Inferring Semantic Information from User Mobility Data

BY

JAMES P. BIAGIONI
B.S., University of Illinois at Chicago, 2006

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2014

Chicago, Illinois

Defense Committee:

Dr. Jakob Eriksson, Chair and Advisor
Dr. Peter Nelson
Dr. Ouri Wolfson
Dr. Isabel Cruz
Dr. John Krumm, Microsoft Research

To Emma

ACKNOWLEDGMENTS

First and foremost, I would like to extend my sincere thanks to Dr. Jakob Eriksson, my advisor and mentor. Of the innumerable lessons I have learned from him, how to *find*, *articulate*, and *pursue* a research problem has been far and away the most transformational, and will continue to serve me well into the future. Without his steadfast support and infinite patience, the work contained in this thesis would not have been possible.

I would also like to extend heartfelt thanks to my committee members, Dr. John Krumm, Dr. Peter Nelson, Dr. Isabel Cruz, and Dr. Ouri Wolfson, for their valuable time and effort in guiding and reviewing my work. Special thanks are due to Dr. Krumm, who graciously invited me to intern with him at Microsoft Research in the Summer of 2012. His mentorship there, and through countless interactions since, has had a lasting impact on me both personally and professionally. Special thanks also to Dr. Nelson, who recognized that my interests in Computer Science lie beyond my undergraduate education, and encouraged me to join the PhD program. I would not have had the opportunity to pursue this undertaking without his guidance.

Thank you to my lab-mates both past and present, especially Timothy Merrifield, Tomas Gerlich, ABM Musa, Peter Snyder, Yanzi Jin, Xiang Huo, Sepideh Roghanchi, Krystian Chmiel and Dr. Chris Kanich. Our lab has been my home away from home for the past several years, and the long hours spent there would not have been the same without them. Their friendship made coming to work every day a joy and pleasure, and has truly meant the world to me.

ACKNOWLEDGMENTS (Continued)

Profound thanks are due to my good friend, Michelle Savage, whose unwavering encouragement and support throughout the course of my degree made it possible to see this endeavor through to its end. Our countless hours-long conversations always provided me with the perspective I needed to see the forest for the trees, and find a path forward.

Thank you to my parents, Peter and Beverley, for impressing upon me the value of a good education, instilling in me a strong work ethic, and providing all I ever needed to fuel and explore my intellectual curiosities. I am especially grateful for my first computer, which immediately provoked the question, "how does this work?" and has lead me directly to this point. Finally, thank you to my sisters, Emma and Lizzie, whose genuine excitement and enthusiasm over my academic pursuits has always provided an inexhaustible source of motivation.

JPB

CONTRIBUTION OF AUTHORS

I would like to gratefully acknowledge my collaborators, whose work forms an integral part of this thesis. Specifically, Dr. Jakob Eriksson, ABM Musa, Tomas Gerlich, Timothy Merrifield, Dr. John Krumm, Xuemei Liu, Dr. Yin Wang, Dr. George Forman, and Dr. Yanmin Zhu.

Chapter 2 represents a published manuscript (1; 2) for which I was the first author. Dr. Eriksson provided the implementation of the kernel density estimation algorithm described in Section 2.1.4, and contributed to the writing of the manuscript.

Chapter 3 represents a published manuscript (3) for which I was the second author. Xuemei, Dr. Wang, Dr. Forman, and Dr. Zhu provided the implementation and description of the TC1 algorithm in Section 3.1.2, the evaluation framework described in Section 3.2, the ground truth map for Shanghai described in Section 3.3.1, the evaluation of the TC1 algorithm in Section 3.3.2 and Section 3.3.3, provided Figures 8–16, and contributed to the writing of the manuscript. Dr. Eriksson also contributed to the writing of the manuscript.

Chapter 4 represents a published manuscript (4) for which I was the first author. Dr. Eriksson provided the implementation and description of the geometry refinement algorithm in Section 4.6, and contributed to the writing of the manuscript.

Chapter 5 represents a published manuscript (5) for which I was the first author. Dr. Eriksson provided the field study of GPS tracking described in Section 5.1.1, and contributed to the writing of the manuscript. ABM provided the implementation, description, and analysis

CONTRIBUTION OF AUTHORS (Continued)

of the adaptive sampling techniques described in Section 5.4, provided Figures 38–39, and also contributed to the writing of the manuscript.

Chapter 6 represents a published manuscript (6) for which I was the first author. Dr. Eriksson provided the system architecture described in Section 6.2, the prototype driver interface described in Section 6.7.2, contributed to the development of the web-based system prototype described in Section 6.7.3, and contributed to the writing of the manuscript. Timothy provided the implementation, description, and analysis of the schedule extraction algorithm in Section 6.5, the arrival time prediction evaluation in Section 6.6.2, Figures 48–52 and Figure 56, and contributed to the writing of the manuscript. Tomas contributed to the development of the web-based system prototype described in Section 6.7.3, and to the writing of the manuscript.

Chapter 7 represents a published manuscript (7) for which I was the first author. Dr. Krumm recruited our user study participants, organized and conducted many of the interviews described in Section 7.2, provided the statistical analysis of our evaluation results in Section 7.4, provided Figures 60–61 and Figure 63, and contributed to the writing of the manuscript.

Chapter 8 contains excerpts from several published manuscripts (1; 2; 4; 5; 6; 7) for which I was the first author. Dr. Eriksson contributed to the related work write-up in Section 8.1–Section 8.1.3.3. Dr. Krumm provided the related work write-up in Section 8.4.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

CHAPTER **PAGE**

TABLE OF CONTENTS (Continued)

CHAPTER **PAGE**

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

FIGURE **PAGE**

LIST OF FIGURES (Continued)

FIGURE **PAGE**

LIST OF FIGURES (Continued)

LIST OF ABBREVIATIONS

ANOVA	Analysis of Variance
CA	Constant Acceleration
CD	Constant Deceleration
CDF	Cumulative Distribution Function
CL	Constant Location
CTA	Chicago Transit Authority
CV	Constant Velocity
DTW	Dynamic Time Warping
GPS	Global Positioning System
GTFS	General Transit Feed Specification
HMM	Hidden Markov Model
KDE	Kernel Density Estimate
MB	Map Based
MM	Markov Model
OSM	OpenStreetMap
UIC	University of Illinois at Chicago
USD	United States Dollar

SUMMARY

Thanks to the ubiquity of Global Positioning System (GPS) sensors in a variety of everyday devices, and the myriad applications for which they are currently used, there is presently a tremendous amount of location-data being generated and collected concerning users' mobility patterns. As a result, we now have an unprecedented opportunity to leverage these rich datasets to infer a range of useful phenomena, in several different application areas.

One such area concerns the inference of road maps. Here, using opportunistically collected datasets we can design algorithms to automatically infer entirely new sections of the map, reducing the need for expensive road surveys. In this thesis we undertake a thorough study of the current state of the art in map inference algorithms, and identify areas for improvement. Based on these findings, we investigate algorithms for mitigating the problems presented by infrequently-sampled data, and develop a new hybrid map inference method for overcoming existing sensitivities to GPS noise and disparity.

We then change gears, and look at ways to reduce the data uplink usage of online GPS tracking systems, while preserving accuracy. With the most commonly used method for reducing usage in practice being simple uniform periodic sampling, there is ample room for improvement. Given the rising popularity of tracking applications, improvements here can have real practical impact. To this end, we develop a system that combines a unified extrapolator which is able to infer users' future movements based on their current and historical patterns, with an adaptive sampler that allows a system operator to specify an error or budget-based performance target

while it intelligently optimizes the other. We evaluate its performance using real-world GPS traces, and compare it against the current status quo.

Next, we look at inferring the necessary phenomena to create a fully automatic transit tracking and arrival time prediction system. Although commercial products exist for this purpose, their installation, configuration, and maintenance can be cost-prohibitive for small agencies. Our goal is to enable the same functionality, with smartphones being the only required hardware. By developing the appropriate algorithms, we are able to create an end-to-end system that automatically infers routes, stops, and schedules, as well as provides real-time arrival time predictions, and demonstrate its accuracy against real-world data.

Finally, we work to find algorithms for measuring the similarity of a person's days, based on location traces recorded by GPS, in a way that approximates human assessments. Having this capability allows us to provide a useful metric for systems that identify anomalous behavior, or predict how a day is likely to evolve. By conducting a user-study we are able to identify suitable algorithms for this task, and evaluate their ability to cluster days.

CHAPTER 1

INTRODUCTION

Due to the widespread availability of GPS sensors in a variety of everyday devices, as well as their near-ubiquitous application in modern cellphones and vehicles, the collection of massive amounts of location-data describing the time and position of individuals' movement patterns is becoming increasingly common.

Very often this data collection is entirely intentional, as cellphones and vehicle telematics systems are commonly co-opted to support GPS tracking applications, including real-time taxi dispatching, freight logistics, and public transit arrival time predictions. However, a large amount of data is also being generated and collected as a side-effect of supporting end-user applications. One of the most prevalent examples of this is crowd-sourced traffic maps, where applications such as the pervasive "Maps" application automatically upload their current location to a central server while they are running, in order to power updates.

Regardless of the purposes for which this data is collected, the end result is that system operators, application developers, and corporations now have access to large, rich collections of user mobility data. With that in mind, the overall goal of this work has been to develop algorithms for application areas that stand to benefit from the rich semantic information that can be inferred from this high-fidelity data.

One potential use for this wealth of data is to infer and update the geometry and connectivity of road maps, using what are known as map inference algorithms. These algorithms offer

a tremendous advantage when no existing road map data is present; rather than incur the expense of a complete road survey, GPS trace-data can be used to infer entirely new sections of the road map at a fraction of the cost. Road map inference can also be valuable in cases where a road map is already available. Here, they not only help to increase the accuracy of drawn road maps, but also help to detect new road construction and dynamically adapt to road closures—useful features for digital road maps being used for in-car navigation.

In Chapter 2, we provide a survey of the contemporary literature on this subject, and through a qualitative, quantitative, and comparative evaluation of existing algorithms, identify ways to improve upon the current state of the art methods.

One weakness with currently available map inference algorithms is their inability to infer maps from GPS traces that are sparsely sampled (i.e., recorded with low temporal frequency). This type of data poses a problem, because as the distance between consecutive GPS samples grows, the likelihood of correctly estimating the path between them rapidly declines. Most often, a simple straight-line path is assumed. However, this frequently results in paths being inferred over areas where no road actually exists.

In Chapter 3, we take a look at two different map inference algorithms for dealing with this problem: one specifically designed for sparse GPS data, and one which is an adaptation of the best method found in Chapter 2. We assess their trade-offs and effectiveness in both qualitative and quantitative comparisons, using both an artificially sub-sampled dataset from Chicago, and a sparsely-sampled dataset collected from thousands of taxis in Shanghai.

Another identified weakness with current map inference algorithms is their sensitivity to GPS noise and trace disparity. As is common with opportunistically collected GPS data, there is often a great disparity in terms of coverage. For example, a freeway may be represented by thousands of trips, whereas a residential road may only have a handful of observations. Additionally, while modern GPS receivers typically produce high-quality location estimates, errors over 100 meters are not uncommon, especially near tall buildings or under dense tree coverage. Combined, GPS trace disparity and error present a formidable challenge for the current state of the art in map inference. To address this issue we may tune the parameters of existing algorithms, choosing to remove spurious roads created by GPS noise, or admit less-frequently traveled roads, but not both.

In Chapter 4 we present an extensible map inference pipeline, designed to mitigate GPS error, admit less-frequently traveled roads, and scale to large datasets. We also demonstrate and compare the performance of our proposed pipeline against existing methods, both qualitatively and quantitatively, using a real-world dataset that includes both high disparity and noise.

In Chapter 5 we shift away from creating road maps, and instead look at the problem of minimizing the data uplink usage of online GPS tracking systems, while maintaining accuracy. Given the increasing popularity of tracking assets and people with GPS, reducing cost is often an important concern. In a field study of large-scale GPS tracking, we find that uniform periodic sampling is the overwhelming policy of choice for achieving this goal. One likely reason for this is that it provides direct control over cost. Unfortunately, however, any savings here necessarily result in a net loss in terms of tracking accuracy and timeliness.

Our approach provides a more flexible solution to this problem. First, we develop a unified extrapolator that is able to infer users' future movements based on their current and historical movement patterns. Then, we combine it with an adaptive sampler which allows a system operator to specify error or budget-based performance targets, while it automatically optimizes the other. We evaluate our end-to-end system using a large collection of real-world GPS traces, and compare it against the current status quo.

In Chapter 6 we shift gears again, and investigate how to infer the necessary components for creating an automatic vehicle tracking and arrival time prediction system for transit agencies. Creating such a system manually can be a daunting task: commercial systems require tracking devices to be installed in vehicles, drivers to be trained in their usage, and a great deal of human processing to create and maintain route maps and schedules in the appropriate format. For smaller transit agencies, such as school bus operators or campus shuttle services, this can be a significant barrier to adoption.

In order to facilitate the introduction of transit tracking and arrival time prediction in smaller transit agencies, we describe a smartphone-based system which we call EasyTracker. To use EasyTracker, a transit agency must obtain smartphones, install an application, and place a phone in each transit vehicle. Our goal is to require no other input. This level of automation is made possible through a set of algorithms that use GPS traces collected from instrumented transit vehicles to determine routes served, locate stops, and infer schedules. In addition, online algorithms automatically determine the route served by a given vehicle at a given time, and predict its arrival time at upcoming stops.

We evaluate our algorithms on real datasets from two existing transit services, demonstrate our ability to accurately reconstruct routes, stops, and schedules, and compare our system's arrival time prediction performance with the current "state of the art" for smaller transit operators: the official schedule.

In Chapter 7 we change tack one final time and look at the problem of assessing the similarity of a person's days based on their recorded location traces. An accurate similarity measure could be useful for finding anomalous behavior, for clustering similar days, or for predicting future travel. While the problem of location trace similarity has been studied extensively in the literature, those efforts have been primarily focused on machine processing, whereas we are interested in matching human assessments of similarity. To this end, we conduct a user study wherein volunteer subjects record their own GPS traces over the course of several weeks, and then assess their similarity using custom-designed software. We test several different similarity algorithms in an effort to accurately reproduce our subjects' assessments, and then apply one such algorithm to the task of clustering days using location traces.

Finally, in Chapter 8 we close by taking a broad look at the existing academic literature that is related to the topics covered in this thesis.

CHAPTER 2

INFERRING ROAD MAPS FROM GPS TRACES: SURVEY AND COMPARATIVE EVALUATION

Road map inference is the process of automatically producing a directed and annotated road map from GPS traces, typically collected opportunistically from vehicles that are already traveling the roads for some other purpose. Map inference can be used to rapidly map unknown or constantly changing territory, to update and improve upon existing road maps, or to quickly adapt to detours and new construction. Moreover, it can also be used to produce custom maps for certain classes of travelers, including pedestrians, bicyclists, transit riders (6), truckers, or tourists, by feeding in data from different sources.

The past decade has witnessed considerable interest in this problem, with a variety of innovative solutions being presented in the open literature. Unfortunately, an almost-exclusive focus on qualitative evaluation, and a lack of comparative evaluation, has made it difficult to understand the relative merits of the various proposed methods. Virtually every published paper on the topic relies on visual inspection of the results, manually comparing inferred maps against existing maps or satellite imagery. Furthermore, comparisons against existing work are more or less absent, with only one paper out of eleven we surveyed providing any comparison against prior work (see Chapter 8 for more details).

We conjecture that the lack of quantitative, comparative evaluation of map inference algorithms is due to three missing elements: (i) sufficiently expressive and robust methods of

quantitatively evaluating the accuracy of inferred maps, (ii) publicly available implementations of proposed map inference methods, and (iii) common sets of publicly available GPS traces and ground truth maps for use in evaluations.

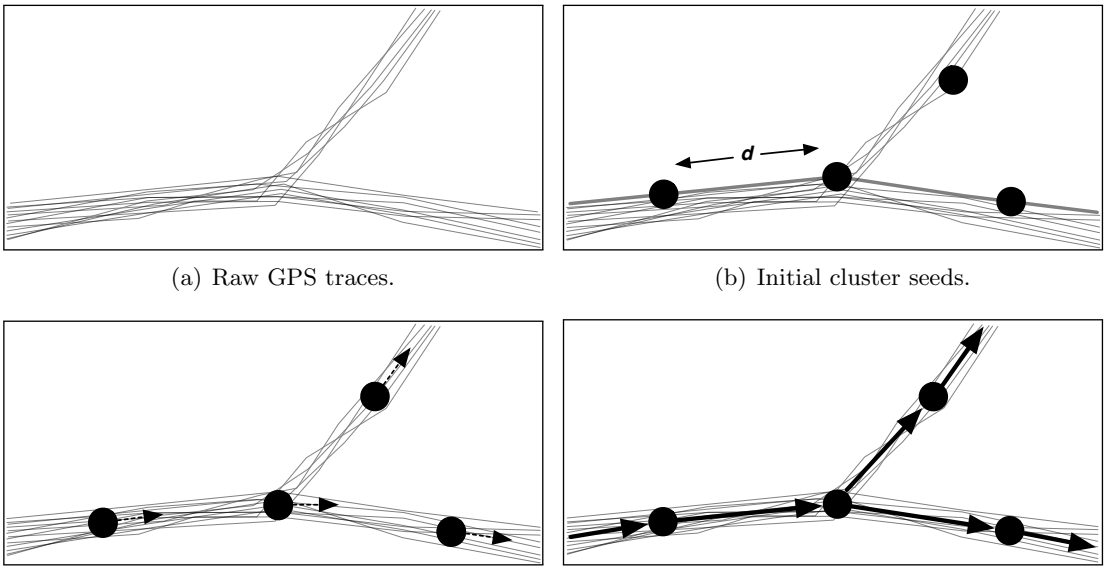
In order to address the problems identified above, and provide a better understanding of the strengths and weaknesses of existing algorithms, in this chapter we: (i) provide an overview of the current literature on map inference, (ii) describe the first quantitative evaluation method for map inference algorithms, and (iii) provide a qualitative, quantitative, and comparative evaluation of three reference algorithms. Additionally, we make available open-source implementations of the three reference algorithms, and a 118-hour trace dataset and ground truth map for unrestricted use by the map inference community on our website (8).

2.1 Background

The base-line requirement for a map inference algorithm is to automatically turn raw GPS traces into a directed and annotated graph representing the connectivity and geometry of the underlying road network. Beyond such basic map inference, various additional objectives have been proposed, such as detecting (9) and extracting (10) detailed intersection geometries, the number and centerlines of lanes (10; 11), and speed limits and road types (12). In this thesis, we focus primarily on basic map inference, and only briefly mention these other aspects.

2.1.1 Operational Overview

The map inference process is typically preceded by a filtering step, where GPS traces are checked for any irregularities with regard to expected distance between points, speed traveled, acceleration, and abrupt direction change. Any point along a trace that fails to satisfy these



(a) Raw GPS traces. (b) Initial cluster seeds.

(c) Updated means with headings. (d) Final map after linking means.

Figure 1. Map inference using the k-means algorithm.

criteria is removed, with an interpolated point being inserted in its place. After filtering, the approaches described in the literature can be categorized by their algorithmic foundations into three categories: the k-means algorithm (13), trace merging, or kernel density estimation (KDE) (14). Below, we introduce the three categories of map inference algorithms in more detail.

2.1.2 Map Inference Based on the k-means Algorithm

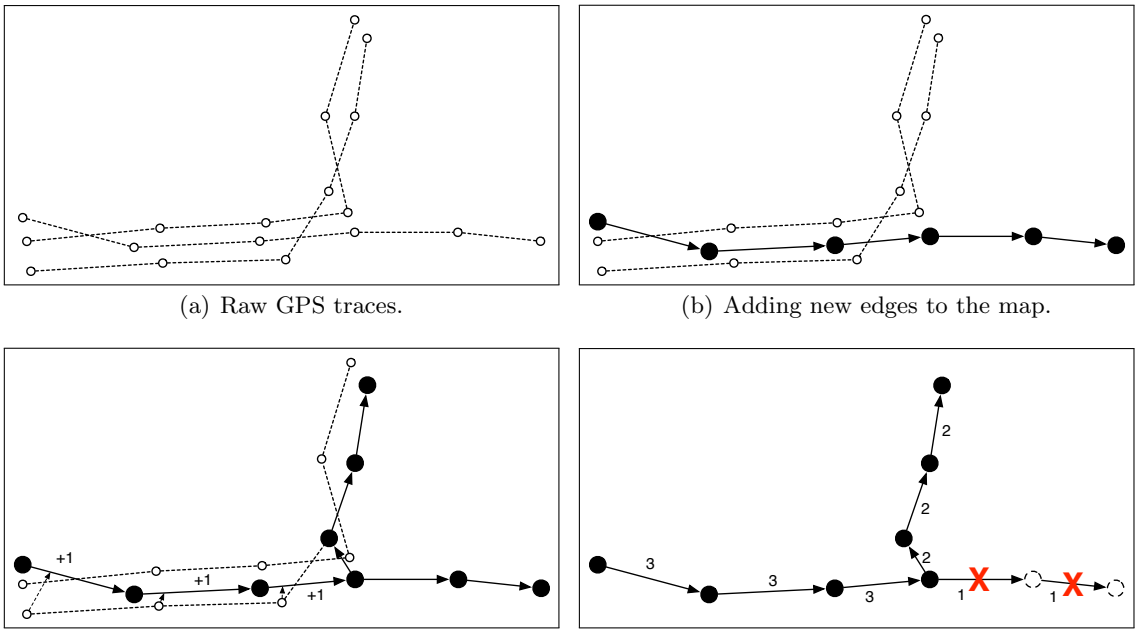
The k-means approach was originally described by Edelkamp and Schrödl (15), and is arguably the most popular map inference technique in the current literature (10; 16; 17; 18; 19). The basic operation of this algorithm is illustrated in Figure 1, and it begins by taking the set of raw GPS traces (Figure 1(a)) and distributing a series of "cluster seeds" along their lengths, with the constraint that every trace point must be located within a fixed distance d

and bearing difference δ of a cluster seed (Figure 1(b)). This results in a sparse representation of the underlying traces based on the collection of seeds. Using the cluster seeds as an initial approximation, minor variations on the k-means algorithm are then used to find seed locations and headings that best describe the raw traces (Figure 1(c)). Once the final seed locations are determined, they are then linked to form road segments based on the pattern of raw traces passing between them (Figure 1(d)). These segments then represent the map inferred using this technique. We provide a qualitative and quantitative evaluation of the basic k-means algorithm in Section 2.3 and 2.4.

2.1.3 Map Inference Based on Trace Merging

The trace merging approach was simultaneously introduced by Cao and Krumm (20) and Niehoefer, Burda, Wietfeld, Bauer and Lueert (12), and its basic operation is illustrated in Figure 2. The process begins with the set of raw GPS traces and an empty map (Figure 2(a)). It then iterates through each recorded trace adding unit weight edges (corresponding to pairs of GPS samples) to the map (Figure 2(b)), unless an existing edge is sufficiently similar in location and heading. Should such an edge already exist, its weight is instead incremented (Figure 2(c)). As a final refinement step, in post-processing any map edges with weight below a specified threshold are removed, such as those marked with an "X" in Figure 2(d). Those edges that remain form the map inferred using this technique.

In addition to the basic operation described above, the method proposed by Cao and Krumm in (20) precedes the standard trace merging method with a "clarification" step. This is a type of particle simulation (21), where a strong, but short-range attractive force is applied to pull



(a) Raw GPS traces.

(b) Adding new edges to the map.

(c) Incrementing weight of matching edges.

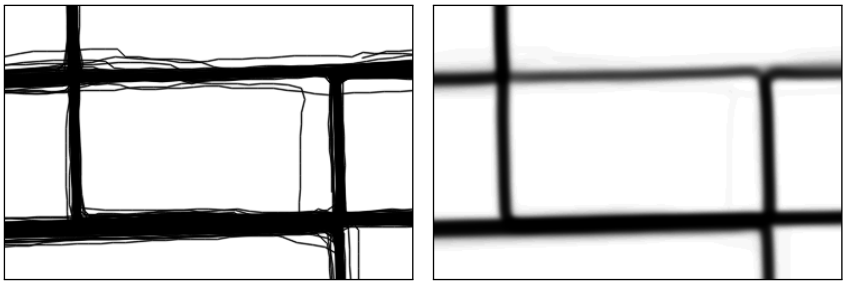
(d) Final map after post-processing.

Figure 2. Map inference using trace merging approach.

together nearby traces that originate from the same road, reducing the effects of GPS noise, and forming tight bands along the road centerlines. We provide a qualitative and quantitative evaluation of this trace merging algorithm in Section 2.3 and 2.4.

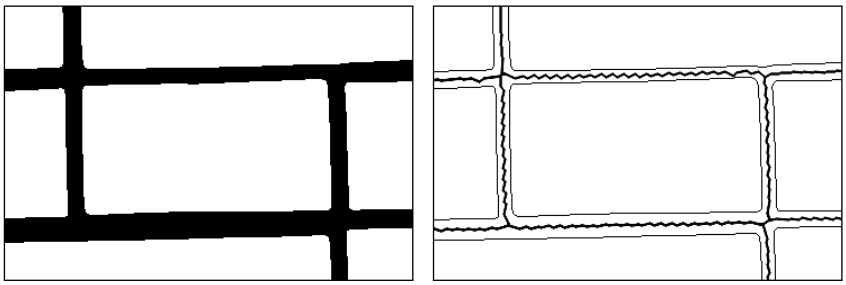
2.1.4 Map Inference Based on Kernel Density Estimation

The kernel density estimation approach was first described by Davies, Beresford and Hopper (22), and has since formed the basis for several derivative map inference techniques (23; 24; 25). The basic operation of this algorithm is illustrated in Figure 3, and it begins by splitting the geographic area covered by the raw GPS traces into a two-dimensional grid of cells. Iterating over each of the raw GPS points or edges, a two-dimensional histogram is then produced,



(a) 2-dimensional histogram of raw GPS traces.

(b) Initial density estimate.



(c) Thresholded density estimate.

(d) Road centerlines are computed.

Figure 3. Map inference using kernel density estimation approach.

representing the number of points or edges that fall in each grid cell (Figure 3(a)). This histogram is then convolved with a Gaussian smoothing function (26), to produce an approximate density estimate of the underlying data (Figure 3(b)). This is an approximation due to the discretization created by the use of grid cells, where the accuracy of the approximation is inversely proportional to the grid size. The density estimate is then passed through a threshold function to produce a binary image of the underlying road outlines (Figure 3(c)), and finally the centerline between the road outlines is extracted (Figure 3(d)). The resulting centerlines represent the map inferred using this technique.

Beyond the basic operation described above, the technique proposed by Davies et al. in (22) accounts for the edges between GPS points in the histogram, by an amount proportional to the edge-length passing through each grid cell. Moreover, they utilize a novel method for extracting road centerlines based on Voronoi tessellation (27), and they also produce KDEs of traces in each of the 8 cardinal and ordinal directions, using the information to annotate each road segment with its allowed directions of travel. We provide a qualitative and quantitative evaluation of this KDE-based algorithm in Section 2.3 and 2.4.

2.2 Robust Quantitative Evaluation of Inferred Maps

In this section, we describe a method for evaluating the accuracy of a map, with respect to a second "ground truth" map. This is a necessary requirement for a quantitative, comparative evaluation of competing map inference algorithms.

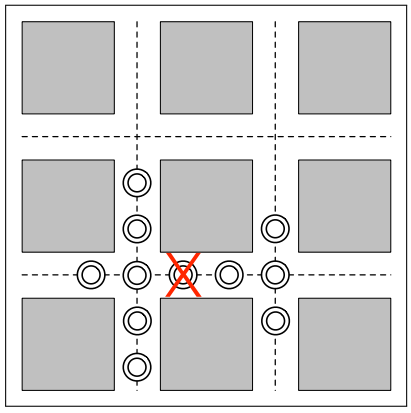
The accuracy of an inferred map depends on two primary aspects of the map: geometry and topology. Here, the geometry of the map describes the geographic location of roads, while the topology describes the interconnections between roads.

A large body of work on graph similarity looks at the problem of comparing graphs (28). This includes exact methods testing for graph isomorphism (29), and inexact methods measuring graph edit distance (30). However, the problem of measuring graph similarity is fundamentally different from that of measuring map similarity: crucially, graphs and their nodes and edges lack any notion of geographic location. Thus, while graph similarity algorithms are able to measure the degree of topological similarity, their nodes and edges can be freely transposed

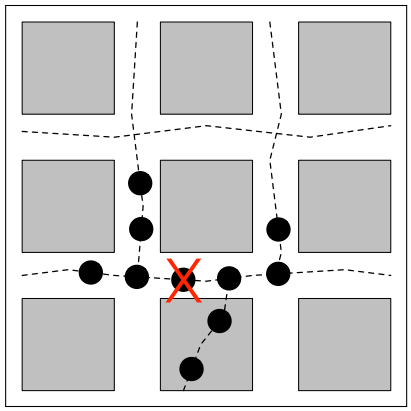
in order to find the closest match. In a map however, the geographic location of nodes and edges must be taken into account, in addition to their topological relationships.

To simultaneously measure the geometric and topological similarity of maps, we propose a method based on the following idea: starting from a random street location, explore each map outward within a maximum radius. This produces two sets of locations, which are essentially spatial samples of a local map neighborhood. By comparing the two sets of samples, and repeatedly sampling the maps in this fashion, we obtain a robust measure of the difference between the two maps. If one of the maps is the ground truth, this difference represents the accuracy of the other.

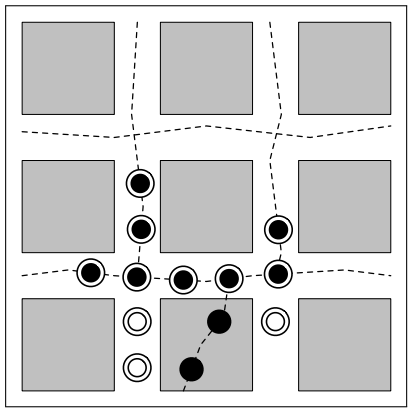
The operation of our map comparison algorithm is depicted in Figure 4. First, we select a start location uniformly at random from the ground truth map. This point is marked with an "X" in Figure 4(a). From the start location, we follow all road segments within a small matching distance d, dropping virtual "holes" at fixed intervals until a maximum radius r from the start location is reached, or a previously followed segment is encountered. When an intersection is encountered, we follow all connecting road segments that lead away from the original location, as turn restrictions and one-way streets allow. Restricting the process to segments leading away from the origin elegantly de-emphasizes unlikely driving patterns such as u-turns, and improves the robustness of the map comparison operation. We then repeat this process starting from the closest point on the inferred map, marked with an "X" in Figure 4(b). Following the same procedure, we drop virtual "marbles" at fixed intervals out to a radius r.



(a) Holes are dropped at fixed intervals along edges of the ground truth map.



(b) Marbles are dropped at fixed intervals along edges of the inferred map.



(c) Marbles from the inferred map fill holes where the maps overlap.

Figure 4. Overview of map comparison algorithm.

Intuitively, if a marble lands close to a hole, it falls in. This represents our matching process. As illustrated in Figure 4(c), marbles that are too far from a hole remain where they landed, and holes with no marbles nearby remain empty. In the figure, holes that are filled correspond to "matched locations," where the geometry and topology of the two maps overlap. Unmatched marbles correspond to spurious parts of the inferred map, and holes that remain empty correspond to parts of the ground truth map that are missing in the inferred map. Counting the number of unmatched marbles and empty holes, we quantify the accuracy of the inferred map with respect to the ground truth according to two metrics: the proportion of spurious marbles,

$$spurious = \frac{spurious\_marbles}{spurious\_marbles + matched\_marbles},$$

and the proportion of missing locations (empty holes),

$$missing = \frac{empty\_holes}{empty\_holes + matched\_holes}.$$

To produce a combined performance measure from these two values, we compute their harmonic mean using the well-known F-score (31),

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall} = 2 \cdot \frac{(1 - spurious)(1 - missing)}{(1 - spurious) + (1 - missing)},$$

where the higher the F-score, the closer the match.

Sampling the maps locally is an important aspect of our approach as it provides us the ability to capture the connectivity of the maps at a very fine-grained level (i.e., as they would be traveled), therefore allowing us to measure topological similarity. Conversely, one could imagine taking a global approach to this problem where we simply cover every edge in the ground truth map with holes, and every edge in the inferred map with marbles, yielding a single pair of sets to match. While such a process would capture the geometric similarity between maps, it would fail to capture local topological similarity, a crucial aspect of overall map similarity. Repeated local sampling at randomly chosen locations yields an accurate view of local geometry and topology throughout the map.

2.2.1 Constructing the Ground Truth

In order to measure the accuracy of the inferred map, we need an accurate ground truth map for comparison. We base our ground truth map on the OpenStreetMap (OSM) (32) database. However, because this map contains many road segments that were never traversed by the vehicles in our dataset, we restrict our ground truth map to those street segments that were visited at least once. This reduced ground truth map reflects the most accurate road topology that can be inferred by the available traces.

2.3 Qualitative Evaluation

In this section, we qualitatively evaluate the road maps generated by three representative algorithms, one from each of the three classes described in Section 2.1. To make this comparison, we first re-implemented the algorithms as described in their respective papers. For k-means we use the algorithm by Edelkamp and Schrödl ("Edelkamp") (15), for trace merging we use

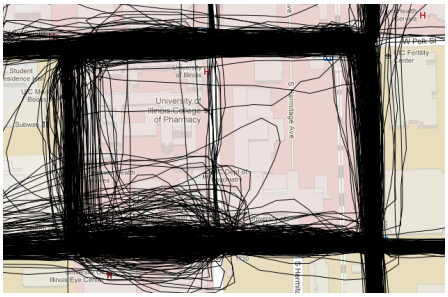
the algorithm by Cao and Krumm ("Cao") (20), and for KDE-based map inference we use the algorithm by Davies et al. ("Davies") (22).

2.3.1 Evaluation Data

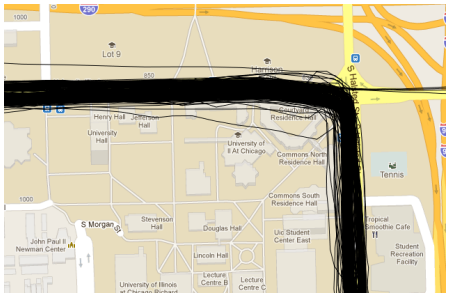
For trace data, we use 118 hours of GPS traces from the University of Illinois at Chicago (UIC) campus shuttles. In addition to traveling around campus, these vehicles pass through two different areas containing relatively tall buildings and significant GPS error. Figure 5(a) provides an example of the distribution of traces and GPS error found in our dataset in one of these high-error areas. In our evaluation, we study both the entire dataset and a subset of the data drawn from an area of low-rise buildings where there is very little GPS error (see sample in Figure 5(b)). Because GPS error can be a problem for map inference algorithms, partitioning our data this way allows us to test and compare their performance on both a realistic, and somewhat idealized dataset. The road maps inferred by each algorithm are depicted in Figures 5(c)–(h), and described below.

2.3.2 Davies et al.

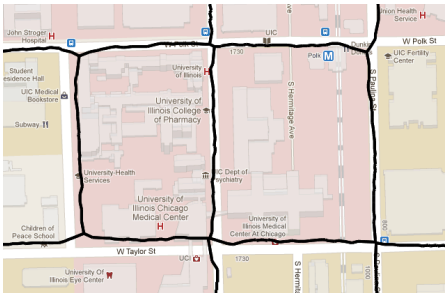
Visually, this algorithm produces superior maps in areas with high GPS error. Because this method avoids treating GPS traces individually, and instead uses them in aggregate to find the road boundaries, it is able to create one road from a large collection of relatively diverse traces. We can see this aspect of the algorithm illustrated in the difficult high-error case in Figure 5(c), where it accurately extracts the road topology without adding any extraneous edges. In the low-error case in Figure 5(d), the result is largely the same. However, we must note the absence of the less-traveled road segment on the right-hand side. This illustrates the fundamental trade-



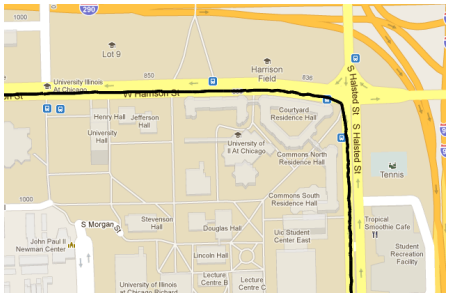
(a) Raw data, high-error sample.



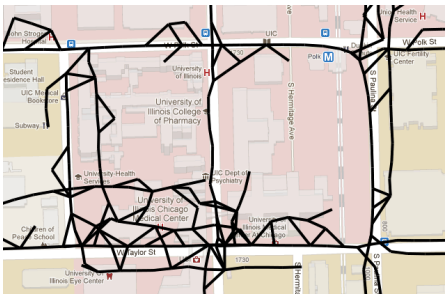
(b) Raw data, low-error sample.



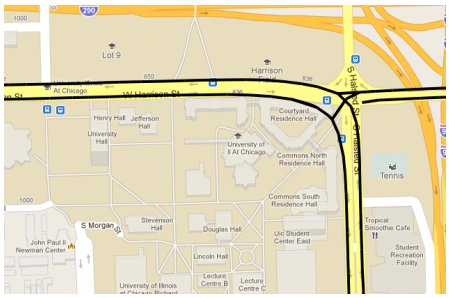
(c) Davies, high-error sample.



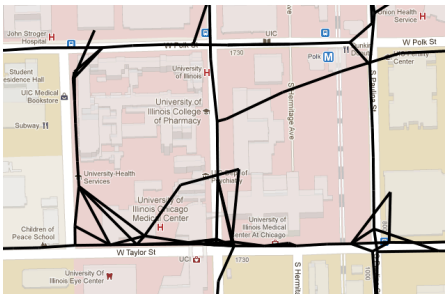
(d) Davies, low-error sample.



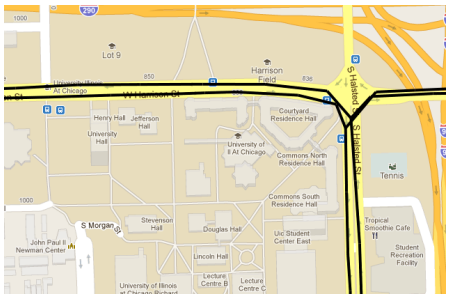
(e) Cao, high-error sample.



(f) Cao, low-error sample.



(g) Edelkamp, high-error sample.



(h) Edelkamp, low-error sample.

Figure 5. Raw data and map inference results in areas with high and low GPS error.

off with this algorithm: any given threshold value must compromise between introducing noise in high-density areas, and losing infrequently traveled edges in low-density areas.

2.3.3 Cao and Krumm

The clarification pre-processing step performed in this algorithm helps reduce GPS noise prior to map inference, and in noise-free areas results in cleanly inferred maps as demonstrated in Figure 5(f). However, clarification has its limitations in areas with high GPS error, where spatially dispersed traces are unable to become tightly banded. As a result, when the trace merging method is applied to the clarified data, residual noise results in the spurious roads seen in Figure 5(e). Although this algorithm attempts to prune spurious roads after map inference, its efforts are largely futile in areas of high GPS noise where edge volume is widely distributed.

2.3.4 Edelkamp and Schrödl

This algorithm creates road segments by linking clusters based on the underlying trace data, and works well in areas with low GPS noise as we can see in Figure 5(h). However, this cluster-linking method is easily led astray by GPS noise and results in several spurious roads being produced, as illustrated in Figure 5(g). Because this algorithm does not attempt to prune spurious roads after inference, all of these roads remain in the final map. Worth noting here is that our implementation does not include intersection refinement and lane-finding. While these additional steps would have likely improved its results on the low-error dataset, they would have had little or no effect on the high-error dataset.

2.4 Quantitative Evaluation

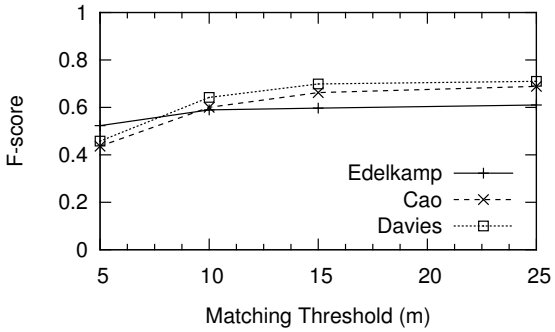
In this section, we use the map comparison method described in Section 2.2 to quantitatively evaluate the three representative algorithms described in Section 2.3. We also discuss parameter sensitivity and implementation details below.

2.4.1 Main Quantitative Results

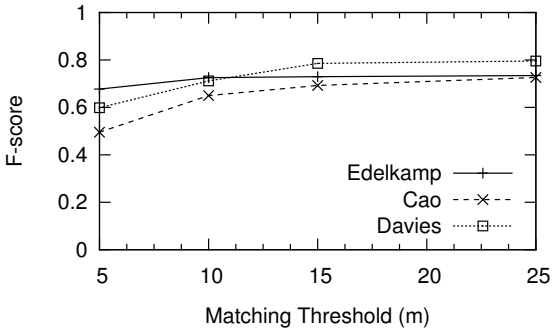
Figures 6(a) and 6(b) illustrate our main results, showing the performance of the three algorithms for a varying "matching threshold." The matching threshold is the allowable distance between a marble and hole before it falls in. Detailed discussion of these results follows.

In Figure 6(a) we can see the performance of the three algorithms as tested on the full dataset. For small matching thresholds, the fine-grained spatial accuracy of the algorithms is significant, with both Davies and Cao underperforming Edelkamp at the 5-meter threshold. For Davies, this is explained by the fact that it generates a single bi-directional centerline, whereas the others produce one edge in each direction. On a wide road, the distance between the centerline and the center of the road in one direction may well exceed 5 meters. A similar problem is exhibited by Cao, where lanes in opposite directions are artificially spread apart to improve legibility. This introduces a slight error, which shows up as poor matching performance for a 5-meter threshold.

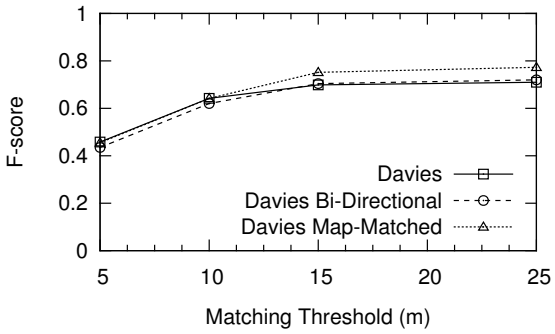
On the low-error dataset, all of the algorithms unsurprisingly achieve a higher F-score (see Figure 6(b)). However, while Davies continues to outperform both Cao and Edelkamp at 15- and 25-meter matching thresholds, the prevalence of wide, median-separated two-way roads in this dataset allows Edelkamp to outperform Davies at 5- and 10-meter matching thresholds.



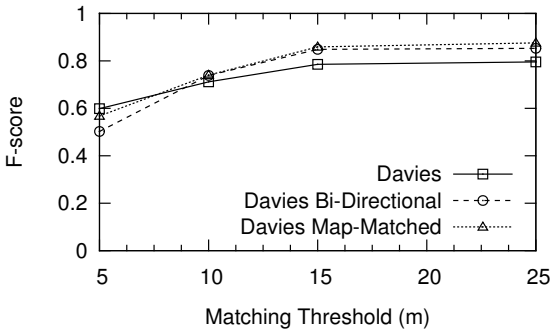
(a) All algorithms, full dataset.



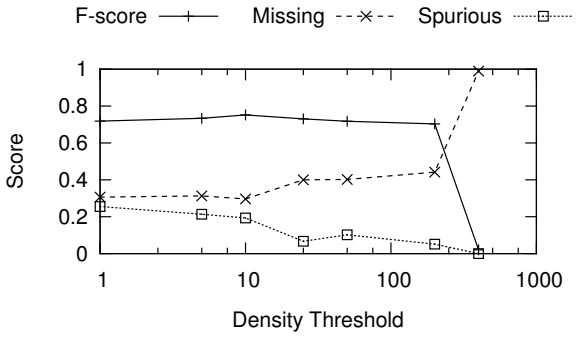
(b) All algorithms, low-error dataset.



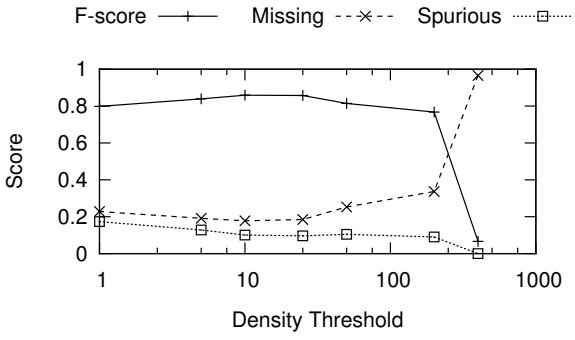
(c) Davies algorithms, full dataset.



(d) Davies algorithms, low-error dataset.



(e) Davies map-matched algorithm, full dataset.



(f) Davies map-matched algorithm, low-error dataset.

Figure 6. Overall results on the full and low-error datasets.

2.4.1.1 Addressing Directionality Finding in Davies

In contrast with the other two approaches, Davies generates a single centerline for each street, with a directionality annotation. The authors in (22) use a technique that measures the direction of travel through each grid cell (as mentioned in Section 2.1.4) to extract the allowed directions of travel. However, in our testing this method did not fare well: many streets that are bi-directional in the ground truth, were not represented as such in the inferred map. To correct for this problem we experimented with two different modifications to the Davies algorithm.

First, we simply discarded the directionality-finding component of the Davies algorithm, making every road bi-directional. While this now allowed for bi-directional roads whose directionality was incorrectly inferred to be correctly represented, it also introduced the problem of incorrectly representing one-way streets as bi-directional. On the full dataset, which consists of a large number of one-way and bi-directional streets, this tradeoff resulted in no net performance gain, as can be seen in Figure 6(c). In our low-error dataset however, which consists predominantly of bi-directional roads, we can see that the bi-directional Davies implementation achieved a notable performance gain (see Figure 6(d)).

As an alternative to the directionality-finding technique in (22), we map-matched our GPS traces onto the inferred bi-directional map described above, using Viterbi map-matching (33). We then used the resulting sequences of road segments to infer road directionality. The relative performance of the three techniques is shown in Figures 6(c) and 6(d). Here, the line designated "Davies" is identical to the one in Figures 6(a) and 6(b). Building on an already

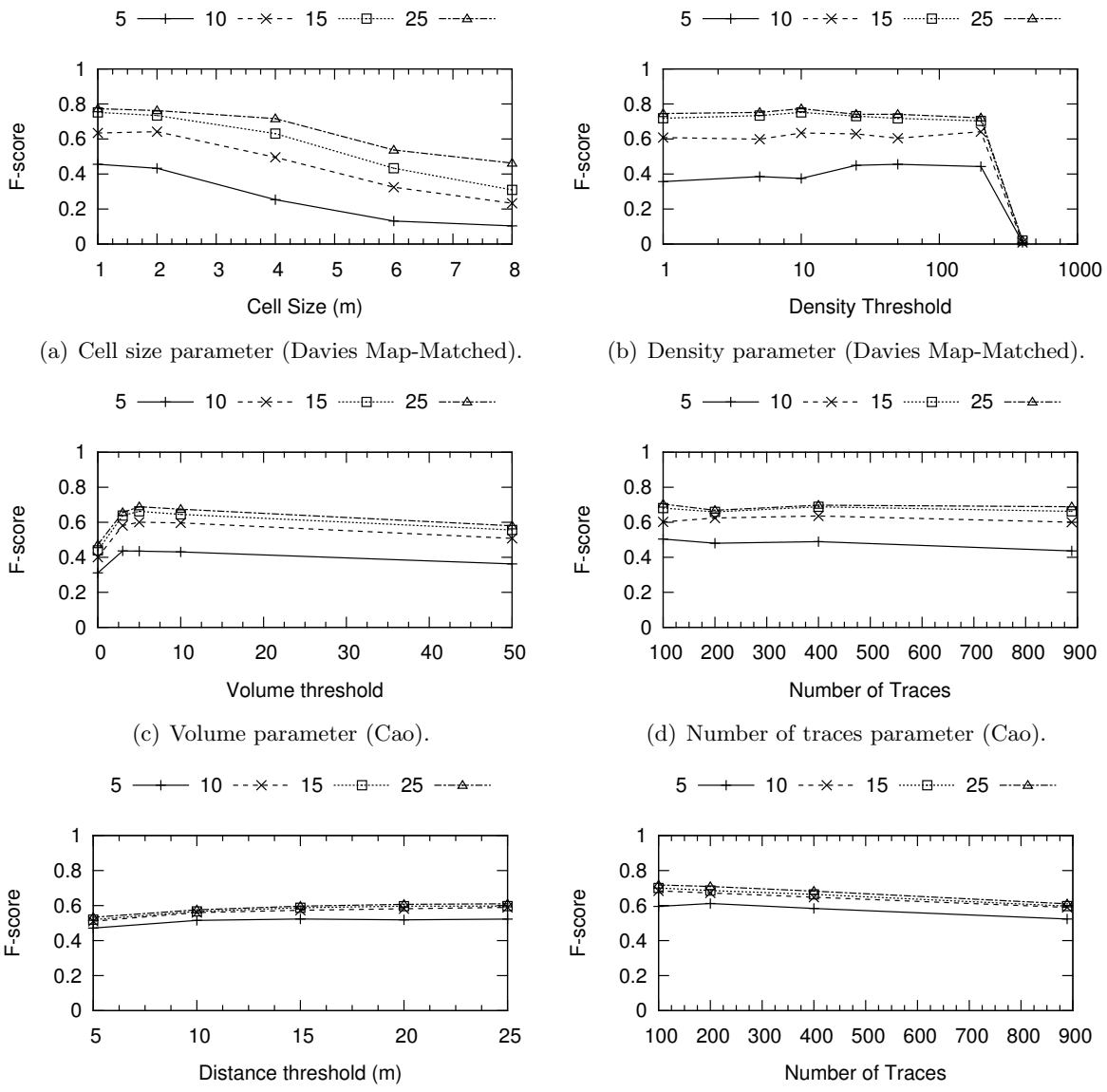
strong performance, this map-matching technique gives the KDE-based algorithm by Davies a significant advantage over the alternative approaches.

2.4.1.2 Remaining Challenges

Even with our map-matching improvements to the Davies algorithm, some challenges remain. Primarily, the choice of a density threshold for Davies is made globally across the entire map. If the chosen threshold value is too low, excess noise is added to the map in the form of spurious roads. If the value is too high, portions of the map with relatively low density are treated as spurious and removed, resulting in missing roads. This behavior is illustrated in Figures 6(e) and 6(f). Here, we can see that the F-score is largely constant over a wide range of density threshold values. However, by studying the components that make up the F-score (i.e., missing and spurious), the trade-off is clearly seen. As the density threshold increases, the proportion of spurious edges decreases (i.e., precision increases), while the proportion of missing edges increases (i.e., recall decreases) as low density roads are pruned from the map. We believe this insight is the key to further improvements to the KDE-based method—only marginal improvements will be made as long as the constant threshold used in the current algorithm remains.

2.4.2 Parameter Sensitivity and Implementation Details

Each of the algorithms described above includes several tuning parameters. We conducted a sensitivity analysis in order to determine which of those most significantly impact performance, by fixing the values of one parameter and allowing all others to vary. Some of the results of this analysis are discussed below.



(a) Cell size parameter (Davies Map-Matched).

(b) Density parameter (Davies Map-Matched).

(c) Volume parameter (Cao).

(d) Number of traces parameter (Cao).

(e) Distance parameter (Edelkamp).

(f) Number of traces parameter (Edelkamp).

Figure 7. Parameter sensitivity testing for the three reference algorithms.

2.4.2.1 Davies et al.

Our implementation of this algorithm closely follows the description in (22), with the additional modifications discussed in Section 2.4.1.1. The parameters for this algorithm are: cell size, density threshold, kernel bandwidth, and number of traces used to infer the map. Figures 7(a) and 7(b) illustrate this algorithm's sensitivity to cell size and density threshold. In Figure 7(a) we see that a smaller cell size will always produce a better result with a fixed bandwidth kernel, as this simply improves granularity. We can see in Figure 7(b) that the density threshold is not a particularly sensitive parameter over a wide range of values, and this is directly related to the trade-off between missing and spurious roads, as discussed in Section 2.4.1.2. However, once the threshold reaches a value that is too high, no roads remain in the map and F-score reaches zero.

2.4.2.2 Cao and Krumm

Our implementation of this algorithm follows the description in (20) closely, with map inference being preceded by clarification. The parameters for this algorithm are: edge volume threshold, location distance threshold, location bearing difference threshold, and number of traces used to infer the map. Figures 7(c) and 7(d) illustrate this algorithm's sensitivity to edge volume threshold and the number of traces used to infer the map. In Figure 7(c) we see that increasing the edge volume threshold (used for pruning spurious edges from the inferred map) increases map accuracy, as spurious edges are removed. However, we also see that if we prune too aggressively legitimate roads may end up being incorrectly removed, decreasing accuracy. We can see in Figure 7(d) that performance decreases with the number of traces

used to infer the map, a result of the increased noise that comes with a larger dataset, and this algorithm's inability to overcome that error.

2.4.2.3 Edelkamp and Schrödl

Our implementation of this algorithm follows the description in (15), with the exception that no intersection refinement or lane-finding is done. The parameters for this algorithm are: cluster seed interval, intra-cluster bearing difference threshold, intra-cluster distance threshold, and number of traces used to infer the map. Figures 7(e) and 7(f) illustrate this algorithm's sensitivity to intra-cluster distance threshold and the number of traces used to infer the map. In Figure 7(e) we see that performance improves with increasing intra-cluster distance, as this increases the clusters' resistance to noise in the GPS traces. We can see in Figure 7(f) that performance decreases with the number of traces used to infer the map, as similar to Cao, the larger dataset includes more noise which this algorithm is unable to overcome.

2.4.3 Algorithm Runtime

The runtime of these algorithms vary dramatically, due to differences in algorithmic complexity. In particular, Cao suffers in dense neighborhoods where it exhibits quadratic complexity. On a subset of 100 traces, Cao finished in 2.5 hours, Edelkamp in 73 seconds, and Davies in 8 seconds. Our map-matched version of Davies finished in 106 seconds. On the full set of 899 traces, Cao required 2.5 days, Edelkamp 15 minutes, and Davies 25 seconds. The map-matched version of Davies finished in 14 minutes.

2.5 Conclusion

Robust quantitative evaluation methods, and rigorous comparison against prior work are important tools for furthering any field of scientific inquiry. Using the new tools presented here, we compared three algorithms from the literature. Overall, the algorithm by Davies et. al (22) was found to significantly outperform the others under a variety of conditions. Using our quantitative evaluation method, we were able to identify opportunities for further improvement to this algorithm. Some of these were implemented and evaluated here yielding a further, significant performance improvement.

In Chapter 4 we will leverage these lessons to build a new map inference algorithm that is resistant to both noise and disparity in GPS data. First, however, in Chapter 3 we will look at extending Davies et al.'s algorithm to the problem of inferring road maps from GPS trace data that is sampled infrequently (or, "sparsely"). This is a phenomenon we did not encounter with our UIC shuttle dataset, and as we will see, it is one that poses significant additional challenges.

2.6 Copyright Information

CHAPTER 3

INFERRING ROAD MAPS IN THE FACE OF DATA SPARSITY

With the ever-increasing use of GPS-enabled smartphones and telematics systems, vehicles equipped with these devices now cover road networks around the globe. Fortunately, GPS traces are often readily available from these vehicles. For example, taxi dispatching that utilizes GPS technology has been around for more than a decade (34), and real-time taxi traces are available in several major cities (35; 36; 37; 38), enabling an unprecedented opportunity to infer maps for many major population centers.

However, the nature of these GPS traces differs significantly from those used in the literature. Most importantly, due to bandwidth and storage cost concerns, sampling rates much lower than 1 Hz are common (39; 38). Although the much larger volume of data may well compensate for this limitation, none of the algorithms referenced in Chapter 2 are designed for this type of sparse sampling. For example, a key design tradeoff in map inference algorithms is whether to use GPS samples alone (15; 10; 16; 23; 24; 19) or the segments that connect consecutive samples (22; 12; 20). Using individual samples alone, sparsely sampled roads may be fragmented or missed entirely. Using segments, we may recover sparsely sampled roads, but we may also recover segments that pass over areas where no road actually exists. For example, two samples taken on either side of a right-angle turn may suggest a non-existent diagonal street.

Figure 8 illustrates this tradeoff with sparse taxi data from Shanghai (40). Here, there are 2,300 taxis reporting their locations, with a sampling rate of 16 seconds when vacant, and 61

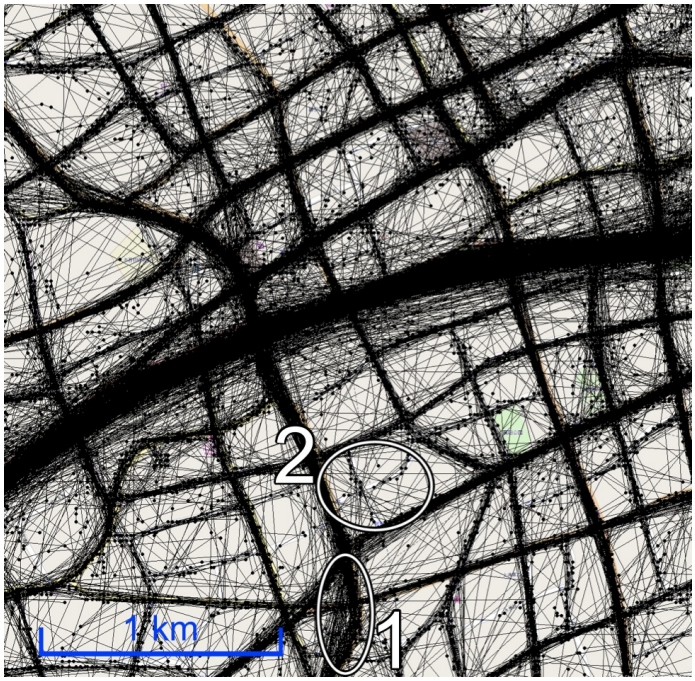


Figure 8. Sample of raw GPS data from 2,300 taxis in Shanghai.

seconds when occupied (38). We draw each sample as a black dot, and connect consecutive samples by line segments. As we can see, most dots and lines are well-aligned with roads. The thick horizontal curve in the center corresponds to an elevated freeway, where its greater thickness can be attributed to several factors, including road width, number of samples, and the lines connecting these samples incorrectly widening the curve inward. This "widening" problem is clearly seen in the highlighted area "1" in Figure 8. Area "2" in Figure 8 illustrates the problem with having an uneven distribution of samples. Inside the area there are two local roads that intersect orthogonally, which are covered by samples with a density much lower than

major roads, almost on par with noise density. In this case, we cannot visually detect such roads without connecting consecutive samples or obtaining substantially more data.

In light of these issues, in this chapter we investigate two techniques for inferring maps from sparse data: (i) a sparse-data adaptation of the KDE algorithm by Davies et al. (22), and (ii) a trace clustering algorithm specifically designed for sparse data by Liu et al. (41). For completeness, we also examine Edelkamp and Schrödl's k-means algorithm (15), and provide a qualitative and quantitative comparison of these techniques on two sparsely sampled datasets.

3.1 Map Inference Using Sparse Data

For sparse GPS traces, the straight line between successive samples does not necessarily correspond to the actual road driven. Consider a car turning a corner: with one sample well before the turn, and one sample well after the turn, the straight line between samples would be likely to cut straight through a building. There is a natural precision/recall tradeoff between using the sample locations alone vs. also using the line segments of the trace paths: if an algorithm uses the samples alone, it risks having lower recall for roads less traveled. For example, consider area "2" labeled in Figure 8. Such a road may be better detected by an algorithm that takes into account the line segment between samples. However, due to turns and curvy roads, this is likely to result in lower precision, as illustrated by area "1" of Figure 8.

In this section, we describe two map inference methods for sparse trace data. One is a modification to the existing KDE algorithm by Davies et al. (22), and the other is a trace clustering algorithm by Liu et al. (41), designed specifically for sparse data.

3.1.1 Kernel Density Estimation

We implement two variants of the KDE map inference algorithm from (22): point-based KDE and segment-based KDE. In both cases, the map is represented by a fine grid of cells. A two-dimensional histogram is produced by counting, in the case of point-based KDE, how many samples are within each cell, or in the case of line-based KDE, how many segments pass through each cell. Here, the line-based KDE is the original method as proposed in (22), and the point-based method is our modified version targeted at sparse data. As described in Section 2.1.4, this histogram is then convolved with a Gaussian smoothing function to produce a density estimate, which is thresholded, and then the road centerlines are extracted using Voronoi tessellation (27).

Besides the choice of using either points or line segments, the fundamental control parameter for the KDE algorithm is the *density threshold* used to find the road outlines. As discussed in Section 2.4.1.2, a high threshold may miss roads with few samples, and a low threshold may produce spurious roads due to noise. This problem is prominent with the taxi data, illustrated by Figure 8. We evaluate these design choices in detail in Section 3.3. To ensure portability of thresholds between datasets, in this chapter we express thresholds in terms of density percentiles, rather than absolute values. For example, grid cells with a density above the 80th percentile may be considered roads, independent of the size of the map, or number of sample points.

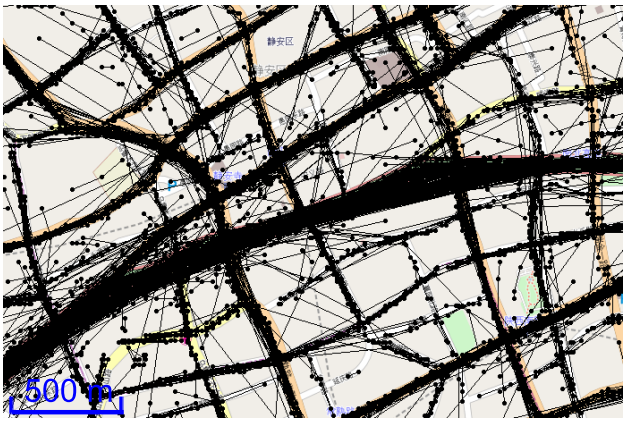
3.1.2 Trace Clustering Algorithm

To explore the use of line segments for inferring maps from sparse GPS traces, Liu et al. (41) designed a trace clustering algorithm named "TC1" (41). Its operation proceeds in four main steps, detailed below.

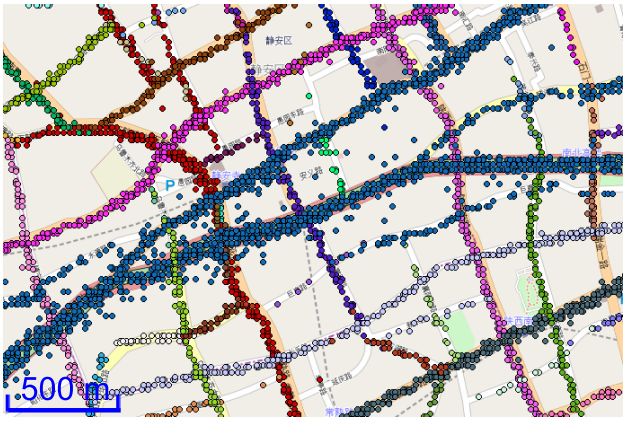
1. **Segment Selection.** As a first step, this algorithm selects a subset of trace segments from the raw GPS data that are likely to fall on an actual road, as opposed to across a building. These are selected using a simple heuristic: for a given pair of GPS samples (a, b), if a has roughly the same bearing as b then the vehicle likely followed a straight path from a to b, and is therefore selected. Figure 9(a) demonstrates the effect of segment selection on the dataset illustrated in Figure 8.

2. **Clustering.** After segment selection, the remaining segments are clustered using a method based on single-linkage clustering (42). Initially, each segment represents its own cluster, and clusters are iteratively merged that are similar in both orientation and geographic distance. The segments of Figure 9(a) have been clustered (represented by color) in Figure 9(b), and replaced by their end-points. After this step, segments are no longer used, and therefore we display each cluster by samples alone.

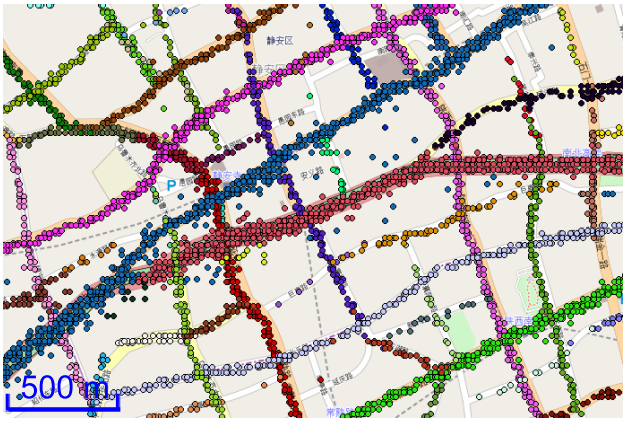
3. **Y-split separation.** The clustering in Step 2 performs well for orthogonal roads, but for roads that have Y-splits of a sufficiently narrow angle, multiple roads can become merged into one cluster. To separate these Y-splits, the major axis of each cluster is traversed, building a polyline incrementally with nodes at least 100 meters apart, selected so that pairs of adjacent segments do not turn excessively. The result of this process can be seen



(a) Trace segment selection.



(b) Clustering.



(c) Y-split separation.

Figure 9. Three steps of the trace clustering algorithm TC1.

in Figure 9(c), where the large blue cluster that covers the horizontal freeway in Figure 9(b) is now separated into several sub-clusters representing distinct roads, as shown by their different colors.

4. **Centerline fitting.** Lastly, the centerline of each road is computed using B-spline fitting (10), resulting in road centerlines that curve naturally to fit the actual GPS trace points, which we display later in Figure 16(c). Similar to the density threshold used by the KDE algorithms, here a global threshold determines the minimum support needed to fit a cluster, called the "support threshold" hereafter.

3.1.3 Other Map Inference Algorithms

Map inference algorithms designed for high-density, high-accuracy data, typically work poorly on low-density or low-accuracy data. For example, Edelkamp and Schrödl's k-means (15), and Cao and Krumm's trace merging (20) algorithm implementations in Chapter 2 produce numerous spurious roads when applied to high-error samples obtained near high-rise buildings.

These algorithms perform even worse when both error and sampling interval are high. For example, we applied Edelkamp and Schrödl's k-means algorithm (15) to our Shanghai taxi data set. While the clustering step is able to locate many clusters near road centers after several iterations, the topology discovery step produces an almost completely connected graph among all cluster centers. This is because a vehicle may travel through multiple clusters between two sparse samples, and the algorithm simply connects the two end-points. Finding the correct sequence of clusters it passes through is non-trivial.

3.2 Evaluation Framework

To evaluate the quality of the results from our map inference algorithms, we measure their performance both qualitatively and quantitatively, using manually verified portions of OpenStreetMap (32) as our ground truth data. Unfortunately, because the TC1 algorithm (41) is only capable of inferring the geometry of a map (and not its topology), in order to measure quantitative performance we must resort to a purely geometric evaluation.

As was the case with our evaluation technique presented in Section 2.2, here we perform our geometric-only quantitative evaluation by measuring the precision and recall of the inferred road map M with respect to a ground truth map $Truth$. To do this we determine the "true positive length," $tp = M \bigcap Truth$, as a measure of common road length which we define shortly. Then we compute,

$$precision = \frac{tp}{||M||},\ recall = \frac{tp}{||Truth||}$$

where $||\cdot||$ measures the total length of the roads in the set. We then combine these two measurements into a single performance value by computing their F-score (31).

To compute the intersection $M \bigcap Truth$, we start by traversing each ground truth road segment $t \in Truth$ and take samples at 1-meter intervals. We then calculate how many of those samples fall within a perpendicular distance threshold m of a (nearest) segment $t' \in M$, such that the orientation difference between t and t' does not exceed 60°. The angle restriction avoids matching portions of roads that cross at intersections, but allows for substantial misalignment of the inferred road.

In order to additionally measure the typical road centerline offsets from the ground truth, we separately compute a histogram of perpendicular road distances of each 1-meter sample from each segment $t' \in M$ to the nearest road segment $t \in Truth$. For an inferred map with accurate centerline positions, we should obtain a sharp histogram within a small distance from the centerline.

3.3 Experimental Results

In this section we perform a qualitative and quantitative evaluation of both KDE variants and the TC1 algorithm, on two separate datasets described below.

3.3.1 Datasets and Ground Truth Map

For our evaluation we use two different datasets: 118 hours of traces from the UIC campus shuttles in Chicago (as in Chapter 2), and one month of traces from 2,300 taxis in Shanghai (40). The Chicago data is sampled at 1 Hz, using commodity, low-cost SiRF-3 GPS receivers installed in 13 vehicles. Spatial resolution of this data is 0.000001 degrees latitude and longitude (less than 0.1 meters), offset by GPS noise with a standard deviation of 3.3 meters. Additionally, as discussed in Section 2.3.1, certain portions of the Chicago data contain significant noise (tens of meters) due to nearby high-rise buildings. This dataset represents the best quality data one might expect to receive from passive data collection using non-survey grade equipment. However, the coverage of this dataset is limited, due to the small number of vehicles and their regular driving patterns.

The Shanghai data was collected by 2,300 taxis, providing excellent coverage of freeways and arterials. However, compared to our Chicago data, the quality of the recorded traces is decidedly

less stellar. The taxis report their location at various intervals, the most common being 16 seconds (when vacant) and 61 seconds (when occupied). These longer sampling intervals are quite common among the taxi dispatching systems we are aware of (35; 39; 36; 37). In addition to this, the spatial resolution of these traces varies between 0.0001 and 0.0002 degrees latitude and longitude, or about 10–20 meters, and bearing is reported at a resolution of 45 degrees (38). We speculate that the taxis were using an early generation of GPS devices, resulting in these poor specifications. The test-area for our Shanghai dataset is shown in Figure 8.

For our ground truth, we select portions of OpenStreetMap (OSM) data that we have manually verified. Curiously, judging by our raw GPS traces, the OSM map in the Shanghai region suffers from a bias to the northeast, which is visible in Figure 8 when viewed at high magnification. We observe that most roads in Shanghai agree with the aerial images and were likely created by manually tracing the shape of the roads on the satellite imagery. This suggests that the aerial images are misaligned (see (38) for more detail). Based on a visual comparison with the locations reported by the taxi fleet, we shift the ground truth map 15 meters to the east and 12 meters to the north for a more faithful evaluation.

As discussed in Section 3.1, we need to determine the density threshold for our KDE algorithms and the support threshold for TC1. In addition to these map inference control parameters, we also need to pick a *matching distance threshold* for our evaluation (i.e., the maximum distance between inferred and ground truth roads that we consider as a match). Since our Chicago dataset is relatively small in terms of the number of traces and area covered, we do not separate training and testing data for parameter tuning. Instead, we set the matching distance

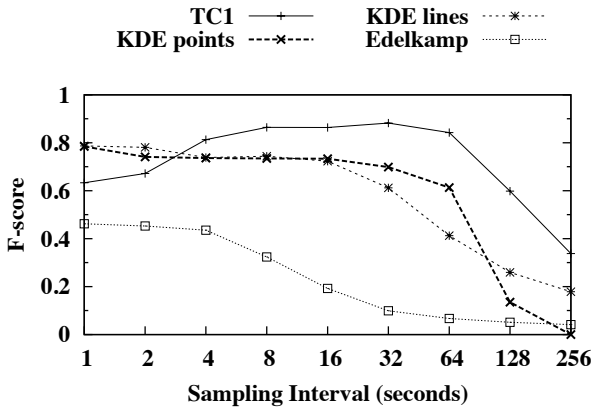
threshold to 20 meters, based on typical GPS error ranges and road widths (38), and manually select the optimal density and support thresholds for our experiments. We use this dataset primarily to evaluate the accuracy of our algorithms under different temporal resolutions.

On the other hand, our Shanghai dataset covers the entire metropolis for one month. Here, we first examine each algorithm threshold separately to understand its effect. Then we use data from a different area in Shanghai for training, and apply the tuned parameters to our test set for a fair comparison of all algorithms. The Shanghai GPS data and its road network are decidedly more diverse than our Chicago dataset. For example, while the Chicago dataset covers only straight roads aligned to the four cardinal directions, the Shanghai dataset covers curved roads, Y-splits, five-way intersections, and several different road types. Ultimately, "stress testing" using the Shanghai data will exhibit many important issues for map inference using sparse data.

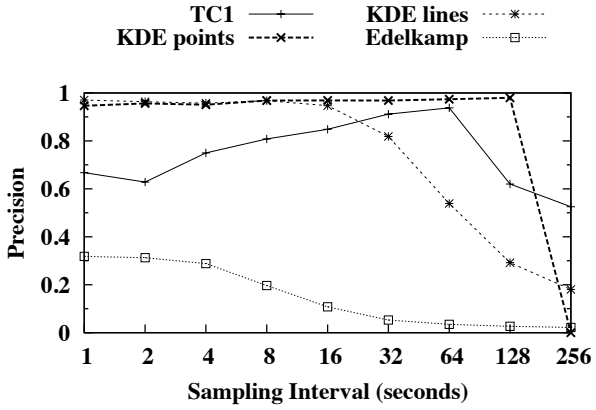
3.3.2 Chicago Shuttle Bus Dataset Results

The Chicago dataset in its original form has high resolution in both the spatial and temporal dimensions. To better understand how temporal resolution impacts map inference, we vary the temporal resolution of this data via sub-sampling. Figure 10(a) shows the F-score of each algorithm at sampling intervals of 1, 2, 4, ..., 256 seconds.

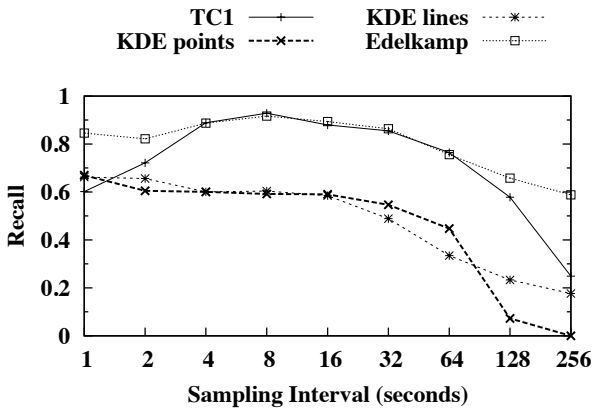
As the sampling interval increases, performance of the KDE-based methods gradually decreases, with the line-based method ("KDE lines") dropping off considerably when sampled at 32 second intervals and beyond. Intuitively, for larger intervals, the line-based method is likely to produce many diagonal segments, cutting across buildings instead of following the roads. This intuition is corroborated by Figure 10(b), where the precision of the line-based



(a) F-score for varying sampling interval.



(b) Precision for varying sampling interval.



(c) Recall for varying sampling interval.

Figure 10. Performance results for all algorithms on the Chicago Shuttle Bus dataset.

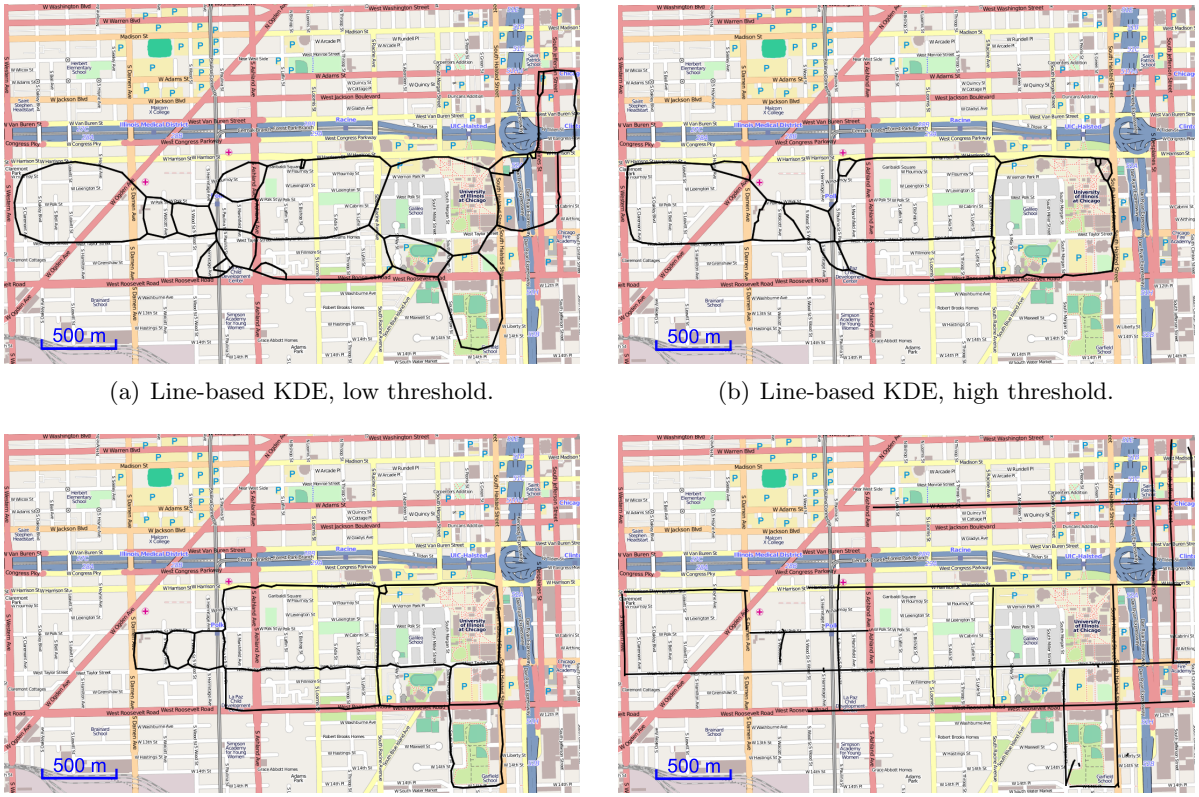
KDE method drops off quickly at 32 seconds or more, while recall decays more slowly, as seen in Figure 10(c).

The point-based KDE variant ("KDE points") shows somewhat increased tolerance for temporal sparsity, with its F-score being largely maintained up to the 64 second interval. Beyond 64 seconds, the decreased density fails to fully cover the roads, resulting in a map with poor recall. However, for the roads it does infer, their precision remains high up until 256 seconds, where no roads were found at all. Overall, both KDE methods suffer from relatively low recall, as a large section of the ground truth map is very lightly traveled compared to the rest of the map, which the global density threshold discards as noise.

The TC1 algorithm offered the best overall performance across the range of sampling intervals, with both high recall and precision. However, interestingly, TC1's performance does not consistently improve with higher sampling rates. Anecdotally, we found that dense data at intersections sometimes linked two perpendicular roads into one cluster, with the resulting centerline matching neither road.

Edelkamp and Schrödl's k-means algorithm (15) ("Edelkamp") produced poor results overall. We believe that this class of algorithm is unsuitable for sparse data, due to its strong tendency to produce spurious segments as data sparsity increases. This is evidenced by the very low precision scores achieved by this algorithm in Figure 10(b). Due to its poor performance in this initial evaluation, we omit Edelkamp from any further comparisons.

Figure 11 shows the road maps inferred by all of our algorithms at a 60-second sampling interval. In Figure 11(a) we can see that the line-based KDE method suffers from a large number



(a) Line-based KDE, low threshold.

(b) Line-based KDE, high threshold.

(c) Point-based KDE, low threshold.

(d) TC1.

Figure 11. Map inference results in Chicago with 60-second sampling interval.

of spurious edges, as corners are pulled inward along shuttle routes. While a higher density threshold mitigates this problem (see Figure 11(b)), it results in reduced coverage instead. In Figure 11(c) we see that the point-based KDE method does not suffer from the "corner problem" and produces a high-precision map. However, because it is unable to infer roads that are not *entirely* covered by samples, many ground truth segments are missing. Overall, since the point-based KDE method is less prone to producing spurious density, it tends to produce better results with a low density threshold. Finally, in Figure 11(d) we see that TC1 has good coverage, producing centerlines that match the road shape well as a result of spline fitting. Qualitatively, we observe a large number of small gaps where road segments meet—an artifact of this algorithm deliberately discarding turns. However, this does not impact precision and recall, and simply connecting nearby segments may resolve this issue.

3.3.3 Shanghai Taxi Dataset Results

The Shanghai dataset presents a much larger and more complex problem than the Chicago dataset. Here, we first study how the matching distance threshold affects the evaluation results. Recall that the Shanghai dataset represents locations in 0.0001–0.0002 increments of latitude and longitude, resulting in 10–20 meter spatial resolution.

Figure 12 shows, for each algorithm, how the F-score varies with the matching distance threshold. These results are based on a 6-hour dataset from the area shown in Figure 8, and we manually select the density thresholds for KDE methods and the support threshold for TC1 that produce the highest scores. A larger matching distance naturally increases the number of

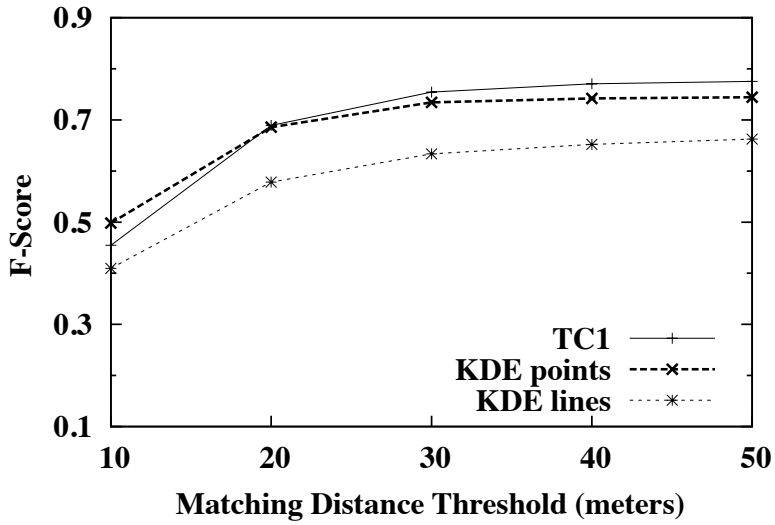


Figure 12. F-score for each algorithm on a 6-hour dataset, for varying matching distance thresholds. All three algorithms level out around 20 meters.

matched roads, allowing both precision and recall to monotonically increase. However, we note that the increase levels off near the 20 meter threshold used previously in this chapter.

Based on the same dataset and algorithm parameters, Figure 13 shows the distribution of distances between the inferred roads and the ground truth map for all three algorithms. Again, few inferred roads fall outside our default 20 meter matching distance threshold. However, there is a relatively heavy tail of distances over 60 meters, suggesting room for further improvement.

Figure 14 illustrates the trade-off in selecting the density threshold for KDE and the support threshold for TC1, using the same 6-hour dataset. As F-score is the harmonic mean of precision and recall, the best F-score is attained towards the top right corner, and the isoclines in the background connect precision-recall points with equal F-score.

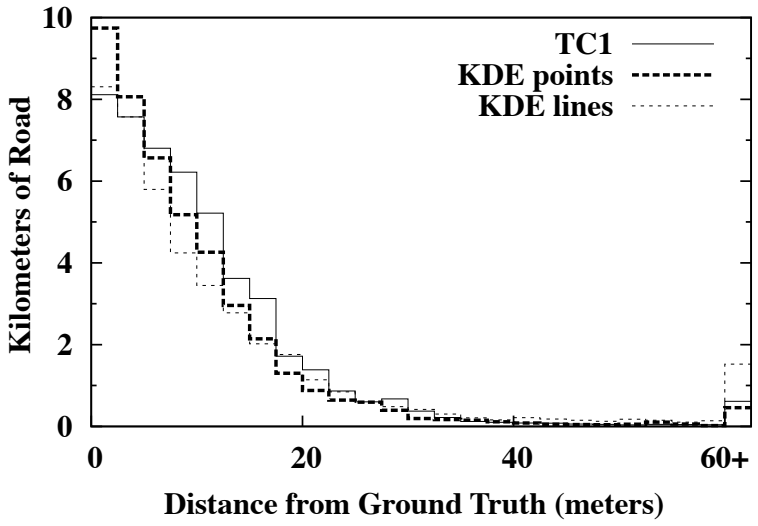


Figure 13. Histogram of distance between generated roads and ground truth.

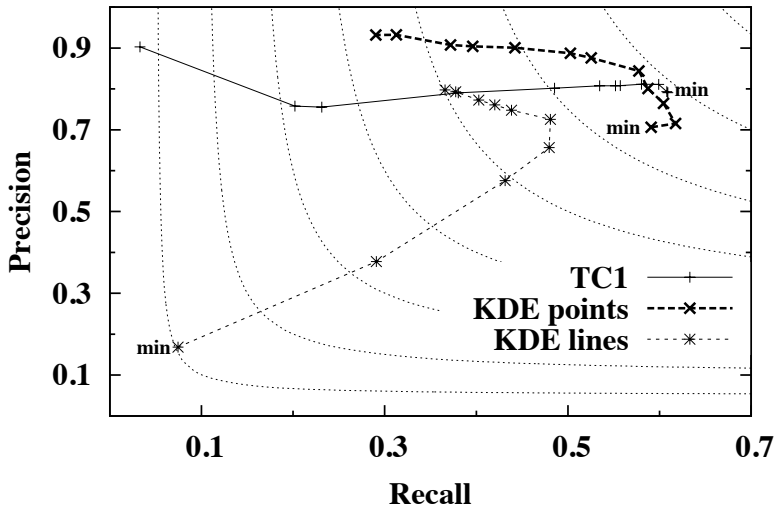


Figure 14. Precision/recall trade-off as density threshold is varied for KDE, and support threshold is varied for TC1.

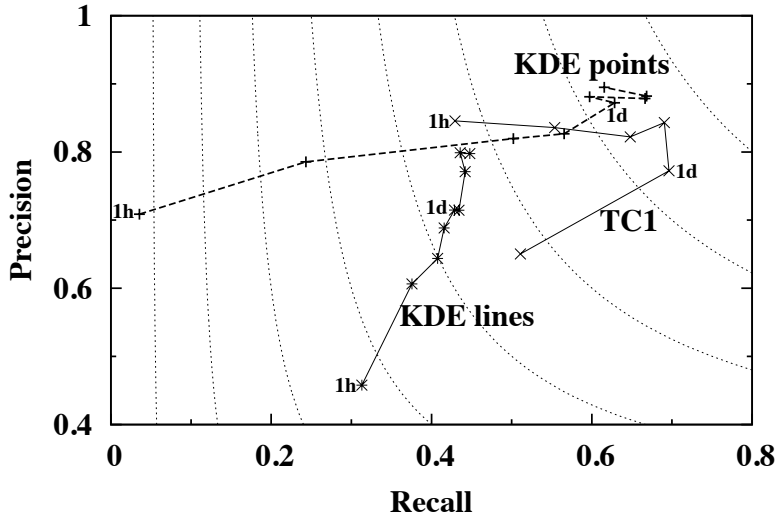
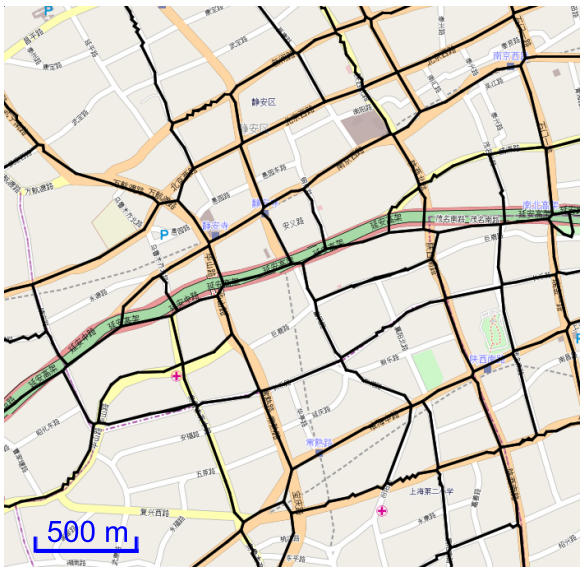


Figure 15. Precision and recall as the number of hours of Shanghai taxi data is varied.

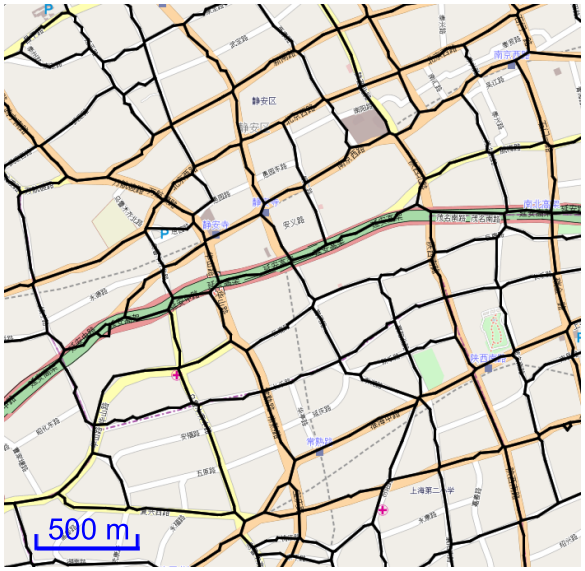
For the KDE methods we see that increasing the threshold reduces the number of spurious roads (increasing precision), but also discards roads less frequently traveled (reducing recall). Perhaps counterintuitively, the line-based KDE method is unable to match the recall performance of its point-based variant despite drawing the full line between each pair of points. This apparent contradiction is explained by the centerline-finding step in the KDE algorithm: with a low threshold, the line-based KDE method will produce large areas of solid "road surface," for which the centerline-finding method then produces a single road, significantly limiting recall for this algorithm. A lower support threshold increases the recall of TC1, but it is less correlated with precision. The pruning step of TC1 removes most of the noise, and the clusters obtained afterwards are generally accurate.

Next we study how each algorithm performs as we vary the amount of input data. We select a different training area of similar size in Shanghai for parameter tuning, and find the density thresholds for our KDE methods and the support threshold for TC1 that produce the best F-scores. For TC1, the same threshold is then used for the test area depicted in Figure 8. For our KDE methods, the density percentile of the threshold is preserved, rather than the absolute value. Figure 15 shows the precision, recall, and F-score results as we vary the input data from one hour to one month.

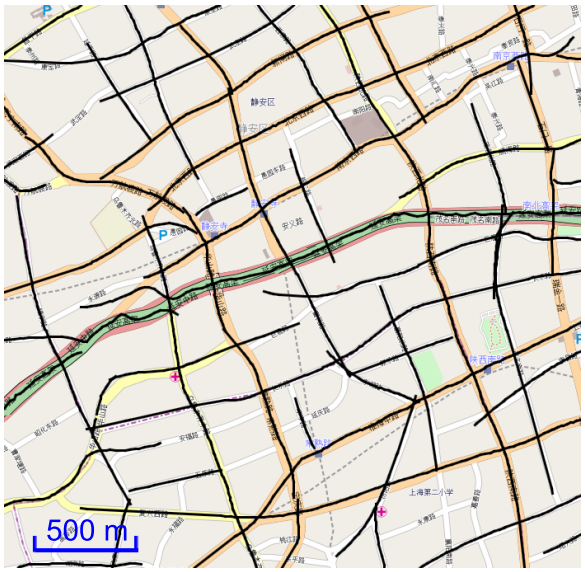
As one might expect, recall initially increases rapidly as trace coverage improves. However, increasing the amount of data has relatively little effect on precision. This is because we train the algorithms separately for each data size, which automatically adjusts the thresholds to compensate for the extra data. Overall, any performance improvement starts to level off around one week of data (for the point-based KDE method), and 24 hours (for TC1). For TC1, this can be explained by the use of single-linkage clustering: additional data increases the diversity of points, which raises the probability of nearby clusters merging incorrectly. We did not run TC1 for datasets larger than two days due to its high runtime complexity. For the point-based KDE method, over-fitting to training data appears to be one culprit: for the larger datasets, a lower threshold than that produced in training results in a considerably better map for the test area. This suggests that there is room for improvement in the way the point-based algorithm is trained. Overall, we find that the point-based KDE method and TC1 exhibit similar numerical performance, with the line-based KDE algorithm falling far behind.



(a) Line-based KDE (1 month).



(b) Point-based KDE (2 weeks).



(c) TC1 (8 hours).

Figure 16. Map inference results in Shanghai.

Finally, Figure 16 shows the inferred maps from all three algorithms, using the bounding box from Figure 8. This yields some valuable qualitative insight into the relative performance of the three algorithms. The line-based KDE method chooses a high threshold out of necessity, in order to distinguish between roads. This, in turn, reduces recall leading to a clean, but sparse-looking map (Figure 16(a)). The point-based KDE method produces what appears to be a very accurate map, with very few spurious road segments (Figure 16(b)). This is somewhat surprising, given the merely 90% precision reported in Figure 15. However, upon closer inspection, the point-based KDE map exhibits road jaggedness (an artifact of the KDE centerline-extraction method) in several places, significantly increasing total road length, thus reducing precision according to the definition in Section 3.2. The TC1 map (Figure 16(c)) shows very good coverage, but its precision is hampered by a number of spurious road segments. This map also exhibits the disconnected road segments discussed in Section 3.3.2.

3.4 Conclusion

In this chapter, we have highlighted the problem of inferring road maps from sparsely sampled GPS trace data, which will increasingly become available as a byproduct of other processes. Given the continuing desire to minimize communication costs and the very large install base of sparsely recording GPS tracking systems, we hold that there will be an ongoing need for inference algorithms that can deal with sparse data. Such robust algorithms may succeed partly by the large volume of data available in this form. While this chapter has assessed existing algorithms, the results show there is ample space for ongoing algorithms research.

3.5 Copyright Information

CHAPTER 4

INFERRING ROAD MAPS THROUGH GPS NOISE AND DATA DISPARITY

Inferred maps today do not offer the quality we expect of real maps. The existing state of the art in map inference (20; 15; 22) is highly sensitive to disparities in the number of trips made on different roads and to the high levels of GPS noise often encountered in urban areas, typically resulting in maps with either poor coverage or a multitude of spurious and misaligned roads. In Chapter 2 we compared the performance of several existing map inference algorithms on a dataset with significant noise and disparity. While each algorithm showed some strengths, the primary conclusion drawn was that all left significant room for improvement.

In this chapter, we propose a hybrid map inference method that combines the best aspects of these existing algorithms with several new innovations to produce the most accurate map inference method to date. Specifically, we: (i) develop an extensible, hybrid map inference pipeline with high tolerance to disparities in coverage and GPS noise, (ii) describe a gray-scale skeletonization algorithm for extracting map centerlines from a density estimate, (iii) present a trajectory-based topology refinement technique for edge pruning and intersection merging, (iv) demonstrate a trajectory-based geometric refinement technique for estimating intersection geometries, and (v) conduct a comparative evaluation of the proposed pipeline against several existing map inference techniques from the literature.

4.1 A Hybrid Map Inference Pipeline

In this section, we present a scalable, extensible map inference pipeline, and motivate its design. In addition to creating a high-performance map inference engine, we aim to provide a reusable framework within which improvements to individual components may be made and evaluated by the community. To this end, the source code of our map inference engine, together with example trace data and ground truth maps, are made available on our website (8).

The list below gives a general overview of our pipeline, where each step builds upon the output of the one before it. In this chapter, we propose an effective method for each step in turn, but expect future research to offer further improvements for each of the various steps.

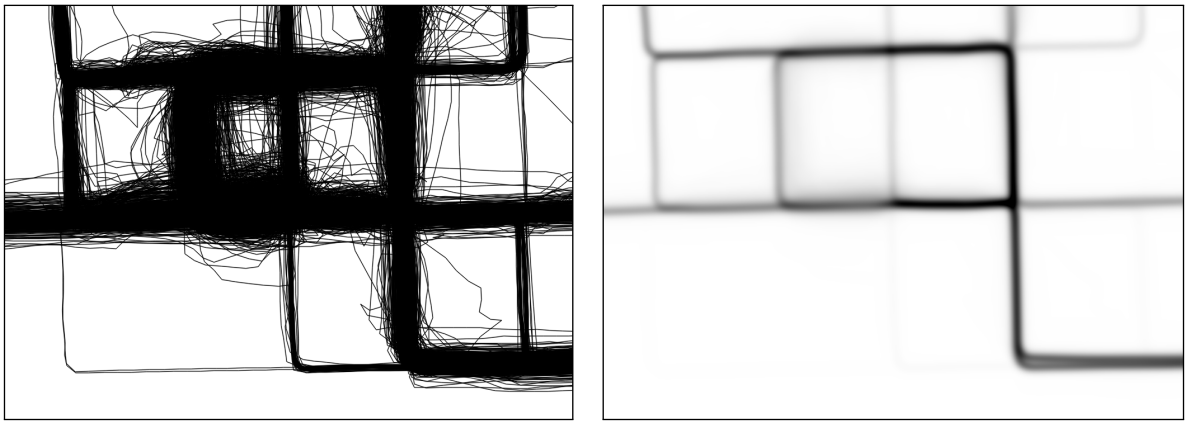
1. **Density Estimation.** The full set of GPS traces is passed through a kernel density estimator, producing a single density estimate for the area of interest.
2. **Initial Map Generation.** Road centerlines are extracted using our new gray-scale skeletonization algorithm.
3. **Trace Map-Matching.** Viterbi map-matching (33; 43) is used to associate each GPS sample in the original traces with an edge in the initial map, weighted by the mean density beneath each edge.
4. **Topology Refinement.** The map-matched traces are studied to remove low-confidence edges, merge redundant nodes, and infer allowable edge transitions.

5. **Geometry Refinement.** Intersections and road segments are aligned, turn lanes are extracted, transforming our topologically-accurate road map into a more geometrically-accurate finished map.

In a major departure from prior work, we combine initial density processing as in (22; 25; 23; 24) with subsequent trajectory processing (20; 15; 16; 17; 18; 19; 12). As shown in Chapter 2, density processing holds a significant advantage over trajectory processing in terms of robustness to noise and computational complexity, both important considerations as we grow the amount of trace data used. Density estimation's ability to very efficiently consider all traces simultaneously allows us to find an optimal road skeleton, rather than resorting to greedy approximations.

Existing density-based approaches are highly sensitive to density disparities between roads—we address this in Section 4.3. More critically, density-based approaches are poorly suited to detecting turn restrictions and grade-separated roadways, as they discard the relationships between points. By preserving these relationships, trajectory-based techniques are better able to perform fine-grained trajectory analysis such as lane detection (15; 10; 11), allowable turn detection (10), and spline fitting of road curvatures (15; 10; 16; 19). However, the prohibitive complexity of computing globally optimal solutions based on trajectories has led to a variety of greedy solutions in the literature. None of these are robust to noisy GPS data, resulting in poor map quality (2).

By map-matching the original traces to a density-based map in Step 3, we recover the relationships between successive samples without the noise sensitivity and scalability problems



(a) Binary mask of original traces. Coverage density varies by three orders of magnitude, including very significant noise in some areas.

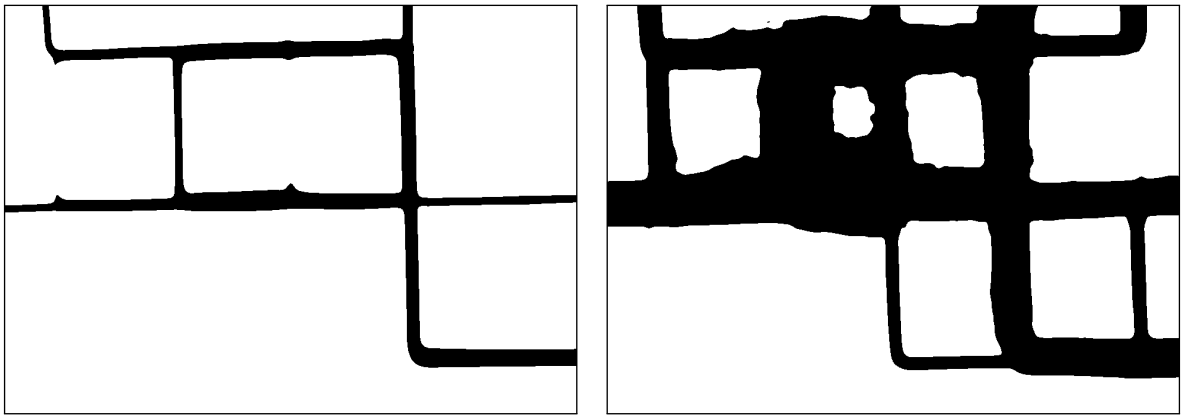
(b) Kernel density estimate. Darker regions correspond to frequently traveled road segments. Some roads are very faint.

Figure 17. Kernel density estimation produces a continuous distribution out of a noisy set of GPS traces.

associated with trajectory-based map inference algorithms. One may interpret map-matching noisy traces to a noise-resilient scaffold as a replacement for the clustering or merging step in existing trajectory-based methods. After matching, we may safely assume that every point matched to an edge is a (potentially noisy) sample from a single road. On rarely traveled roads, however, the noise problem persists: determining whether an underlying road is accurately represented by a single, potentially noisy trace, is an open problem. We now discuss each step in more detail.

4.2 Density Estimation

In the first stage of the pipeline, the full set of GPS traces is condensed into a single two-dimensional density estimate using the KDE algorithm described in Section 2.1.4.



(a) Binary map image derived by applying a high threshold to the KDE.

(b) Binary map image derived by applying a low threshold to the KDE.

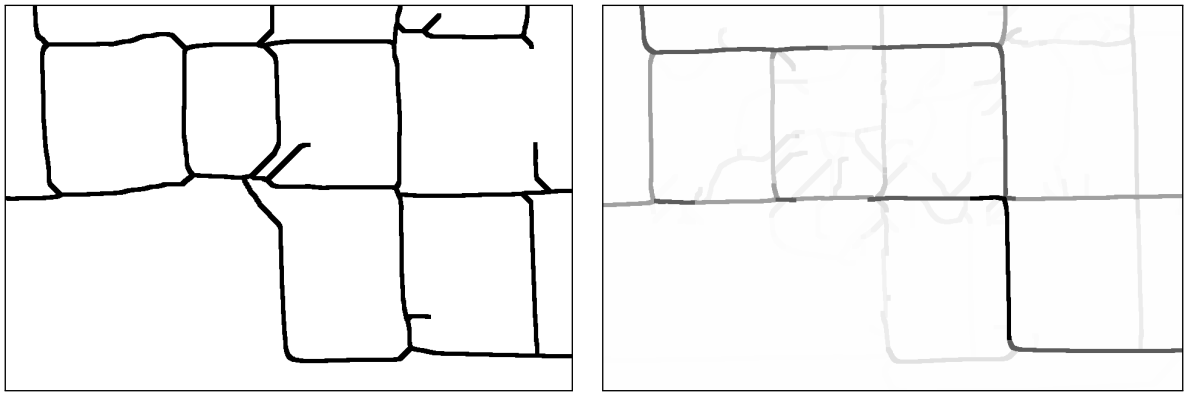
Figure 18. Simple binary thresholding does not work well, as no single threshold achieves both high accuracy and high coverage.

Figure 17(a) shows a sample of raw GPS traces, with its corresponding kernel density estimate shown in Figure 17(b). The density estimate shows clear and smooth peaks along the most heavily traveled roads in our dataset. In the next section, we extract an initial road network based on the density estimate computed here.

4.3 Gray-scale Skeletonization for Road Centerline Finding

As shown in Figure 18(a) a simple binary threshold may be used to produce a binary mask of the most popular roads. However, as seen in Figure 18(b), simply lowering the threshold to include less popular roads also admits a great deal of GPS noise in some areas, creating a difficult dilemma.

The canonical skeletonization algorithm, due to Zhang and Suen (44), produces a characteristic skeleton from a binary image, as illustrated in Figure 19(a). As mentioned above however,



(a) Skeleton from binary image.

(b) Gray-scale skeleton from KDE.

Figure 19. Binary skeletonization vs. gray-scale skeletonization. Gray-scale skeletonization produces more edges, but each edge is more accurate, and annotated by a gray-scale level.

no single threshold can produce a binary image that is both inclusive and accurate in the presence of density discrepancies and GPS noise. Below, we extend the original skeletonization technique to multi-intensity (gray-scale) images, so that we may produce a skeleton without the use of a binary threshold, resulting in the gray-scale skeleton seen in Figure 19(b).

At a high level, our algorithm repeatedly performs the binary skeletonization operation, once per integer density level, starting with the maximum density. At each level, new parts are potentially added to the skeleton, but none are ever removed. This process accurately captures the centerlines of high-density ridges, and at the same time it is able to produce centerlines for roads that were only driven once. More formally, the gray-scale skeletonization algorithm proceeds as follows.

P_9 (i - 1, j - 1)	P_2 (i - 1, j)	P_3 (i - 1, j + 1)
P_8 (i, j - 1)	P_1 (i, j)	P_4 (i, j + 1)
P_7 (i + 1, j - 1)	P_6 (i + 1, j)	P_5 (i + 1, j + 1)

Figure 20. The 8-neighborhood of pixel P_1, used in the process of skeletonization.

4.3.1 Algorithm Description

Let $\mathbf{D}(x, y)$ be the density at location x, y. For each density level $l \in l_{max} \ldots 1$, let $\mathbf{T}_l$ be a binary (thresholded) image, such that $\mathbf{T}_l(x, y) = 1$ iff $\mathbf{D}(x, y) \geq l$. Our algorithm recursively produces a skeleton image $\mathbf{S}_l$ for level l, such that $\mathbf{S}_l = skeletonize(\mathbf{T}_l + \mathbf{S}_{l+1})$, where $\mathbf{S}_{l_{max}} = skeletonize(\mathbf{T}_{l_{max}})$. Thus, the skeleton pixels at the end of the process have values in the range $l_{max} \ldots 1$ depending on the level at which they were introduced.

The procedure *skeletonize* repeatedly "thins" the image until it converges. Each thinning step consists of applying several local conditions to any pixel of value 1 (unit pixels) in the image, to decide whether to set the pixel to zero. Here, we use the notation from (44), partially illustrated in Figure 20. For every pixel $P_1 = 1$, the decision to keep it or set $P_1 := 0$ is made based on the values of the neighboring pixels P_2 through P_9. Each major thinning iteration consists of two sub-iterations. In the first sub-iteration, $P_1 := 0$ iff all of the following conditions are satisfied:

$$B(P_1) \geq 2 \tag{a}$$

$$B(P_1) \leq 6 \tag{b}$$

$$A(P_1) = 1 \tag{c}$$

$$P_2 \cdot P_4 \cdot P_6 \neq 1 \tag{d}$$

$$P_4 \cdot P_6 \cdot P_8 \neq 1, \tag{e}$$

where $A(P_1)$ is the number of $(0, \geq 1)$-pairs in the ordered set $P_2, P_3, \ldots, P_9, P_2$, and $B(P_1)$ is the number of non-zero neighbors of P_1. In the second sub-iteration, Conditions (d') and (e') are substituted for Conditions (d) and (e):

$$P_2 \cdot P_4 \cdot P_8 \neq 1 \tag{d'}$$

$$P_2 \cdot P_6 \cdot P_8 \neq 1. \tag{e'}$$

Upon convergence, any remaining unit pixels that satisfy the following rule are set to zero:

$$B(P_1) \geq 7 \tag{f}$$

4.3.2 Algorithm Intuition

By proceeding level by density level, and preserving any skeleton pixels from higher levels as we progress toward lower densities, we ensure that high-density pixels are accurately represented in the final skeleton. We now briefly summarize the purpose of each of the conditions above.

Conditions (a)–(e') closely resemble those in (44), adjusted to accept values other than $\{0, 1\}$. Specifically, Condition (a) preserves unit pixels at the end-points of lines. Condition (b) forces thinning to progress inward from the edges of contiguous unit pixel areas, consistent with the thinning concept. Condition (c) preserves any unit pixel that is the sole bridge between two or more non-zero pixels. Here, two or more $(0, \geq 1)$-pairs split the 8-neighborhood into disjoint sets, connected only through P_1. Finally, alternating between Conditions (d)–(e) and (d')–(e') enforces a degree of synchronization in this otherwise highly parallelizable algorithm. This is to avoid a race condition which may thin two-pixel wide lines in a single step, rather than thinning these to a one-pixel wide line.

In the final step, applying Condition (f) to the image after convergence is necessary to counter a phenomenon that occurs only in gray-scale skeletonization. Occasionally, an area surrounded by high-density ridges may be entirely filled with cells of non-zero density. Using binary skeletonization, the high density ridges would first be flattened to level 1, and the image thinned until a single line remained through the middle of the low-density area. In gray-scale skeletonization however, the high-density ridges appear early in the process, and cannot be thinned away later. Due to Condition (b), these ridges then prevent pixels in the surrounded low-density area from being removed. Condition (f) returns the image to its desired skeleton shape by hollowing out any such surrounded areas in a single step.

4.3.3 Performance Optimizations

Both the density estimation and skeletonization techniques described are highly parallelizable algorithms that are well suited for GPU or MapReduce implementations. However, as

described, the gray-scale skeletonization algorithm runs one binary skeletonization per level. For maps with very high densities, this may grow to an unmanageable number of iterations, even on GPU hardware. In practice, we have found that restricting the levels to powers of two produces largely identical initial maps, with an exponential improvement in execution time.

4.3.4 Edge Extraction from Skeleton Image

Given a skeleton image, our goal is to produce an initial road map consisting of nodes and edges. We use the "combustion" technique described in (24) to associate each pixel with an edge, and the Douglas-Peucker algorithm (45) to produce the edges that make up the shape of each road segment.

4.4 Density-Aware Map-Matching

We now match our original traces to a contiguous sequence of edges from the initial map produced by the skeletonization algorithm above. This accomplishes two things. First, it sets an upper bound on the number of nodes and edges. Later steps may prune and shift nodes and edges, but may not add to the topology. This avoids the tendency of trajectory-based techniques to produce spurious edges. Second, by assigning each point to an edge, it reduces the computational complexity and improves the parallelism of downstream methods that can now operate on each edge independently.

Our map-matching technique uses Viterbi's algorithm, and is based on (33). Relative to (33), we make the following modifications. To enforce a speed limit, every edge is represented by several consecutive fixed-length states, each of which must be traversed before transitioning to a new edge. This replaces the speed limit heuristic in (33). Moreover, transition probabilities,

which were uniform in (33), are instead assigned values proportional to the edge weight, computed as the mean density level of the pixels that make up the gray-scale skeleton. Weighting transitions based on edge weight encourages the matcher to use popular roads when the trace allows it, effectively reducing the number of traces that traverse spurious roads.

4.5 Topology Refinement

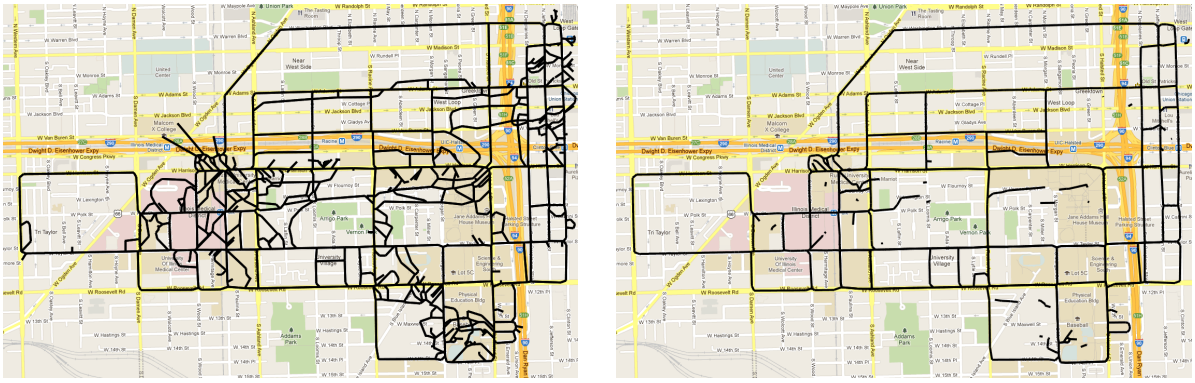
In the topology refinement step, the initial map produced through density processing is updated based on the map-matched traces. Edges with zero or one traversals are discarded, pairs of intersections are merged when trace evidence supports it, and statistics are produced to determine the allowable transitions between edges. In the sub-sections below, we describe each step in more detail.

4.5.1 Pruning Spurious Edges

Through edge pruning, edges that see less than two well-matched traversals are removed. More formally, for each map-matched trace t and edge e, we compute n_e^t, the number of traversals of e such that $RMSD(\tau, e) < RMSD_{max}$, where τ is a traversal of e and

$$RMSD(\tau, e) = \sqrt{\frac{1}{|\tau|} \sum_{p \in \tau} dist(p, e)^2}$$

Here $p \in \tau$ are GPS points, and $dist$ is the distance between p and the nearest point on e. We only consider traversals with $RMSD(\tau, e) < RMSD_{max}$ good matches; these are used as evidence of a road segment's existence.



(a) Map before pruning. (b) Map after first round of pruning.

Figure 21. Our trajectory-based pruning process. Using the original traces map-matched against the initial map removes most spurious edges.

Aggregating n_e^t across all traces, we have $n_e = \sum_t n_e^t$, the number of well-matched traversals for each edge. Intuitively, any edge with $n_e = 0$ is unlikely to represent an actual road. Such an edge may, for example, have been created by a noisy GPS trace, which was later matched to a more popular nearby road by our weighted map-matching technique in Section 4.4.

We argue that the same is true for edges with $n_e = 1$. With a single traversal, the trace will naturally fit the map edge perfectly. After all, the edge was most likely drawn based on that single trace. Thus, the $RMSD(\tau, e)$ is pointless when $n_e = 1$. For this reason, we require two traces to support the existence of any given road segment.

Figure 21 shows our map before (a) and after (b) this pruning process, clearly illustrating the effectiveness of this technique in reducing the number of spurious road segments.

(a) Intersection before collapsing nodes.

(b) Intersection after collapsing nodes.

Figure 22. By collapsing nearby nodes into intersections, the final map topology is produced.

4.5.2 Collapsing Nodes into Intersections

While the gray-scale skeletonization algorithm in Section 4.3 produces an accurate road skeleton for a given density estimate, the generated topology does not necessarily correspond well to typical road designs. One common case of this, caused by an uneven density distribution in the intersection, is illustrated in Figure 22(a). Here, a single four-way intersection is represented as two adjacent three-way intersections.

To address this common problem, we first sort all pairs of adjacent intersection nodes in order of increasing distance. We then consider collapsing each pair in order, replacing the pair's two m, n-degree intersections with a single (up to) $m + n - 2$-degree intersection at their mean location. If this refinement does not reduce the total number of well matched traces, it is made permanent. Figure 22(b) shows the result after collapsing. This process effectively transforms the map into a topologically accurate representation of the underlying road network.

4.5.3 Map-Matching Do-Over

After completing the pruning and collapsing steps above, all traces are map-matched once more, this time using the actual number of traversals rather than edge densities to compute transition probabilities. Any traces initially matched to now-deleted edges are re-matched to a more likely route, and edge pruning is performed once more. Here, the first pruning round breaks a number of spurious cycles, and the second round removes the remaining spurs after map-matching re-routes traces to more probable routes. The final result from this step is shown in Figure 30(d).

4.5.4 Detecting Allowable Edge Transitions

Finally, for each trace we compute a list of all adjacent pairs of distinct edges $e : d$, in order. We then compute the number of occurrences of each pair, $count(e : d)$ across all traces. To enforce allowable edge transitions we use a strict interpretation of this data: a transition from edge e to edge d is allowed iff $count(e : d) > 0$.

4.6 Geometry Refinement

In the geometry refinement step, the map is updated to model intersections in more detail, and to improve the alignment of the inferred map using its original traces. Here, the segment-level topology does not change—segments may shift, but they are not added or deleted. One minor exception to this are turn-lanes, which do contribute additional detail to the topology of an intersection, but do not add new road segments.

4.6.1 k-means Refinement

Our goal here is to simultaneously refine the entire map, including intersections, rather than refine each segment piecemeal. We propose a geometry refinement technique based on the k-means algorithm. The input to the k-means algorithm is the k initial means and a set of sample points to be clustered. We adapt k-means to the geometry refinement problem by: (i) creating an initial estimate based on the input map, and (ii) restricting clustering eligibility—which sample points may be assigned to what means, based on the map-matching results from Section 4.4.

4.6.1.1 Initialization

Key to a successful application of the k-means algorithm is a good initial estimate. Since the goal here is to refine an existing map, we base our estimate on this map. We produce two classes of means: intersection means and segment means. For each intersection and end-point (i.e., dead-end) in the input map, we add one intersection mean to our initial estimate. Each intersection mean is associated with all segments incident on the intersection. For each road segment in the input map, treating each direction separately, we produce $\lceil \frac{L}{m} - 2 \rceil$ means, where L is the length of the segment, and m is the maximum distance between means. The first and last points of the segment are excluded, as these are already represented by the intersection means. The rest are uniformly distributed along the length of the segment. These means are associated only with the segment from which they originated.

(a) Same intersection with directional street segments.

(b) Same intersection with turn-lanes added.

Figure 23. Geometry refinement, separating directions and adding turn lanes.

4.6.1.2 Assignment

Each GPS sample is then assigned to its nearest eligible mean. The set of eligible means include those from the segment that the sample was matched to, as well as the intersections or end-points that delimit the segment at each end. Note that intersection means are eligible for GPS sample assignment from all segments incident on that intersection. This optimizes intersection alignment, taking all neighboring segments into account. Simultaneously, segment-based means automatically find the shape of each road segment.

4.6.1.3 Update

In the update step, each mean is moved to reflect its new sample membership. The typical update function simply takes the mean location of all member samples as the new mean location. This can be further refined by first removing points that lie too far from the mean (outliers), and by taking into account the location of neighboring means.

This produces a map with correct lane separation, but with intersections still represented as points in the map topology. Where bi-directional segments exist, this leads to an hourglass-shaped intersection geometry (see Figure 23(a)). To produce a correct intersection geometry, we need to estimate each segment transition separately.

4.6.2 Estimating Transition Trajectories

As a naïve solution, simply replacing the intersection node with direct edges between segments produces a significantly improved map; the hour-glass shape is removed, while the topology is preserved. This approach however, does not accurately capture the individual shape of each lane, often leading to crooked-looking intersections. Below, we describe a solution that separately estimates the transition between each pair of segments, producing a complex, but accurate, intersection geometry.

For each pair of road segments, taken as a single "transition," we prepare a separate set of means according to the initialization method described above. We then perform k-means clustering again, this time using the transition means. Here, a given mean is eligible for assignment only if the current sample came from a matching transition. The generated transition segments and the original street segments are then merged, using a simple constant set-back from the intersection as the merge-point. Figure 23(b) shows the final result, after adding turn-lanes.

Note how the location of the intersection is initially offset to the south-west in Figure 23(a). This intersection is highly asymmetric in density, with a large majority of traces in the south-west corner. Density-based processing finds this peak, and places the intersection here. After breaking out the turn-lanes however, this density asymmetry no longer has an effect as each

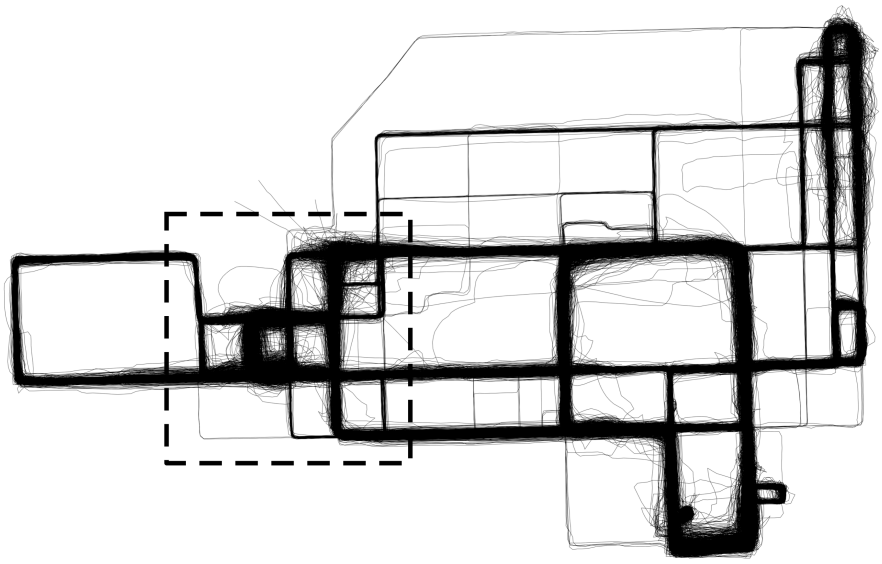


Figure 24. 7 months of raw traces, displaying both high disparity and high GPS noise. The dashed square indicates the "hospital area."

turn-lane has an independent set of traces matched to it. This example emphasizes the power of combining density- and trajectory-based processing for map inference.

4.7 Evaluation

In this section, we perform a quantitative and qualitative evaluation of our hybrid map inference pipeline. Where possible, we compare our results to those of the map inference algorithms evaluated in Chapter 2, demonstrating dramatically improved performance both quantitatively and qualitatively.

4.7.1 Dataset

For our evaluation, we use up to 7 months of data collected from the UIC shuttle buses. A mask of the full set of traces is shown in Figure 24. As we discussed in Section 2.3.1, the area served contains a mix of low-rise residential buildings and high-rise office and hospital towers. The dashed square highlights the problematic high-error "hospital area," where urban canyons result in poor GPS reception, with some traces showing consistent errors well over 100 meters.

4.7.2 Other Algorithms

In addition to our proposed map inference pipeline, we evaluate three existing algorithms for comparison purposes: the KDE-based method by Davies et. al (22) ("Davies"), the k-means approach by Edelkamp and Schrödl (15) ("Edelkamp") and the trace-merging technique by Cao and Krumm (20) ("Cao"). For this comparison, we limit our data to a 1-month subset for two reasons, previously discussed in Chapter 2. First, and most importantly, the inherent scalability problem of Cao makes it infeasible to evaluate this algorithm with a larger dataset. Second, due to noise sensitivity, the performance of Edelkamp declines as the amount of data exceeds the 1-month mark.

4.7.3 Evaluation Methodology

The purpose of our evaluation is to determine the accuracy with which each map inference algorithm represents the underlying road network. As in previous chapters, for our ground truth we use a manually-verified section of OpenStreetMap (32) covering our region of interest.

We use two different evaluation methods from previous chapters to compare our inferred maps to the ground truth. The first method (GEO) is taken from Chapter 3, and evaluates

map geometry only. Here, the connectivity of the map is ignored entirely, but every segment of both maps is taken into account. The second method (TOPO) is taken from Chapter 2, with one modification. This sampling-based method evaluates the topology of the map as well as the geometry. We modify the method to ignore parts of the map where no correspondence could be found between the inferred and ground truth maps. Thus GEO partially evaluates the entire map, whereas TOPO fully evaluates those parts where the maps overlap. As before, we use F-score, the geometric mean of precision and recall, as our primary evaluation metric.

For the results below, we use the topology refinement output only. While geometry refinement produces a more descriptive map, our ground truth does not contain this level of detail, making it infeasible to quantitatively evaluate the performance of the geometry refinement step. Visual samples of the geometry refinement output are provided in Figures 23(a) and 23(b).

4.7.4 Results

Our overall results are shown in Figure 25 and Figure 26, and as we can see, our new hybrid method offers a significant improvement over the previous state of the art in both GEO and TOPO evaluations. Despite its encouraging performance, Figure 27 and Figure 28 offer insight into the limitations of our proposed pipeline. While precision falls in the 0.9–1.0 range for matching thresholds of 15 meters and above, recall falls below 0.8 for both GEO and TOPO evaluations. Visually inspecting the generated maps, the explanation behind this is clear: many roads in the ground truth were traversed only once, and in the current topology refinement step, roads with a single traversal are pruned. Future work will need to address this problem, in order to improve performance further.

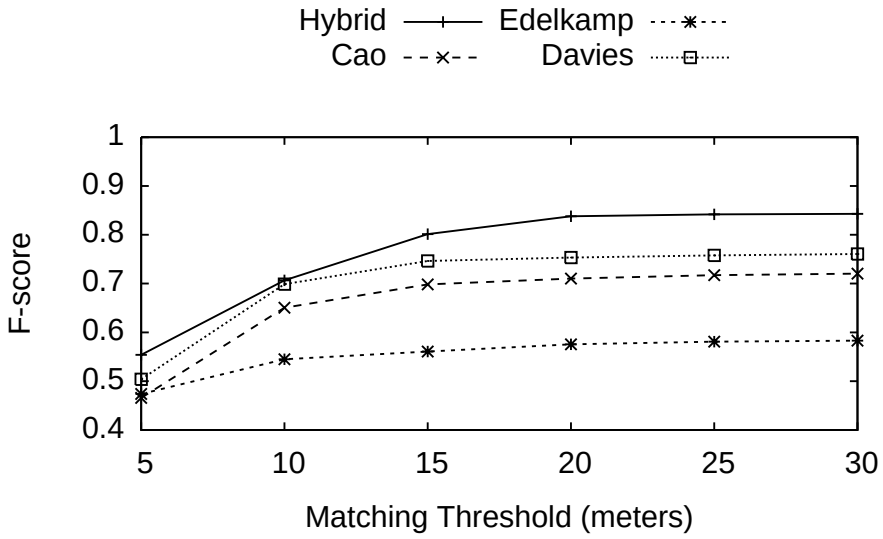


Figure 25. GEO F-scores of our method vs. existing methods on the 1-month dataset.

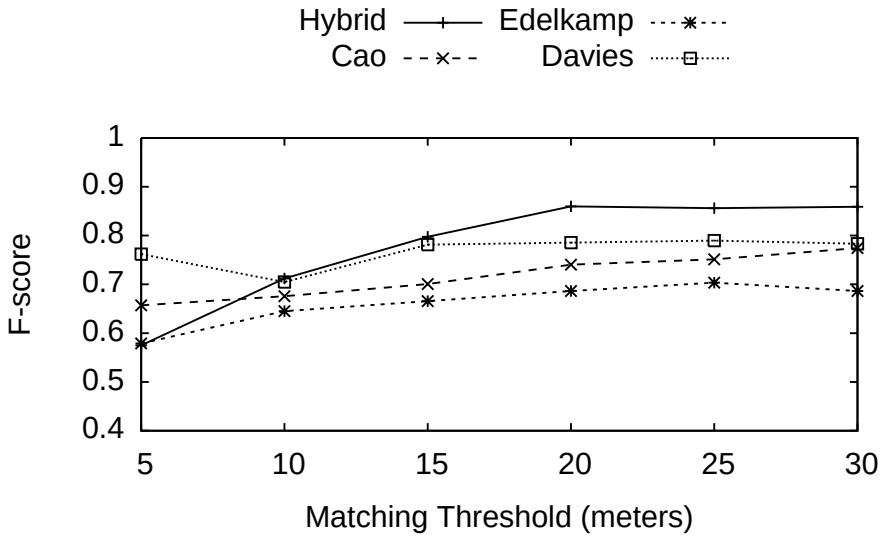


Figure 26. TOPO F-scores of our method vs. existing methods on the 1-month dataset.

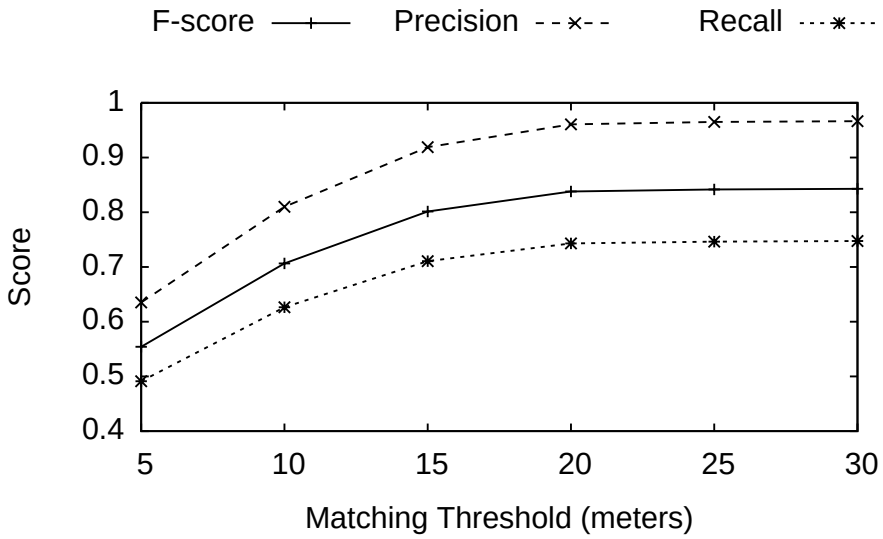


Figure 27. Precision/recall and F-score on 1-month dataset. GEO evaluation metric.

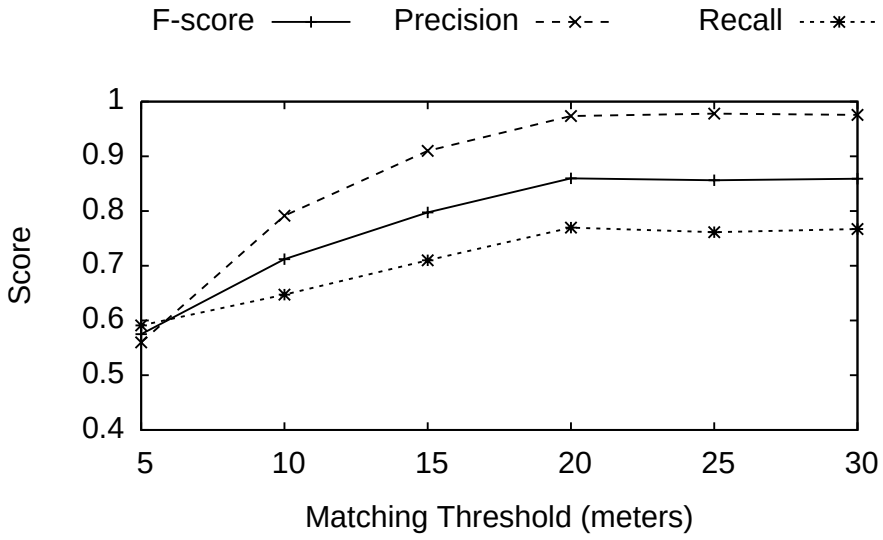


Figure 28. Precision/recall and F-score on 1-month dataset. TOPO evaluation metric.

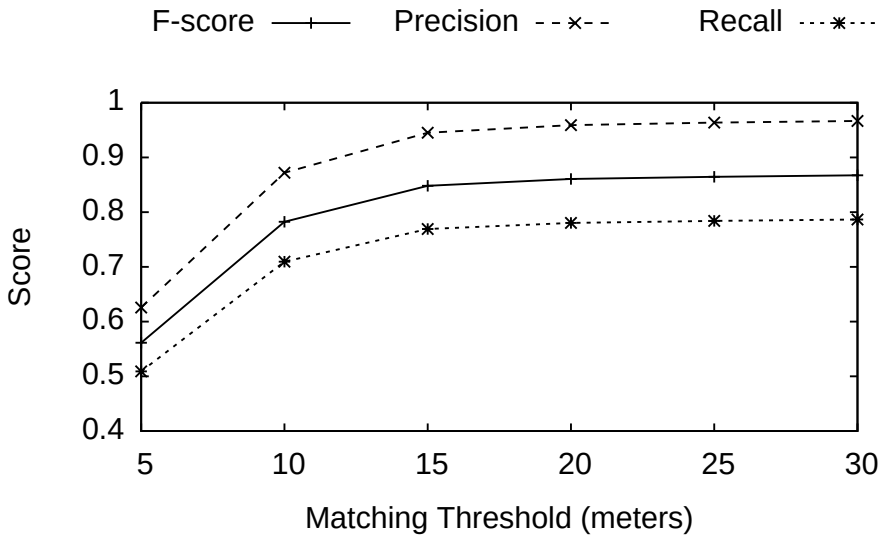
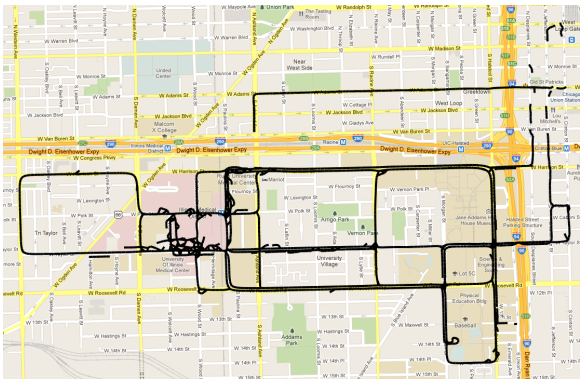


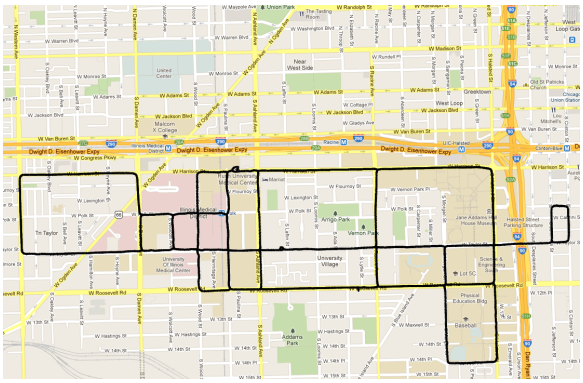
Figure 29. Precision/recall and F-score on 7-month dataset. GEO evaluation metric.

Finally, Figure 29 shows the GEO performance on 7 months of data. The slightly sharper bend of the curve suggests that additional data was helpful in producing more accurate centerlines. However, when using a higher matching threshold, performance is essentially unchanged. In this case, additional data simultaneously improves the map quality and introduces additional ground truth edges. With more evenly distributed data, we expect to see a monotonic performance increase with larger data sets.

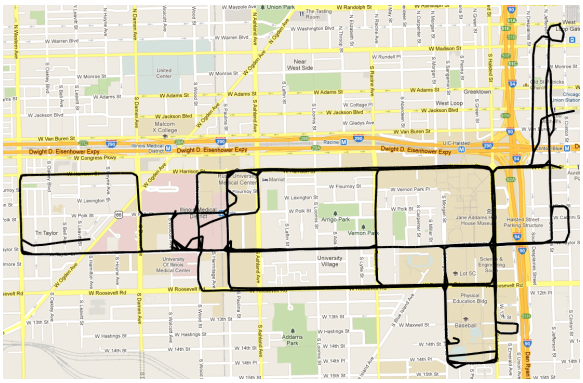
Figures 30(a)–30(d) visually illustrate the performance of each algorithm across the entire region of interest. Our proposed method produces a significantly more complete map, with very few spurious edges. Focusing our attention on the "hospital area" which has high density,



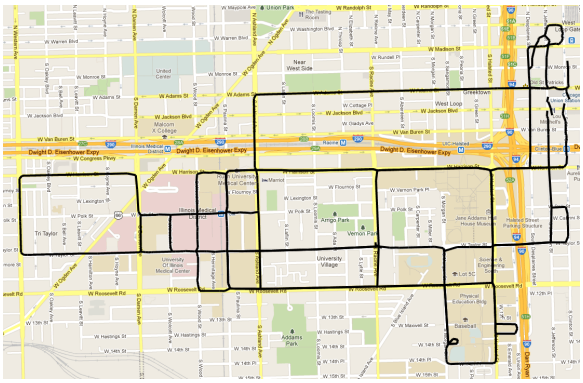
(a) Full map generated using Cao method with 1 month of raw data, overlaid on a map.



(b) Full map generated using Davies method with 1 month of raw data, overlaid on a map.



(c) Full map generated using Edelkamp method with 1 month of raw data, overlaid on a map.



(d) Full map generated using our hybrid method with 1 month of raw data, overlaid on a map.

Figure 30. Illustration of the performance of all algorithms across the entire region of interest.

disparity, and GPS error, we also see that our method produces a considerably higher quality map, with better coverage and improved alignment.

4.8 Conclusion

In this chapter we have presented a hybrid map inference pipeline, which significantly advances the state of the art when considering noisy and disparate datasets. Key to our work is the combination of initial density processing, with its ability to consider all traces in aggregate, followed by trajectory processing, with its capacity for capturing topological and geometric details. While the results on this dataset are very good, more work remains to validate our approach on different datasets, and tune each step for optimal performance.

4.9 Copyright Information

The material in this chapter originally appeared in the following publication: Biagioni, J. and Eriksson, J.: Map Inference in the Face of Noise and Disparity. In Proceedings of the 20th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pages 79–88. © 2012 Association for Computing Machinery, Inc. Reprinted by permission. http://doi.acm.org/10.1145/2424321.2424333.

CHAPTER 5

ONLINE GPS TRACKING WITH LOW DATA UPLINK USAGE

Tracking assets and people using the global positioning system is becoming increasingly popular. Applications of GPS tracking are widespread, including anti-theft lowjack devices, freight logistics and public transit arrival time prediction. A less obvious, but perhaps even more pervasive application of GPS tracking is crowd-sourced traffic maps: many popular smartphone applications, such as the ubiquitous "Maps" application, automatically upload the locations of smartphones to a central server when they are running.

The energy consumption of GPS devices, and methods of reducing the same, has been the subject of intense scrutiny in the past several years (46; 47; 48; 49; 33; 50). In GPS tracking applications, however, energy is arguably a secondary concern: in many applications of GPS tracking, energy is either abundant, as in most vehicular applications, or already expended, as in the crowd-sourced traffic maps example above. For this reason, our focus is on the uplink.

The data usage requirements of a GPS tracking system are arguably modest: a bare-bones differential update could be represented using just a few bytes. At the typical 1 Hz update frequency of an off-the-shelf GPS receiver, this would seem a negligible amount. However, during testing of a simple GPS tracking application, that application alone came very close to exceeding our 250 MB/month limit. Here, we examine how to reduce this data usage while preserving tracking performance.

In an online GPS tracking system, a fundamental trade-off exists between *accuracy*, the average error between the actual location and the reported location; and *uplink cost*, the average amount of "data usage" charged per second of tracking. For example, we may achieve accuracy close to the capability of the GPS receiver itself, but only at significant uplink cost, or we may achieve virtually zero uplink cost, but only with tracking accuracy on the order of several kilometers. In this chapter we propose an end-to-end "thrifty tracking" system that adaptively makes the trade-off between these performance parameters.

To realize this system, we: (i) design a unified extrapolation method that infers future movements based on current and historical conditions, and (ii) develop an adaptive sampling framework that allows the user to specify a performance target for *error* or *budget* (along with an adjustable *delay* parameter), while it optimizes the other. Additionally, we present a characterization of current tracking behavior based on large-scale GPS probe data, and an end-to-end evaluation of the above methods on real-world GPS traces.

5.1 Principles and Practice

GPS tracking comes in two basic flavors—offline and online. Offline tracking simply records the trajectory for subsequent, typically manual retrieval, while online tracking continuously transmits location reports over a wireless link to a central, Internet-based receiver.

In addition to a wide variety of first-class applications, GPS tracking also occurs as a side-effect of other applications, most commonly in GPS-enabled smartphone applications. Here, a GPS-enabled application continuously transmits the location of the user to the application

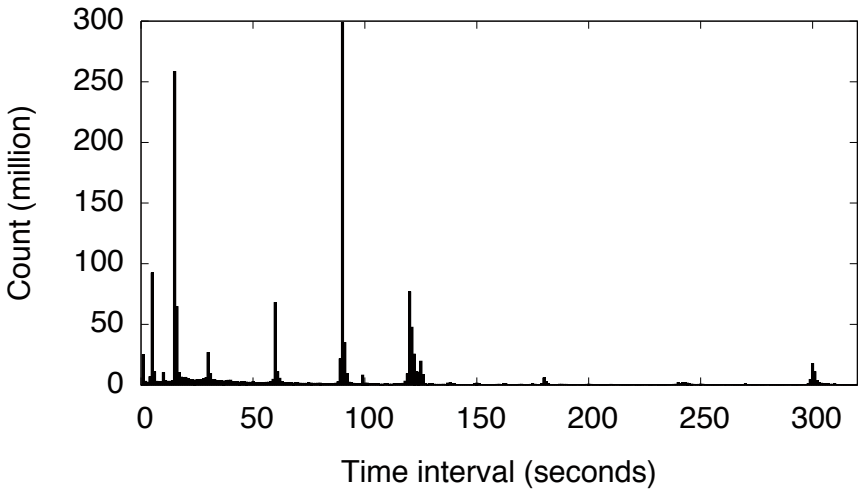


Figure 31. Histogram of intervals between ∼1.6 billion location reports from 25 providers, illustrating the periodic nature of contemporary GPS tracking.

provider or network operator, for inferring traffic information, fine-tuning cellular-network designs, and other tasks that require large amounts of GPS trace data.

5.1.1 Field Study of Large-Scale GPS Tracking

To gain a better understanding of typical online GPS tracking behavior we studied a dataset consisting of 1.6 billion GPS points, collected by 25 different data providers from 2010–2012. For privacy reasons, individual traces were split into short, de-identified and disconnected "probes" consisting of a smaller number of GPS points, before we received them. Despite this, we can gain an accurate statistical picture by studying the sampling behavior exhibited within each individual probe. Figure 31 shows a histogram of time intervals between samples, across all probes, and a clear pattern of periodic reporting emerges with tell-tale peaks at periods 1, 5,

15, 30, 60, 90, 120, 180, 240 and 300 seconds. After removing these clearly periodic samples, only 11.4% remain.

To better understand the origin of these remaining samples, we manually inspected several dozen representative traces from the four data producers with the largest fraction of non-periodic transmissions. One appeared to be a logistics company, with frequent visits to loading docks, and a target period of 300 seconds. Here, a majority of non-periodic transmissions coincided with likely stops and CAN-bus events such as ignition on/off events. Note that a trace with a period P, mixed with intermittent additional reports with a mean interval at or below P, will appear non-periodic in Figure 31. For example, samples at 0, 60, and 120 seconds have strict 60-second inter-sample spacings, but adding intermittent samples at 17 and 99 seconds produces inter-sample spacings at 17, 43, 39, and 21 seconds, a seemingly non-periodic sequence. For some data providers this was a frequent occurrence. Thus, the 11.4% reported above is an over-estimate of the proportion of non-periodic transmissions.

Similar behaviors were observed for other data providers, and throughout we were unable to find any evidence of spatial sampling (i.e., every so many meters), speed or bearing change-based sampling, or even policies as simple as not transmitting when stationary: all providers show several back-to-back transmissions with identical locations. Thus, both anecdotally and quantitatively speaking, we believe there is ample room for improvement to the status quo in online GPS tracking. Related work from the academic literature is reviewed in Chapter 8.

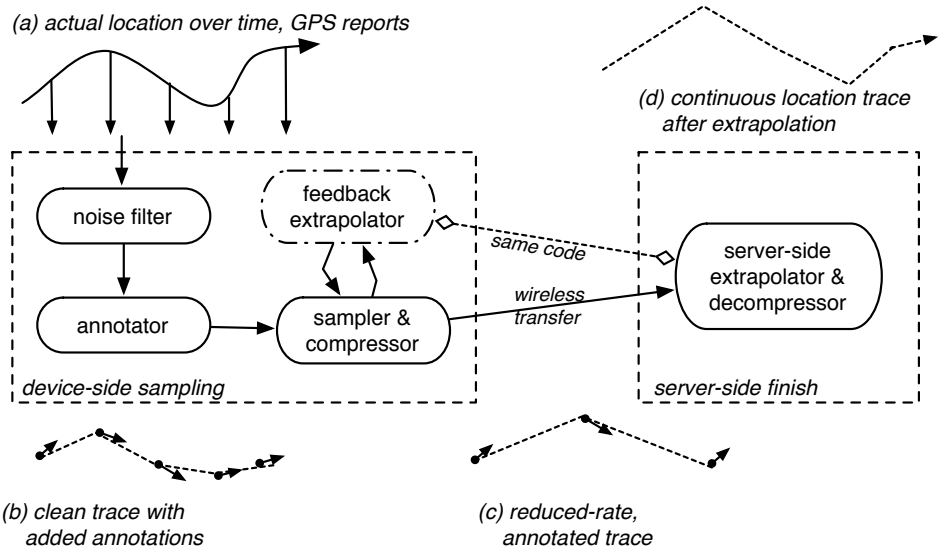


Figure 32. Thrifty tracking system architectural diagram. Adaptive sampling techniques rely on feedback from an extrapolator executing on the device, mirroring the extrapolation done on the server.

5.2 Thrifty Tracking Overview

Figure 32 illustrates our general thrifty tracking architecture which accommodates all known, and a variety of new tracking methods. Starting in the top-left of the figure, a GPS receiver samples the (continuous) device location, often with a frequency of 1 Hz or higher. If a GPS energy conservation mechanism is in use, this is done before the thrifty tracking system receives the GPS samples.

The incoming *raw* GPS trace (a) is first passed through a filter to remove any obvious outliers (based on impossible velocities), and then decorated by an annotator with additional information not provided by the GPS receiver (e.g., heading and acceleration). The decorated

GPS trace (b) is then passed to a sampler that unilaterally decides whether to forward a given trace point, with any necessary annotations, over the wide-area link based on the configured error or budget target (see below). The resulting sampled trace (c) is then fed to two identical extrapolators: one running on the remote server, and one running on the mobile device. On the server side, the extrapolator output (d) is made available for use by the trace consumer. On the client side, an identical extrapolator produces a continuous location estimate for local use. By comparing the output of the local extrapolator against the incoming raw GPS location, an error-aware sampler can then make its forwarding decision based on the difference between the current estimate and the measured location.

Our evaluation of this system is based on data retrieved from the OpenStreetMap (OSM) (32) website, but originally donated by volunteers from Moscow, Russia, and its outlying areas. In total, our evaluation spans over 5,000 individuals, 12.4 million GPS locations, and almost 3,500 hours of recorded traces.

5.3 GPS Trace Extrapolation

By inferring the future location of a device, or *extrapolating* its location trace, improvements can be made to both the timeliness and accuracy of tracking. The most basic free-space extrapolation method ("Constant Location" (CL)) predicts that the future location, for all times in the future, will be the same as the most recently reported location. By applying this extrapolation method, we improve the timeliness of our tracking: we are now able to provide an immediate estimate, at any point in time. However, this gain in timeliness is matched by a

loss in accuracy: for a moving device, predictions made by this extrapolator grow increasingly inaccurate with the time since the last update.

In this section, we explore how more sophisticated extrapolation methods described in prior work such as (51; 52; 53; 54) can be used to improve accuracy. Specifically, we evaluate the following three free-space techniques: Constant Velocity (CV), which produces a straight-line path from the most recent location, at the velocity from the most recent report, and Constant Acceleration (CA), which is based on CV but also includes estimated acceleration. We also introduce a minor variation, named Constant Deceleration (CD) which performs the same function as CA when acceleration is less than zero, otherwise it performs the same function at CV. Finally, we evaluate the use of advanced map-based extrapolators, described below. In Section 5.3.3, we propose a unified extrapolation technique that combines these free-space and map-based methods using a machine learning approach, to further improve extrapolation performance and therefore tracking efficiency.

5.3.1 Map-Based Extrapolation

If movements are restricted to a known map, or if past movement history is available, this information may be used to improve extrapolation performance over the free-space methods above. Map-based techniques naturally rely on online map-matching. For this, we use a Viterbi map-matcher similar to those described in (43; 33). These techniques are both batch-oriented, and output the maximum probability path at the end of trace processing. To make Viterbi map-matching work for online purposes, we instead use the maximum probability edge at each time step, at the cost of producing less accurate results.

Our map-based extrapolator operates by traveling along the current road until an intersection is reached. Once at the intersection, it may either stop there (MB0), continue straight through the intersection if possible (MB1), or decide how and whether to turn based on past movement history (MM0/MM1).

5.3.1.1 Trace-Driven Turn Prediction

If historical traces are available, these may be used to learn the most likely choice at each decision point as shown in (55). This can be thought of as an n^{th}-order Markov model, where street segments are represented by states, and turns by transition probabilities. During extrapolation, when encountering an intersection the turn with the highest probability indicated by the model is taken and extrapolation continues along the new subsequent street. If the model contains no applicable information for a given intersection, we have a choice of either stopping at the intersection (MM0), or continuing straight through (MM1). For our evaluation we elected to use a 10^{th}-order Markov model, based on the good performance shown on this task in (55). We use separate sets of past traces to train and test the model.

5.3.2 Individual Extrapolator Performance

Given the variability of GPS traces, it is unclear which of these extrapolators most accurately predicts future movements for an arbitrary trace. To evaluate the extrapolators described above, we compare their predicted locations to the measured locations throughout the OSM dataset described in Section 5.2. More specifically, for each trace L, and for each time i of the trace, we compute the distance $\delta^i(j) = |\hat{L}^i_j - L_j|$, between the extrapolated locations $\hat{L}^i_j$, and the corresponding actual locations L_j, $\forall j > i$. For each time i, we then produce two numbers:

D_i^{max}, the number of seconds that elapse before $\delta^i(j) > max$, where max is a maximum tolerable distance error threshold, and

$$E_i^{\Delta t} = max(\{\delta^i(j) : j = i+1, ..., i+\Delta t\})$$

the maximum error incurred over an interval of Δt seconds following the location update. For each extrapolator we then report its mean duration

$$\overline{D^{max}} = \frac{1}{|L|}\sum_{i=0}^{|L|} D_i^{max}$$

over all traces, which indicates how long an extrapolator "lasts" given a maximum error threshold, and its mean maximum error

$$\overline{E^{\Delta t}} = \frac{1}{|L|}\sum_{i=0}^{|L|} E_i^{\Delta t}$$

over all traces, which describes how well it performs over a fixed time interval.

Figure 33 and Figure 34 show the value of $\overline{D^{max}}$ and $\overline{E^{\Delta t}}$ for eight extrapolators on our OSM dataset, for varying values of max and Δt. Overall we observe that the free-space extrapolation methods outperform their map-based counterparts when the values of max and Δt are low. However, as we increase max and Δt the map-based methods eventually reach performance parity, and then outperform the free-space methods by an increasingly large margin.

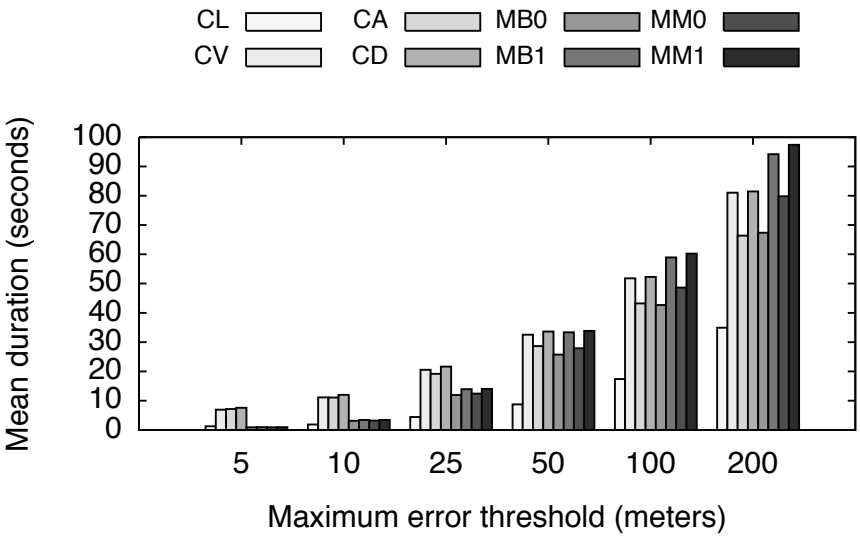


Figure 33. $\overline{D^{max}}$ for eight extrapolators on our OSM dataset, for varying values of max.

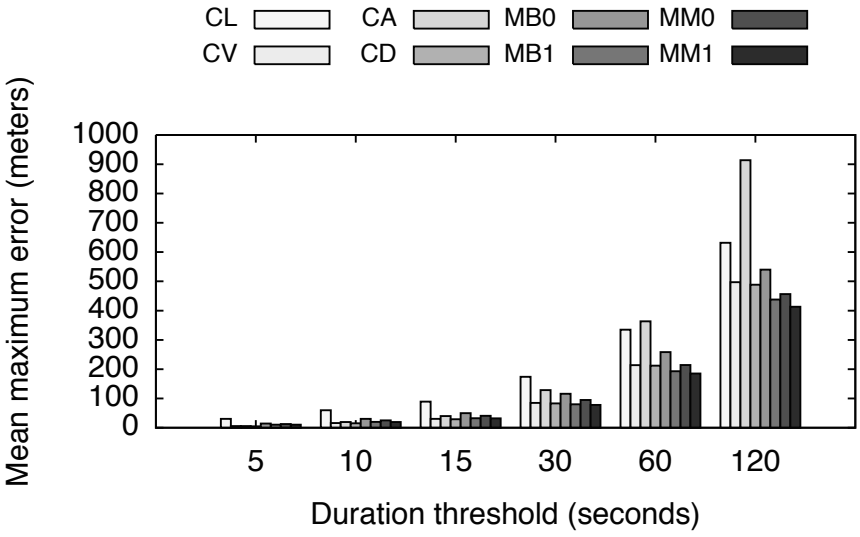


Figure 34. $\overline{E^{\Delta t}}$ for eight extrapolators on our OSM dataset, for varying values of Δt.

Within the collection of free-space extrapolators we find that CD performs best in all cases, followed closely by CV, then CA, and lastly CL. The superior performance of CD can be attributed to its harnessing the best aspects of CV and CA, while not inheriting their limitations. In almost all cases CD performs identically to CV, by assuming the subject continues moving in a constant direction at a constant velocity. However, in those cases where deceleration is detected, CD assumes the subject is coming to a stop and gradually reduces velocity to zero. CD's ability to replicate this real-world phenomenon is key to its performance advantage. CV follows closely behind CD in performance, limited only by its inability to reduce an extrapolated subject's velocity. CA follows further behind, as its tendency to continuously *increase* velocity when acceleration is detected (up to 60 mph) is largely inconsistent with reality. Finally, the universally poor performance of CL is a direct result of our OSM data consisting of traces where the subject is often moving. Because CL is only able to predict its current location for all times in the future, this results in quickly growing extrapolation errors.

Within the collection of map-based methods we find that trace-driven turn prediction (MM0/MM1) performs better than un-trained techniques (MB0/MB1). This finding is a direct result of trace-driven turn prediction leveraging past behavior to predict the future. Because it doesn't have to stop at intersections, it is able to extrapolate further into the future than MB0, and since it has knowledge of past turns at intersections, it has a better idea of which direction to follow than MB1. We also observe that given the option of either stopping or continuing straight through an intersection, in both the trained (MM0/MM1) and un-trained (MB0/MB1) cases, continuing straight through always results in better extrapolation performance. This is a

largely intuitive result, as people traveling along roads tend to proceed straight through intersections far more often than they turn (approximately 90% of the time in our dataset), making an extrapolation of "straight-ahead" a reasonable prediction in the absence of any additional information. Because MB1 and MM1 consistently outperform MB0 and MM0 (respectively), we only present results from MB1 and MM1 going forward.

5.3.2.1 Overall Performance Analysis

Our finding that map-based extrapolation methods fail to work well at low error-thresholds is seemingly counterintuitive, as they have far more information with which to predict the path of future travel. However, there are at least two factors that limit their accuracy in situations where strict error tolerance is desired: road position and GPS error.

Because digital maps are a simplification of reality, often representing multi-lane roads by simple bidirectional paths, they are unable to represent the position of a subject in a way that accurately reflects their real-world location. Instead, they have to "snap" the subject to their closest likely road position, often introducing one or two lane-widths of error before extrapolation along the map even begins. Moreover, because the extrapolator is tasked with replicating our real-world location traces, its performance suffers further as a result of the measurement error in our recorded locations. Commodity GPS receivers typically report locations with 5–10 meters of error under ideal conditions, and in the presence of tall buildings, loss of signal and multi-path effects occasionally produce errors in excess of 100 meters. Because these errors are a phenomenon our map-based extrapolators are unable to predict and reproduce, when they occur in our traces and deviate from the road, they will be reported as extrapolation errors.

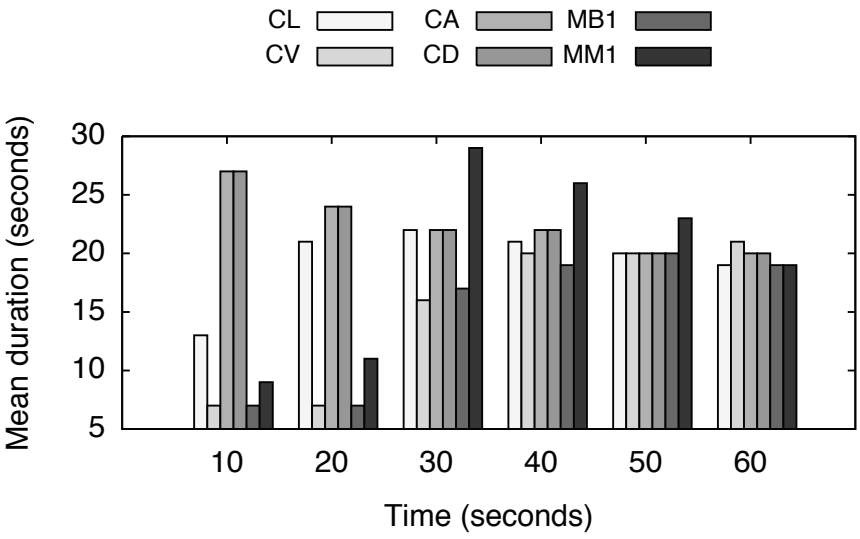


Figure 35. Values of D_t^{max} over time for six extrapolators along a particular trace, for a fixed 25-meter *max* error threshold. Here we observe that no single extrapolator consistently outperforms the others.

From this we conclude that free-space extrapolation techniques thrive in low error-bound conditions because they begin their extrapolation from the measured GPS location directly, and proceed without their predicted path having to follow an inflexible model. It is worth noting here that while the vehicle may well be located on the street, much as a map-based extrapolator would suggest, our evaluation is based on reproducing the original GPS trace, not estimating the "true location" of the vehicle.

5.3.3 Unified Extrapolation

We must bear in mind that the values we are seeing in Figure 33 and Figure 34 are means (i.e., $\overline{D^{max}}$ and $\overline{E^{\Delta t}}$), and that the best method for extrapolation can vary between samples

based on the current position of the vehicle and the short-term history of previous travel. This behavior is clearly illustrated in Figure 35, where we've plotted the raw D_t^{max} values for a fixed (25-meter) error threshold over time (along a particular trace). We can see that while certain extrapolators work well under particular conditions, no single extrapolator offers the best performance in all circumstances. To address this, we propose a unified extrapolator which automatically selects the best extrapolation method for the current circumstances.

5.3.3.1 Machine Learning Approach

The goal of the unified extrapolator is to select the best extrapolation method at any given time. Since it cannot know what will happen in the future, this is a challenging task with no guaranteed results. We cast this as a classification problem, relying on past experience to train a classifier: supervised learning applies here as the recorded history reveals exactly which extrapolator works best.

Given that CD and MM1 are the best performing extrapolators on our OSM dataset for low and high (respectively) error and duration thresholds, our first approach was to build a simple two-class decision tree that selected either the CD or MM1 extrapolator. While this classifier performed well in practice on our OSM dataset, we found that it did not generalize well to other trace datasets, where other extrapolators outperform CD and MM1 over the same range of error and duration thresholds.

Therefore, in order to make our classifier more general-purpose we adopted a multi-stage classification approach. Our experiments with the simple CD/MM1 classifier taught us an important lesson: making a classification decision between the free-space and map-based ex-

trapolators is very practically useful, as there is a distinct cross-over point along the threshold ranges where the performance advantage between these two classes of methods is exchanged. With this in mind, we constructed the first stage of our multi-stage classifier as a decision tree trained to select between the free-space and map-based extrapolators. Then, to ensure the generality of our approach, we separately trained two additional decision trees: one to select amongst the free-space extrapolators, and one to select amongst the map-based extrapolators.

Putting it all together, our multi-stage classifier operates as follows: input samples are first presented to our first-stage decision tree in order to determine the appropriate free-space/map-based extrapolator class. Then, based on that decision they are either passed to the second-stage free-space extrapolator decision tree, or second-stage map-based extrapolator decision tree, in order to determine the specific extrapolation method to use. For all of our decision trees we used the implementation provided by the Scikit-learn (56) machine learning library.

We train all of our decision trees using the following set of features, drawn from a 60-second sliding window of locations immediately preceding the current sample: distance between the current sample and its map-matched edge, previous samples' mean speed, difference between the current sample's speed and previous samples' mean speed, difference between the current sample's acceleration and previous samples' mean acceleration, current sample's distance to the previous samples' mean location, and the current sample's distance to the immediately preceding sample. These features were identified among a larger collection using a Tree-based estimator (57) to compute their classification importance, and determined to be significant factors in selecting the best extrapolation method.

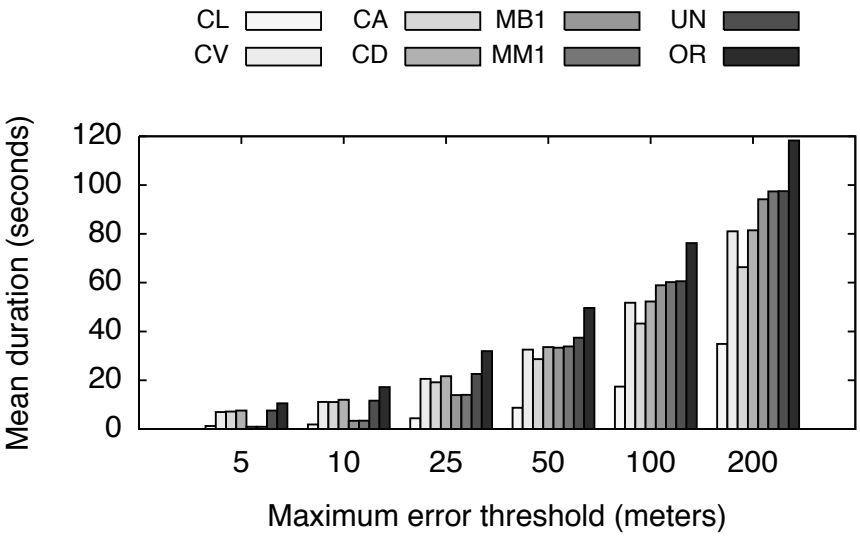


Figure 36. $\overline{D^{max}}$ for eight extrapolators on our OSM dataset, for varying values of max.

5.3.3.2 Unified Extrapolator Performance Evaluation

To determine a loose upper bound on the performance of our unified extrapolator, we use an "Oracle" extrapolator that looks into the future to select the extrapolation method that will work best among all possible options. The performance of the Oracle extrapolator is shown as the column labeled "OR" in Figure 36 and Figure 37. In the worst case we can see that the Oracle's performance matches that of the single-best extrapolation method, and in the best case far exceeds the performance of any alternative, suggesting that dynamically selecting which extrapolator to use can potentially result in significant performance gains.

As Figure 36 and Figure 37 show, the Unified extrapolator (column labeled "UN") meets the performance of the best individual extrapolator for any given value of max or Δt, and in some

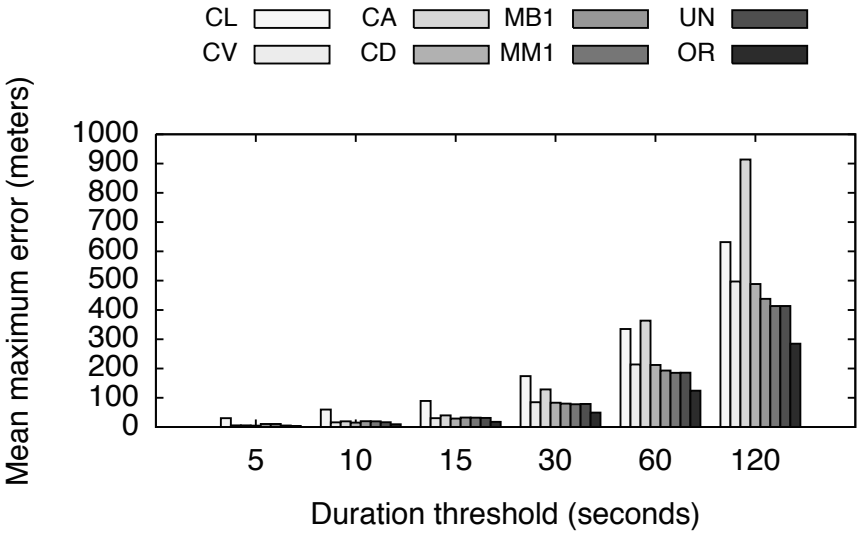


Figure 37. $\overline{E^{\Delta t}}$ for eight extrapolators on our OSM dataset, for varying values of Δt.

ranges exceeds the performance of all individual extrapolators by adaptively selecting the best method to use at any particular moment in time. Generally speaking, the Unified extrapolator performs robustly across the full range of *max* and Δt values, achieving our goal of creating a *single* extrapolation method that flexibly adapts to its circumstances. The remaining disparity between the UN and OR columns suggest there may still be room for improvement in adaptively selecting the best extrapolation method, but this is a challenge we leave for future work.

5.4 Adaptive Sampling

Online tracking systems trade off two performance attributes: accuracy and cost. While periodic sampling provides highly predictable cost, it also provides very loose guarantees on accuracy. Alternatively, sampling at a fixed distance interval provides a strict error bound, but

very loose constraints on cost. As shown in Section 5.1, uniform periodic sampling is the policy of choice in today's tracking systems. It gives the user direct control over cost—a likely explanation for its popularity, but sacrifices timeliness and accuracy. Introducing the extrapolation methods from Section 5.3 can significantly improve accuracy and timeliness, as they provide an instantaneous location estimate at any point in time. However, further improvements can be achieved by replacing uniform periodic sampling with adaptive sampling techniques.

5.4.1 Feedback, Delay, and GPS Compression

Replicating the extrapolation process performed at the receiver by the sender enables it to directly observe any incurred error. This allows the sender to choose the samples it transmits to maximize the accuracy gained from each transmission. For maximum timeliness, a sampler must decide whether or not to transmit each sample as soon as it arrives, allowing relatively little room for maximizing sampling efficiency. However, if the user is willing to tolerate a fixed delay in reporting, significant gains can be made by choosing when to transmit a sample. Adding a delay window also provides an opportunity for GPS trace compression, which can yield further data usage savings.

Two types of GPS trace compression are required for our samplers: error-bound and size-bound. For error-bound compression we use the algorithm proposed by Meratnia and Rolf (58). Our size-bound GPS compressor uses the same approach, but stops when a maximum sample count has been reached.

5.4.2 Sampling with Configured Error and Delay

With an error bound and fixed delay configured by the user, the task of the sampler is to minimize cost while enforcing the error bound. For each incoming location from the GPS receiver, the sampler measures the distance between the extrapolated trace and the current location. If the distance exceeds the error bound, this sample must be transmitted. If zero delay is configured, the sample is transmitted immediately, updating the server and restarting the extrapolation.

With a non-zero delay of T seconds, the sample (and the surrounding window of samples) may be transmitted at any time between the present and T seconds into the future. The optimal time to transmit is when the resulting error is minimized. Since future errors are unknown, we need to estimate the future extrapolation error. For this, we maintain statistics on the extrapolator's past performance: its expected error over a given time interval, and the expected duration it stays below some maximum error. Then, at a given time-step we can decide whether to transmit the current window containing the oldest sample or defer transmitting in the hopes of finding larger errors (to be corrected) in the future. Therefore, the current window is only sent if it has a greater mean error than the expected mean error of all other candidate transmission windows.

Figure 38 shows budget usage as the error threshold is varied, for different delays. These results are based on a constant-location extrapolator, to highlight the behavior of the sampler in isolation (see Section 5.5 for our combined adaptive sampler/unified extrapolator evaluation). Here, we compare our sampler with a basic error-bound straw man solution. This solution

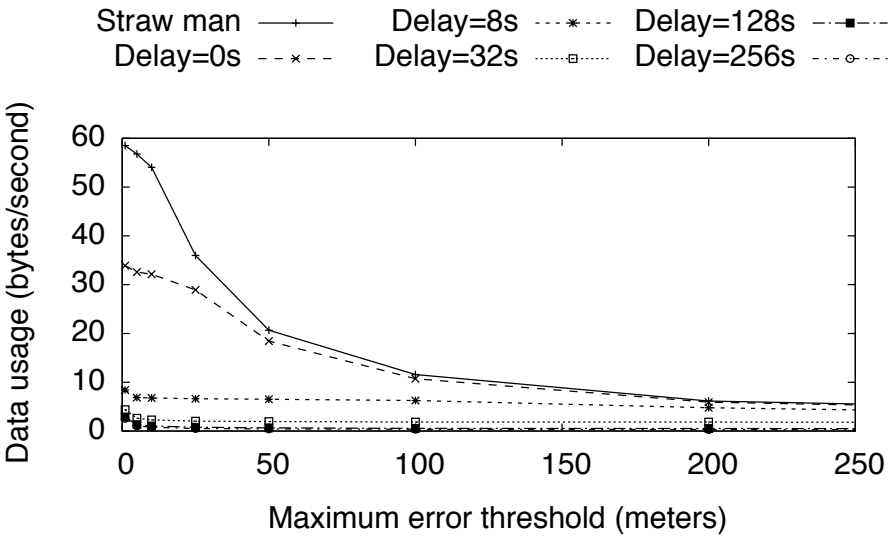


Figure 38. Budget usage with increasing maximum error bound for various delay bounds.

transmits a single sample with a fixed distance interval equal to the error threshold, thus guaranteeing the error never goes above threshold. While the straw man provides a guaranteed error bound, it does so at high cost. Our sampler outperforms the straw man by a considerable margin, even with zero delay configured, demonstrating that while compression is important, adaptive sampling offers a substantial advantage on its own. Furthermore, as one might expect, as error-tolerance increases the cost decreases.

5.4.3 Sampling with Configured Budget and Delay

With a target budget and delay configured by the user, the sampler minimizes error while meeting the budget and delay targets. Intuitively, the larger the budget and the larger the delay, the smaller the error. When sampling on a budget we must first have enough savings

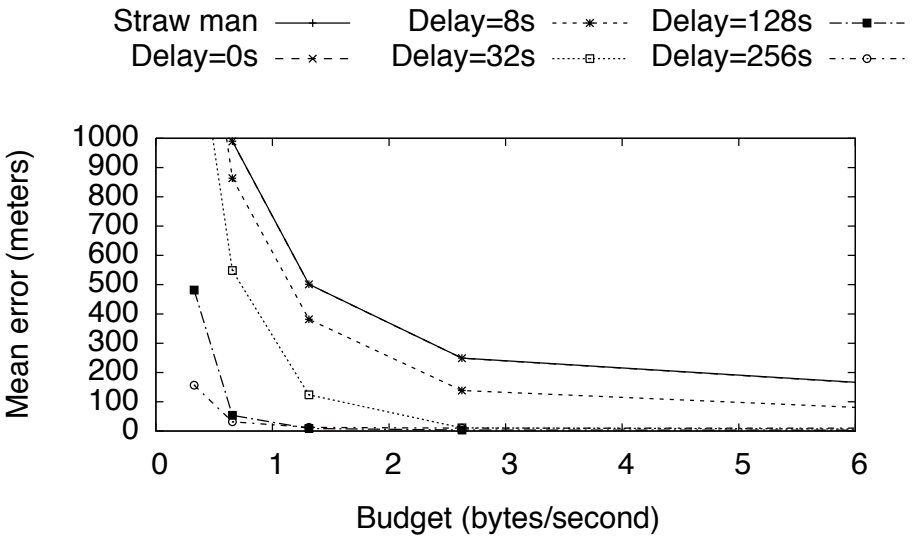


Figure 39. Mean error with increasing budget for various delay bounds.

to make a minimum-size transmission. Then, to decide when to transmit, we compare the errors produced by extrapolation and the errors produced by compressed transmission using our current savings, and transmit if the improvement exceeds the expected GPS error. Intuitively, if the improvement does not exceed the expected GPS error, it is not worth spending our hard-earned bytes transmitting this window.

Figure 39 shows the mean error incurred vs. specified budget, for several specified delays. Here, the straw man solution transmits a single sample whenever it has the savings. This figure shows that our sampler reduces error quickly when given more budget, validating the effectiveness of our technique. Moreover, as we increase the delay, mean error is reduced as our sampler is able to select more appropriate windows to transmit. Interestingly, the straw

man matches our sampler's performance when we have zero delay. This is because compressing a single sample always yields a perfect result, causing the sampler to transmit whenever the extrapolator error exceeds GPS error.

5.5 End-to-end Evaluation

In this section, we look at the overall performance of our system with the unified extrapolator and adaptive sampler coupled together into an end-to-end system. In our study of typical GPS tracking behavior discussed in Section 5.1, we identified a clear pattern of periodic reporting among tracked individuals, with substantial peaks at 15 and 90 seconds covering almost half of the 1.6 billion recorded samples. In this evaluation we will use these two periods to guide our analysis by characterizing them as two common *types* of system operator: one whose chief concern is tracking *accuracy*, and the other whose chief concern is tracking *cost*.

5.5.1 Tracking for Accuracy

For our accuracy-concerned system operator we adopt the 15-second sampling interval, where each of the individuals being tracked will send 5760 samples per day. According to our experiments with the AT&T wireless network, if we assume the operator uses UDP as their transmission protocol each sample will cost 84 bytes to transmit. This will result in a data usage budget of 5.6 bytes/second, and mean tracking error of 110 meters.

As shown in Table I, using our system this operator can reduce their data usage to 0.85 bytes/second while maintaining the same mean error ("w/o delay" column). Moreover, if they are willing to accept a 7.5 second delay in transmissions (1/2 the sampling interval), they can further reduce their data usage to 0.75 bytes/second ("w/delay" column). We argue that a

Optimization Criteria	**Fixed interval**	**Thrifty w/o delay**	**Thrifty w/delay**
Data usage (bytes/second)	5.6	0.85	0.75
Mean error (meters)	750	175	35

TABLE I

END-TO-END EVALUATION RESULTS.

delay of 1/2 the sampling interval is a reasonable value, as this is equal to the mean delay of fixed interval sampling. Overall, using our system this operator can see a reduction in their data usage of 85% with no additional delay, and if they are willing to accept a 7.5 second delay, they can realize a further 12% reduction in data usage (for a total reduction of 87%).

5.5.2 Tracking for Cost

For our cost-concerned system operator we adopt the 90-second sampling interval, where each of the individuals being tracked will send 960 samples per day. This will result in a budget of 0.93 bytes/second (using UDP), and mean tracking error of 750 meters.

As shown in Table I, using our system this operator can reduce their mean error to 175 meters while maintaining the same budget (w/o delay), or 35 meters (w/delay) if they are willing to accept a 45 second delay in transmissions (again, 1/2 the sampling interval). Therefore, using our system this operator can see an improvement in their tracking accuracy of 77% with no additional delay, and if they are willing to allow a 45 second delay they can realize a further 80% improvement in accuracy (for a total improvement of 95%).

Overall, we can see that our system is capable of substantial reductions in data usage and error based on the needs of the provider. Crucially, this benefit is afforded by simply providing

a target accuracy or budget-bound, and the system is able to adaptively reduce cost with no further intervention.

5.6 Conclusion

Given the rising popularity of GPS tracking applications, reducing their data usage requirements is a pressing need. To this end, we designed a unified extrapolator that provides a single method for accurately inferring an object's future location, by using machine learning to harness the capabilities of several free-space and map-based extrapolation techniques. We also developed an adaptive sampler, that allows a system operator to specify a performance target for error or budget (along with an adjustable delay parameter), while it automatically optimizes the other. We then combined these two components into a unified thrifty tracking system, which in our experiments was shown to outperform the status quo by up to 20× while providing improved guarantees and flexibility.

CHAPTER 6

AUTOMATIC TRANSIT TRACKING, MAPPING, AND ARRIVAL TIME PREDICTION USING SMARTPHONES

Transit information has come a long way from the printed paper schedules of decades past. Today, virtually every transit agency in the developed world, regardless of size, makes their schedule available on the web in one form or another. However, for smaller operations, this is often where it ends. More advanced services, such as integration with Google Maps, automatic transit directions, or real-time tracking and arrival time prediction, are typically reserved for large transit agencies who have the necessary expertise in-house or can afford to have it done for them.

Through these advanced services, transit agencies can dramatically improve the transit user experience. Real-time tracking and arrival time prediction are particularly powerful: transit riders who would otherwise be enduring a potentially frustrating wait can now adjust their travel plans to minimize waiting time. Even where buses typically run on time, real-time tracking can improve rider confidence in the transit service, allowing users to schedule closer connections with less built-in margin of error.

For these reasons, real-time transit tracking is a highly desired feature among transit riders. That said, the implementation of a full transit tracking system, including complete and accurate digital route maps, in-vehicle tracking devices, an online tracking website, up-to-date schedules, and accurate arrival time predictions, can be a daunting and costly process for smaller transit

operators. While commercial providers exist, with NextBus (59) and Clever Devices (60) being the two main vendors, use of these services incurs substantial initial and recurring fees.

Our goal is to reduce the cost and complexity of offering these services by creating EasyTracker, an automatic system for transit tracking, mapping, and arrival time prediction. The system consists of four main components: (i) an off-the-shelf smartphone, installed in each bus or carried by each driver, functioning as an automatic vehicle location system or tracking device, (ii) batch processing on a back-end server which turns stored vehicle trajectories into route maps, schedules, and prediction parameters, (iii) online processing on a back-end server which uses the real-time location of a vehicle to produce arrival time predictions, and (iv) an interface that allows a user to access current vehicle locations and predicted arrival times.

Using EasyTracker, a transit agency can implement a sophisticated bus-tracking and arrival time prediction system by simply purchasing a number of smartphones and downloading the bus-tracking application to each phone. EasyTracker has considerable advantages over the current state of the art. First, the use of standard smartphone hardware reduces both the one-time and recurring costs involved in establishing a real-time transit tracking system. Second, since the system automates the process of route map and schedule creation, cost and required user input are dramatically reduced. Third, due to its automated nature, our system is able to adjust the published routes and schedules in response to road construction or predictable congestion events.

To realize this system, in this chapter we: (i) develop an algorithm for inferring the set of serviced routes from a collection of unlabeled GPS traces, requiring no driver interaction

or other user input, (ii) design an algorithm for inferring the locations of transit stops along these routes, and (iii) provide a means of automatically inferring an estimate of the route schedule, describing hours of operation and intended arrival times for arrival time prediction. Additionally, we present a thorough evaluation of our system using eleven months of data across seven routes from two transit agencies.

6.1 Background and Motivation

The minimum requirement for a real-time transit tracking system is an in-vehicle device (sometimes referred to as an "automatic vehicle locator") and a back-office component. The in-vehicle device determines the vehicle's current location using GPS and communicates this via a wireless link (typically cellular service) to the back-office. The back-office component is a central server that processes the incoming time-ordered sequences of locations and typically provides a live tracking website for the public as well as status monitoring for dispatch purposes.

This type of vehicle tracking, which simply reports the locations of all active vehicles, is widely available today. While this is a useful service, its utility for transit applications is somewhat diminished by a lack of sufficient navigation metadata: what route is each bus driving, and at what time will it arrive at my stop? State of the art systems provide this metadata by means of an in-vehicle device which accepts driver input, such as the current route, as well as by estimating arrival times based on current vehicle location, past travel times, and the official route schedule.

In order to make arrival time predictions, these navigation-enabled systems require the following additional information:

1. A route shape file containing the road segments traversed by each route, for matching a vehicle's current GPS location to a location along the route.
2. A list of stops for each route in traversal order, for producing trip directions.
3. The planned schedule for each route and stop, to handle corner-cases such as the first and last trip of the day.
4. The route driven by each active vehicle at all times, in order to know where each vehicle is going next.

Manually collecting this information can be a time-consuming and complex task. We have first-hand experience working with four different transit agencies, which serve between 1,000 and 500,000 trips per day. Anecdotally, one such agency, despite an annual budget of $250 million USD, lacks the resources to produce route shape files for their existing bus routes. As a consequence, their routes do not appear in Google Maps (61) and other trip planning services.

6.1.1 Back-end Processing

What primarily sets EasyTracker apart from the current state of the art is the aim to require no manual input. Using EasyTracker, items 1–3 (above) are automatically derived from recorded GPS traces. This not only reduces the amount of manual labor required, but also enables agencies that lack the necessary technical expertise in-house to set up a transit tracking system without seeking external assistance. Item 4, the current route of a vehicle, is automatically determined based on its recent movements and the known route map (item 1). This automatic classification allows drivers to focus on safety, and avoids the need for additional driver training.

6.1.2 In-vehicle Device

For our purposes, the type of in-vehicle device used is of relatively lesser significance: the quality of GPS coordinates provided by a smartphone GPS unit are not dramatically different from those provided by a dedicated vehicle tracking device. However, the initial and recurring costs can differ dramatically, given the benefits of mass-production of smartphones. In urban environments, where vehicles often travel in the GPS shadow of tall buildings, a sophisticated vehicle tracking device may use inertial navigation to augment the outage-prone GPS sensor. While inertial navigation itself is outside the scope of this thesis, modern smartphones are equipped with several suitable sensors (accelerometer, gyroscope, electronic compass, cellular and WiFi radio) which may be leveraged to improve GPS accuracy in challenging environments. Any technique that improves GPS localization will only improve the performance of this system.

6.2 System Architecture and Overview

We call our system EasyTracker, for the ease with which it allows transit agencies to add transit tracking to their list of services. As illustrated in Figure 40, the EasyTracker architecture consists of a data collection unit in each vehicle, a number of algorithms for online and batch data processing, and one or more user interfaces for the transit user. The primary function of the in-vehicle device is to periodically transmit its GPS coordinates to a central location server, using a cellular uplink. The in-vehicle device can either be permanently installed in the vehicle, or may be carried by the driver. The central location server receives location updates from all in-vehicle devices.

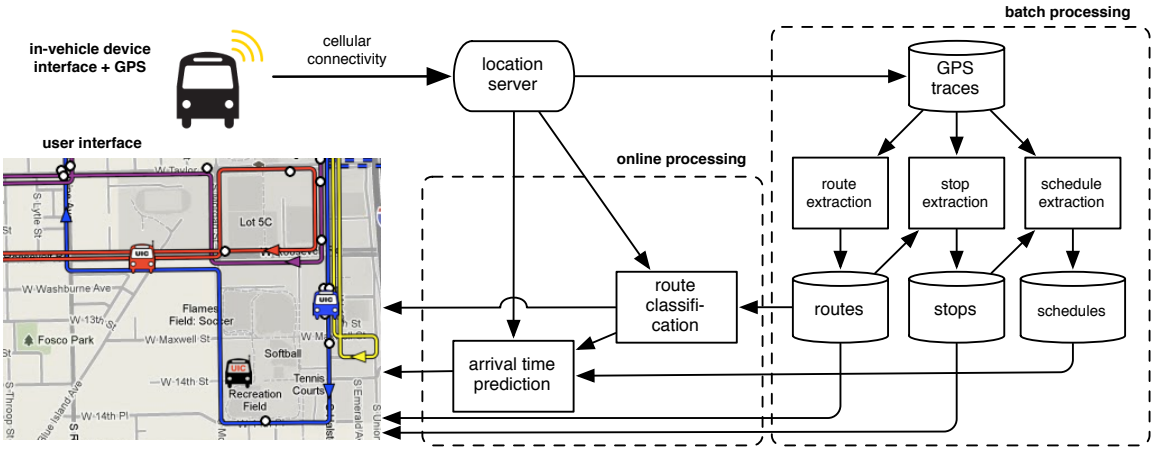


Figure 40. Architectural overview of the EasyTracker system. Data produced by in-vehicle devices are passed through batch and online processing, yielding route shapes, stop locations, route classifications, and arrival time predictions, which are displayed through a user interface.

The received records are then put through batch and online processing. In batch processing, a large set of recorded GPS traces are processed to produce route shapes, stop locations, and schedules. Online processing matches vehicles to routes, and performs arrival time prediction.

6.2.1 Batch Processing

Our *route extraction* algorithm is based purely on GPS traces, and does not make use of an existing road map. This design is based on three observations:

1. A sufficiently accurate digital road map may not be freely available for the area of interest.
2. Since routes do not necessarily follow public roadways, a road map may be misleading.
3. Our route extraction algorithm performs more reliably when using an automatically inferred road map.

Given the route shapes, the recorded traces are separated based on the shape they follow. Each set is then fed to our *stop extraction* algorithm, which produces a set of bus stops based on the vehicles' movements along the shape.

Given the stop locations, we then determine the raw arrival times: the approximate time each vehicle arrived at (or passed by) each stop along the route. These are processed by our *schedule extraction* algorithm to estimate the planned schedule of the route.

6.2.2 Online Processing

Automatic *route classification* classifies vehicles as "in-service," serving a particular route as determined by the route shapes, or "out-of-service" if the recent location trace does not match up with a known route. Once a vehicle is deemed "in-service," *arrival time predictions* are made using the recent location trace of the vehicle and the relevant route schedule.

6.2.3 Prototype User Interface

Shown in the bottom-left corner of Figure 40 is a cropped image from a prototype system, currently in use by UIC's campus shuttle service. Vehicles are matched to routes by their color, and clicking on a shuttle stop or vehicle brings up their arrival time predictions. Given route shapes, stop locations and predicted arrival times, a number of variations on this interface are easily constructed.

6.2.4 GPS Traces and Ground Truth Data

The majority of our evaluation is based on recorded GPS traces from UIC's campus shuttle service, as well as captured data from the publicly available Chicago Transit Authority (CTA) real-time bus tracker feed (62).

For the campus shuttle, we use seven months of GPS traces recorded from thirteen buses, operating four routes (six including minor variations). Campus shuttle GPS traces have the following characteristics:

- They are labeled only with a vehicle ID number—these numbers do not correspond to routes, as every vehicle can be serving any route at any given time.
- Locations are recorded once every second any time the ignition is on.
- The shuttles frequently take trips to locations off of the official routes, for mechanical service, outreach operations, or chartered University outings.

We also manually collected the ground truth location of each campus shuttle stop, and the exact route followed by the campus shuttles to serve as ground truth for route extraction.

For the CTA data, we use 100 days of traces from a single route. Since the CTA bus tracker system only provides bus locations once every sixty seconds, we interpolate these traces down to one second intervals to allow for uniform processing. For the CTA, their official General Transit Feed Specification (GTFS) (63) feed, which serves as our ground truth, specifies both stop locations and route shapes.

6.3 Route Extraction

Route extraction is the process of turning unlabeled GPS traces into the set of route shapes followed by instrumented transit vehicles. These route shapes can then be used to label real-time GPS traces and classify transit vehicles as belonging to a particular route. Additionally, route-labeled traces are used to perform stop extraction, and the route shapes themselves may be used in drawing a user interface.

Since many transit vehicles travel on public roads, it may seem natural to base a route extraction algorithm on an existing road map. While this is a reasonable initial approach, it comes with two main drawbacks:

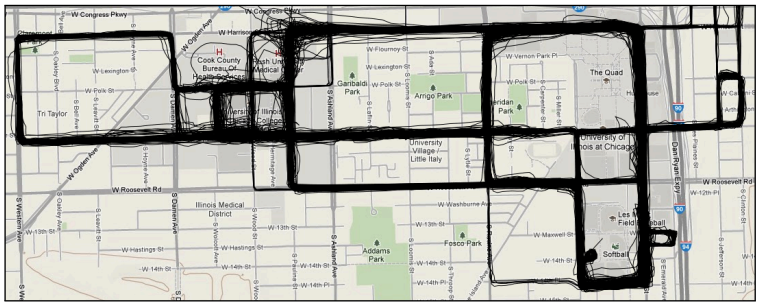
1. A completely accurate road map may not be freely available for the service area. While a free map such as OpenStreetMap (32) may visually appear accurate, errors in turn restrictions and connectivity are (anecdotally) common. This can result in significant errors in the routes produced.

2. Because transit vehicles may use limited-access service roads, or exclusive right-of-way transit lanes, we cannot rely on existing digital road maps to supply us with the unique road features that may be used by transit vehicles.

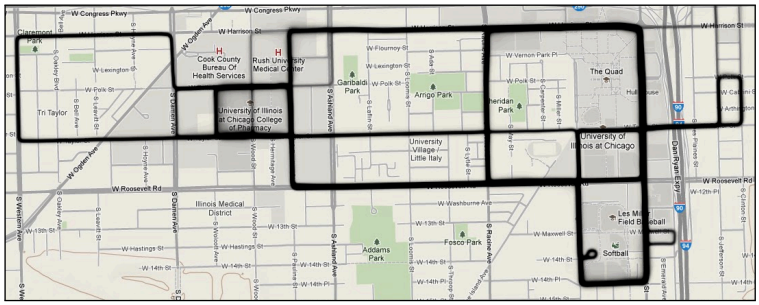
As an alternative, we evaluate the use of a KDE-based map inference algorithm to generate our own model of the road network from the GPS traces produced by our vehicles. This approach allows new road segments to be added on-demand: as soon as GPS trace data is available from transit vehicles, the portion of the road network that is newly utilized may be added to the map.

Figure 41 illustrates the complete automated route extraction process at a high level. Starting with an input of raw GPS traces (Figure 41(a)), a kernel density estimate of the full set of traces is computed (Figure 41(b)), and then the road map is constructed (Figure 41(c)).

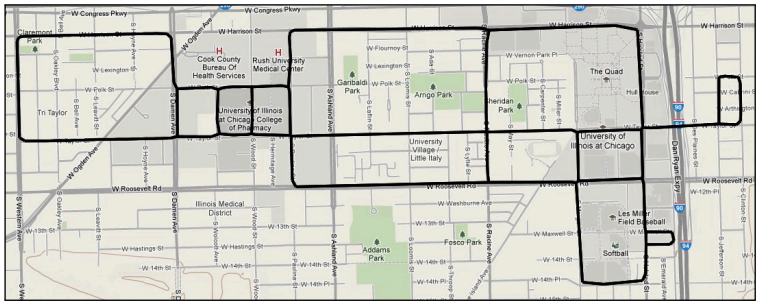
The inferred road map is used to map-match the raw GPS traces, turning each trace into a series of discrete road segments. These road segment series are then analyzed to find frequently



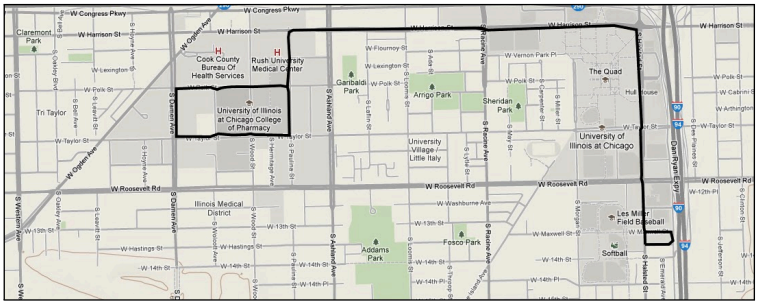
(a) Raw GPS traces drawn as a separate thin black lines.



(b) Kernel density estimate as a gray-scale overlay.



(c) Road map inferred from all traces.



(d) Single extracted route, out of several.

Figure 41. High-level overview of the route extraction process, overlaid on the local road map.

repeated sequences, which are output as the set of official transit routes. Figure 41(d) illustrates one of several extracted routes. Below, we describe the route extraction process in more detail.

6.3.1 Raw Data Pre-Processing

The first step in our process is to clean up and organize the raw data. Each GPS location along a trace is accompanied by a MAC address (identifying the vehicle) and a timestamp. Each trace is broken up into several *drives*, separated by long (10 minute) intervals without location reports. Such intervals typically indicate a parked vehicle, making them a natural delimiter.

Depending on the frequency with which the in-vehicle device is configured to record location data, we may collect a large amount of location points that are very close to each other, when a vehicle is stopped or moving slowly. For the purpose of route extraction we prefer a sparse representation of the traveled path, as extra points along an edge afford us no advantage. Therefore, we thin each trace to produce a linear density of locations in each direction of one point every 20 meters. This value was selected empirically, to balance between sufficient data density and reasonable runtime.

6.3.2 Route Extraction

To identify the routes followed by our transit vehicles, we first map-match our drives onto the edges of our inferred road map. This is done using the Viterbi algorithm, as described in (33; 43). The output of the Viterbi algorithm provides us with the maximum-probability sequence of edges traversed. This sequence of edges is then processed further to identify our set of routes.

We iterate through the edge-sequences sequentially until a repeated series of edges of a minimum sum length (we use a distance of 100 meters, or half a block) is encountered, taking direction of traversal into account. Note that we detect a series of edges with a minimum sum length, rather than a single repeated edge. This helps avoid problems where repeated traversals of the same intersection or circulation point may otherwise trigger the repetition detector.

The detected repetition conceptually completes a circuit in our graph, and the resulting subset of edges is considered a "route candidate." The route candidate is then stored, and processing continues from the first repeated edge onwards through the rest of the edge-sequence data. Once this process is complete, we have produced a collection of edge-sequences representing all of our route candidates. Note that we assume each route is cyclical (i.e., it eventually repeats). If a transit system were to contain non-cyclical routes, a different heuristic would be required for detecting and separating route candidates.

In order to identify the true routes from among all of the possible candidates, we count the number of instances of each route candidate. Figure 42 shows the traversal count for each identified route (normalized by the total number of drives, for readability), using several quantities of UIC shuttle data. Starting with the route with the highest count, we incrementally add routes to the set of official routes in decreasing count-order. For each added route, we use Welch's t-test (64), to compute a p-value describing the statistical difference between the current set of official routes and the set of remaining candidate routes. The location where the p-value is lowest delimits the set of official routes, completing the route extraction process.

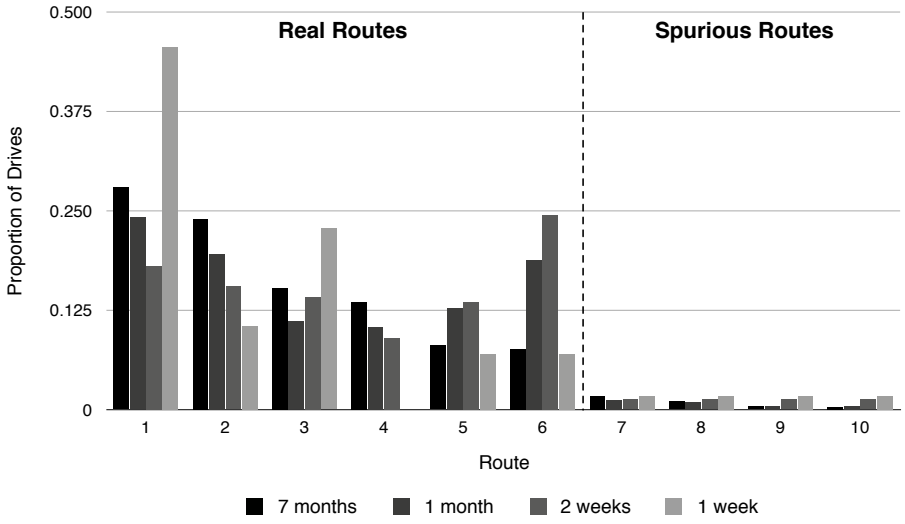


Figure 42. A collection of real and spurious routes, and the corresponding proportion of drives they represent over several quantities of data from the UIC campus shuttles.

In Figure 42, the six routes on the left with the highest counts were manually verified to coincide with the actual campus shuttle routes, whereas the four routes on the right are the four spurious routes with the highest counts, with many more left out. Spurious routes represent either noise in the underlying trips, or GPS noise. Trip noise may be one-off charter trips, and trips to and from the bus depot that do not represent actual campus shuttle routes. GPS noise on the other hand can produce spurious edges during a normal trip, resulting in a unique route.

6.3.3 Route Extraction Performance

To evaluate the accuracy of our route extraction algorithm we compare our set of extracted routes against the ground truth routes provided to us by the campus shuttle team, which are based on OpenStreetMap (32). The comparison is performed as follows: along the length of the

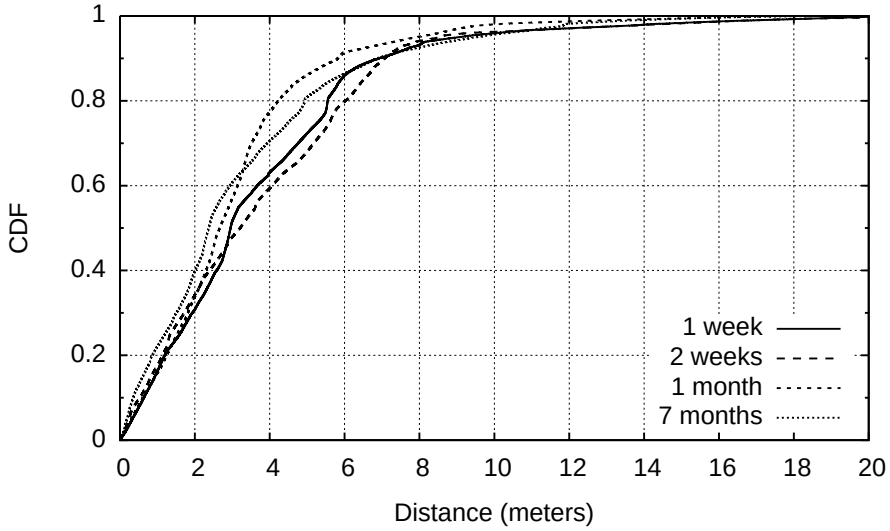


Figure 43. CDF of distance between generated routes and ground truth, for several quantities of data.

extracted route, select locations at 1-meter intervals. For each location, measure the distance to the closest ground truth edge.

Figure 43 shows the Cumulative Distribution Function (CDF) of the resulting distances for all routes, over several quantities of data. For all quantities, the 90th percentile is below eight meters. The ground truth is based on a route map which describes a four-lane road using a single centerline. Considering that the average lane width in the United States is approximately 3.35 meters (11 feet), an error below eight meters falls within the boundaries of a typical road. Hence, a difference of eight meters is not unreasonable for a bus route, which tends to stay on the right-hand side of the road.

Quantity of data	Mean error	Median error	Max error
1 week	3.7 m	3.0 m	21.3 m
2 weeks	3.8 m	3.2 m	22.8 m
1 month	3.1 m	2.7 m	17.8 m
7 months	3.2 m	2.4 m	20.6 m

TABLE II

ROUTE EXTRACTION ERROR PERFORMANCE.

However, for a small number of locations, we observe significantly larger errors. In Table II, we see that there exist errors in excess of 17 meters for all quantities of data. Through manual inspection, we found that these errors arise in the high-error "hospital area" of campus, described in Section 2.3.1 and Section 4.7. In this area the extracted roads are significantly offset from those in the ground truth, explaining the problem. The largest error observed across all quantities of data was 22.8 meters, or about one-tenth of a standard Chicago city block.

6.4 Stop Extraction

Stop extraction is the process of turning a set of raw GPS traces belonging to a given route, into a small set of coordinates indicating the locations of transit stops. The generated locations are used for producing arrival times for schedule extraction and arrival time prediction, and for drawing stop locations on a route map.

For schedule extraction and arrival time prediction, perfect accuracy of stop extraction is not required as long as most actual stops are represented. However, any omissions will result in the inability to predict arrival times for that stop. Furthermore, inaccuracies in stop locations

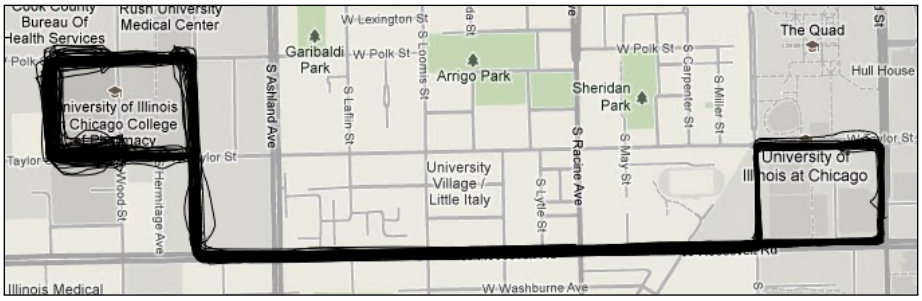


Figure 44. Raw GPS traces from a single route. Traces reveal little about amount of time spent.

when drawing the map can lead to missed buses and upset users. Therefore, our goal is to find as many actual stops as possible, while minimizing the number of spurious stops reported.

This is a challenging problem, as the movement pattern of a bus at a true bus stop is deceivingly similar to the behavior at traffic signals and stop signs. Moreover, error in GPS location introduces noise in the traces which complicates the identification of stopping events.

Raw GPS traces along an example route are depicted in Figure 44. Intuitively, buses spend more time at bus stops than in other locations. An estimate of the proportion of time spent in any location along the route is produced using kernel density estimation (14). This is accomplished by first map-matching the route-specific GPS traces to the route shape, and then producing a density estimate based on the distance along the route for each GPS location. Figure 45 shows the computed density estimate along the example route, where the taller peaks in the figure indicate regions with a higher density of GPS points—these are the locations where the bus spent the greatest amount of time.

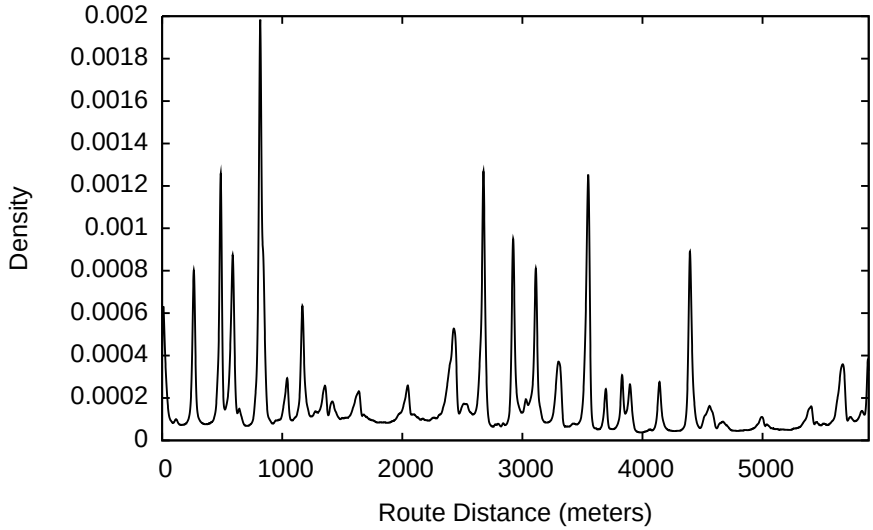


Figure 45. Kernel density estimate of raw GPS points. Vehicles spend more time at the taller peaks.

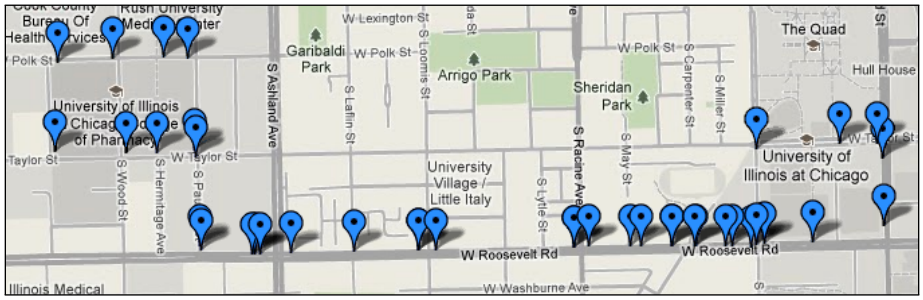


Figure 46. Detected stop locations after applying threshold and finding the maxima.

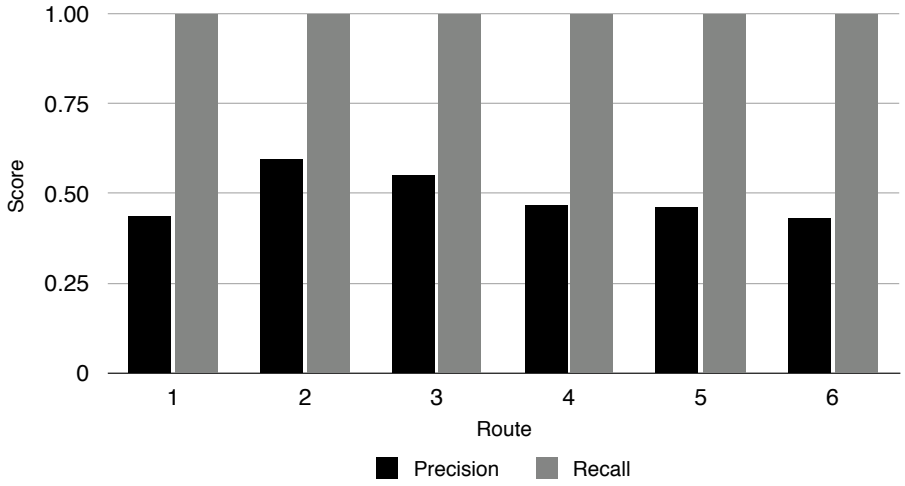


Figure 47. Precision and recall performance of the stop extraction algorithm on six routes. All stops were identified, but many spurious stops were also reported.

In order to identify stop locations, the density estimate is first passed through a threshold function. The threshold value used here is the median value of the density estimate. We reason that the median value represents the density from typical driving along the route without stopping, and therefore stops will not be found in regions with density less than or equal to the median. After thresholding, we locate the remaining maxima in the density estimate, and use their locations as stop locations. Figure 46 shows the detected stops along our example route.

6.4.1 Stop Extraction Performance

To evaluate the performance of the stop extraction algorithm, data from six different campus shuttle bus routes over seven months was used. Stops generated by our algorithm were compared against the set of ground truth bus stop locations for each route, and were determined to be

accurate representations of their ground truth counterparts if they were less than half a block (100 meters) away. Figure 47 illustrates the performance of the stop extraction algorithm in terms of precision and recall. Weekday, weekend, and an express route were all included, exhibiting different stopping behavior characteristics. The precision metric captures the fraction of correctly detected stops with respect to the total number of detected bus stops. On the other hand, the recall metric describes the fraction of correctly detected stops with respect to the total number of ground truth bus stops. As can be seen in the graph, the stop extraction algorithm achieves 100% recall, with around 50% precision across the six routes.

While a higher precision metric is desirable, it is difficult to achieve due to the strong similarity of bus stopping behavior at bus and non-bus stop locations. Intersections, traffic signals, traffic congestion, and stop signs can easily be confused with true bus stops. We investigated several methods based on both stop time and spatial distributions. Unfortunately, any performance improvements on training data came at the cost of over-training to specific cases. Thus, while our goal is to create a fully automatic system, and while the system does a great deal to whittle down the number of possible stop locations, it does to some extent fall short in the case of stop extraction. In Section 6.7, we discuss means by which these weaknesses can be addressed through manual intervention.

6.5 Schedule Extraction

Schedule extraction is the process of turning raw arrival times at stops along a given route into a service schedule for each stop. While the final derived schedule may be displayed to end users, its primary purpose is to support the internal arrival time prediction system (see

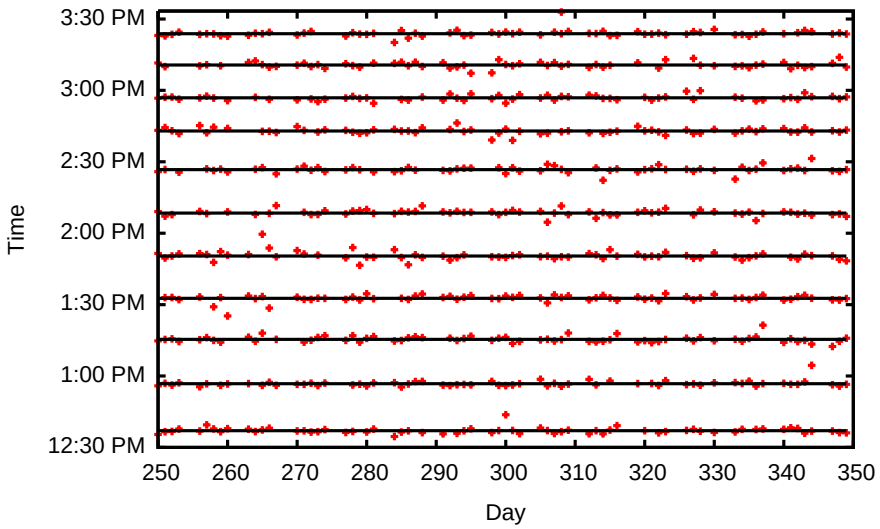


Figure 48. Arrival times for the first stop on a route. The horizontal lines indicate the k-means computed cluster centers at this stop for each daily trip.

Section 6.6). The schedule provides a fall-back alternative for predictions far into the future, at the beginning or end of the day, or when vehicle tracking data is unavailable or unusable. Automating the schedule extraction process helps to reduce deployment overhead and may in some cases be helpful if the transit agency lacks a formal schedule. Additionally, using an extracted schedule from GPS trace data can correct inaccuracies in the pre-existing schedule or detect undocumented changes in behavior.

Figure 48 and Figure 49 provide a graphical illustration of the challenges involved in deriving a static schedule from recorded arrival time data. In Figure 48, dots mark arrival times at the first stop of CTA route #157 in the interval 12:30–3:30pm over a span of 100 days (no service on weekends). Here, the underlying schedule is evident from the horizontal bands of arrivals

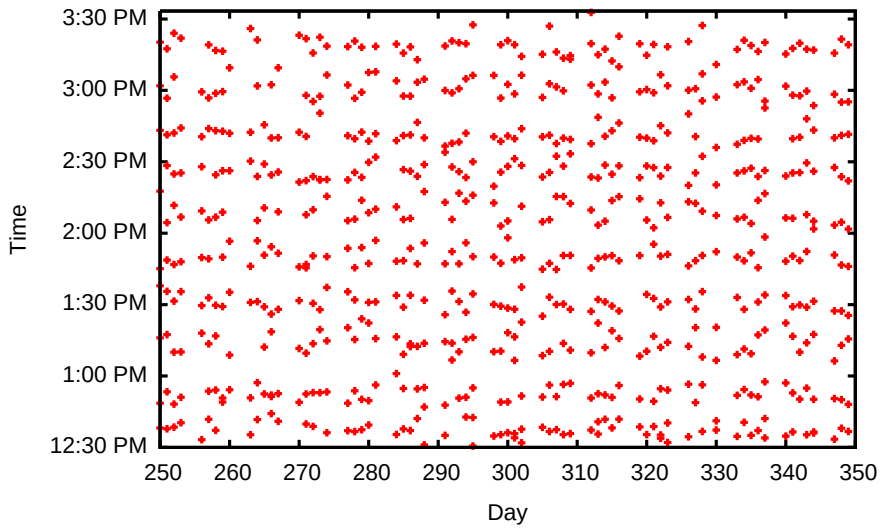


Figure 49. Arrival times at the last stop on a route. The data is too noisy to use the same (clustering) approach used in Figure 48.

at approximately the same time every day. The lines indicate the result of k-means clustering (65) these raw arrival times (discussed below), producing a high-quality schedule estimate. In contrast with the first stop, Figure 49 shows arrival times at the last stop on the route. Here, while some banding is evident, a schedule is very difficult to discern, and is beyond straightforward clustering.

Figure 50 further illustrates this difference in regularity: while travel-time uncertainty is present throughout the route, variance gradually builds as the bus travels away from the first stop, resulting in the disorganized arrival times shown in Figure 49. This invariably means that any schedule at the end of a route is going to be somewhat unreliable. For the route shown in Figure 50 the standard deviation at the last stop is roughly 10× greater than at the first stop.

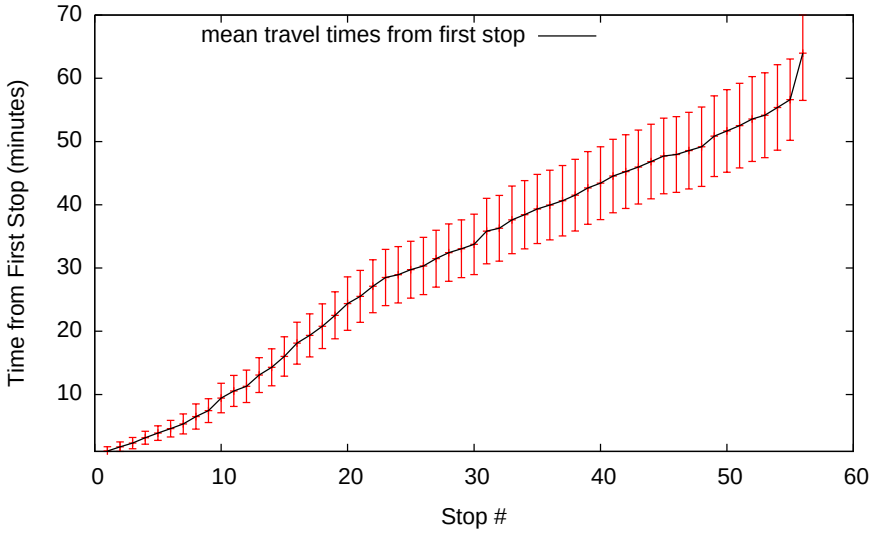


Figure 50. Per stop mean travel times from the first stop on a route, bars show standard deviation. Travel time variance increases with distance from the first stop.

Due to the high variance at later stops, we cannot rely on k-means clustering alone in order to produce an accurate schedule for every stop along the route. Below, we define the problem formally, and present our solution. In short, we use k-means clustering on data from the first stop to determine its schedule, and compute downstream schedules from a combination of the first-stop schedule and the estimated travel time from the first stop at a given time of day.

6.5.1 Problem Description

The input to the schedule extraction algorithm is a set of stops $S = \{s_1, ..., s_{|S|}\}$ and a set of trips $\mathbf{T}_D = \{T_1^D, T_2^D, ..., T_{|T_D|}^D\}$ on day D along a route, where each trip T_i^D contains a series of arrival times, one per stop along the route $T_i^D = \{a_1^i, a_2^i, ..., a_{|S|}^i\}$. When considering only a single trip, we simplify our notation for arrival times to a_s where s is the stop.

The schedule extraction problem is given $\mathbf{T}$, produce a set of schedule times $K_s^{D'} = \{k_1^s, k_2^s, ..., k_{|T_{D'}|}^s\}$, where k_i^s is the i^{th} scheduled arrival time at stop s, and D' is either the day of the week, or a member of the set $\{weekday, weekend\}$. For example, Figure 48 plots a_1^i (the arrival time points) for a range of i over all D, and k_i^1 (the horizontal lines) for the same range of i.

6.5.2 Estimating the First-Stop Schedule

As mentioned above, we produce only the first-stop schedule K_1 from the raw arrivals T_1^D. Schedules $K_2...K_{|S|}$ are derived from K_1 and the mean trip times for the time of day, as described in the next sub-section.

To extract the intended first stop arrival times, we use k-means clustering over all arrival times a_1^i in all trips T_i. We choose our value for k to be $median(|T^D|)$, the median number of trips observed on this route per day. The choice of initial values has a significant effect on k-means performance. After experimenting with several initialization methods from (66), we settled on the max-min approach described in (67). This approach initializes k seeds incrementally, choosing the next seed from a_s^i which is furthest away from the current collection of seeds. After we compute the k-means clusters, we have an accurate estimate of K_1, as shown in Figure 48.

6.5.3 Deriving Downstream Schedules

For two stops i, j and trip t, we define:

$$travel\_time(i, j, t) = \frac{1}{|D|} \sum_D a_j^t - a_i^t, \tag{6.1}$$

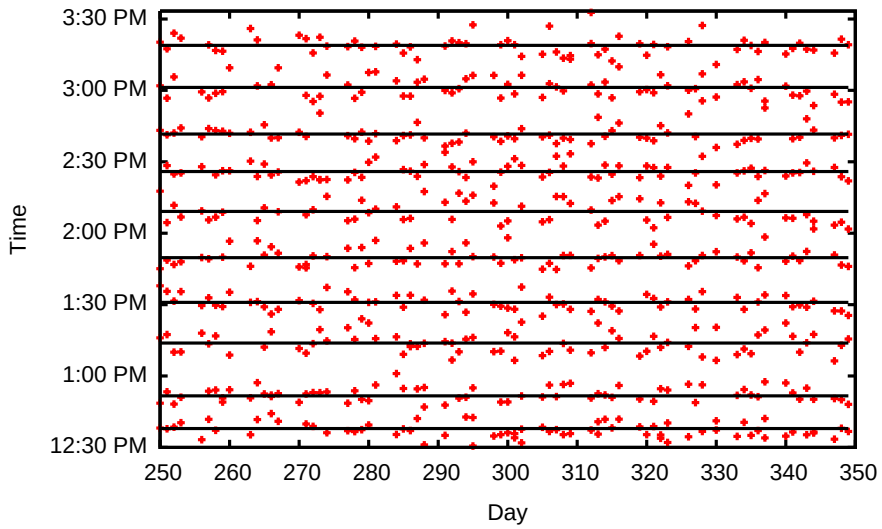


Figure 51. The schedule (horizontal lines) for the last stop based on the first stop schedule and mean travel times.

the mean arrival time difference for trip t between stops i, j over all days in D. Given the schedule for the first stop, K_1, we can then compute the schedule for a later stop s as $K_s = \{k_1^s, ..., k_{|T_D|}^s\}$ where:

$$k_t^s = k_t^1 + travel\_time(1, s, t) \tag{6.2}$$

As illustrated in Figure 50, the travel time variance increases as the bus travels further along the route, meaning the schedule will be increasingly inaccurate. Unfortunately, this is the nature of bus travel during traffic congestion—our goal is simply to produce the best schedule estimate.

Figure 51 illustrates an example output from the solution described above. Here, the black lines represent the schedule estimate computed from the first-stop departure schedule and the

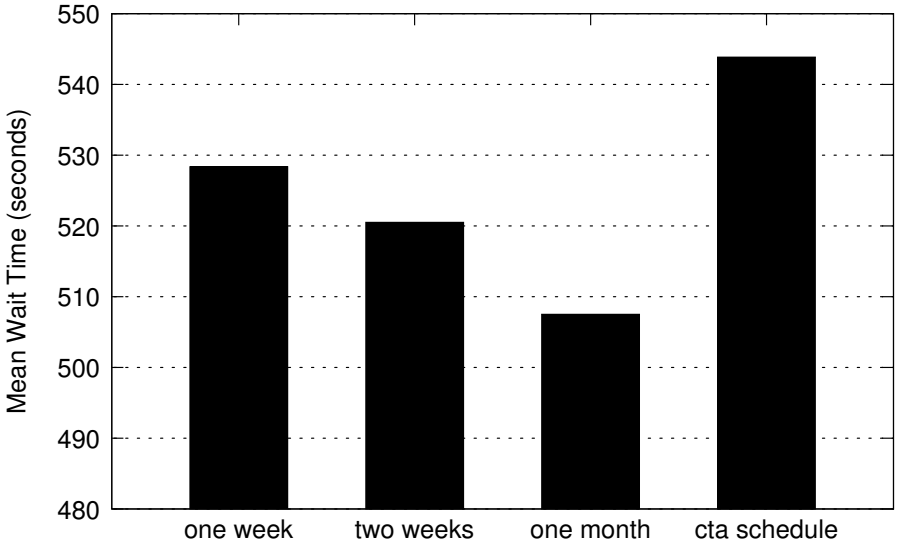


Figure 52. Mean wait times for a selected bus route for several training set sizes and the official CTA schedule.

mean travel times from the first stop. The resulting schedule shows reasonable arrival time intervals and a good fit with the raw data.

6.5.4 Extracted Schedule Accuracy

We evaluate the extracted schedule using actual CTA bus arrival time traces. For this evaluation, we repeatedly choose a random bus stop and time, and use our extracted schedule for that stop to predict when the next bus is due to arrive. To simulate the experience of a typical bus rider, we "arrive" at the bus stop 2 minutes before the scheduled arrival time (building in a small margin of error), and then wait until the next bus actually arrives, based on our recorded bus arrival time traces.

We refer to this as the *wait time*—if a passenger arrived at the stop based on the schedule, this is how long they would have actually been waiting for the bus to arrive. To see how the size of our training set impacts performance, we evaluated schedules generated from one week, two weeks and one month of data. For a given route, we performed this test 5,000 times and recorded the mean wait time for each schedule. As shown in Figure 52, the extracted schedule actually outperforms the official CTA schedule, potentially saving travelers an average of over 30 seconds. However, note the absolute values on the y-axis: even our most accurate schedule, based on one month's worth of data, does a terrible job at accurately predicting arrival times. Using the schedule for guidance, the mean time spent waiting for a bus is over 8 minutes.

Given the inadequacy of static schedules in predicting bus arrival times, real-time transit tracking and arrival time prediction is clearly called for. In Section 6.6.2, we evaluate the accuracy of a real-time arrival time predictor, and compare this to using the static schedule.

6.6 Online Processing

The online processing components combine the routes, stops, and schedules produced in batch processing, with the recent GPS trace of a vehicle to: (i) determine if the vehicle is in service (and if so, on which route), and (ii) estimate the arrival time of the vehicle at subsequent stops. Below, we describe how this is accomplished.

6.6.1 Route Classification

Given a sequence of GPS points $G = \{g_1, ..., g_{t-1}, g_t\}$ recorded up until time t, and a set of candidate service routes R, our goal is to determine whether the vehicle is currently driving

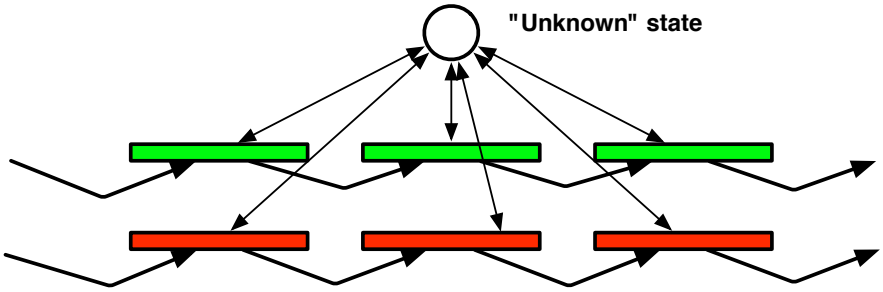


Figure 53. Route-matching Hidden Markov Model. Transitions between routes are only possible through the unknown state.

route $r \in R$. Assuming a vehicle serves at most one route in each uninterrupted drive, this can be determined by computing, for each route r,

$$dist = \frac{1}{|G|} \sum_{i=1}^{|G|} dist(g_i, r),$$

where $dist(g_i, r)$ is the minimum distance between point g_i and any segment of route r. To reflect a more realistic usage model, we need to relax these assumptions as follows:

- Vehicles may serve multiple routes in a single drive.
- Vehicles may change between in-service and out-of-service within a single drive.
- Vehicles may occasionally detour around closed roads or accident sites.

Thus, rather than make a single decision for an entire drive, we need to determine the status of each vehicle in an online fashion, as new GPS points arrive. To do this, we make use of Hidden Markov Model (HMM) map-matching (43; 33), with a map constructed from the known routes,

as illustrated in Figure 53. Here, the segments of each route are added as *separate* states, creating overlapping road segments where routes coincide, with no direct transitions between them. Transition and emission probabilities are left unmodified, except for the introduction of a single "unknown" state, representing out-of-service driving and detours, and serving as a place-holder for transitions between routes. Transitions are possible from each state to the unknown state, and from the unknown state to every other state, though the probability is a small non-zero constant. The emission probability of the unknown state is a small non-zero constant as well.

We use the Viterbi algorithm to decode the most probable sequence of states. As a consequence, once a vehicle is classified as belonging to a given route, it will remain in that state as long as it follows the route somewhat closely, independent of overlaps with other routes. Once it diverges significantly, it will either transition to the unknown state and stay there (if it is now out of service), or transition to the unknown state and then to another route (if it is now serving a different route).

We evaluate our HMM-based vehicle classifier on one month of labeled GPS trace data from the UIC campus shuttles. Specifically, we first split the labeled trace data into individual drives of each route. Then, for each drive we select a random starting point and run through the trace, measuring the distance traveled until a route classification is made. If we travel more than seven kilometers without making a route classification, we treat the route as "unclassified."

Out of our six routes, two are extended versions of two others. The existence of several different patterns belonging to a common route is a regular occurrence in transit networks.

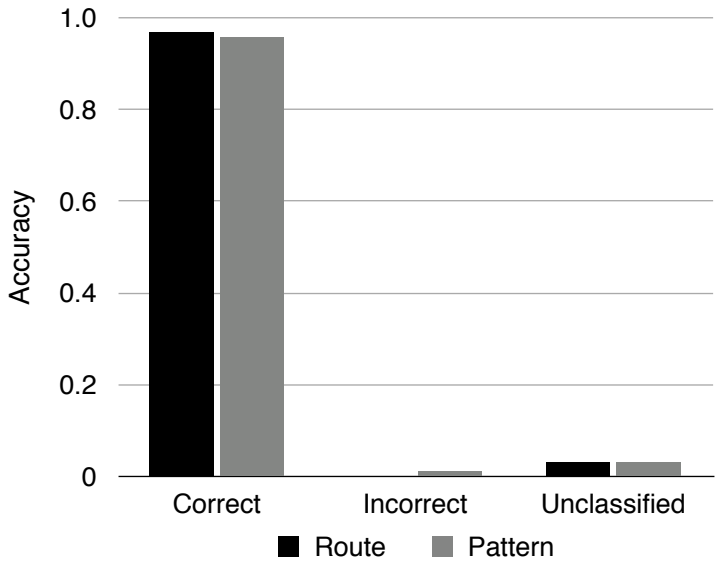


Figure 54. Classification performance of HMM-based vehicle classifier, using one month of labeled data from the UIC campus shuttles.

Typically, the variation served by a given vehicle depends on the time of the day, or the day of the week, which means they can be distinguished based on time, in combination with the spatial matching discussed here. To give an accurate picture of the route matching algorithm's performance in the face of such variations, we evaluate the accuracy of both unique pattern determination as well as aggregate route determination for these routes. To make the route or pattern classification decision, we analyze the Viterbi state probabilities with every new GPS point. When the most probable route/pattern is 10× greater than the next most probable route/pattern, the algorithm makes its determination.

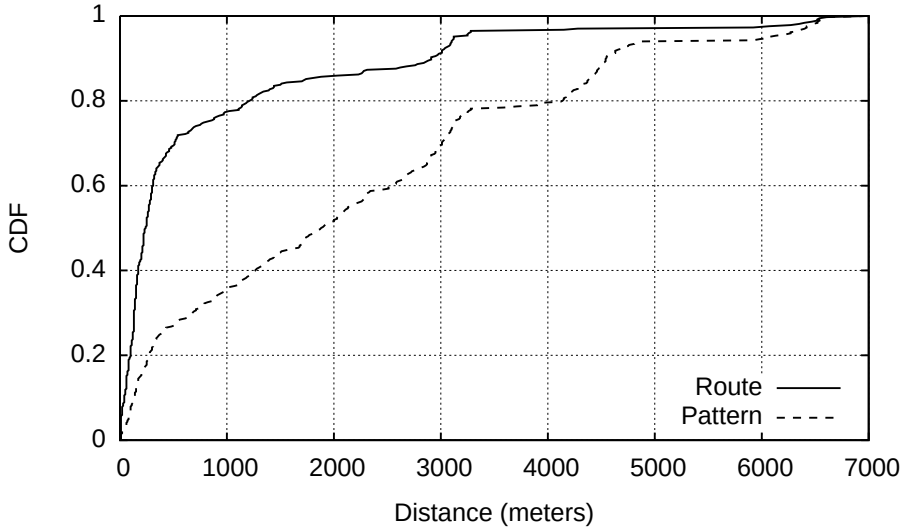


Figure 55. CDF of distance traveled before a correct pattern or route classification was made.

Figure 54 shows the overall performance of our HMM-based vehicle classifier for route and pattern determination. We see see that 97% and 96% of the time we are able to correctly classify the route and pattern, respectively. We can also see that 1% of patterns and 0% of routes are incorrectly classified, while 3% of patterns and routes remain unclassified after seven kilometers of driving.

From this, we can see that the algorithm is, with high probability, eventually able to accurately determine the route and pattern driven. But how long does it take to make this determination? Figure 55 is a CDF of the distance traveled before a decision was made. As expected, routes are classified more quickly than patterns, as no pattern classification can be made without taking time into account, until the patterns for a given route diverge. In 75%

of cases, 500 meters of travel is sufficient to distinguish the route traveled. For routes with patterns that largely overlap (as is the case here), or for routes that overlap substantially, the distance that needs to be traveled can be significantly longer. Overall, the performance of route classification depends heavily on the amount of overlap present in the transit network in question, as well as the typical driving patterns of transit vehicles. Once a decision is made, it will remain until the vehicle leaves the route. Hence, the initial classification delay is incurred only at the beginning of each shift.

6.6.2 Arrival Time Prediction

Arrival time prediction continuously estimates the next arrival time of a vehicle serving route r at stop s_i given a schedule estimate and (when available) the current location of vehicles currently serving the route. Typical circumstances under which the current vehicle location may be unavailable or insufficient include:

- The first trip of the day, before any vehicle has started serving the route.
- The first several stops of the route, when the next vehicle has not departed the first stop.
- The last trip of the day. Here, a schedule is needed to predict that the vehicle will subsequently be taken out of service.

We assume that every vehicle serving the route is equipped with a working in-vehicle device, and is reporting its location periodically. If no vehicle is present on the route preceding s_i, we estimate the arrival time based on the next departure from the first stop s_1 according to the extracted schedule, plus $travel\_time(s_1, s_i, t)$, computed as described in Section 6.5. Conversely,

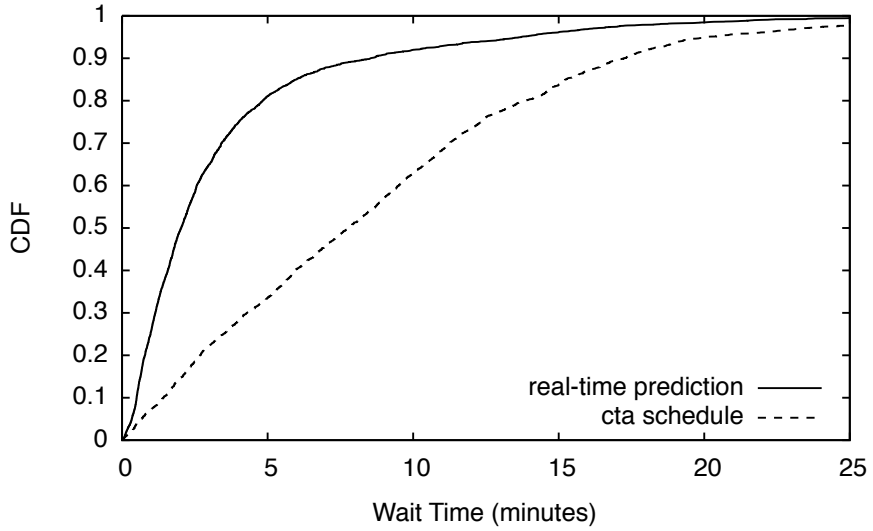


Figure 56. CDF of wait times for 5,000 real-time arrival predictions vs. the CTA schedule.

if a vehicle is present, then the time until its arrival at s_i is estimated based on the mean pairwise travel time between its current location and s_i, computed as:

$$(1-\gamma)\ travel\_time(s_{prev}, s_i, t) + \gamma\ travel\_time(s_{prev+1}, s_i, t), \tag{6.3}$$

where s_{prev} is the most recently served stop, γ is the fraction of the distance between s_{prev} and s_{prev+1} already traversed, and t is the current trip.

To evaluate the performance of our arrival time predictor, we perform the same experiment described for the extracted schedule. However, instead of consulting the schedule, we use the arrival time predictor, based on the location of the next bus that will arrive at the stop. Figure 56 clearly illustrates the benefit of incorporating real-time tracking when predicting bus

arrival times. Wait times are significantly reduced by the use of real-time data, bringing the median wait down from over 8 minutes to just 1 minute.

The reason behind this dramatic improvement can be seen in Figure 50. The route used for this evaluation has a mean service interval of approximately 15 minutes. Without real-time tracking, each vehicle spends the majority of its time in the latter parts of the route, where variance is high. With real-time tracking and a 15-minute service interval however, the bus is on average 7.5 minutes away from the stop in question. Assuming the travel time variance is loosely a function of travel time, it is easy to see from Figure 50 that the expected variance (at 7.5 minutes out) is very small. Hence, no matter where the bus is when we consult the predictor, we are likely to get an accurate prediction.

6.7 From Paper Product to Production System

In this section, we briefly discuss additional steps required to take the proposed system from its current form to production use. We also briefly mention additional features that may be incorporated to further improve the rider experience.

The system described thus far takes no manual input: routes, stop locations, schedules, vehicle classification, and arrival time predictions are all based purely on unlabeled GPS traces. As we have shown, our system is able to (with the exception of stop locations and occasional slow vehicle classifications) produce results similar to or better than data entered by hand.

However, EasyTracker cannot produce the kind of transit tracking system users have come to expect without a bit of manual input. In addition to spurious stop locations, and the occasional

slow route classification, the system lacks several kinds of annotations, such as stop and route labels, that transit riders would typically expect to see.

From a technical point of view, the stop extraction algorithm tends to produce spurious stop locations at a number of traffic lights and stop signs, and the vehicle classification algorithm is sometimes unable to distinguish between similar routes. These errors need to be corrected, either by additional sensor modalities, or by human intervention.

Below, we discuss two optional system components: an administrative web interface and a driver interface, that provide means of integrating a small amount of manual input into the system to significantly improve the user experience.

6.7.1 Optional Management Interface

We propose to complement the automatic system with an optional web-based management interface. The purpose of this interface is to allow a dispatcher (or other office personnel) to enter additional, static annotations to the system which cannot be automatically derived from GPS traces:

- Route labels, such as “Lakeshore Drive” or “Route #60”.
- Stop labels, such as “City Hall,” or “Roosevelt/Halsted.” Reasonable stop labels can be inferred from a road map, but these may not correspond to the labels on stop signs or in paper schedules.
- Accessibility information, such as “elevator available” or “has bike rack.”

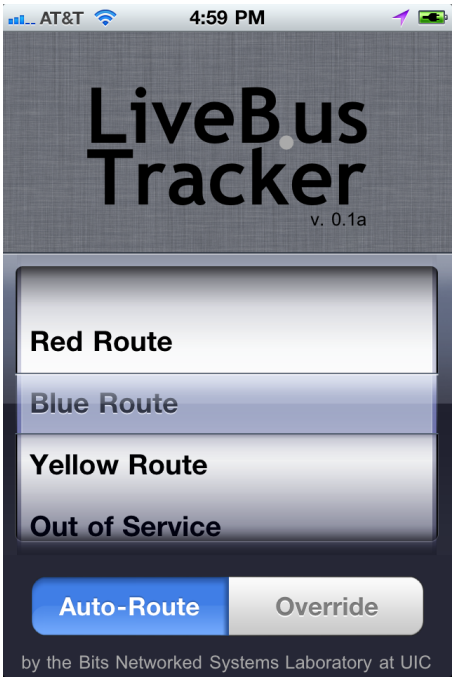


Figure 57. Prototype driver interface for optional manual input and communication with dispatch.

Through the management interface, the transit operator is presented with the tools to correct mistakes, add route and stop labels, and other relevant annotations. In addition, the management interface may allow an operator that is aware of impending route or schedule changes to proactively "reset" selected routes, to avoid inertia in the acquisition of updated routes and schedules.

6.7.2 Optional Driver Interface

In addition to the static annotations that may be entered through the administrative web interface, drivers may optionally be trained to provide additional information through an in-

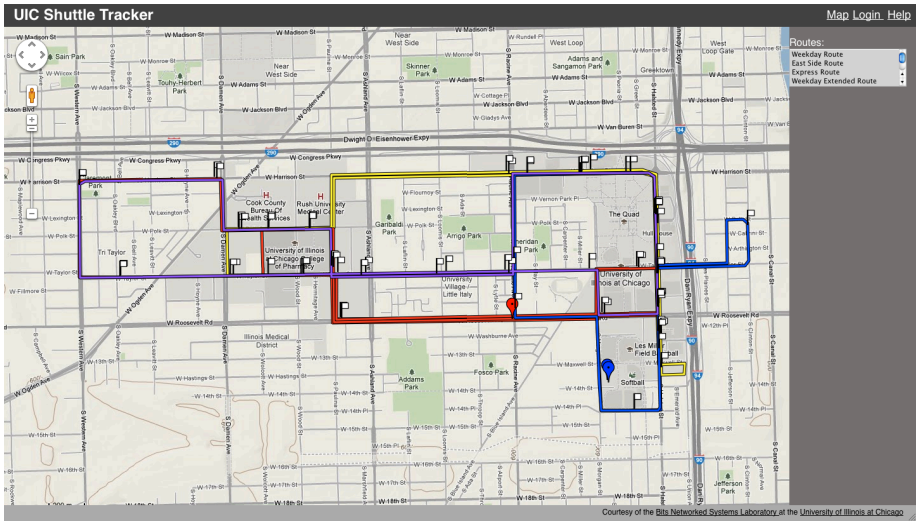


Figure 58. Screenshot of current EasyTracker prototype, currently in production use on the UIC shuttle bus system.

terface on the in-vehicle device. Figure 57 illustrates an envisioned driver interface. Here, the driver may manually override the automatic route classification in case of a misclassification. Other data that the driver may be asked to provide includes passenger occupancy, availability of seats, room for wheelchairs/strollers, and bike rack occupancy.

To further improve the passenger experience, the smartphone may be connected to the vehicle speaker system to provide voice prompts to passengers, notifying them of the next stop. Finally, the smartphone interface may be used as a means for central dispatch to communicate with the driver in the form of text prompts when the vehicle is not moving.

6.7.3 Current Prototype System

Parts of EasyTracker are currently in use on the UIC campus shuttle system. Figure 58 is a screenshot of the web interface we provide to campus shuttle riders today. In the current prototype, vehicles are automatically classified as belonging to a red, blue, purple, or yellow route using the algorithm in Section 6.6.1 and arrival times are predicted using the method in Section 6.6.2. Routes, stop locations, and schedules from Section 6.3–Section 6.5 were manually corrected to remove any mistakes.

6.8 Conclusion

In this chapter we have presented EasyTracker, an automatic system for low-cost, real-time transit tracking, mapping and arrival time prediction. Based on our experience with building a campus shuttle tracking system for UIC, we have discovered how labor intensive the collection of this data can be. To address this problem, we have demonstrated how high-value data such as routes, stops, and transit schedules, can be inferred automatically from simple GPS traces. Our system produces high-fidelity route maps, extracts transit stop locations, and constructs transit schedules that consistently out-perform the official schedules produced by the Chicago Transit Authority. Last but not least, EasyTracker provides accurate transit tracking and real-time arrival time predictions, all without manual intervention.

6.9 Copyright Information

The material in this chapter originally appeared in the following publication: Biagioni, J., Gerlich, T., Merrifield, T., and Eriksson, J.: EasyTracker: Automatic Transit Tracking, Mapping, and Arrival Time Prediction Using Smartphones. In Proceedings of the 9th ACM

Conference on Embedded Networked Sensor Systems, pages 68–81. http://doi.acm.org/10.1145/2070942.2070950.

CHAPTER 7

ASSESSING DAY SIMILARITY FROM LOCATION TRACES

Both consumers and corporations recognize the value of location traces for understanding people's daily habits and anticipating occasional needs. These traces can help in understanding our daily activities; in particular, we can use location traces to find anomalous days and to cluster similar days, leading to a better understanding of daily routines. Both of these tasks require a way to compare days to one another.

In this chapter we develop and test algorithms to measure the similarity of days represented by location traces, tested against similarity assessments from real users. With a reliable way to measure similarity we can find days that stand out from the rest as anomalies, which may indicate confusion (an important phenomenon to detect among populations of users with cognitive impairments (68)) or a change of habits. We can also make sensible clusters of days that belong together to assess variety and make predictions about how a day will evolve, providing useful basic knowledge for adaptive systems to leverage.

While a variety of sensors could be used to characterize a day, such as activity measured on a person's mobile phone, desktop computer, vehicle, social networking sites, biometric sensors, etc., here we use location traces, measured with GPS. One advantage of this is that location is a constantly existent state (if not always measurable) as opposed to event-based activities, such as texting events, that only happen occasionally. Location is also dynamic for most people and

easy to measure outdoors with GPS. These characteristics make location a convenient variable to use for measuring the similarity between a person's days.

Existing similarity techniques (see Chapter 8 for more details) depend on learning a model of normal behavior from observation, which means they must be trained anew for new subjects. One of our goals is to find a single similarity measure that works well for multiple people, without requiring any training. In addition, previous techniques detect dissimilar behavior based on an algorithm or threshold designed by the researcher. Instead, another of our goals is to find a similarity measure that approximates what a human subject would say about their own data. Achieving these goals will allow us to provide future systems with a way to accurately reproduce human assessments of day similarity that works well for the general population, and requires no training time; perhaps helping to mitigate the cold-start problem in relevant applications.

To achieve this goal, we: (i) gather an average of 46 days of GPS traces from 30 volunteer subjects, (ii) conduct a user study, wherein our 30 volunteer subjects are randomly shown pairs of pairs of their days and asked to assess their similarity using custom-designed software, (iii) implement and evaluate eight different similarity algorithms in an effort to accurately reproduce our subjects' assessments, (iv) perform statistical analyses to find those algorithms that best reproduce the assessments from our subjects, and (v) apply one of our similarity algorithms to the task of clustering days using location traces.

7.1 GPS Data and Preprocessing

In order to perform our experiments for assessing day similarity based on location traces, we gathered GPS data from the vehicles of volunteers. This section describes our data logging and preprocessing for the experiment described in Section 7.2.

7.1.1 GPS Data from Volunteers

We logged GPS data from 30 volunteers (8 female). Each volunteer borrowed a RoyalTek RBT-2300 GPS logger and placed it in their main vehicle, powered by the cigarette lighter. All of our subjects were employees of Microsoft Corporation in Redmond, WA, USA, and most were compensated with a $30 USD cafeteria spending card. A few subjects agreed to participate without any compensation. Our goal was to collect at least six weeks of data from each subject. In the end we obtained GPS data for an average of 46 days from each subject, varying from 20 to 60 days, where the majority of the recorded drives consisted of simple weekday home/work commute trips and weekend drives in the local community; a dataset we believe generalizes well to the larger population of people with regular work routines. Each subject was in possession of the GPS logger for at least six weeks, but some did not drive every day. In order to reach 30 subjects, we started logging with 39 subjects, but later found that 9 did not provide suitable data due to mysterious stoppages in logging, a late refusal to log, frequent sharing of their vehicle (which violated our survey criteria), and two unexpected departures. We also ignored two subjects who had only 14 and 18 days of logging.

The loggers were set to record a time-stamped coordinate pair every 10 seconds. Figure 59 shows a short sequence of GPS points from one of our subjects with 10-second sampling. Since

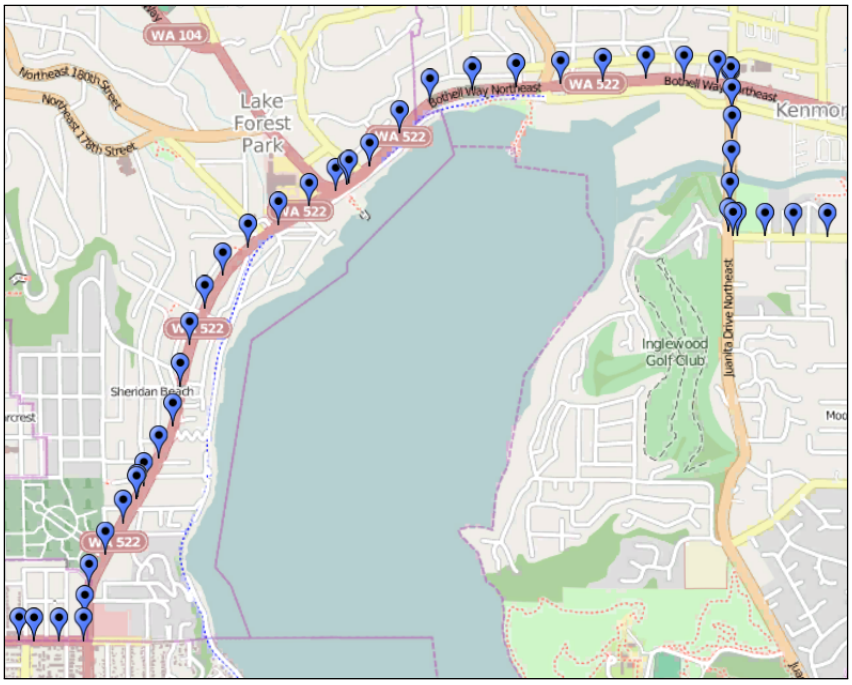


Figure 59. A short sequence of GPS points sampled at an interval of 10 seconds.

we ran our loggers without their rechargeable batteries, they logged only when the vehicle's cigarette lighter was powered. For some vehicles, this happens only when the vehicle is turned on, and for others the cigarette lighter is powered continuously. In our preprocessing phase, detailed below, we filled in gaps corresponding to these and other limitations of the recorded GPS stream.

7.1.2 GPS Data Preprocessing

In order to attach some semantic information to the raw GPS data, our first preprocessing step was to automatically detect the time and location of all stops in the raw traces. For our purposes, a stop is defined as any location in the GPS data where we detect that the

subject/vehicle remained within a 300-meter (radius) circular region for five or more minutes. These parameters were chosen based on a training dataset, whose subjects were not included in our final evaluation.

In order to produce an initial set of candidate stops, we first made a linear time-ordered pass through the GPS trace data and marked those locations that met our stop criterion, defined above. Because a stop location that was visited more than once during the course of the recorded GPS trace would have more than one stop representation in our data, we then collapsed those redundant representations into one final stop. Doing so allowed us to associate a set of aggregate knowledge with the actual stop location. For example, consider the case of a subject's work location; over the course of a typical work week their trace data will initially represent "work" with five separate stop representations (one for each day). By collapsing these five representations into one, we obtain one stop location that represents the aggregate knowledge of the original five (i.e., days of the week the location was visited, times the subject arrived/departed, etc.), which is significantly more useful than five disparate time/location observations. In order to collapse the stops, we applied agglomerative clustering (26) to the candidate stops using the same 300-meter threshold (as above) as the criterion for merging.

Once we determined the final set of stops, we then leveraged the aggregate information contained therein to apply semantic labels to certain stops. Specifically, we used data from the American Time Use Survey (69) to classify the most-likely pair of stops as either *Home* or *Work* locations. Since our final stops contained knowledge of the days and times of arrival/departure, length of stay, and frequency of visits, we built and trained a classifier to perform probabilistic

Home/Work labeling based entirely on these criteria. Since Home/Work stops occur very frequently in many subjects' GPS datasets, it was important to be able to distinguish them for our subjects' later assessment of their data. Specifically, having these labels helped our subjects orient themselves quickly and easily to the type of days they were observing (e.g., weekday/weekend), and distinguish between regular and anomalous days more easily.

Finally, as one last preprocessing step, we created a "symbolized stop representation" of each day of data from the raw GPS traces (where a *day* is defined from 4:00am–3:59am). Specifically, for each location in the raw GPS data, we replaced its coordinate pair with its associated "Stop ID" (a unique identifier associated with each stop), and interpolated in time for those vehicles that logged only when they were turned on. If a given coordinate pair was not associated with (i.e., located at) a stop location, it was replaced with a "From Stop ID–To Stop ID" pair, denoting travel between stops. Simplifying the raw trace data into a series of symbols denoting time spent at (and traveling between) stops not only provides us with a more compact representation of the trace data, but also a more abstract representation for use with evaluation algorithms that aren't geographically-aware (see Section 7.3).

7.2 Human Assessment of Day Similarity

Our goal is to find an algorithm that can assess the similarity of days in a way that matches human assessment. Toward this end, each of our 30 subjects was invited to run a custom program that displayed, and asked them to make similarity assessments on their own recently recorded data. The program started by displaying a calendar indicating the days for which we

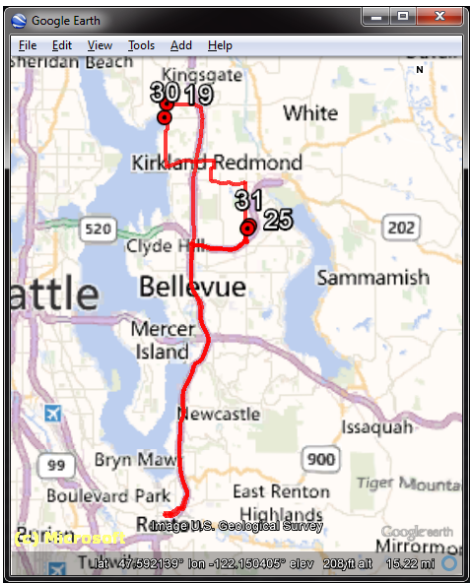
had GPS data available for the subject. For a selected day, the program showed that day's location traces in three different ways:

1. **Map**. An interactive map, shown in Figure 60(a), displayed the stops we found (as described in Section 7.1.2), each with its unique ID number. It also showed the GPS traces between the stops. This visualization emphasized the spatial layout of the day's trips and stops.

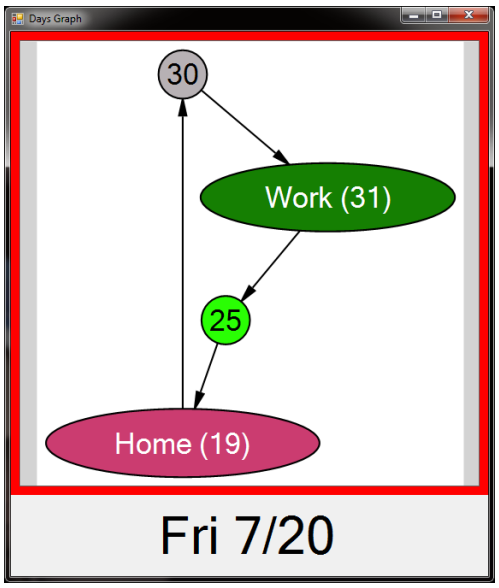
2. **Graph**. An interactive graph, as in Figure 60(b), showed the subject's stops as nodes and their trips as edges. Thicker edges indicated more trips between their connected stops. The Home and Work stops were labeled if we found them, otherwise stops were labeled with only their unique ID number that matched the numbers on the map. Clicking on a node or edge in the graph highlighted the corresponding stop or GPS trace on the map, making for convenient exploration. This visualization emphasized the number of stops and the transitions between them.

3. **Timeline**. A timeline, as in Figure 60(c), displayed each stop in a different color block, laid out along a horizontal timeline. The time periods denoting trips between stops were colored black. This visualization gave a temporal view of the day that was lacking in the other two.

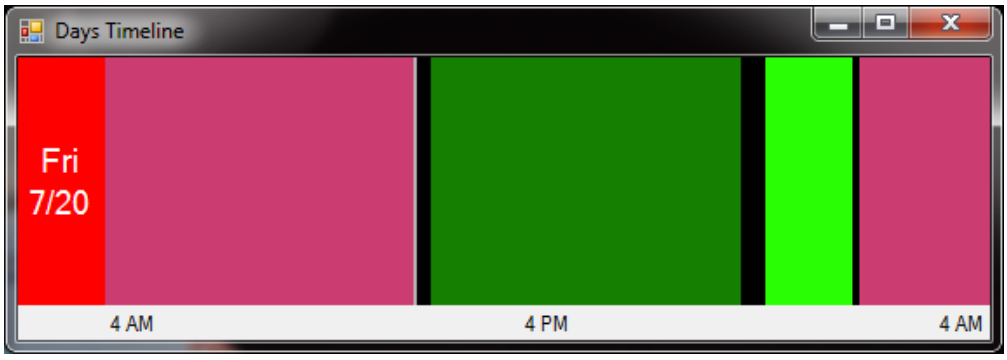
After starting the program, we asked each of our subjects to familiarize themselves with the visualizations by picking a day and briefly describing it to us using the visualizations. The main part of our user study came next: each subject was asked to assess the relative similarity



(a) An interactive map for viewing a day's GPS data.



(b) A graph view of a day's GPS data, showing stops and the trips between them.



(c) Timeline view of a day, showing stops as blocks of color and trips between stops as narrower bands of black.

Figure 60. Visualizations of a day for our subjects.

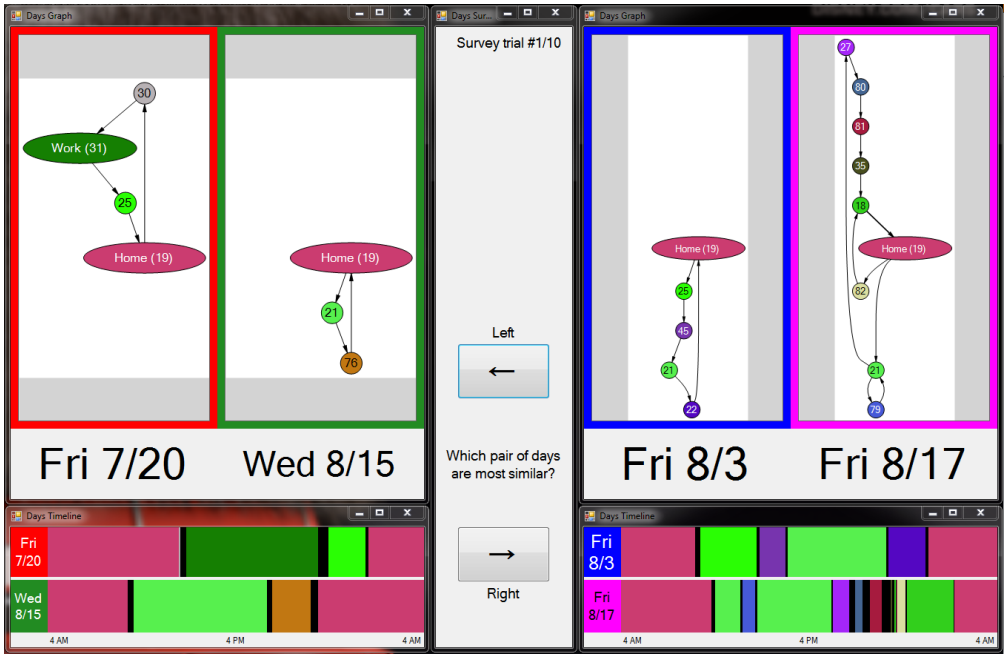


Figure 61. The main part of our user study, where we asked subjects to indicate which pair of days were most similar to each other.

of pairs of pairs of their days. That is, each subject was shown four randomly selected days simultaneously, using the visualizations described above, and as shown in Figure 61. We then asked the subject to indicate which of the two pairs was most similar. For instance, if the two pairs of days were A and B, and C and D, we asked the subject to indicate if A and B were more similar to each other than C and D, or vice-versa. We chose this simple assessment after first piloting a different survey that asked subjects to give a numerical similarity rating to a pair of days. This proved too difficult, so we reverted to this simpler question about the relative similarity of pairs of days. The example shown in Figure 61 is a good representation of the

complexity of the typical comparison problem presented to our subjects; with an average of five stops per day, the left-most pair of days represents a simpler case, and the right-most pair a more complex case. Each subject rated 30 pairs of pairs, which took approximately 30 minutes in total for each subject.

7.3 Algorithms for Assessing Day Similarity

To find an algorithm that computes a numerical similarity (or, "distance") score between pairs of days that matches the similarity rankings of our subjects, we implemented and evaluated four trajectory similarity algorithms in both *standard* and *modified* forms. The standard form of each algorithm is that given by its original definition, described in the following sub-sections. The modified form of each algorithm consists of its original definition being adapted to use Dynamic Time Warping (DTW) (70), a technique which allows us to relax the assumption that activities between pairs of days be aligned in time. For example, consider two days A and B consisting of the same simple "Home → Work → Home" activity pattern. On Day A, the subject leaves home at 8:30am, arrives at work at 9am, departs work at 6pm, and returns home at 6:30pm. On Day B, the subject leaves home at 8am, arrives at work at 8:30am, departs work at 5:30pm, and returns home at 6pm. Since days A and B both consist of a 9-hour work-day with a 30-minute commute from/to home, subjectively speaking they are virtually identical. However, because of the 30-minute time-shift between them, they will necessarily incur a penalty from any objective similarity measure. Therefore, our motivation behind evaluating a DTW-modified version of each algorithm was to establish whether our subjects ignore these shifts

in time, and if so, to more accurately capture and reproduce the subjective nature of their evaluations. Below, we describe the four standard trajectory similarity algorithms.

7.3.1 Edit Distance

Edit distance measures the number of "edit" operations needed to transform one string of symbols into another. In our case, this algorithm operates on the symbolized stop representation of our trace data (as discussed in Section 7.1.2), and therefore the "symbols" referred to here correspond to Stop IDs and From Stop ID–To Stop ID pairs.

Valid edit operations include: insertion, deletion, and substitution. In our evaluation, we used the canonical Levenshtein (71) implementation of this algorithm, where a unit cost is assigned to each of these operations. The performance of this similarity metric, in both its standard (denoted "without Dynamic Time Warping") and modified (denoted "with Dynamic Time Warping") form can be seen in Figure 62.

7.3.2 Distance Sensitive Edit Distance

The standard edit distance algorithm (described above) operates entirely on the symbolized stop representation of a given day, without taking into consideration the stops' geographic locations. In order to account for the geographic location of stops we modified the standard Levenshtein algorithm (71) to use great-circle distance, measured using the Haversine formula (72), as its cost function for each of the edit operations. This means, for example, that the cost of performing the *substitution* operation for two Stops #60 and #157 is no longer 1, but rather the distance in meters between Stops #60 and #157 according to their coordinate locations. The performance of this similarity metric can be seen in Figure 62.

7.3.3 Stop Type Distance

The symbolized stop representation of a subject's days requires an exact correspondence between IDs to be considered a match. Because this definition can be overly restrictive, we generalized the representation of each stop by classifying its location *type*. In order to perform this classification, we provided the coordinates of each stop to Bing Local Search, which returned a list of categorized businesses and their distances from our stop within a radius of 250 meters. Example business types include "Restaurant," "Grocery and Food Stores," and "Banks and Credit Unions," among many others. Using this data we then built a location-type probability distribution for each stop, based on the proportion of returned business types and weighted by their distance from the original stop location.

Replacing each Stop ID with its corresponding location-type probability distribution, we then computed the distance between days as the sum of the KL-divergence (73) scores between their probability distributions. The performance of this similarity metric can be seen in Figure 62.

7.3.4 Sum of Pairs Distance

This metric (74) computes the distance between days based on their raw location traces, rather than the symbolized stop representations used above. As a result, this metric does not take into account any of the related semantic information.

Sum of pairs distance measures the sum of the great-circle distances between every pair of trace locations. Since this metric requires that the traces for days A and B be of equal length,

we first perform simple linear interpolation and then compute their distance. The performance of this similarity metric can be seen in Figure 62.

7.4 Results

We evaluated our similarity algorithms both on the task of matching our subjects' similarity assessments, and on a clustering task.

7.4.1 Matching Subjects' Similarity Assessments

We ran our eight similarity algorithms on the data from our 30 subjects. Recall that each subject was shown 30 sets of 4 days each. Each set of four days was split into two pairs, and the subject chose which pair was most similar. We gave these same sets of days to our similarity algorithms and recorded their assessment of which days were most similar. The accuracy results we report show the proportion of human decisions our algorithms were able to correctly reproduce.

The accuracy results are shown in Figure 62. Ignoring statistical significance, the best performing algorithm was Sum of Pairs Distance w/DTW, with a mean accuracy of 75.5% (SD=10.4%). This algorithm looks at the great circle distance between points in the two location traces, with local adjustments for time shifts. In second place was Distance Sensitive Edit Distance w/DTW with an overall mean accuracy of 74.2% (SD=9.3%). The fact that our two best-performing algorithms both base their distance metric on actual geographic distance is telling; clearly our subjects associate day similarity with geographic proximity.

Since we computed the accuracy for each subject, this provided 30 sample accuracies for each algorithm, allowing for a statistical analysis. We began with a one-way, repeated-measures

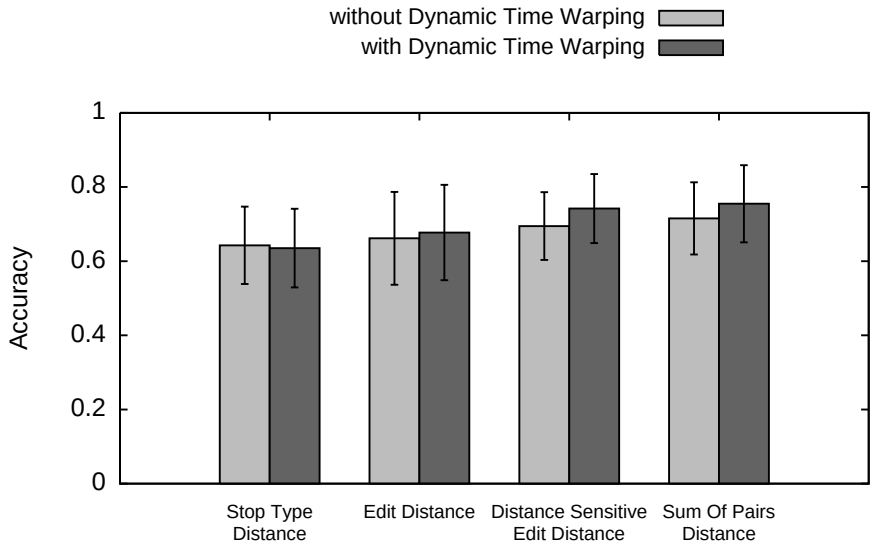


Figure 62. Accuracy results for our eight similarity algorithms. Error bars show +/− 1 standard deviation over all 30 test subjects.

Analysis of Variance (ANOVA) test, which resulted in $F(7, 29) = 11.22$, $p = 5.45 \times 10^{-12}$. This is evidence that the choice of algorithm has a statistically significant effect on accuracy. We next performed one-tailed, paired-sample t-tests of the means between each pair of algorithms, with a Holm-Bonferroni (75) correction to account for the multiple t-tests. Of the 28 possible pairs of algorithms, 16 pairs had statistically significant mean accuracy differences at the $\alpha = 0.05$ level. Table III tallies the wins and losses of each algorithm. The algorithm with the best performance record is Distance Sensitive Edit Distance w/DTW, with five wins and no losses. The next best algorithm is Sum of Pairs Distance w/DTW, with four wins and no losses. There was no statistically significant difference in performance between these two algorithms. Of these two, Sum of Pairs Distance w/DTW is easier to implement, since it does not require

Algorithm	**Wins**	**Losses**
Edit Distance w/o DTW	0	3
Edit Distance w/DTW	0	1
Distance Sensitive Edit Distance w/o DTW	2	2
Distance Sensitive Edit Distance w/DTW	5	0
Stops Categories Distance w/o DTW	0	4
Stops Categories Distance w/DTW	0	4
Sum of Pairs Distance w/o DTW	3	0
Sum of Pairs Distance w/DTW	4	0

TABLE III

NUMBER OF STATISTICALLY SIGNIFICANT WINS AND LOSSES FOR OUR SIMILARITY ALGORITHMS.

the identification of stops in the location traces. While the two best-performing algorithms both used DTW, it produced a statistically significant performance improvement for only the Distance Sensitive Edit Distance algorithm, over its non-DTW counterpart.

Overall, for accuracy and ease of implementation, we are inclined to recommend Sum of Pairs Distance w/DTW as the best algorithm we tested for assessing the similarity of days.

7.4.2 Application to Clustering

One application of our similarity measure is clustering, where we can find groups of similar days. We tested this by having our 30 subjects assess clusterings of their own days. We clustered days with a spectral clustering algorithm (76), using the Edit Distance w/o DTW algorithm as our distance metric. Edit Distance w/o DTW had a mean accuracy of 66.2% (SD=12.5%), slightly lower than the best accuracy of 75.5% for Sum of Pairs Distance w/DTW. We used

Edit Distance w/o DTW for our survey, because at the time we conducted our study we hadn't yet been able to test for the best performing algorithm.

For the clustering portion of the survey, we asked each subject to increment through the number of clusters, k, starting at two. For each k, the program displayed the clustered days in groups using the same visualizations described in Section 7.2. An example of a timeline showing three clusters is depicted in Figure 63, where the day-groupings are indicated by the colored labels on the left-hand side of each row.

Each subject was asked to pick the best k and then to rate the clustering on a Likert scale by indicating their level of agreement with the statement, "My days have been accurately separated into sensible groups." The results of this question are shown in Figure 64, where we see that 20 out of 30 subjects answered either "Agree" or "Strongly agree," indicating that the clustering was generally successful. This, in turn, further supports the assertion that our Edit Distance w/o DTW similarity algorithm comes close to matching human similarity assessments. We would expect Sum of Pairs Distance w/DTW to work even better, since it was the most accurate algorithm based on our analysis in Section 7.4.1.

7.4.3 Visualization Usefulness

Finally, as part of our survey, we asked each subject about the usefulness of our three visualizations for assessing the similarity of days. The three visualizations were the map, the graph, and the timeline (depicted in Figure 60). After each subject assessed 30 pairs of paired days, we asked them to rate the usefulness of each visualization on a Likert scale with the

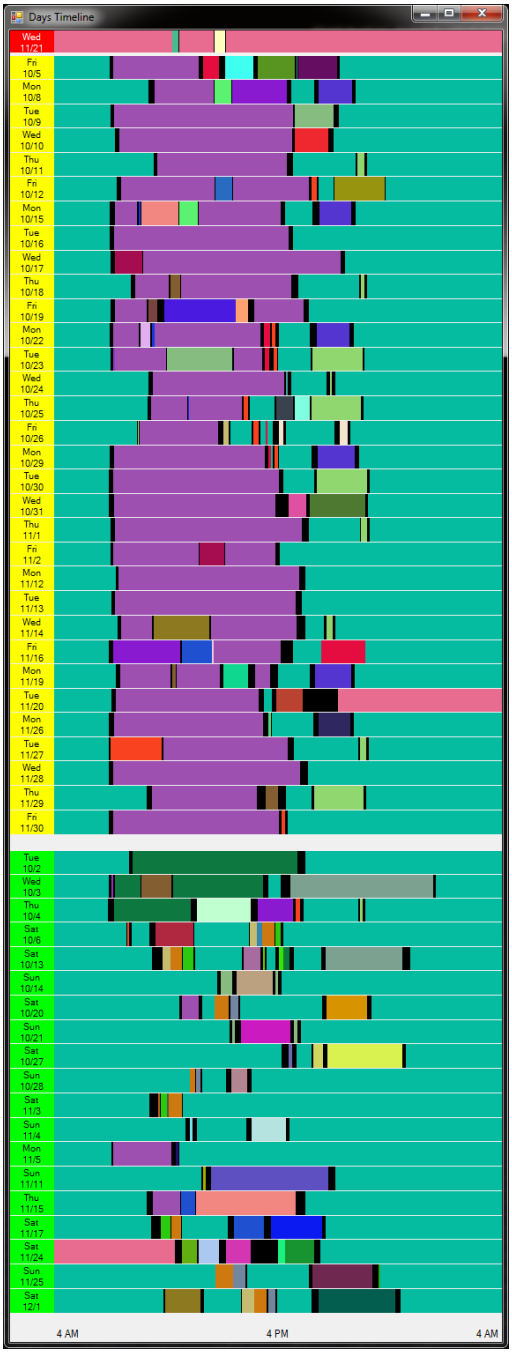


Figure 63. Three clusters shown on a timeline. Each row is one day. The single day in the top cluster is an outlier. The main central cluster shows 32 work days, and the bottom cluster shows 19 non-work days.

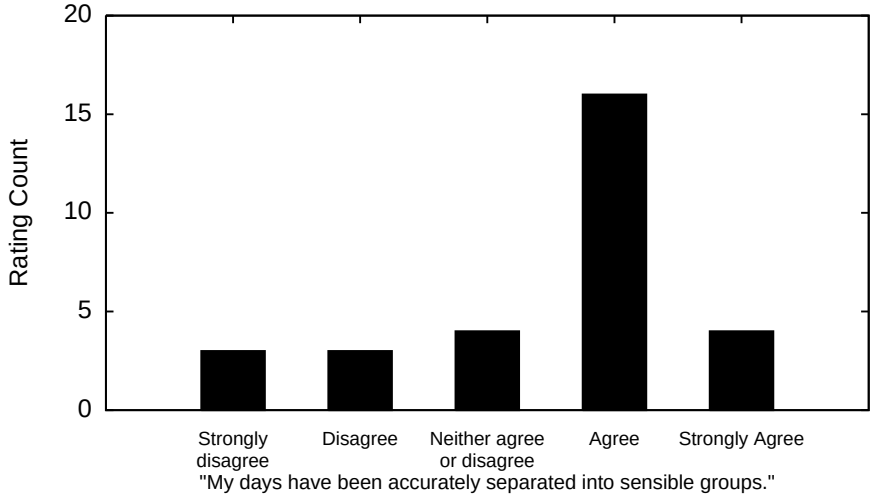


Figure 64. Most of our subjects were happy with the clustering results.

assertion, "The <visualization> was a useful way to assess the similarity of pairs of days," where <visualization> was "map," "graph," or "timeline."

Figure 65 shows the numerical results. To detect any statistically significant difference in responses, we performed a one-way, repeated-measures ANOVA test. We found $F(2, 29) = 21.58$, $p = 9.86 \times 10^{-8}$, indicating there is a statistically significant difference of opinion among the three different visualizations. Furthermore, one-tailed, paired-sample t-tests of the means, with a Holm-Bonferroni (75) correction at the $\alpha = 0.05$ level, showed that the map was considered significantly more useful than both the graph and timeline, and the graph was significantly more useful than the timeline.

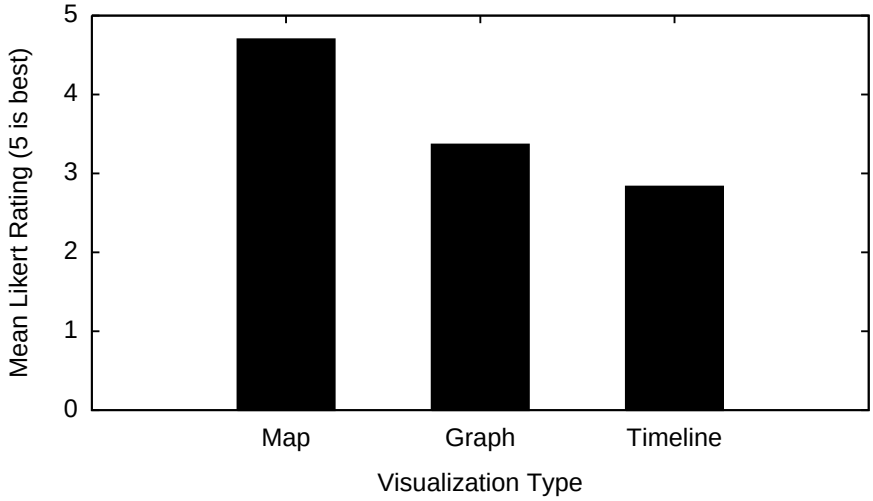


Figure 65. Our subjects rated the usefulness of our three different visualizations for assessing the similarity of days.

We can relate these preference results to the relative accuracies of our similarity measures: both Sum of Pairs Distance and Distance Sensitive Edit Distance are highly geographic in nature, a quality that is emphasized in the map visualization. However, we must also note that the rating of a visualization is necessarily influenced by its layout, colors, and other design elements, in addition to the data it shows; qualities we did not assess in our evaluation.

7.5 Conclusion

Based on a survey of 30 subjects, we evaluated the accuracy of 8 different similarity algorithms on their location traces. We found that two algorithms, Sum of Pairs Distance w/DTW and Distance Sensitive Edit Distance w/DTW, worked best at matching human assessments of

day similarity. We also showed that one of our similarity algorithms worked well for clustering days of location traces, based on an evaluation from our subjects.

In addition to clustering, these similarity algorithms can potentially be used to find anomalies and help predict behavior. None of our algorithms depend on training, so they are generic across all users, and therefore relatively easy to use.

We envision future work in this area may explore other similarity algorithms as well as experiments to detect anomalies. We would expect anomaly detection to work well because of the good performance shown here by our algorithms at reproducing human assessments of the similarity of days.

7.6 Copyright Information

CHAPTER 8

RELATED WORK

In this chapter, we review the existing academic literature that pertains to each of the topics presented in this thesis: (i) inferring road maps from GPS traces, (ii) online GPS tracking, (iii) automatic transit tracking, mapping, and arrival time prediction, and (iv) assessing day similarity from location traces.

8.1 Inferring Road Maps from GPS Traces

As discussed in Chapter 2, there has been considerable interest in the problem of inferring road maps from GPS traces over the past decade, with many innovative solutions being presented. Those papers that exist in the open literature are summarized in Table IV. Out of the 11 papers, 6 report k-means based methods, 2 report trace merging methods, and 3 report KDE methods.

As noted in the "Evaluation Method" column, the literature has thus far almost exclusively relied on a qualitative method ("eyeballing") for evaluating performance. Typically, a sample of the generated road geometry is overlaid on a map or satellite image, from which conclusions are drawn manually. In the cases where any type of numerical accuracy result is reported (15; 10; 17; 12), no comparison is made to related work. In cases where a comparison is made against related work (19), only a qualitative comparison is offered, with no numbers reported.

Paper	**Class**	**Data**	**Ground Truth**	**Evaluation Method**	**Features**
Edelkamp & Schrödl (15)	k-means	250 synthetically perturbed traces	gen. from GPS traces	lane error vs. amount of data	lane finding
Schroedl, Wagstaff, Rogers, Langley & Wilson (10)	k-means	250 synthetically perturbed traces	gen. from GPS traces	lane error vs. amount of data	intersection geometry
Davies, Beresford & Hopper (22)	KDE	1 million GPS points	UK ordnance survey	eyeball vs. ground truth	
Worrall & Nebot (16)	k-means	traces from mining vehicles	none	compact vs. raw	compact representation
Guo, Iwamura & Koga (17)	k-means	synthetic GPS traces	none	relative error vs. amount of data	
Chen & Cheng (23)	KDE	traces from automobiles	Google Earth	eyeball vs. ground truth	
Niehoefer, Burda, Wietfeld, Bauer & Lueert (12)	trace merging	7 traces	Google Maps	eyeball vs. ground truth, relative error vs. amount of data	edge classification
Cao & Krumm (20)	trace merging	20M GPS points from campus shuttles	Bing Maps	eyeball vs. ground truth, route query vs. Bing Maps	GPS trace clarification
Shi, Shen & Liu (24)	KDE	"massive amounts" of GPS traces	Google Earth	eyeball vs. ground truth	
Jang, Kim & Lee (18)	k-means	GPS traces	Naver maps (77)	eyeball vs. ground truth	
Agamennoni, Neito & Nebot (19)	k-means	5 days/15 open mine vehicle GPS traces	none	eyeball vs. (22) and (10)	principal road path

TABLE IV. MAP INFERENCE LITERATURE IN CHRONOLOGICAL ORDER.

Below we briefly reintroduce the three categories of map inference algorithms, and discuss the variations on each offered in the literature.

8.1.1 Map Inference Based on the k-means Algorithm

The k-means based approach begins by distributing a series of "cluster seeds" at locations drawn from the set of trace data, with the constraint that every trace point must be within a fixed distance d and bearing difference δ of a cluster seed. Using the cluster seeds as initial estimates, minor variations on the k-means algorithm are then used to find seed locations and headings that best describe the raw traces. Once the seed locations are settled, they are linked to form road segments, based on the pattern of raw traces passing between them. These segments then represent the map inferred using this technique.

8.1.1.1 Edelkamp and Schrödl (15)

In addition to being the original k-means based method, this paper further refines the road network model by fine-tuning the location of intersections, representing the road centerline by a smooth curve rather than a series of straight lines, and performing lane finding. To refine the location of intersections, clusters located at the end-points of contiguous road segments are incrementally advanced towards identified "merge-zones," until they are determined to overlap. The road centerline is refined by performing spline-fitting (78) through the series of clusters located along contiguous road segments, and lane finding is then performed by clustering the raw traces based on their respective distance-offset from the road centerline. This work was evaluated by comparing the number of lanes detected, and the lane position error against a ground truth map generated by the authors, using survey-grade differential GPS equipment.

8.1.1.2 Schroedl, Wagstaff, Rogers, Langley and Wilson (10)

Based on Edelkamp and Schrödl (15), this paper describes a process for additionally refining the intersection geometry. Specifically, instead of treating intersections as simple point features, they model the individual lanes involved and the transitions and turn restrictions between them. This is accomplished by first identifying the location of intersections and determining their bounding boxes. Traces that pass through an intersection's bounding box are then grouped by their entry and exit points, and spline-fitting is applied to each group to produce the final turn-lane geometry of the intersection. This work was evaluated by visually comparing the generated intersection geometries against a ground truth map provided by NAVTEQ (79).

8.1.1.3 Worrall and Nebot (16)

The aim of this paper is to infer a compressed road map, represented as a set of lines and arcs. In order to create this representation, the authors first use a method based on Edelkamp and Schrödl (15) to infer the set of road clusters, and then segment them into regions of "constant curvature." The best line or arc is then fitted to each segment. Standard regression analysis (80) is used for regions representing straight lines, and non-linear least-squares fitting (80) is used for regions representing arcs. This work was evaluated by measuring the error between the compressed map representation and a cluster-based (15; 10) alternative.

8.1.1.4 Guo, Iwamura and Koga (17)

Similar to the goal of Worrall and Nebot (16), the aim of this paper is to generate a set of spline curves representing the road network. To accomplish this, least squares approximation is first used to derive a set of "feature points" (akin to cluster seeds in (15)) from the raw

trace data. Spline curves are then fit to these points and used to represent the centerline of the road. This work was evaluated by studying the stability of the inferred map with respect to the amount of trace data used. No comparison against a ground truth map or competing methods was made.

8.1.1.5 Jang, Kim and Lee (18)

This paper proposes a heuristic performance optimization for the method described by Edelkamp and Schrödl (15), by first splitting the geographical space covered by the trace data into minimum bounding rectangles. This work was evaluated by visually comparing the inferred map against a ground truth map provided by Korean web portal Naver (77).

8.1.1.6 Agamennoni, Neito and Nebot (19)

The approach taken in this paper is similar to Schroedl et al. (10), except the focus is to extract "principal road paths," which are curves as defined in (81). This work was evaluated by visually comparing the inferred map against its underlying trace data. In an extended technical report of this work (82), the authors perform a visual comparison against the k-means approach described by Schroedl et al. (10), and the kernel density estimation method provided by Davies et al. (22). A limited quantitative evaluation of this method is also presented in (82), using a GPS trace dataset collected by the authors.

8.1.2 Map Inference Based on Trace Merging

Trace merging methods begin by iterating through each recorded GPS trace, adding edges from the raw trace data to the map unless an edge sufficiently similar in location and bearing already exists. Should such an edge already exist, its "weight" is instead incremented, and in

post-processing any edges with weight below a threshold are removed. Those edges that remain form the map inferred using this technique.

8.1.2.1 Cao and Krumm (20)

The method proposed in this paper precedes the standard trace merging approach with a "clarification" step. The clarification step is a type of particle simulation (21), where a strong, but short-range attractive force is applied to pull together nearby traces, and a weaker, but long-range retractive force is used to keep traces from straying too far from their original location. This reduces the effects of GPS noise by pulling together traces that originate from the same road, forming tight bands along the road centerlines. The clarification process also separates lanes of traffic traveling in opposite directions, by repelling traces moving in opposite directions rather than attracting them. This work was evaluated by visually comparing the inferred map against satellite imagery from Bing Maps, and also by visually comparing shortest-path routes from the inferred map against those generated by Bing Maps.

8.1.2.2 Niehoefer, Burda, Wietfeld, Bauer and Lueert (12)

This paper modifies the standard trace merging method by adjusting the position of existing road segments when merging a new trace segment into the already-existing map. This technique allows the location of road segments in the base map to be steadily refined as more traces are added, in a similar spirit to the "clarification" pre-processing step used by Cao and Krumm (20). This paper also describes techniques for automatically classifying road types, such as highways, streets or walkways, as well as entry and exit ramps, bridges, and tunnels. A qualitative evaluation of this work was conducted by visually comparing the inferred map against the

same region depicted in Google Maps. A quantitative evaluation was also performed against a "reference map" inferred using all available traces, whereby the relative position error of a road segment is shown to rapidly decrease with increasing amounts of data.

8.1.3 Map Inference Based on Kernel Density Estimation

Map inference methods based on kernel density estimation begin by computing a two-dimensional histogram of trace points or edges over the area of interest. They then convolve the histogram with a Gaussian smoothing function (26) to compute an approximate kernel density estimate, which is thresholded to produce a binary image of the road outlines. Finally, they apply a variety of methods to produce road centerlines from this binary image, and the resulting centerlines represent the map inferred using this technique.

8.1.3.1 Davies, Beresford and Hopper (22)

In the approach proposed in this paper, not only are the individual GPS samples quantified in the histogram, but also the lines between them. Specifically, they are accounted for in each grid cell they pass through by an amount proportional to the length of line that passes through each cell, akin to the anti-aliasing (83) method common in computer graphics. After thresholding the density estimate, the outlines of the produced road "bitmap" are extracted using a contour follower (84). To find the centerlines of these outlines, which are likely to coincide with the centerlines of the underlying roads, the Voronoi graph (27) of points evenly spaced along the contours is produced, followed by the removal of edges that fall outside the contour, or that are of insufficient length. Finally, separately produced KDEs of traces in each of the 8 cardinal and ordinal directions are used to annotate each road segment with its allowed

directions of travel. This work was evaluated by visually comparing the inferred map against that of a UK Ordnance Survey.

8.1.3.2 Chen and Cheng (23)

The method described in this paper starts by producing a point-based density estimate, and then extracts the road map from the bitmap image produced after thresholding, using image-processing techniques. Specifically, they use morphological "dilation" and "closing" to produce a smooth and contiguous image of the road boundaries. Then, a "thinning" operation deletes all pixels on the boundaries of the pattern until only a skeleton remains along the road centerlines, which is then converted into road segments. This technique was evaluated by visually comparing the inferred map against the same region depicted in Google Earth.

8.1.3.3 Shi, Shen and Liu (24)

The approach taken in this paper is very similar to that of Chen and Cheng (23), wherein image-processing techniques are used to extract a "thinned" road network skeleton. However, the authors of this paper also introduce a novel algorithm known as "combustion" for robustly extracting the road network from a pixelated skeleton. This technique was evaluated by visually comparing the inferred map against satellite imagery from Google Earth.

8.1.4 Gray-scale Skeletonization

Related to our gray-scale skeletonization algorithm in Section 4.3, there is existing work by Li, Bai and Liu (85), and Yim, Choyke and Summers (86). Li et al. (85) look at the problem of skeletonizing gray-scale images that lack a contiguous contour. Starting from the boundary segments of the gray-scale image, their algorithm develops a so-called "strength map" from

which they are able to extract the skeleton. Yim et al. (86) look at the skeletonization of gray-scale images from magnetic resonance angiography, as a means to identify the paths of small vessels and their tree structures. The technique employed in this work starts by modeling the image as a directed acyclic graph, and then applies selection and pruning methods to isolate the most salient features in order to extract a final skeleton.

8.1.5 Road Centerline Finding

Related to our centerline finding approach (using gray-scale skeletonization in Section 4.3), prior work on KDE algorithms have employed a variety of novel techniques. In Davies et al. (22), a form of Voronoi tessellation is used within the boundaries of the binary road network. By pruning away short "dead-end" segments, they are able to recover a contiguous spline along the approximate road centerline. In Chen and Cheng (23), morphological "dilation" followed by "closing" operations are used to fill gaps in the binary road network representation, and then morphological "thinning" is used to extract the centerline of the resulting shape.

Finally, the watershed transform (87) can also be used to extract road centerlines. However, this transform suffers from two major weaknesses that make it unsuitable for our purposes. First, it depends on a method for identifying a set of local minima from which to initiate the flooding process, and reliably choosing minima that produce a good set of ridges is non-trivial. Second, the watershed transform is fundamentally unable to detect dead-end roads, since these cannot form a separate flooding basin.

8.2 Online GPS Tracking

In Chapter 5 we present a system for online GPS tracking that seeks to reduce data uplink usage while preserving tracking performance. This is an active area of research, with a substantial publication record over the past two decades, on topics including: (i) energy conservation, (ii) GPS trace compression, and (iii) moving objects databases.

Due to the high power consumption of GPS receivers and their popular use in mobile, energy-limited devices, many researchers have focused on improving the energy efficiency of GPS tracking (46; 88; 89; 47; 48; 49; 33; 50). By contrast, in our work we assume that power is plentiful, or that the GPS is already active for a primary application. GPS trace compression (45; 90; 91; 58; 92; 93; 88; 94) is an often-used approach for producing a compact representation of a trace, however, while these techniques can be helpful in reducing the size of each transmission, they cannot reduce the number of transmissions without sacrificing timeliness.

Although we do make use of existing GPS compression techniques, the focus of our work is online tracking, where forwarding decisions are made as GPS points become available. The seminal work in this area was published by Wolfson and Sistla et al. (95; 51; 96; 52), where they present methods for updating databases that track moving objects. Most similar to our work is (52), where they propose update policies based on dead-reckoning, wherein the moving object and database maintain a synchronized notion of the object's position between updates, and update the object's database position only when the distance between the object's actual and expected location deviate by too large of a margin. Our work builds upon theirs in two ways: (i) whereas their system requires knowledge of the object's expected path of travel in

order to perform extrapolation, our system does not impose any such requirement, instead extrapolating the object's future path of travel using techniques that only rely on its history of previous locations, and (ii) while their system permits tuning of the update cost coefficients (used to control the trade-off between factors in optimizing performance), our system permits direct control over the costs themselves, giving users the ability to set hard limits on their range of values, and enabling concrete performance guarantees.

Work by Čivilis and Jensen et al. (97; 53; 54) builds directly on the efforts of Wolfson and Sistla et al., and develops an architecture that lays the foundation for our work by adding support for several alternate extrapolation techniques. Our work extends upon theirs in two specific ways: (i) whereas their system requires the *manual* selection of one particular extrapolation method, our system provides a unified extrapolator that *automatically* selects the best method for the current conditions, and (ii) while their system controls the transmission of location updates through the specification of a maximum error-threshold, our system permits specifying either an error- or budget-threshold, while it automatically optimizes the other.

Existing work by Lange et al. (98; 99; 100) looks at the problem of online trajectory reduction. Here, the objective is to store an approximation of a moving object's trajectory with the fewest possible vertices, while simultaneously ensuring it doesn't deviate from the actual trajectory by more than some specified accuracy bound. While this work bears some similarity to ours, it differs in two ways: (i) the authors' primary concern is storing a compact and accurate trajectory in real-time, with object tracking being only a secondary concern, whereas we're primarily concerned with object tracking, with accurate trajectory representation and storage

being only an incidental concern, and (ii) the technique their system uses for extrapolation is limited to simple linear dead-reckoning, whereas we provide myriad alternatives through the use of our unified extrapolator.

8.3 Automatic Transit Tracking, Mapping, and Arrival Time Prediction

While transit tracking is already a popular service offered by commercial providers, the EasyTracker system presented in Chapter 6 is to our knowledge the first to automate the entire process, from raw GPS traces to a complete transit tracking and arrival time prediction system.

In (101) and (102), a system for cooperative transit tracking is described. Here, it is assumed that the routes and schedules are known, but that the transit agency is not willing to install tracking devices. Instead, users cooperatively track transit vehicles through software that automatically reports their location when they are riding in a bus or train.

TransitGenie (103) is a transit navigation service for smartphones that computes route recommendations based on real-time transit information, as opposed to static schedules. TransitGenie is complementary to EasyTracker, in that it makes use of the tracking information produced by the service to provide travel advice to end-users.

8.3.1 Arrival Time Prediction

In Section 6.6.2 we present a very simple arrival time prediction system that relies on the schedule and mean trip times computed in Section 6.5.3. While this technique produces satisfactory results, more sophisticated bus arrival time prediction methods can be found in (104; 105) and (106).

8.4 Assessing Day Similarity From Location Traces

In Chapter 7 we develop and test algorithms for assessing the similarity of a person's days based on location traces recorded from GPS sensors. While the GIS community has looked extensively at location trace similarity (e.g., Deng, Xie, Zheng and Zhou (107)), these efforts are aimed primarily at machine processing. In this thesis we are interested in matching human assessments of similarity, which appears more commonly in research for anomaly detection. In (108) Ma detects anomalies from GPS traces by first representing a normal trace as a sequence of rectangles on the ground. An anomaly is declared if a new trace's rectangles are sufficiently different from those of the normal trace. Here the similarity measure is explicit in that it depends on a quantity measuring the geographic difference between the normal trip and the query trip. It also ignores time. In (68) Patterson et al. detect anomalous behavior based on GPS tracking. They train a dynamic, probabilistic model from a person's historical GPS traces. If the uncertainty of the trained model exceeds the uncertainty of a general prior model of human motion, then the system declares an anomaly. This is an example of an implicit similarity measure. Both (108) and (68) are aimed at detecting anomalies in the lives of the cognitively impaired. The system of Giroux, Bauchet, Pigot, Lussier-Desrochers and Lachappelle (109) has the same goal, only they use sensors in a home to detect anomalies in predefined daily routines, like making coffee. An anomaly is declared if the normal sequence of events is violated or if the timing of the sequence is sufficiently different from normal. Researchers have also detected anomalies in video, such as Xiang and Gong (110), whose system automatically builds models of normality from training video.

All of these approaches differ from ours in that they must be trained anew on a per-subject basis, whereas we find a single measure that doesn't require any training and generalizes well to multiple people. Moreover, we find a measure that approximates a human subject's assessment of their own data, without making any a priori assumptions.

8.5 Copyright Information

CHAPTER 9

CONCLUSION

In this thesis we have developed algorithms for application areas that stand to benefit from the inference of semantic information from user mobility data. Specifically, we have: (i) developed a robust quantitative evaluation method for map inference algorithms, (ii) investigated and evaluated two techniques for inferring road maps from sparsely sampled GPS traces, (iii) presented a hybrid map inference pipeline that combines the best aspects of existing algorithms with several new innovations in order to provide the most accurate method to date, (iv) developed an online GPS tracking system that reduces data uplink usage and provides improved guarantees and flexibility to system operators, (v) presented a system for automatic transit tracking and arrival time prediction for small transit agencies that requires minimal equipment and labor cost, and (vi) identified two algorithms that closely approximate human evaluations in assessing the similarity of people's days.

As the ubiquity of GPS sensors continues to expand into more and more everyday devices, it is our hope that the work presented here will provide both the inspiration, and technical basis for future developments in inferring semantic information from their generated data.

CITED LITERATURE

1. Biagioni, J. and Eriksson, J.: Inferring Road Maps from GPS Traces: Survey and Comparative Evaluation. In 91st Annual Meeting of the Transportation Research Board, 2012. 21 pages.

2. Biagioni, J. and Eriksson, J.: Inferring Road Maps from Global Positioning System Traces: Survey and Comparative Evaluation. Transportation Research Record: Journal of the Transportation Research Board, No. 2291, pages 61–71, 2012.

3. Liu, X., Biagioni, J., Eriksson, J., Wang, Y., Forman, G., and Zhu, Y.: Mining Large-Scale, Sparse GPS Traces for Map Inference: Comparison of Approaches. In Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 669–677. ACM, 2012.

4. Biagioni, J. and Eriksson, J.: Map Inference in the Face of Noise and Disparity. In Proceedings of the 20th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pages 79–88. ACM, 2012.

5. Biagioni, J., Musa, A., and Eriksson, J.: Thrifty Tracking: Online GPS Tracking with Low Data Uplink Usage. In Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pages 506–509. ACM, 2013.

6. Biagioni, J., Gerlich, T., Merrifield, T., and Eriksson, J.: EasyTracker: Automatic Transit Tracking, Mapping, and Arrival Time Prediction Using Smartphones. In Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems, pages 68–81. ACM, 2011.

7. Biagioni, J. and Krumm, J.: Days of Our Lives: Assessing Day Similarity from Location Traces. In User Modeling, Adaptation, and Personalization, volume 7899 of Lecture Notes in Computer Science, pages 89–101. Springer Berlin Heidelberg, 2013.

8. Biagioni, J. Personal website. http://phdthesis.biagioni.net/.

9. Fathi, A. and Krumm, J.: Detecting road intersections from GPS traces. In Geographic Information Science, pages 56–69. Springer, 2010.

10. Schroedl, S., Wagstaff, K., Rogers, S., Langley, P., and Wilson, C.: Mining GPS traces for map refinement. Data Mining and Knowledge Discovery, 9(1):59–87, 2004.

11. Chen, Y. and Krumm, J.: Probabilistic modeling of traffic lanes from GPS traces. In Proceedings of the 18th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pages 81–88. ACM, 2010.

12. Niehoefer, B., Burda, R., Wietfeld, C., Bauer, F., and Lueert, O.: GPS community map generation for enhanced routing methods based on trace-collection by mobile phones. In 1st International Conference on Advances in Satellite and Space Communications, pages 156–161. IEEE, 2009.

13. Mitchell, T.: Machine Learning. McGraw-Hill Education, 1st edition, 1997.

14. Scott, D. W.: Kernel Density Estimators, pages 125–193. John Wiley and Sons, Inc., 2008.

15. Edelkamp, S. and Schrödl, S.: Route planning and map inference with global positioning traces. In Computer Science in Perspective, pages 128–151. Springer, 2003.

16. Worrall, S. and Nebot, E.: Automated process for generating digitised maps through GPS data compression. In Australasian Conference on Robotics and Automation, 2007.

17. Guo, T., Iwamura, K., and Koga, M.: Towards high accuracy road maps generation from massive GPS traces data. In International Geoscience and Remote Sensing Symposium, pages 667–670. IEEE, 2007.

18. Jang, S., Kim, T., and Lee, S.: Map generation system with lightweight GPS trace data. In 12th International Conference on Advanced Communication Technology, volume 2, pages 1489–1493. IEEE, 2010.

19. Agamennoni, G., Nieto, J. I., and Nebot, E. M.: Robust inference of principal road paths for intelligent transportation systems. IEEE Transactions on Intelligent Transportation Systems, 12(1):298–308, 2011.

20. Cao, L. and Krumm, J.: From GPS traces to a routable road map. In Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pages 3–12. ACM, 2009.

21. Khanna, M.: Introduction to Particle Physics. Prentice-Hall of India, 2004.

22. Davies, J., Beresford, A. R., and Hopper, A.: Scalable, distributed, real-time map generation. IEEE Pervasive Computing, 5(4):47–54, 2006.

23. Chen, C. and Cheng, Y.: Roads digital map generation with multi-track GPS data. In International Workshop on Geoscience and Remote Sensing, volume 1, pages 508–511. IEEE, 2008.

24. Shi, W., Shen, S., and Liu, Y.: Automatic generation of road network map from massive GPS, vehicle trajectories. In 12th International Conference on Intelligent Transportation Systems, pages 1–6. IEEE, 2009.

25. Steiner, A. and Leonhardt, A.: A map generation algorithm using low frequency vehicle position data. 90th Annual Meeting of the Transportation Research Board, 2011.

26. Hastie, T., Tibshirani, R., and Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, 2009.

27. Aurenhammer, F.: Voronoi diagramsa survey of a fundamental geometric data structure. ACM Computing Surveys (CSUR), 23(3):345–405, 1991.

28. Conte, D., Foggia, P., Sansone, C., and Vento, M.: Thirty years of graph matching in pattern recognition. International Journal of Pattern Recognition and Artificial Intelligence, 18(03):265–298, 2004.

29. Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C., et al.: Introduction to Algorithms, volume 2. MIT Press Cambridge, 2001.

30. Bunke, H.: On a relation between graph edit distance and maximum common subgraph. Pattern Recognition Letters, 18(8):689–694, 1997.

31. Liu, B.: Web Data Mining. Springer, 2007.

32. OpenStreetMap. http://www.openstreetmap.org/.

33. Thiagarajan, A., Ravindranath, L., LaCurts, K., Madden, S., Balakrishnan, H., Toledo, S., and Eriksson, J.: Vtrack: accurate, energy-aware road traffic delay estimation using mobile phones. In Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, pages 85–98. ACM, 2009.

34. Liao, Z.: Real-time taxi dispatching using global positioning systems. Communications of the ACM, 46(5):81–83, 2003.

35. Liu, K., Yamamoto, T., and Morikawa, T.: Feasibility of using taxi dispatch system as probes for collecting traffic information. Journal of Intelligent Transportation Systems, 13(1):16–27, 2009.

36. Biem, A., Bouillet, E., Feng, H., Ranganathan, A., Riabov, A., Verscheure, O., Koutsopoulos, H., and Moran, C.: IBM infosphere streams for scalable, real-time, intelligent transportation services. In Proceedings of the 36th ACM SIGMOD International Conference on Management of Data, pages 1093–1104. ACM, 2010.

37. Balan, R. K., Nguyen, K. X., and Jiang, L.: Real-time trip information service for a large taxi fleet. In Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services, pages 99–112. ACM, 2011.

38. Wang, Y., Zhu, Y., He, Z., Yue, Y., and Li, Q.: Challenges and opportunities in exploiting large-scale GPS probe data. Technical Report HPL-2011-109, HP Labs, 2011.

39. Lou, Y., Zhang, C., Zheng, Y., Xie, X., Wang, W., and Huang, Y.: Map-matching for low-sampling-rate GPS trajectories. In Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pages 352–361. ACM, 2009.

40. SUVnet-trace data. http://wirelesslab.sjtu.edu.cn/.

41. Liu, X., Zhu, Y., Wang, Y., Forman, G., Ni, L. M., Fang, Y., and Li, M.: Road recognition using coarse-grained vehicular footprints. Technical Report HPL-2012-26, HP Labs, 2012.

42. Sibson, R.: SLINK: an optimally efficient algorithm for the single-link cluster method. The Computer Journal, 16(1):30–34, 1973.

43. Newson, P. and Krumm, J.: Hidden markov map matching through noise and sparseness. In Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pages 336–343. ACM, 2009.

44. Zhang, T. and Suen, C. Y.: A fast parallel algorithm for thinning digital patterns. Communications of the ACM, 27(3):236–239, 1984.

45. Douglas, D. H. and Peucker, T. K.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Cartographica: The International Journal for Geographic Information and Geovisualization, 10(2):112–122, 1973.

46. Kim, D. H., Kim, Y., Estrin, D., and Srivastava, M. B.: Sensloc: sensing everyday places and paths using less energy. In Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems, pages 43–56. ACM, 2010.

47. Paek, J., Kim, J., and Govindan, R.: Energy-efficient rate-adaptive GPS-based positioning for smartphones. In Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, pages 299–314. ACM, 2010.

48. Lin, K., Kansal, A., Lymberopoulos, D., and Zhao, F.: Energy-accuracy aware localization for mobile devices. In Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services. ACM, 2010.

49. Zhuang, Z., Kim, K.-H., and Singh, J. P.: Improving energy efficiency of location sensing on smartphones. In Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, pages 315–330. ACM, 2010.

50. Kjærgaard, M. B., Langdal, J., Godsk, T., and Toftkjær, T.: Entracked: energy-efficient robust position tracking for mobile devices. In Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services, pages 221–234. ACM, 2009.

51. Wolfson, O., Jiang, L., Sistla, A. P., Deng, M., Chamberlain, S., and Rishe, N.: Databases for tracking mobile units in real time. In Database Theory–ICDT 1999, pages 169–186. Springer, 1999.

52. Wolfson, O., Sistla, A. P., Chamberlain, S., and Yesha, Y.: Updating and querying databases that track mobile units. In Mobile Data Management and Applications, pages 3–33. Springer, 1999.

53. Civilis, A., Jensen, C. S., and Pakalnis, S.: Techniques for efficient road-network-based tracking of moving objects. IEEE Transactions on Knowledge and Data Engineering, 17(5):698–712, 2005.

54. Civilis, A., Jensen, C. S., Nenortaite, J., and Pakalnis, S.: Efficient tracking of moving objects with precision guarantees. In 1st International Conference on Mobile and Ubiquitous Systems: Networking and Services, pages 164–173. IEEE, 2004.

55. Krumm, J.: A markov model for driver turn prediction. SAE SP, 2193(1), 2008.

56. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: Machine learning in python. The Journal of Machine Learning Research, 12:2825–2830, 2011.

57. Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A.: Classification and Regression Trees. CRC press, 1984.

58. Meratnia, N. and Rolf, A.: Spatiotemporal compression techniques for moving point objects. In Advances in Database Technology–EDBT 2004, pages 765–782. Springer, 2004.

59. NextBus, Inc. http://www.nextbus.com/.

60. Clever Devices, Inc. http://www.cleverdevices.com/.

61. Google Maps. http://maps.google.com/.

62. CTA Real-Time Feed. http://www.ctabustracker.com/.

63. General Transit Feed Specification (GTFS). http://code.google.com/transit/spec/transit_feed_specification.html.

64. Walpole, R., Myers, R., and Ye, K.: Probability and Statistics for Engineers and Scientists. Pearson Education, 2002.

65. Hartigan, J. A. and Wong, M. A.: Algorithm AS 136: A k-means clustering algorithm. Applied Statistics, pages 100–108, 1979.

66. Steinley, D. and Brusco, M. J.: Initializing k-means batch clustering: A critical evaluation of several techniques. Journal of Classification, 24(1):99–121, 2007.

67. Mirkin, B.: Clustering for Data Mining: A Data Recovery Approach. Chapman & Hall/CRC, 2005.

68. Patterson, D. J., Liao, L., Gajos, K., Collier, M., Livic, N., Olson, K., Wang, S., Fox, D., and Kautz, H.: Opportunity knocks: A system to provide cognitive assistance with transportation services. In Ubiquitous Computing, pages 433–450. Springer, 2004.

69. United States Bureau of Labor Statistics. American Time Use Survey (ATUS). http://www.bls.gov/tus/.

70. Yi, B.-K., Jagadish, H., and Faloutsos, C.: Efficient retrieval of similar time sequences under time warping. In 14th International Conference on Data Engineering, pages 201–208. IEEE, 1998.

71. Levenshtein, V. I.: Binary codes capable of correcting deletions, insertions and reversals. In Soviet Physics Doklady, volume 10, page 707, 1966.

72. Sinnott, R. W.: Virtues of the haversine. Sky and Telescope, 68:158, 1984.

73. Kullback, S.: Information Theory and Statistics. Courier Dover Publications, 1968.

74. Agrawal, R., Faloutsos, C., and Swami, A.: Efficient Similarity Search in Sequence Databases. Springer, 1993.

75. Holm, S.: A simple sequentially rejective multiple test procedure. Scandinavian Journal of Statistics, pages 65–70, 1979.

76. Von Luxburg, U.: A tutorial on spectral clustering. Statistics and Computing, 17(4):395–416, 2007.

77. Naver. http://dev.naver.com/openapi/apis/map/.

78. Piegl, L. and Tiller, W.: The NURBS book. New York, NY, USA, Springer-Verlag New York, Inc., 2nd edition, 1997.

79. NAVTEQ Maps and Traffic. http://www.navteq.com/.

80. Wong, S. S. M.: Computational Methods in Physics and Engineering. World Scientific, 1997.

81. Hastie, T. and Stuetzle, W.: Principal curves. Technical report, University of Washington, 1988.

82. Agamennoni, G., Nieto, J., and Nebot, E.: Technical report: Inference of principal road paths using GPS data. Technical report, The University of Sydney, Australian Centre For Field Robotics, June 2010.

83. Leler, W. J.: Human vision, anti-aliasing, and the cheap 4000 line display. ACM SIGGRAPH Computer Graphics, 14(3):308–313, 1980.

84. Yokoi, S., Toriwaki, J.-I., and Fukumura, T.: An analysis of topological properties of digitized binary pictures using local features. Computer Graphics and Image Processing, 4(1):63–73, 1975.

85. Li, Q., Bai, X., and Liu, W.: Skeletonization of gray-scale image from incomplete boundaries. In 15th International Conference on Image Processing, pages 877–880. IEEE, 2008.

86. Yim, P. J., Choyke, P. L., and Summers, R. M.: Gray-scale skeletonization of small vessels in magnetic resonance angiography. IEEE Transactions on Medical Imaging, 19(6):568–576, 2000.

87. Meyer, F.: Topographic distance and watershed lines. Signal Processing, 38(1):113–125, 1994.

88. Ashbrook, D. and Starner, T.: Using gps to learn significant locations and predict movement across multiple users. Personal and Ubiquitous Computing, 7(5):275–286, 2003.

89. Jurdak, R., Corke, P., Dharman, D., and Salagnac, G.: Adaptive gps duty cycling and radio ranging for energy-efficient localization. In Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems, pages 57–70. ACM, 2010.

90. Hershberger, J. and Snoeyink, J.: An o (n log n) implementation of the douglas-peucker algorithm for line simplification. In Proceedings of the 10th Annual Symposium on Computational Geometry, pages 383–384. ACM, 1994.

91. Kellaris, G., Pelekis, N., and Theodoridis, Y.: Trajectory compression under network constraints. In Advances in Spatial and Temporal Databases, pages 392–398. Springer, 2009.

92. Vu, T. H. N., Ryu, K. H., and Park, N.: A method for predicting future location of mobile user for location-based services system. Computers & Industrial Engineering, 57(1):91–105, 2009.

93. Ohsawa, Y., Fujino, K., Htoo, H., Hlaing, A. T., and Sonehara, N.: Real-time monitoring of moving objects using frequently used routes. In Database Systems for Advanced Applications, pages 119–133. Springer, 2011.

94. Muckell, J., Hwang, J.-H., Patil, V., Lawson, C. T., Ping, F., and Ravi, S.: SQUISH: an online approach for GPS trajectory compression. In Proceedings of the 2nd International Conference on Computing for Geospatial Research & Applications, pages 13:1–13:8. ACM, 2011.

95. Sistla, A. P., Dao, S., Wolfson, O., and Chamberlain, S.: Modeling and querying moving objects. In 29th International Conference on Data Engineering, page 422. IEEE, 1997.

96. Wolfson, O., Chamberlain, S., Dao, S., Jiang, L., and Mendez, G.: Cost and imprecision in modeling the position of moving objects. In 14th International Conference on Data Engineering, pages 588–596. IEEE, 1998.

97. Jensen, C. S. and Pakalnis, S.: Trax: real-world tracking of moving objects. In Proceedings of the 33rd International Conference on Very Large Data Bases, pages 1362–1365. VLDB Endowment, 2007.

98. Lange, R., Dürr, F., and Rothermel, K.: Online trajectory data reduction using connection-preserving dead reckoning. In Proceedings of the 5th Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services, pages 52–61. Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, 2008.

99. Lange, R., Farrell, T., Durr, F., and Rothermel, K.: Remote real-time trajectory simplification. In 7th International Conference on Pervasive Computing and Communications, pages 1–10. IEEE, 2009.

100. Lange, R., Dürr, F., and Rothermel, K.: Efficient real-time trajectory tracking. The VLDB Journal, 20(5):671–694, 2011.

101. Thiagarajan, A., Biagioni, J., Gerlich, T., and Eriksson, J.: Cooperative Transit Tracking using Smart-phones. In Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems, pages 85–98. ACM, 2010.

102. Biagioni, J., Agresta, A., Gerlich, T., and Eriksson, J.: Demo Abstract: TransitGenie - A Context-Aware, Real-Time Transit Navigator. In Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, pages 329–330. ACM, 2009.

103. TransitGenie Chicago. http://www.transitgenie.com/.

104. Bin, Y., Zhongzhen, Y., and Baozhen, Y.: Bus arrival time prediction using support vector machines. Journal of Intelligent Transportation Systems, 10(4):151–158, 2006.

105. Tiesyte, D. and Jensen, C. S.: Similarity-based prediction of travel times for vehicles traveling on known routes. In Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pages 14:1–14:10. ACM, 2008.

106. Chien, S. I.-J., Ding, Y., and Wei, C.: Dynamic bus arrival time prediction with artificial neural networks. Journal of Transportation Engineering, 128(5):429–438, 2002.

107. Deng, K., Xie, K., Zheng, K., and Zhou, X.: Trajectory indexing and retrieval. In Computing with Spatial Trajectories, pages 35–60. Springer, 2011.

108. Ma, T.-S.: Real-time anomaly detection for traveling individuals. In Proceedings of the 11th International ACM SIGACCESS Conference on Computers and Accessibility, pages 273–274. ACM, 2009.

109. Giroux, S., Bauchet, J., Pigot, H., Lussier-Desrochers, D., and Lachappelle, Y.: Pervasive behavior tracking for cognitive assistance. In Proceedings of the 1st International Conference on PErvasive Technologies Related to Assistive Environments, pages 86:1–86:7. ACM, 2008.

110. Xiang, T. and Gong, S.: Video behavior profiling for anomaly detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, 30(5):893–908, 2008.

VITA

Name James P. Biagioni

Education Ph.D., Computer Science, University of Illinois at Chicago, 2014.
B.S., Computer Science, University of Illinois at Chicago, 2006.

Honors Fast Forward Preview Session Runner-Up, ACM SIGSPATIAL, 2013.
Dean's Scholar Award, University of Illinois at Chicago, 2012–2013.
Paper Session Best Presentation, ACM SenSys, 2011.
Demo Session Honorable Mention, ACM MobiCom, 2010.
NSF-IGERT Fellowship, University of Illinois at Chicago, 2007–2011.

Publications

Biagioni, J., Musa, A., and Eriksson, J.: Thrifty Tracking: Online GPS Tracking with Low Data Uplink Usage. In Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pages 506–509. ACM, 2013.

Biagioni, J. and Krumm, J.: Days of Our Lives: Assessing Day Similarity from Location Traces. In User Modeling, Adaptation, and Personalization, volume 7899 of Lecture Notes in Computer Science, pages 89–101. Springer Berlin Heidelberg, 2013.

Biagioni, J. and Eriksson, J.: Map Inference in the Face of Noise and Disparity. In Proceedings of the 20th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pages 79–88. ACM, 2012.

Liu, X., Biagioni, J., Eriksson, J., Wang, Y., Forman, G., and Zhu, Y.: Mining Large-Scale, Sparse GPS Traces for Map Inference: Comparison of Approaches. In Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 669–677. ACM, 2012.

Biagioni, J. and Eriksson, J.: Inferring Road Maps from Global Positioning System Traces: Survey and Comparative Evaluation. Transportation Research Record: Journal of the Transportation Research Board, No. 2291, pages 61–71, 2012.

Publications (Continued)

Biagioni, J. and Eriksson, J.: Inferring Road Maps from GPS Traces: Survey and Comparative Evaluation. In 91st Annual Meeting of the Transportation Research Board, 2012. 21 pages.

Gerlich, T., Biagioni, J., Merrifield, T., and Eriksson, J.: Demo: Tracking Transit with EasyTracker. In Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems, pages 401–402. ACM, 2011.

Biagioni, J., Gerlich, T., Merrifield, T., and Eriksson, J.: EasyTracker: Automatic Transit Tracking, Mapping, and Arrival Time Prediction Using Smartphones. In Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems, pages 68–81. ACM, 2011.

Thiagarajan, A., Biagioni, J., Gerlich, T., and Eriksson, J.: Cooperative Transit Tracking using Smart-phones. In Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems, pages 85–98. ACM, 2010.

Biagioni, J., Agresta, A., Gerlich, T., and Eriksson, J.: Demo Abstract: TransitGenie - A Context-Aware, Real-Time Transit Navigator. In Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, pages 329–330. ACM, 2009.

Biagioni, J., Szczurek, P., Nelson, P., and Mohammadian, A.: Tour-Based Mode Choice Modeling: Using an Ensemble of Conditional and Unconditional Data Mining Classifiers. In 88th Annual Meeting of the Transportation Research Board, 2009. 16 pages.

Invited Talks

"Inferring Road Maps from GPS Traces (revised)," Google, 2013.
"Inferring Road Maps from GPS Traces," Microsoft, 2012.

Professional Service

Reviewer for GeoInformatica, 2014.
Reviewer for ACM SIGMOD Record, 2013.
External Reviewer for Int'l Conference on Pervasive Computing, 2012.
Student Volunteer for ACM SIGSPATIAL GIS, 2012–2013.