from partition
6:     for each sub-node in SN do
7:         RID = write sub-node to a data page
8:     end for
9: end for
```

edge information. The min-cut graph partitioning algorithm used requires a predefined number of partitions; therefore Line (1) estimates the number of pages needed based on the size of the spatio-temporal network and the size of the data page. Line (2) expands the spatio-temporal network into a time-expanded network for the min-cut partitioning in Line (3). The result of the partitioning method is an array of partitions of the time-expanded network which are then converted to sub-nodes in Line (5). Iterating through these partitions with Lines (4-9), each partition is converted into a set of sub-nodes in Line (6) and written to disk in Line (8).

Figure 4.9 illustrates the results of LCP applied to the same input STN as in the other orthogonal partitioning methods. A min-cut partitioning was run on the modified time-expanded graph (wait edges are removed to emphasize Lagrangian connectivity) and then stored as sub-nodes on four data pages. For efficiency, we used a bulk load operation, which sorts these statements along the block number and inserts them into data pages. Physically, sub-node data records are stored in data pages and a B+tree index is created to support retrieve operations.

4.2.2 Cost Model

Traditional spatial networks use a connectivity ratio to measure predicted disk I/O [103]. We extend this connectivity ratio to formulate a spatio-temporal measurement we call

LRatio (Lagrangian connectivity Ratio). LRatio measures the connectivity along time and space in a STN. In Equation 4.1, *Lagrangian edges* refer to edges connecting nodes through time, such as the edges displayed in a time-expanded network. This metric ignores the 'wait' edges in a time-expanded network, maximizing LRatio minimize disk I/O for STN Lagrangian operators.

$$LRatio = \frac{Total\ number\ of\ unsplit\ Lagrangian\ edges}{Total\ number\ of\ Lagrangian\ edges} \tag{4.1}$$

Intuitively, the average number of disk accesses (DA) in Equation 4.2 according to a STN route can be expressed as a function of the LRatio and the number of accessed nodes:

$$DA(route) = 1 + (1 - LRatio) * (Route\ Length - 1) \tag{4.2}$$

where *RouteLength* is the number of edges traversed in the route. To be specific, for a route with n edges, the first edge is accessed by one disk I/O. The remaining $n - 1$ edges, are accessed by $(n - 1) * (1 - LRatio)$ disk I/O because each edge has a $(1 - LRatio)$ chance to cause additional disk I/O. This approximation is reasonable because, in a time-expanded network, 1) accessing nodes causes a page fault only when the access method meets a split edge and 2) all nodes are always accessed in increasing temporal order.

4.3 Experimental Evaluation

In this section we evaluate our proposed method against traditional approaches using route evaluation queries and relevant STN operators from Table 4.1. All experiments were performed on an Intel Core 2 Duo CPU machine running Microsoft Windows XP with 4GB of RAM.

4.3.1 Experiment Setup:

Figure 4.11 gives our experimental setup. Using a Minneapolis, MN roadmap from the Minnesota Department of Transportation [6], we created and stored three data files, one generated by LCP, and the other two using orthogonal partitioning methods. These

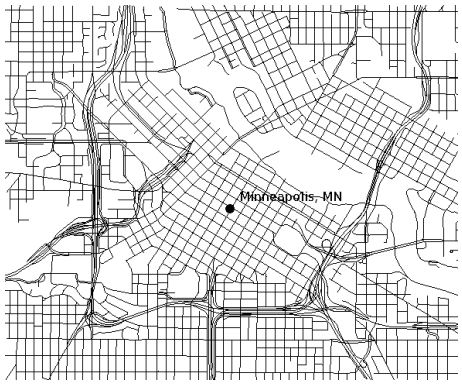


Figure 4.10: Minneapolis, MN road network [6]

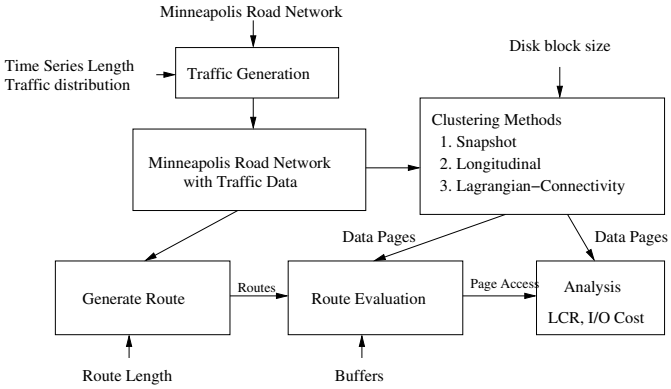


Figure 4.11: Experimental Setup

stored networks were then evaluated with a route evaluation workload, specific to each experiment, and sent for analysis.

The dataset consisted of 1,140 nodes and 3,764 edges. The edge data were classified into three types: highways, county roads, and side streets. The travel time attribute series was synthetically generated based on various Gaussian distributions for each road type and took into account activity levels over a day, e.g., rush hour. We set the length of time steps for the STN to 288 instants (5 minute slots) and generated the time-expanded network based on replicated nodes and travel times.

Route Evaluation: In our study, we focused on a popular query distinct to spatio-temporal networks. This query is an encapsulation of the operator first mentioned in Table 4.1 and returns the travel time between two nodes given a set route and start

time. This query represents real-world queries calculating estimated travel time for a given route at a given start time.

4.3.2 LCP Approximation: ATSS

We performed experiments using three different STN storage candidates. The first two, the orthogonal Spatial and Longitudinal methods are compared against the LCP method. A fourth candidate, the aggregated time-stamped snapshot (ATSS) method, is presented here as a simple alternative to LCP, attempting non-orthogonal partitioning without the need to create a time-expanded graph.

ATSS partitions the network based on static network connectivity using CCAM [103] and then the time-series information is divided into temporal chunks, similar to the Snapshot method, only with multiple sequential time instants instead of one time instant. The temporal information for each node is segmented based on this time interval length and the proposed non-orthogonal record format is used to store the nodes with their temporal subsets.

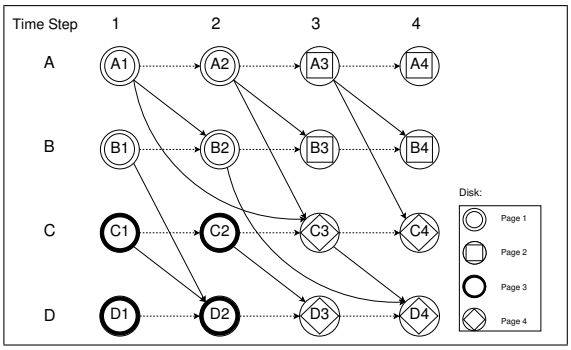


Figure 4.12: Aggregated Time-Stamped Snapshot

The ATSS method can be seen as a trade-off between Snapshot and Longitudinal partitioning. Since the network is sliced with respect to time intervals, we need to define a parameter as time intervals to slice the graph, see Figure 4.12 where the time interval parameter is 2 time steps. It is difficult to determine an adequate value for the time length parameter. The strength of the aggregated time-stamped snapshot model is that it is relatively practical when the travel time is fairly uniform. The main disadvantage is that it is not possible to determine the appropriate time interval parameter value to

yield a better performance.

4.3.3 Experimental Results

In our experiments, we compare the LCP storage method against the Snapshot and Longitudinal approaches, and discuss results. Lastly, we compare LCP and ATSS in Experiment 4 as the only non-orthogonal partitioning methods.

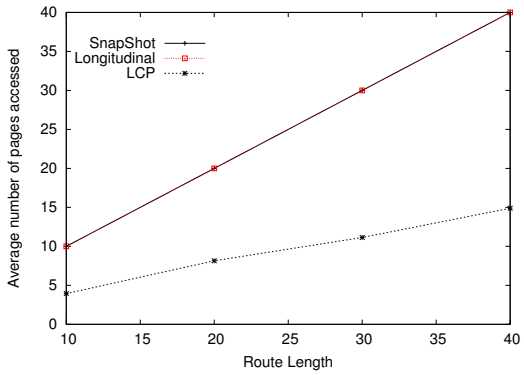
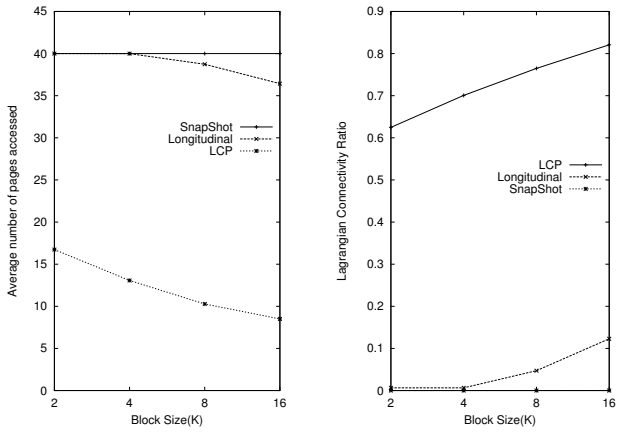


Figure 4.13: Experiment 1 - The effect of the route length. Note that Snapshot and Longitudinal are overlapping.

Experiment 1: Effect of Route Length.
To evaluate query performance, we varied route length in terms of nodes traversed along the route, and compared the number of data pages accessed by the three storage methods, Snapshot, Longitudinal and LCP. We used 1,500 randomly generated simple-path routes over the STN and varied the route length from 10 to 40 edges traversed. We used 4KB block size and one buffer cache. The number of buffers showed no effect on performance due to the progression of time and the properties of simple-paths, in that neither time steps nor nodes can be revisited.

Figure 4.13 shows the performance comparison using all three models or different route lengths. For all methods, the number of data page accesses for route evaluation queries increases along with the increase in the route length. However, Snapshot and Longitudinal partitioning perform worse in this experiment at every step. This is due to the orthogonal partitioning of data which requires accessing a new data page for every next node accessed in a path. Snapshot accesses a new data page for every node because it stores each time instant on separate pages. Longitudinal does this due to the long

time series used, therefore filling entire disk pages with a single node's attribute data. Only LCP requires less data pages accessed than nodes accessed.



(a) Data Page Access (b) LRatio

Figure 4.14: Experiment 2 - Effects of varying the size of data pages

Experiment 2: Size of Data Page.

This experiment evaluates the effect of varying the size of the data page on data page access. If more information can stored on each data page, this potentially increases the disk access efficiency of the orthogonal storage methods if temporally connected information can be collocated on a single page, as LCP is designed to do. Figure 4.14 shows average I/O costs and the LRatio as we varied the block size from 2 KB to 16 KB. We observed an improvement in disk I/O efficiency for LCP, as expected, due to storing more temporally connected information on a data page. With the Longitudinal partitioning model, increasing the size of the disk block allowed more than one partition to fit on a data page, resulting in an improvement in data page access and LRatio measurement.

Experiment 3: Cost Model Evaluation.

We measured the accuracy of the LRatio cost model from Section 4.2.2. To measure LRatio, we read all data pages stored in a data file for each storage method and count split and unsplit movement edges. Our experimental results matched the predicted I/O cost from LRatio within 10% (Figure 4.15). This gives some credence to the LRatio metric as a cost-estimator for STN operators and we therefore use the LRatio measurement

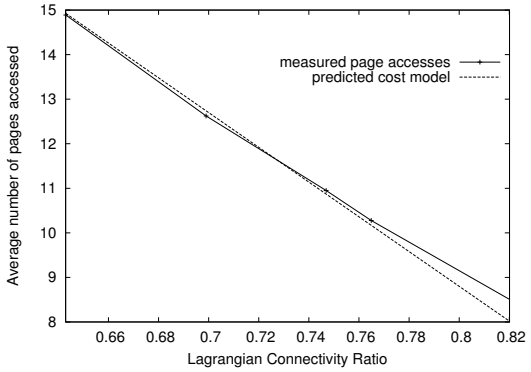


Figure 4.15: Experiment 3 - Accuracy of the cost model in a Lagrangian path evaluation

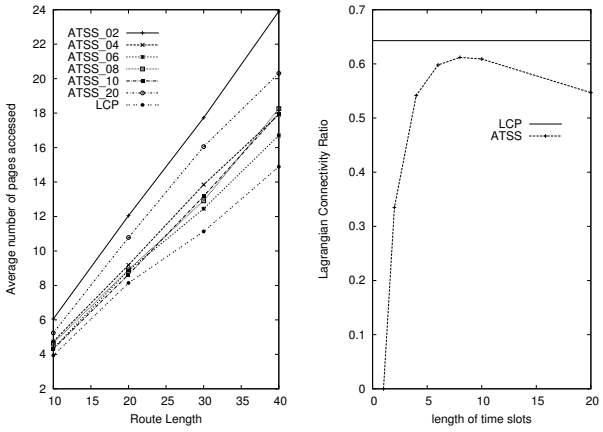
as a performance metric in our experiments.

Experiment 4: Evaluating ATSS.

This experiment compares the LCP and ATSS methods. The intent of this experiment is to evaluate the effects of different kinds of non-orthogonal partitioning. Figure 4.16(a) shows the ATSS method compared to the LCP method. ATSS was run with six different time interval parameter values, which range from a interval length of 2 time steps to 20. Note how if the time window is too long or too short, performance degrades. Also note that LCP consistently outperforms ATSS without any need of 'parameter tuning'. The result is also visualized in Figure 4.16(b) using the LRatio cost model. The LRatio metric varies depending on the time interval parameter selected for the ATSS partitioning. This demonstrates how ATSS is sensitive to parameter tuning and the composition of the dataset.

Experiment 5: Changing the Spatio-Temporal Network.

In this experiment, the spatial connectivity and temporal length were varied on a synthetic dataset in order to gauge robustness. The effect of the length of time steps on I/O costs is shown in Figure 4.17(b). We used different time series lengths and therefore changed the number of pages used based on the increased data. As can be seen, the time series length does not affect performance. This property is desirable when a user stores large time steps incremental. That is, we can slice large time steps and store each section individually. In Figure 4.17(a), we varied the number of connections between each node, referred to as the edge/node ratio or connectivity ratio. A typical road network has low connectivity, between 2 and 3 edges per node. Preliminary experiments



(a) LCP vs ATSS (b) LRatio of ATSS

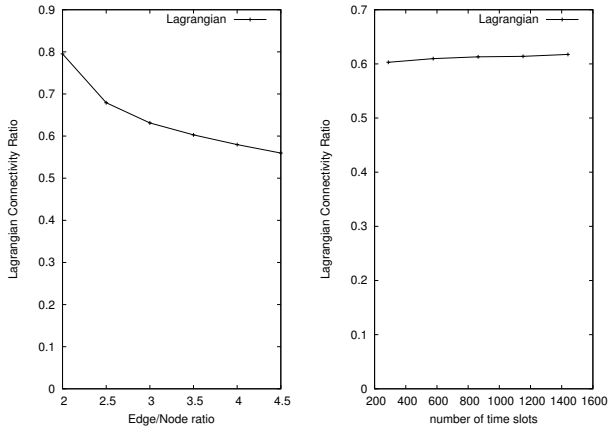
Figure 4.16: Experiment 4 - Comparison of Non-Orthogonal Methods

show LRatio may suffer as the connectivity ratio increases.

4.4 Related Work

A broader set of related work for the SSTN problem is summarized by Table 4.2. Much work has been done in the geometric space, indexes for both space and space-time have been popular for years [108, 102, 101, 104, 109, 110]. However, geometrical approaches, (e.g., Euclidean distance), may not be ideal for spatial networks. CCAM [103] demonstrated that geometric partitioning less efficient when dealing with networks and that topological connectivity may be better. In connectivity based partitioning, orthogonal partitioning methods, such as the Longitudinal or Snapshot method, could capture network connectivity based on either space or time independently. For instance, CCAM considered only spatial connectivity when partitioning and storing spatial networks, and therefore would likely emulate the Longitudinal method when given a spatio-temporal network, especially one with large time series such as the ones mentioned in this paper.

Both Longitudinal and Snapshot method store data as synchronous mode, that is, either space or time is fixed and data are arranged sequentially. In CCAM [103], temporal information is not considered as it is a node-based partitioning and a naive extension would be to use synchronous time grouping for each node, either with some time interval



(a) edge/node ratio (b) time steps length

Figure 4.17: Experiment 5 - Changing the Spatio-Temporal Network

Table 4.2: Related work for Spatio-Temporal Network

	Spatial	Spatio-Temporal
Geometric	R-Tree [101], Quad Tree [102]	MVR-Tree, TPR-Tree [108]
Topological	CCAM [103]	LCP, Snapshot, Longitudinal

or the entire time series (see Figure 4.18). The synchronous data structure, however, could not store the spatial temporal connectivity, such as Lagrangian-connectivity. Our sub-node storage design focuses on non-orthogonal spatio-temporal node storage based on Lagrangian-connectivity. This allows multiple nodes, connected through Lagrangian edges, to be on the same data page and reduce I/O cost.

Topological ordering, traversal partitioning, and graph partitioning have been used to optimize access methods of graphs [111, 112, 113, 114]. Topological ordering and traversal partitioning, however, need preprocessing to layout the graph and is slower

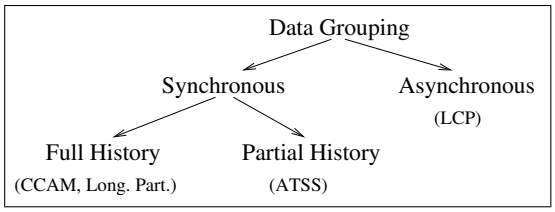


Figure 4.18: Related work in Record Formats for Time Series Storage

than graph partitioning. In graph partitioning literature [115, 116], a multi-level partitioning algorithm which balance cluster size and minimize the min-cut, is one of the more efficient methods and we used Metis [107] as multi-level partitioning way to group the Lagrangian connectivity.

4.5 Conclusions and Future Work

Spatio-temporal networks are becoming increasingly popular for a variety of important societal applications such as transportation management, fuel distribution, airline management, electrical grid usage analysis, etc. Traditional approaches for the SSTN problem have focused on orthogonal partitioning (e.g., snapshot, longitudinal, etc.) of the network, which produces significant I/O costs when performing Lagrangian movement queries (e.g., route evaluation). We proposed a Lagrangian-Connectivity Partitioning (LCP) method to efficiently store and access spatio-temporal networks that utilizes the interaction between nodes and edges in a network. We introduced a sub-node record format for storing STN data based on non-orthogonal spatio-temporal network partitioning. Experimental evaluation of LCP demonstrated significant improvements over previous work.

Our immediate future work is to expand our experiments to a real world dataset, with hundreds of thousands of nodes, edges and time steps. Another interesting experiment would evaluate the lossy compression (speed profiles) approach. Interesting comparisons would look for travel time accuracy along with evaluating interesting events in the data that may be lost through compression. Other future intentions for this work include exploring more complex queries, such as time-dependent shortest path computation, etc. Lastly, the chosen min-cut graph partitioning method was used due to its simplicity and available implementation. Other, more complex graph partitioning algorithms may be

more efficient at capturing unique characteristics of spatio-temporal networks.

Chapter 5

Conclusion and Future Work

The main contributions of this thesis addressed the challenges of defining, storing, processing, and analyzing spatial big data. In Table 5.1 we show our published works related to these topics and First, we defined spatial big data and discussed the challenges and opportunities SBD brings to spatial computing, elaborating on work we did in [11]. Second, we addressed the challenge of analyzing spatial big data with GPS trajectories, motivated by the societal application of finding bicycle corridors from urban cyclists [47, 48]. Third, we addressed the challenge of storing spatial big data in spatial database systems, specifically storing spatio-temporal networks for use in advanced routing services [45, 46]. In this chapter, we will present our preliminary work on the challenge of processing spatial big data in the future work section of the thesis.

5.1 Key Results

This section presents a summary of the major results that were produced as a part of this thesis.

- **Spatial Big Data [11, 117, 118]:** Increasingly, location-aware datasets are of a volume, variety, and velocity that exceed the capability of spatial computing technologies. Spatial Big Data examples include trajectories of cell-phones and GPS devices, vehicle engine measurements, temporally detailed road maps, etc. Spatial Big Data poses an enormous set of challenges in terms of analytics, data processing, capacity, and validation. Work in this thesis defined spatial big data and the challenges it presents for traditional computing technologies.

Table 5.1: Thesis Contributions: Spatial Big Data Analytics for Urban Informatics

Spatial Big Data	**Urban Informatics**		
	Strategic	Tactical	Operational
Long-term Forecasts (climate, demographics, economy)	SBD for Urban Planning [11, 117, 118]		
Location Traces (GPS, UAV, WAMI)		Bicycle Corridor Selection [48, 47]	
Spatio-Temporal Networks			Commuter Information Systems [46, 45, 119]

- **Network Trajectory Similarity [48]:** Given a set of trajectories on a road network, the goal of the All-Pair Network Trajectory Similarity (APNTS) problem is to calculate the similarity between all trajectories using the Network Hausdorff Distance. This problem is important for a variety of societal applications, such as facilitating greener travel via bicycle corridor identification. The APNTS problem is challenging due to the high cost of computing the exact Network Hausdorff Distance between trajectories in spatial big datasets. Previous work on the APNTS problem takes over 16 hours of computation time on a real-world dataset of bicycle GPS trajectories in Minneapolis, MN. In contrast, this paper focuses on a scalable method for the APNTS problem using the idea of row-wise computation, resulting in a computation time of less than 6 minutes on the same datasets. We provide a case study for transportation services using a data-driven approach to identify primary bicycle corridors for public transportation by leveraging emerging GPS trajectory datasets.
- **GPS Trajectory Analysis [47]:** The k-Primary Corridor problem is important due to a number of societal applications, such as city-wide bus route modification or bicycle corridor selection, among other urban development applications. For example, to plan a new bus route, one may analyze commuter trajectories to summarize primary corridors of travel, suggesting new bus routes which may minimize the aggregate walking distance of commuters. Let us consider the problem of determining primary bicycle corridors through a city to facilitate safe and

efficient bicycle travel. By selecting representative trajectories for a given group of commuters, the overall alteration to commuters routes is minimized, encouraging use. Facilitating commuter bicycle traffic has shown in the past to have numerous societal benefits, such as reduced greenhouse gas emissions and healthcare costs [93].

- **Spatio-Temporal Networks [46, 45, 119]:** Given a spatial network and its variations over time (e.g., time-varying travel times on road networks) this chapter discusses how to model, query and store spatio-temporal networks. This problem has application in several domains such as transportation networks, emergency planning, knowledge discovery from sensor data, and crime analysis. Adequately representing the temporal nature of spatial networks would potentially allow us to raise interesting questions (e.g., eco-routing, non-FIFO behavior) and find efficient solutions. In transportation networks, travelers are often interested in finding the best time to start so that they spend the least time on the road. Crime data analysts may be interested in finding temporal patterns of crimes at certain locations or the routes in the network that show significantly high crime rates. Modeling the time dependence of sensor network data would certainly improve the process of discovering patterns such as hot spots. In these application domains, it is often necessary to develop a model that captures both the time dependence of the data and the underlying connectivity of the locations. There are significant challenges in developing a model for spatio-temporal networks. The model needs to balance storage efficiency and expressive power and provide adequate support for the algorithms that process the data.

5.2 Future Directions

We organize the future directions of this thesis into two categories: (a) short-term directions and (b) long-term directions.

5.2.1 Short-term Directions

The component of handling spatial big data we did not touch upon in this thesis so far is the processing step. Recently cloud computing has become quite popular and

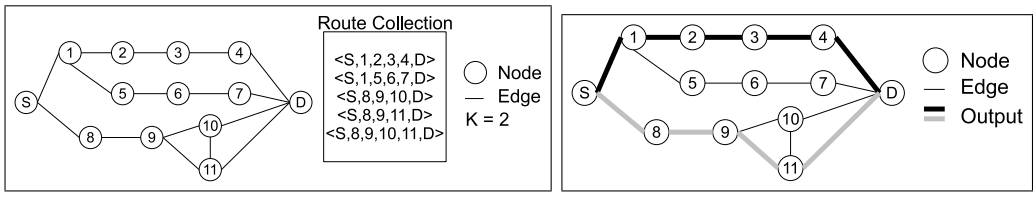
computation platforms such as MapReduce/Hadoop have reached mainstream business audiences. In this thesis we presented work on both the storage and the analysis layers of spatial big data, and we are currently working on work related to the processing layer. As GPS trajectories become more numerous (our case-study had only 819 trajectories, but new datasets are now available with tens and hundreds of thousands), more efficient and scalable processing methods are needed. We are developing a MapReduce-based version of our similarity algorithm to run on the Hadoop distributed computing system. We hope to greatly increase the number of trajectories we can use in our analysis but will need to solve key challenges related to data partitioning and algorithm parallelization.

5.2.2 Long-term Directions

Spatio-temporal networks provide a true conversion between space and time, something difficult to achieve in geometric dimensional space. Given this, we hope to move our trajectory analysis to spatio-temporal networks, where temporal information can truly be integrated with the underlying road network. This will pose significant challenges as the size of real-world spatio-temporal networks provided by companies such as NAVTEQ have millions of nodes. It will likely need to be combined with advanced processing platforms, although MapReduce has been shown to be less than ideal for graph-based iterative algorithms. Spatial Big Data offers the opportunity to address the traditional routing query using a data driven paradigm. Specifically, the ever increasing availability of GPS traces give us the opportunity to add a new dimension to route recommendation systems by using information from personal or popular favorite routes.

One difficulty with many potential route solutions is merging and grouping routes into something coherent. It is possible that many routes found by the algorithmic portfolio may be similar, with only minor deviations, as illustrated in Figure 1.1(b). Clustering can be done to find similar routes, either in terms of path chosen, or some attribute of the path, such as road type (e.g., highways, side-streets, etc). Merging and ranking the paths of routes is a difficult problem that we intend to explore. Ideally, after merging the routes, some subset of that would be merged to reduce the risk of information overload on the end-users.

We propose to leverage our work on trajectory similarity measures and trajectory clustering to return k dissimilar routes. Next we discuss a preliminary approach for



(a) Collection of potential routes between two points.

(b) Sample grouped and merged answers to provide coherent feedback and suggestions to users.

Figure 5.1: Many potential route solutions will require merging and grouping of routes, similar to trajectory similarity.

grouping routes that may be produced from a set of routing algorithms including summarization of GPS trajectories. The K-Median for Route-Collection (KMRC) problem that our proposed algorithm addresses is defined as follows: Given a spatial roadmap graph, a desired number of routes, k, and a collection of routes, R (e.g., GPS trajectories), find a subset of k routes in R that minimizes route similarity. Route similarity is defined as the number of nodes and edges that two routes r_1 and r_2 have in common. If r_1 and r_2 have no nodes or edges in common, then they would have a route similarity of zero; whereas if they had all edges in common and they were of equal length, they would have a route similarity of $|r_1|$, or the length of r_1 or r_2. Figures 5.1(a) and 5.1(b) illustrate an input and output example of KMRC respectively. The input consists of thirteen nodes, fifteen edges (with edge weights of 1 for simplicity), k = 2, indicating that two routes are desired, and the collection of routes (shown in the table in Figure 5.1(a)). The output contains two routes from the given collection of routes that minimize route similarity, $\langle S, 1, 2, 3, 4, D\rangle$ and $\langle S, 8, 9, 11, D\rangle$. Additionally, merging and grouping route candidate answers may be done for all start times. We propose to investigate novel algorithmic refinements such as seeding each subsequent time instant after the first time instant with the results of previous time instants to improve computational savings without reducing result quality. The idea is to minimize iterations by providing the algorithm with an answer that is close to the final answer and avoid redundant calculations for each time instant as would be done in a naïve algorithm.

References

[1] F.J. Harvey and K.J. Krizek. Commuter Bicyclist Behavior and Facility Disruption. Technical Report Report no. MnDOT 2007-15, University of Minnesota, 2007.

[2] Betsy George, Sangho Kim, and Shashi Shekhar. Spatio-temporal network databases and routing algorithms: A summary of results. In *SSTD*, pages 460–477, 2007.

[3] Federal High Administration, www.fhwa.dot.gov.

[4] G. Capps, O. Franzese, B. Knee, MB Lascurain, and P. Otaduy. Class-8 heavy truck duty cycle project final report. *ORNL/TM-2008/122*, 2008.

[5] EIA. "U.S. Natural Gas Pipelines". `http://goo.gl/hprcU`.

[6] Minnesota Department of Transportation, www.dot.state.mn.us/.

[7] Business Week. Spain's Santander, the City That Runs on Sensors. `http://www.businessweek.com/articles/2013-05-16/spains-santander-the-city-that-runs-on-sensors`, 2013.

[8] Y.F. Thomas, D. Richardson, and I. Cheung. *Geography and drug addiction*. Springer Verlag, 2009.

[9] P.J. Lioy and S.M. Rappaport. Exposure science and the exposome: an opportunity for coherence in the environmental health sciences. *Environmental health perspectives*, 119(11):a466, 2011.

[10] S.M. Rappaport. Implications of the exposome for exposure science. *Journal of Exposure Science and Environmental Epidemiology*, 21(1):5–9, 2010.

[11] Shashi Shekhar, Michael R Evans, Viswanath Gunturi, and KwangSoo Yang. Spatial big-data challenges intersecting mobility and cloud computing. *NSF Workshop on Social Networks and Mobility in the Cloud*, page 35, 2012.

[12] The Independent. New Songdo City: Atlantis of the Far East. `http://www.independent.co.uk/news/world/asia/new-songdo-city-atlantis-of-the-far-east-1712252.html`, 2009.

[13] Popular Science. Planned Portuguese Eco-City Is Controlled By A Central Computer Brain. `http://www.popsci.com/technology/article/2010-10/portuguese-smart-city-emulates-biology-using-computer-brain-centralized-control`, 2010.

[14] Marcus Foth, Jaz Hee-jeong Choi, and Christine Satchell. Urban informatics. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, pages 1–8. ACM, 2011.

[15] Eric Paulos, RJ Honicky, and Ben Hooker. Citizen science: Enabling participatory urbanism. *Handbook of Research on Urban Informatics*, pages 414–436, 2008.

[16] Tim Kindberg, Matthew Chalmers, and Eric Paulos. Guest editors' introduction: Urban computing. *Pervasive Computing, IEEE*, 6(3):18–20, 2007.

[17] Mark Weiser. Hot topics-ubiquitous computing. *Computer*, 26(10):71–72, 1993.

[18] Marcus Foth. *Handbook of research on urban informatics: The practice and promise of the real-time city*. Information Science Reference, IGI Global, 2009.

[19] J. Manyika et al. Big data: The next frontier for innovation, competition and productivity. *McKinsey Global Institute, May*, 2011.

[20] Garmin. `http://www.garmin.com/us/`.

[21] Wikipedia. Usage-based insurance — wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Usage-based_insurance&oldid=464698702`, 2011. [Online; accessed 15-December-2011].

[22] TomTom. TomTom GPS Navigation. `http://www.tomtom.com/`, 2011.

[23] Google Maps. `http://maps.google.com`.

[24] Yu Zheng and Xiaofang Zhou. *Computing with Spatial Trajectories*. Springer Publishing Company, Incorporated, 1st edition, 2011.

[25] Ralf Hartmut Güting, Victor Teixeira De Almeida, and Zhiming Ding. Modeling and querying moving objects in networks. *The VLDB Journal*, 15(2):165–190, 2006.

[26] Brendan Morris and Mohan Trivedi. Learning trajectory patterns by clustering: Experimental studies and comparative evaluation. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 312–319. IEEE, 2009.

[27] Daniel P Huttenlocher, Klara Kedem, and Jon M Kleinberg. On dynamic voronoi diagrams and the minimum hausdorff distance for point sets under euclidean motion in the plane. In *Proceedings of the eighth annual symposium on Computational geometry*, pages 110–119. ACM, 1992.

[28] Jeff Henrikson. Completeness and total boundedness of the hausdorff metric. *MIT Undergraduate Journal of Mathematics*, 1:69–79, 1999.

[29] Sarana Nutanong, Edwin H Jacox, and Hanan Samet. An incremental hausdorff distance calculation algorithm. *Proceedings of the VLDB Endowment*, 4(8):506–517, 2011.

[30] Jinyang Chen, Rangding Wang, Liangxu Liu, and Jiatao Song. Clustering of trajectories based on hausdorff distance. In *Electronics, Communications and Control (ICECC), 2011 International Conference on*, pages 1940–1944. IEEE, 2011.

[31] Hu Cao and Ouri Wolfson. Nonmaterialized motion information in transport networks. *Database Theory-ICDT 2005*, pages 173–188, 2005.

[32] G.P. Roh and S. Hwang. Nncluster: An efficient clustering algorithm for road network trajectories. In *Database Systems for Advanced Applications*, pages 47–61. Springer, 2010.

[33] Jung-Rae Hwang, Hye-Young Kang, and Ki-Joune Li. Spatio-temporal similarity analysis between trajectories on road networks. In *Proceedings of the 24th international conference on Perspectives in Conceptual Modeling*, ER'05, pages 280–289, Berlin, Heidelberg, 2005. Springer-Verlag.

[34] Jung-Rae Hwang, Hye-Young Kang, and Ki-Joune Li. Searching for similar trajectories on road networks using spatio-temporal similarity. In *Advances in Databases and Information Systems*, pages 282–295. Springer, 2006.

[35] E. Tiakas, A. N. Papadopoulos, A. Nanopoulos, Y. Manolopoulos, Dragan Stojanovic, and Slobodanka Djordjevic-Kajan. Searching for similar trajectories in spatial networks. *J. Syst. Softw.*, 82(5):772–788, May 2009.

[36] Eleftherios Tiakas, Apostolos N Papadopoulos, Alexandros Nanopoulos, Yannis Manolopoulos, Dragan Stojanovic, and Slobodanka Djordjevic-Kajan. Trajectory similarity search in spatial networks. In *IDEAS'06. 10th International*, pages 185–192. IEEE, 2006.

[37] S. Turner, R. Margiotta, and T. Lomax. Lessons learned: monitoring highway congestion and reliability using archived traffic detector data. *US Department of Transportation, Federal Highway Administration*, 2004.

[38] Betsy George, Sangho Kim, and Shashi Shekhar. Spatio-temporal network databases and routing algorithms: A summary of results. In *SSTD*, pages 460–477, 2007.

[39] D. Delling, P. Sanders, D. Schultes, and D. Wagner. Engineering route planning algorithms. *Algorithmics of large and complex networks*, pages 117–139, 2009.

[40] Ugur Demiryurek, Farnoush Banaei Kashani, and Cyrus Shahabi. Efficient k-nearest neighbor search in time-dependent spatial networks. In *DEXA (1)*, pages 432–449, 2010.

[41] Venkata M. V. Gunturi, Ernesto Nunes, KwangSoo Yang, and Shashi Shekhar. A critical-time-point approach to all-start-time lagrangian shortest paths: A summary of results. In *SSTD*, pages 74–91, 2011.

[42] Ugur Demiryurek, Farnoush Banaei Kashani, Cyrus Shahabi, and Anand Ranganathan. Online computation of fastest path in time-dependent spatial networks. In *SSTD*, pages 92–111, 2011.

[43] Lívia A Cruz, Mario A Nascimento, and José AF de Macêdo. k-nearest neighbors queries in time-dependent road networks. *Journal of Information and Data Management*, 3(3):211, 2012.

[44] M.F. Mokbel, T.M. Ghanem, and W.G. Aref. Spatio-temporal access methods. *IEEE Data Engineering Bulletin*, 26(2):40–49, 2003.

[45] Michael R. Evans, KwangSoo Yang, James M. Kang, and Shashi Shekhar. A lagrangian approach for storage of spatio-temporal network datasets: a summary of results. In *GIS*, pages 212–221, 2010.

[46] KwangSoo Yang, Michael R Evans, Gunturi Venkata M.V., and Shashi Shekhar. Lagrangian Approaches to Storage of Spatio-temporal Network Datasets. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2013 (accepted).

[47] Michael R. Evans, Dev Oliver, Shashi Shekhar, and Francis Harvey. Summarizing trajectories into k-primary corridors: a summary of results. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '12, pages 454–457, New York, NY, USA, 2012. ACM.

[48] Michael R Evans, Dev Oliver, Shashi Shekhar, and Francis Harvey. Fast and Exact Network Trajectory Similarity Computation: A Case-Study on Bicycle Corridor Planning. In *UrbComp 2013 (submitted)*, 2013.

[49] New York Times, 7/12/2007. http://www.nytimes.com/2007/07/12/business/12ups.html.

[50] U.S. Energy Information Adminstration. Monthly Energy Review June 2011. `http://www.eia.gov/totalenergy/data/monthly/`.

[51] Davis, S.C. and Diegel, S.W. and Boundy, R.G. Transportation energy data book: Edition 28. Technical report, Oak Ridge National Laboratory, 2010.

[52] Austin Brown. Transportation Energy Futures: Addressing Key Gaps and Providing Tools for Decision Makers. Technical report, National Renewable Energy Laboratory, 2011.

[53] US Congress. Energy independence and security act of 2007. *Public Law*, (110-140), 2007. http://en.wikipedia.org/wiki/Energy_Independence_and_Security_Act_of_2007.

[54] InformationWeek. Red Cross Unveils Social Media Monitoring Operation. http://www.informationweek.com/government/information-management/red-cross-unveils-social-media-monitorin/232602219, 2012.

[55] G. Levchuk, A. Bobick, and E. Jones. Activity and function recognition for moving and static objects in urban environments from wide-area persistent surveillance inputs. In *Proceedings of SPIE*, volume 7704, page 77040P, 2010.

[56] New York Times. Military Is Awash in Data From Drones. http://www.nytimes.com/2010/01/11/business/11drone.html?pagewanted=all, 2010.

[57] New York Times. Mapping Ancient Civilization, in a Matter of Days. http://www.nytimes.com/2010/05/11/science/11maya.html, 2010.

[58] H. Kargupta, J. Gama, and W. Fan. The next generation of transportation systems, greenhouse emissions, and data mining. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1209–1212. ACM, 2010.

[59] H. Kargupta, V. Puttagunta, M. Klein, and K. Sarkar. On-board vehicle data stream monitoring using minefleet and fast resource constrained monitoring of correlation matrices. *New Generation Computing*, 25(1):5–32, 2006. Springer.

[60] Lynx GIS. http://www.lynxgis.com/.

[61] MasterNaut. Green Solutions. http://www.masternaut.co.uk/carbon-calculator/.

[62] TeleNav. http://www.telenav.com/.

[63] TeloGIS. http://www.telogis.com/.

[64] American Transportation Research Institute (ATRI). Fpm congestion monitoring at 250 freight significant highway location: Final results of the 2010 performance assessment. http://www.atri-online.org/index.php?option=com_content&view=article&id=303:250-freight-significant-locations, 2010.

[65] American Transportation Research Institute (ATRI). Atri and fhwa release bottleneck analysis of 100 freight significant highway locations. http://www.atri-online.org/index.php?option=com_content&view=article&id=248&Itemid=75, 2010.

[66] Daniel. Sperling and D. Gordon. *Two billion cars*. Oxford University Press, 2009.

[67] Federal Highway Administration. Highway Statistics. *HM-63, HM-64*, 2008.

[68] Betsy George and Shashi Shekhar. Road maps, digital. In *Encyclopedia of GIS*, pages 967–972. Springer, 2008.

[69] S. Shekhar and H. Xiong. *Encyclopedia of GIS*. Springer, New York, 2008.

[70] NAVTEQ, www.navteq.com.

[71] P. Bolstad. *GIS Fundamentals: A first text on geographic information systems*. Eider Pr, 2005.

[72] Shashi Shekhar, Michael R Evans, James M Kang, and Pradeep Mohan. Identifying patterns in spatial information: a survey of methods. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(3):193–214, 2011. Wiley Online Library.

[73] T.C. Bailey and A.C. Gatrell. *Interactive spatial data analysis*, volume 413. Longman Scientific & Technical Essex, 1995.

[74] C. Brunsdon, A.S. Fotheringham, and M.E. Charlton. Geographically weighted regression: a method for exploring spatial nonstationarity. *Geographical analysis*, 28(4):281–298, 1996.

[75] A.S. Fotheringham, C. Brunsdon, and M. Charlton. *Geographically weighted regression: the analysis of spatially varying relationships*. John Wiley & Sons Inc, 2002.

[76] R. Agrawal, R. Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, volume 1215, pages 487–499, 1994.

[77] S. Ghemawat, H. Gobioff, and S.T. Leung. The google file system. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 29–43. ACM, 2003.

[78] D. Borthakur. The hadoop distributed file system: Architecture and design. *Hadoop Project Website*, 11:21, 2007.

[79] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[80] S. Shekhar, S. Ravada, D. Chubb, and G. Turner. Declustering and load-balancing methods for parallelizing geographic information systems. *Knowledge and Data Engineering, IEEE Transactions on*, 10(4):632–655, 1998.

[81] Shashi Shekhar and Sanjay Chawla. *Spatial databases - a tour*. Prentice Hall, 2003.

[82] S. Shekhar, P.R. Schrater, R.R. Vatsavai, W. Wu, and S. Chawla. Spatial contextual classification and prediction models for mining geospatial data. *Multimedia, IEEE Transactions on*, 4(2):174–188, 2002. IEEE Computer Society.

[83] N. Cressie. Statistics for spatial data. *Terra Nova*, 4(5):613–617, 1992.

[84] S. Chawla, S. Shekhar, W.L. Wu, Army High Performance Computing Research Center, and University of Minnesota. *Modeling spatial dependencies for mining geospatial data: An introduction*. Army High Performance Computing Research Center, 2000.

[85] B.M. Kazar, S. Shekhar, D.J. Lilja, and D. Boley. A parallel formulation of the spatial auto-regression model for mining large geo-spatial datasets. In *SIAM International Conf. on Data Mining Workshop on High Performance and Distributed Mining (HPDM2004)*. Citeseer, 2004.

[86] S. Shekhar, S. Ravada, V. Kumar, D. Chubb, and G. Turner. Parallelizing a gis on a shared address space architecture. *Computer*, 29(12):42–48, 1996.

[87] G. Malewicz, M.H. Austern, A.J.C. Bik, J.C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 international conference on Management of data*, pages 135–146. ACM, 2010.

[88] U. Kang, C.E. Tsourakakis, and C. Faloutsos. Pegasus: A peta-scale graph mining system implementation and observations. In *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*, pages 229–238. IEEE, 2009.

[89] J. Cohen. Graph twiddling in a mapreduce world. *Computing in Science & Engineering*, 11(4):29–41, 2009.

[90] Günter Rote. Computing the minimum hausdorff distance between two point sets on a line under translation. *Information Processing Letters*, 38(3):123–127, 1991.

[91] Daniel P. Huttenlocher, Gregory A. Klanderman, and William J Rucklidge. Comparing images using the hausdorff distance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 15(9):850–863, 1993.

[92] Daniel P Huttenlocher and Klara Kedem. Computing the minimum hausdorff distance for point sets under translation. In *Proceedings of the sixth annual symposium on Computational geometry*, pages 340–349. ACM, 1990.

[93] Josephine Marcotty. Federal Funding for Bike Routes Pays Off in Twin Cities. `http://www.startribune.com/local/minneapolis/150105625.html`.

[94] T.H. Cormen. *Introduction to algorithms*. The MIT press, 2001.

[95] L.R. Ford and DR Fulkerson. *Flows in networks*. Princeton University Press, 1962.

[96] Helmut Alt and Leonidas J Guibas. Discrete geometric shapes: Matching, interpolation, and approximation. *Handbook of computational geometry*, 1:121–153, 1999.

[97] Yalin Wang, Qilong Han, and Haiwei Pan. A clustering scheme for trajectories in road networks. In *Advanced Technology in Teaching-Proceedings of the 2009 3rd International Conference on Teaching and Computational Science (WTCS 2009)*, pages 11–18. Springer, 2012.

[98] Hongbin Zhao, Qilong Han, Haiwei Pan, and Guisheng Yin. Spatio-temporal similarity measure for trajectories on road networks. In *Internet Computing for Science and Engineering, Fourth International Conference on*, pages 189–193. IEEE, 2009.

[99] Michael R. Evans, KwangSoo Yang, James M. Kang, and Shashi Shekhar. A lagrangian approach for storage of spatio-temporal network datasets: a summary of results. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '10, pages 212–221, New York, NY, USA, 2010. ACM.

[100] S. Shekhar and S. Chawla. *Spatial Databases: A Tour*. Prentice Hall. ISBN 013-017480-7., 2003.

[101] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 47–57. ACM, 1984.

[102] H. Samet. Spatital data structures. In *ACM SIGGRAPH 2007 courses*, page 1. ACM, 2007.

[103] Shashi Shekhar and Duen-Ren Liu. CCAM: A connectivity-clustered access method for networks and network computations. *IEEE Trans. Knowl. Data Eng.*, 9(1):102–119, 1997.

[104] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. Network flows. Englewood Cliffs, New Jersey, 1993. Prentice Hall, Inc.

[105] J.C. Herrera and A.M. Bayen. Incorporation of Lagrangian measurements in freeway traffic state estimation. *Transportation Research Part B: Methodological*, 2009.

[106] L. R Ford and D. R Fulkerson. Constructing maximal dynamic flows from static flows. In *OPERATIONS RESEARCH Vol. 6, No. 3, May-June 1958, pp. 419-433 DOI: 10.1287/opre.6.3.419*, 1958.

[107] A. Abou-Rjeili and G. Karypis. Multilevel algorithms for partitioning power-law graphs. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, page 10, 2006.

[108] Long-Van Nguyen-Dinh, Walid G. Aref, and Mohamed F. Mokbel. Spatio-temporal access methods: Part 2 (2003 - 2010). *IEEE Data Eng. Bull.*, 33(2):46–55, 2010.

[109] A.U. Frank, S. Grumbach, R.H. Gueting, C.S. Jensen, M. Koubarakis, N.A. Lorentzos, Y. Manolopoulos, E. Nardelli, B. Pernici, H.J. Schek, et al. Chorochronos: A research network for spatiotemporal database systems. *SIGMOD Record*, 28(3):12–21, 1999.

[110] B. George and S. Shekhar. Time-aggregated graphs for modeling spatio-temporal networks. *Journal on Data Semantics XI*, pages 191–212, 2008.

[111] J. Banerjee, W. Kim, S. J. Kim, and J. F. Garza. Clustering a DAG for CAD databases. volume 14, page 1684, November 1988.

[112] Erik G. Hoel, Wee-Liang Heng, and Dale Honeycutt. High performance multimodal networks. In *SSTD*, pages 308–327, 2005.

[113] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, page 813. VLDB Endowment, 2003.

[114] H. Samet. *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, 2006.

[115] C.J. Alpert and A.B. Kahng. Multiway partitioning via geometric embeddings, orderings, and dynamic programming. *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, 14(11):1342–1358, 1995.

[116] S.E. Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007.

[117] Michael R Evans, Dev Oliver, KwangSoo Yang, and Shashi Shekhar. Enabling Spatial Big Data via CyberGIS: Challenges and Opportunities. In *CyberGIS: Fostering a New Wave of Geospatial Innovation and Discovery*. Springer Book, 2013 (accepted).

[118] Michael R Evans, Dev Oliver, Viswanath Gunturi, and Shashi Shekhar. Spatial Big Data: Case Studies on Volume, Velocity, and Variety. In *Big Data: Techniques and Technologies in Geoinformatics*. CRC Press, 2014 (accepted).

[119] Michael R Evans, KwangSoo Yang, Viswanath Gunturi, Betsy George, and Shashi Shekhar. Spatio-Temporal Networks: Modeling, Storing, and Querying Temporally-Detailed Roadmaps. In *Space-Time Integration in Geography and GIScience: Research Frontiers in the US and China*. Springer Book, 2013 (accepted).