Large-Scale, Low-Latency State Estimation Of Cyberphysical Systems With An Application To Traffic Estimation

by

Timothy Jason Hunter

A thesis submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Alexandre Bayen, Chair
Professor Pieter Abbeel
Professor Michael Franklin
Professor Alexei Pozdnukhov

Fall 2014

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

Large-Scale, Low-Latency State Estimation Of Cyberphysical Systems With An Application To Traffic Estimation

Abstract

Large-Scale, Low-Latency State Estimation Of Cyberphysical Systems With An Application To Traffic Estimation

by

Timothy Jason Hunter

Doctor of Philosophy in Computer Sciences

University of California, Berkeley

Professor Alexandre Bayen, Chair

Large physical systems are increasingly prevalent, and designing estimation strategies for them has become both a practical necessity and a complicated problem. Their sensing infrastructure is usually ad-hoc, and the estimate of interest is often a complex function of the data. At the same time, computing power is rapidly becoming a commodity. We show with the study of two estimation tasks in urban transportation how the proper design of algorithms can lead to significant gains in scalability compared to existing solutions.

A common problem in trip planning is to make a given deadline such as arriving at the airport within an hour. Existing routing services optimize for the expected time of arrival, but do not provide the most *reliable* route, which accounts for the variability in travel times. Providing statistical information is even harder for trips in cities which undergo a lot of variability. This thesis aims at building scalable algorithms for inferring statistical distributions of travel time over very large road networks, using GPS points from vehicles in real-time. We consider two complementary algorithms that differ in the characteristics of the GPS data input, and in the complexity of the model: a simpler streaming Expectation-Maximization algorithm that leverages very large volumes of extremely noisy data, and a novel Markov Model-Gaussian Markov Random Field that extracts global statistical correlations from high-frequency, privacy-preserving trajectories.

These two algorithms have been implemented and deployed in a pipeline that takes streams of GPS data as input, and produces distributions of travel times accessible as output. This pipeline is shown to scale on a large cluster of machines and can process tens of millions of GPS observations from an area that comprises hundreds of thousands of road segments. This is to our knowledge the first research framework that considers in an integrated fashion the problem of statistical estimation of traffic at a very large scale from streams of GPS data.

To my father and my mother, who gave me the most one can ask from his parents.
To my brother Lucas and my sisters Marine and Sophie, for their constant support.

To the millions of commuters who spend years of their lives behind the wheels.

Contents

List of Figures

List of Algorithms

Acknowledgments

The clock was ticking with a deadly punctuality at each second, answered only by the howling shriek of the coffee machine brewing more of its potent liquid. The fingers were dancing on the keyboards like grains of rice on a drum. Pairs of eyes were staring at abstract charts and symbols written in a cabalistic language. Sometimes, a sentence would drop, more precise than a scalpel in the surgical room: "Experiment one is running. I changed the value of sigma to 0.2 and reduced the number of machines". The confirmation would come a second later: "Wait. I am compiling". Wait. The laptops were scrambling to decipher the cryptic instructions that their masters could barely apprehend. One hour still before the conference deadline. One hour before a clock announces that it is midnight in the Samoa Islands. One hour before a digital Cerberus decides that the academic Styx is closed for business until next year. The sweat starts to glue the fingers to the keyboard. Suddenly, the light shuts off, and the pale glow of the emergency light weakly warms the room. No more electricity. No more internet. And seventeen minutes before the deadline with a half-fleshed article that looks like Frankenstein's monster.

How did things get to this point?

Doing a Ph.D. is like skydiving for the first time: your friends who have done it before assure it is a thrilling and safe experience, and the safety record of the club is spotless, so you sign up, convinced by the arguments. Once aboard the plane, you start to think it was not such a brilliant idea after all. This thought climaxes when you get pushed out of the aircraft. The mid-air flight passes like a short stint inside a hypersonic tumble dryer. Once you have landed, you start to reconstitute the process that lead to this decision. Who was behind all that?

My first thanks go to my family, in particular my parents Soleine and William Hunter, for their understanding throughout all these years, and for teaching me that technological progress should first and foremost be to the benefit of men and women.

My two co-advisers have been a constant support in this endeavor. I would like first to thank my co-adviser, Alexandre Bayen, for his constant encouragements. Ever since we first met between an elevator door and a phone call during visit day, his enthusiasm for research and his vision of a enlightened society has been gospel to me. Alex was kind enough to let me explore traffic datasets under the prism of statistical methods that bore few resemblance with his domain of expertise. Under his guidance, we launched a collaboration with the AMPLab that profoundly shaped my research interests and still bears fruits to this day. In a more relaxed setting, Alex is a masterful entertainer and a most delightful accomplice around a table at Jupiter's. Alex's support was invaluable in more somber times as well. Although I have not made use of his offer to call at any time of the day or night, I am still keeping this option in mind. Thank you for the comic books, Alex.

I still remember the first email of my co-adviser Pieter Abbeel, back in Stanford more than seven years ago, asking if I would like to collaborate on doing some aerobatics with a

toy helicopter. Little did I know that this meeting would have so momentous consequences. Pieter ignited a passion for research that continues. Pieter conveys contagious enthusiasm for exploring novel ground and for making cool demonstrations that express the full power of the research behind it. After Pieter graduated to an Assistant Professorship at Berkeley, he convinced me it would be a great place for doing a Ph.D. Back then, I was unsure if I was more suited for an academic career, but Pieter's energy and trust were major factors in coming here. When the time came to devise new models and methods, Pieter's sharp eye and keen attention to details prevented us from time-consuming pitfalls in numerous occasions. Our repeated conversations were instrumental in what became Chapter 2 and Chapter 4. We still need to have this beer, Pieter.

Michael Franklin has been a great source of encouragement in harnessing the power of the cloud. Back then, it was clear that we needed computers. A lot of computers. Thanks to the impulse of Mike, the frightening prospect of rewriting everything for distributed computing became a joyful endeavor in Scala and Spark. I was at first a bit dubious that the systems community would find a (car) traffic system of interest, but our discussions lead to more and more material that eventually became the SOCC article. Thanks to Mike, the *Mobile Millennium* project was a first class citizen within the AMPLab and lead to a number of great collaborations.

When Alex told me I should talk with a new faculty in Civil Engineering, Alexei Pozdnukov struck me by his understanding of very large systems and his command of the challenges involved in using physical data at a very large scale. Through our conversations, Alexei has been always kept the big picture in mind.

Many thanks to my first research adviser, Andrew Ng. Andrew has always pushed me to achieve the best, and would believe in accomplishments that even I would not think possible. Thank you Andrew for letting me share my passion for Artificial intelligence as a CS221 Teaching Assistant, which was my first teaching experience. Andrew was keen on giving challenging and rewarding assignments, and making me discover the joys of research.

Eric Feron was my first contact with research and I still fondly remember my stay at his lab in Georgia Tech. Working with the helicopter was a fun and challenging problem, and sometimes even a close shave.

During all this effort, the technical support of various groups, and especially the California Center for Intelligent Transportation and PATH, have been essential in dealing with all the subtleties of traffic data. In particular, not much would have been accomplished without the dedicated support of Ryan Herring and Saneesh Apte. Ryan was patient enough to explain the foolhardy first year the basics of collaborative software engineering. His work on the *Mobile Millennium* infrastructure was a foundational step for all the research presented in the subsequent pages, and his authorship is reflected in the copyright notices of the code released along this thesis. Saneesh also had a determining influence throughout our (sometimes opiniated) conversations about data management and code styles, and his help was crucial in collecting the data. Thanks to Joe Butler, who pushed me to implement the map-matching code in a proper manner. What I believed back then to be a pointless exercise of coding style became one of my proudest engineering accomplishments, and in retrospect

it enabled much more research than would have been possible otherwise.

I found a great collaborator in Aude Hofleitner, who inspired me to combine machine learning with traffic research. Aude's sophisticated models were stepping stones for the research in Chapter 4, and her passion combined with a no-nonsense approach have always been a driving force in the lab and for me in particular. Without our invigorating and fruitful exchanges, this thesis would permeate with preposterous nonsense about traffic. Aude approaches everything in life with an amazing energy, and I am glad I could meet her both on and around the campus.

Life would have been quite different without the friendship of so many great people, from which Jack Reilly and Samitha Samaranayake stand in particular. Early work-related discussions would drift into late night musings and further later into ski trips to nearby Lake Tahoe. We always ended up meeting again in the most peculiar places (Tunisia, Switzerland), and I hope this pattern will continue in the future.

The crowd of the basement (Bakers), first floor (Gates), fourth floor (Soda) and seventh floor (Sutardja Dai Hall), you made my day so many times. I wish to thank in particular (in alphabetical order): Leah Anderson, Adam Coates, Orianna DeMasi, Maria Eckstein, Ahmed El-Alaoui, Shiry Ginosar, Woody Hoburg, Judy Hoffman, Thibaud Hottelier, Sergey Karayev, Walid Krichène, Stéphanie Lefèvre, Antonio Lupher, Alan Malek, Teodor Moldovan, Thomas Pumir, David Shinazi, Virginia Smith, Ameet Talwalkar, Jérôme Thai, Cathy Wu, and Matei Zaharia.

The floppy discs (with a k): you were great. It was a pleasure to fly the Frisbee with you every Tuesday.

The whole gang, thank you for the lubrication. You know who you are, and you know drinking is unhealthy, but by Bacchus, why stop such a merry predicament?

Countless thanks to A.-M. B. for weathering my soliloquies throughout these years. G., you have changed my life. I hope our paths will cross again.

Last but not least, my thoughts go to all my friends, who shared their lives across continents, and whose unwavering support brought joy across all these years, in particular Elói Pereira, François-Fabien Fehrani and Patti Lee.

The author wishes to thank the not-so-anonymous reviewers for their critical suggestions on the draft on the thesis: Jack Reilly and Samitha Samaranayake for the introduction and the acknowledgments, Caroline Falck for her eagle eye on the overall manuscript.

Chapter 1

Introduction

> From the midst of this darkness a sudden light broke in upon me–a light so brilliant and wondrous, yet so simple, that while I became dizzy with the immensity of the prospect which it illustrated, I was surprised that among so many men of genius who had directed their inquiries towards the same science, that I alone should be reserved to discover so astonishing a secret.
>
> Mary Shelley, *Frankenstein*

Our society is increasingly dependent on very large infrastructure systems that provide vital services such as water, electricity, information, or energy. Prominent examples are telecommunication systems, power grids, water distribution or transportation systems. These systems are continuously monitored by a vast array of sensors, which gather large streams of digital information. The integration of the physical processes of this system with the computational processes is called a *cyberphysical system*. This network of sensors gives information about the aggregate state of the system, or about local variables such as chemical concentrations, power output, temperatures, etc. This step is commonly referred to as state estimation or filtering. These variables are then used to provide corrective feedback to the system (control). In order to process this stream of information, it will be more and more common in the future to use massive amounts of computing power for the following reasons:

- The sensing network is increasingly complex: it usually offers indirect measurements of the state (the sensors are either not located in the area of interest or measure other variables correlated with the variables of interest).
- Due to coupling effects, estimating the state of a large system is usually much more difficult than estimating the states of the subsystems.

Furthermore, as monitoring networks grow in complexity and versatility, they are usually tasked with answering more complex questions such as predicting the evolution of the system or performing diagnostics on critical sections (and also providing some statistical guarantees about the quality of these diagnostics). These tasks often require much more computing power than aggregating measurements, and yet become an increasingly important part of the monitoring process. Examples include assessing the spread of toxic chemicals in a water system or detecting radioactive surges in a nuclear core. These tasks grow in complexity and will require more and more the use of advanced statistical and computing techniques in order to be completed in a timely manner. Some disciplines such as weather forecasting have already harnessed the potential offered by cost-effective computing platforms. We discuss in this thesis the well-known challenge of vehicular traffic estimation, from the perspective of a large cyberphysical system.

1.1 Car traffic as a cyberphysical system

Traffic. Traffic congestion affects nearly everyone in the world due to the environmental damage and transportation delays it causes. The *2012 Urban Mobility Report* [98] states that traffic congestion causes 5.5 billion hours of extra travel in the United States in 2011, which accounted for 2.9 billion extra gallons of fuel and an additional cost of $121 billion. Amongst the modern man-made plagues, traffic congestion is a universally recognized challenge [36]. Building reliable and cost-effective traffic monitoring systems is a prerequisite to addressing this phenomenon.

One way to alleviate the effects of traffic congestion is to provide accurate and timely information to drivers. Providing drivers with accurate traffic information reduces the stress associated with congestion and allows drivers to make informed decisions, which generally increases the efficiency of the entire road network [18]. Researchers on Traffic Information Systems (TIS) broadly agree that accurate information is critical to increase their usage [30]. One of the main challenges of TIS is to provide reliable, accurate and timely travel information to drivers, and especially travel time information.

The question of estimating traffic and providing accurate travel information is well understood in the case of highways. Modeling highway traffic conditions has been well-studied by the transportation community, with work dating back to the pioneering work of Lighthill, Whitham and Richards [74, 120]. Historically, the estimation of traffic congestion has been limited to highways, and has relied mostly on a static, dedicated sensing infrastructure such as loop detectors or cameras [121]. Recently, researchers demonstrated that estimating highway traffic conditions can be done using only GPS probe vehicle data [122]. Traffic estimation is still an open question in the case of roads within a city, called *arterial roads*. Arterial roads are the major urban city streets that connect population centers within and between cities. The estimation problem is more challenging in the case of arterial roads, due to the higher complexity of city traffic, and to the cost of deploying a wide network of sensors in large metropolitan areas. Dedicated traffic sensors are expensive to install, maintain and

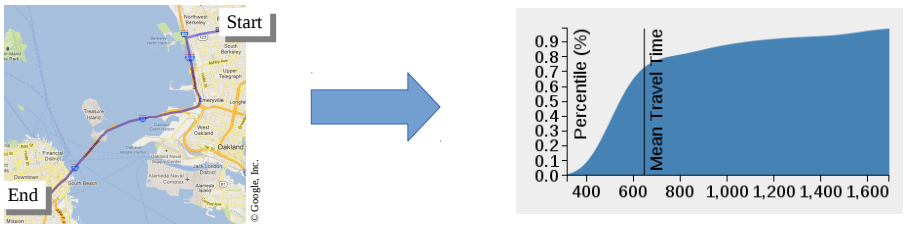


Figure 1.1.1: The travel time function that we desire to compute: given a start time and a path across a road network, we wish to provide a statistical distribution of arrival times (represented here as a cumulative distribution).

operate, which limits the number of sensors that governmental agencies can deploy on the road network. The lack of sensor coverage across the arterial network thus motivates the use of GPS probe vehicle data for estimating traffic conditions.

Goal. The specific problem we address in this study is how to extract travel time distributions from *sparse, noisy* GPS measurements collected *in real-time* from vehicles, and over a *very large network*. We wish to provide a function that, given any route in the road network, returns statistical information of travel times over this route (see Figure 1.1). Commercial providers [2, 4, 108] have so far focused on the easier task of assigning a single number for the travel time on a road, which is usually understood as an average travel time[1]. This number is usually not enough information for the end user. For example, consider a trip to a nearby airport. A passenger would like to know when to leave her[2] current location in order to be at the airport with some reasonable confidence, and the mean does not convey this information, or even worse, may be misleading the user into taking a route that has a smaller mean but a larger variance (i.e., the trips are usually shorter but they are longer, the delays are significant). Seasoned taxi drivers take their past experience into account to develop their routing strategies, but this knowledge is hard to quantify and transfer to casual drivers. Furthermore, computing the full statistical distribution of travel times lets the user decide how much risk she is willing to take[3].

Source of data. *Probe data* is information collected from moving vehicles, and typically includes the location and speed of the vehicle. This type of data have been revolutionized by the introduction of the Global Positioning System (GPS) in the 1980s. The most promising source of data is the GPS receiver in personal smartphones and commercial fleet vehicles [53]. The GPS can measure the location of a receiver across the the globe with relatively high accuracy. The last decade has brought the GPS to the masses by making it a standard

[1]For various legal reasons, these products purposefully omit any labeling that could be interpreted as a statistical performance guarantee on the information provided to drivers.

[2]We follow the economics convention to address the casual driver with a female gender. See [100] for a more complete discussion.

[3]Of course, leaving an infinity of time before the event is the only strategy to arrive with probability one. We assume drivers would object to such a long trip to the airport.

feature in smartphones. According to some studies [99], devices with a data connection and a GPS will represent 80% of the cellphone market by 2015. GPS data collected for traffic estimation purposes comes from two different sources:

- Fleet data. A number of businesses (taxi companies, FedEx, UPS, trucks) operate large fleets of vehicles and optimize their services by tracking the locations of the vehicles. This source of GPS information is usually of high volume (tens of thousands of vehicles across the continent), low frequency (every minute or so), and with limited privacy concerns. Also, their coverage of the road network can be limited to certain areas.

- Smartphone data. Some global companies collect vehicular GPS data by using software installed in the device, like mapping software (Garmin. Google, INRIX, Nokia, Waze, etc.) This data is believed to be of high frequency and to be more or less scrubbed from personal information during the collection process[4]. However, this data is seldom accessible to researchers due to privacy, market fragmentation, and competitivity concerns.

Probe data presents some compelling advantages over static sensors: it is much easier to upgrade (for example by pushing software upgrades to devices), it is cheaper to operate (the energy and transmission costs are shouldered by the end users) and its coverage is usually very extensive (see Figure 1.2.1 for an example). It also presents some drawbacks: achieving reasonable levels of privacy is an open question, the coverage is dependent on the user base and may be poor in some areas, some variables like vehicle counts are not directly observable, and the market fragmentation means that agreements must be negotiated with each company that collects GPS data (often gathered from different users).

Pipeline. All the algorithms and techniques we are about to present have been implemented and deployed within the *Mobile Millennium* traffic information system [20]. *Mobile Millennium* infers real-time traffic conditions using GPS measurements from drivers running cell phone applications, taxicabs, and other mobile and static data sources. This system was initially deployed in the San Francisco Bay area and later expanded to other locations such as Sacramento, Stockholm, and Porto. *Mobile Millennium* is intended to work at the scale of large metropolitan areas: the road network considered in this work is a real road network (a large portion of the greater Bay Area, comprising 506,685 road links) and the data for this work is collected from thousands of vehicles that generate millions of observations per day. Furthermore, *Mobile Millennium* is a research platform and can be used with various models of travel time. We have built upon *Mobile Millennium* to provide a statistical estimation engine for travel time, and expanding it was a significant contribution of this work. In this data-intensive task, the final output is a multi-stage pipeline with each of the stages being responsible for a separate task. This traffic pipeline is decomposed in the following modules:

[4]For example, as of February 2014, the privacy policy of Google maps states: "We you use [Google Maps], we may collect and process information about your actual location, like GPS signal sent by a mobile device. We may also use various technologies to determine location, such as [WiFi and cell tower information]".

- An ingestion module that takes all the feeds and stores them in a database,
- A path inference module that reconstructs trajectories from GPS points,
- Various travel time modules that compute travel time estimates on the road network,
- A validation module that implements different metrics of travel times (mean square error or log-likelihood for example).

Most of this code is available under an open source license, and the output has been deployed as a web service[5].

Scaling/Spark. While a lot of research has focused on a few road intersections, we have decided to work from the outset at the scale of large city. This adds multiple scalability challenges: (1) the data to process is substantially larger, and (2) estimation algorithms need to scale to millions of variables, which is a hard challenge in Machine Learning in general. As a consequence, we investigate the use of novel cloud computing frameworks that let scientific users deploy code across hundreds of computers in a transparent way. We have implemented some of our algorithms on top of the Spark computing framework [124]. After some modifications to Spark, we were able to achieve massive throughput with low-latency (near real-time), an elusive yet necessary goal for complex cyberphysical systems.

Existing systems. Up to now, researchers have mostly tackled traffic estimation by focusing on the different subtasks of data collection, data filtering, road link congestion detection, etc. Research is usually done by using synthetic input or considering a small, controlled environment. Very few researchers have attempted to consider the complete task of traffic estimation in a holistic manner. There are formidable engineering challenges in building a reliable system for mass collection of GPS data, and enlisting enough users to reach a critical mass. This is why fully integrated TIS have been mostly pioneered in the transportation industry, which has developed systems that scale to the whole world and that ingest thousands of observations per second. However, there has been no published attempt to quantify the quality of the solutions offered by the industry. The terms of use generally limit scientific investigation of the output provided to the public. Furthermore, companies that collect such data are loathe to share it with researchers due to obvious legal and privacy concerns. The only successful examples of full platforms developed in academia that we are aware of are Cartel [59] and *Mobile Millennium* [64]. Researchers have made significant strides in tackling each subtask individually. However, this focused approach does not tackle integration and scalability issues that arise when one considers the whole pipeline. We hope that this work will provide some insight as to the problems that are relevant to researchers in the field.

Traffic estimation and control goes beyond the roads and the cars that circulate thereupon. It involves complex, distributed computations that monitor the flow of cars (the physical part of the process). Crucial to these computations is a sensing infrastructure that infers the state of the vehicular network. This tight coupling between physical and logical

[5] `http://traffic.berkeley.edu/navigatesf`

processes characterizes the overall system as a cyberphysical system. It is common in computer science to assume that correctness is decoupled from performance: a slow but correct program is annoying but eventually completes its task. This is not the case in a cyberphysical system. The physics of the process change the state of the system while computations are performed, because these changes happen concurrently, not sequentially, to the computational process. This is why performance and timeliness of the computations are such a critical aspect.

1.2 Complexity of modeling traffic from GPS observations

What sort of GPS data can be processed by the *Mobile Millennium* system? A datum of GPS information contains a time stamp, a unique identifier for the driver or the trip, a location encoded by latitude and longitude, and often some other elements such as heading, speed, status (empty, en route, etc.). These additional attributes were usually found to be too noisy to be used in a systematic fashion, and were not used. Figure 1.2.1 graphically presents a subset of this probe data collected by *Mobile Millennium*.

Our final objective is to give timely and accurate information about travel times to the user. We aim to develop a model of travel times that can give non-trivial, informative statistical bounds on the travel times. This goal informs the complexity of the model that we wish to build. In particular, we aim to develop models that capture significant correlations in travel times between different road sections.

Temporal disparity. Existing GPS data is very heterogeneous, both in a spatial sense and in a temporal sense (see Figure 1.2.2 and Figure 1.2.3). Some small portions of the road network (the highways and the main arterials) concentrate most of the data. It is not always clear if this also means that these portions also concentrate most of the traffic. This GPS data is collected from various sources that were not designed for traffic analysis; hence they may record the location of a vehicle at intervals up to three minutes. These very low frequency collection schemes make the bulk of the data collected these days, and using them for accurate traffic estimation remains a challenge. Furthermore, in the case of individual drivers, in order to limit privacy intrusion and to save battery life, it is common to only record the location of the vehicle during short intervals (a few minutes to an hour). There is no control over when the GPS unit will record the trip. As a conclusion, we have access to large amount of a low-frequency data, or to a small amount of high-frequency data. This is why we will present two models for traffic that correspond to these two typical scenarios.

- A first scenario in which we only have high-frequency data (Scenario "A").
- A second scenario in which we have large amounts of sparse GPS data (Scenario "B").

In both cases, we want to work at the scale of the city, and we will highlight the contributions towards scaling the problem to very large metropolis.

Figure 1.2.1: An example of dataset available to *Mobile Millennium* [20] and processed by the path inference filter [63]: taxicabs in San Francisco from the Cabspotting program [10]. Large circles in red show the position of the taxis at a given time and small dots (in black) show past positions (during the last ten hours) of the taxi fleet. The position of each vehicle is observed every minute.

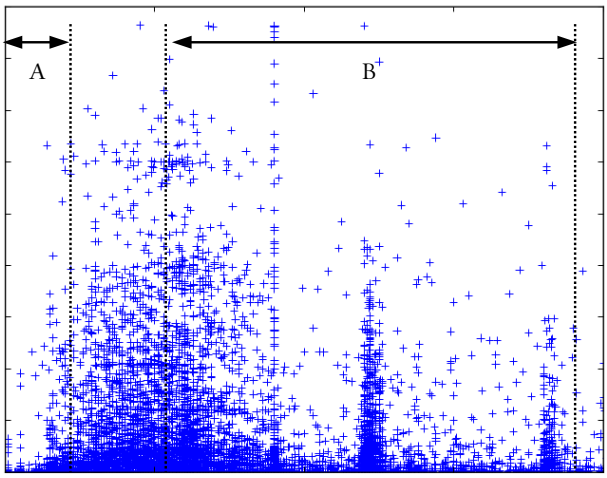


Figure 1.2.2: Temporal distribution of trajectories from a typical source of data. All trajectories are observed in small chunks. The vertical axis corresponds to the overall duration of the chunk, and the horizontal axis corresponds to the sampling rate of the chunk. Most data is programmatically generated at many intervals between 20 and 60 seconds, or at fixed intervals (10, 60, 90, 120 seconds). Most of it is generated at large intervals (¿ 10 seconds). Scale is omitted for confidentiality reasons. The data used in each of the two data scenarios ("A" and "B") is marked by an interval.

Spatial disparity. It is clear to any driver that all the roads do not bear the same traffic demand. As can be seen in Figure 1.2.3, this difference is very large (by more than four orders of magnitude). Highways bear the brunt of heavy traffic and are used almost continuously day and night. Most of the roads however are part of the tertiary network in the suburban areas and are used a few times each day or less. These roads seldom present any congestion, but may have some busy intersections (such as around schools) that are hard to predict *a priori.* Between these two extremes lies a rich variety of road configurations with one, two or three lanes, stop signs, signaled lights, bus and bicycle lanes, etc. Any model that aims at a reasonable level of accuracy must (1) be robust to a scarcity of data and/or noisy data, and (2) not depend too much on prior knowledge of the road network as it is always shifting or inaccurate. Bayesian statistical models provide a compelling solution to address these two issues.

Our approach will focus on extracting structure from data. In the case of traffic, there is a wealth of research focused on explaining observations based on first principles (more on that later). Our data exploration will be motivated by the general physical principals behind the phenomenon. However, it will not try to *explain* as much as *predict* the future or the unobserved data. In particular, some very strong simplifying assumptions will be made, and justified only by the accuracy of the prediction they let us make. These assumptions are justified only in the light that they let us make reasonable *predictions* using limited

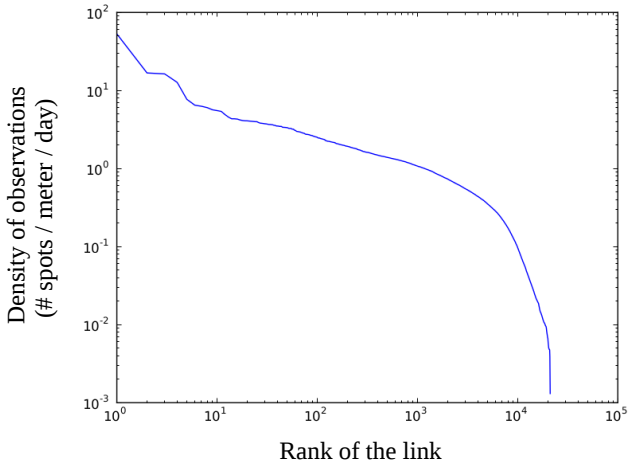


Figure 1.2.3: The number of observations (GPS points projected to the road network) by link, in decreasing order (normalized by the length of the link), for a full day, from a commercial data provider. The exact scale is shifted for confidentiality reasons. The first 100 elements correspond to highways. They contain 30% of all observations and have a very high observation density. In these situations, a fine-grained physical model is effective. For 90% of the links (index greater than 10^3), we barely observe a few vehicles per day on each road (spread-out suburbs), so a simple historical models will be the best we can hope for. For the middle range, we have enough observations for a richer statistical model, but not enough for a finer physical model.

computing resources.

1.3 Organization of the thesis and contributions

This thesis is organized as follows. This first chapter introduced the problem of estimating traffic at scale and gave an overview of the data currently available for this task. The following chapters present a framework for extracting travel information from GPS data.

GPS data can be very noisy and needs to be filtered and map-matched onto the map before any use. This is a challenging issue due to the long intervals between each observation. Chapter 2 presents an algorithm to map-match GPS data from a very wide range of latencies and for a variety of computation/accuracy trade-offs.

Contribution: The chapter formalizes the problem of reconstructing trajectories from high to low-frequency observations, and presents an algorithm, called the Path Inference Filter, to do that.

Publications: [60, 61]

Chapter 3 considers the case of large amounts of low-latency data, which is representative

of datasets currently available. In this case, the accuracy of the estimation problem is mostly bound by computation times. If some further independence assumptions are made about travel time distributions, we can build a model that can scale to hundreds of machines.

Contribution: Chapter 2 presents a model for travel times that can scale linearly to massive inputs of sparse observations and very large road networks. It uses the Spark programming framework [124] to distribute computations on a large cluster of computers. Our implementation can update traffic estimates from hundreds of thousands of observations in a few seconds. This algorithm is the core of an estimation pipeline deployed inside *Mobile Millennium*. This engine gathers GPS observations from participating vehicles and produces estimates of the travel times on the road network. As we will see, our framework can accommodate *any* distribution of travel times, provided they expose a few functionalities (sampling, parameter estimation from observations). This should be of interest to the traffic researchers and practitioners since our framework solves the issues related to using raw GPS samples, and let researchers build traffic estimates at a very large scale with low latency. A probabilistic model of travel times on the arterial network is presented along with an online *Expectation Maximization* (EM) algorithm for learning the parameters of this model (Section 3.2).

Publications: [62, 64].

Chapter 4 makes different assumptions and explores the scenario in which a small amount of high-frequency observations are available to the researcher. In this case, there is not enough data in real-time to achieve real-time traffic estimation. However, one can build a good baseline/historical model of travel times that takes into account the correlation of travel time between different segments of the road network. We will motivate this model with a careful look at the available data.

Contribution: The chapter presents a model for travel time that uses privacy-aware GPS data, and that provides distributions of travel times for any path in the road network. We also present an efficient inference algorithm based on Fast Fourier Transform to compute these travel times. Our implementation can work with long paths (comprising hundreds of road segments) on a large road network with more than half a million road links.

All the models considered so far have a reasonable size (less than a million variables), for which inference and learning algorithms can be derived by adapting traditional techniques. As the size of the problems at hand increases to truly enormous dimensions, some radically different techniques may be required. Learning the Gaussian model presented in Chapter 4 requires evaluating the determinant of a very large matrix, which is a significant problem in linear algebra for very large matrices. Chapter 5 presents a new algorithm that can compute the determinant of a class of matrices (diagonally dominant matrices) in *expected near-linear time*, which is a significant progress compared to the traditional intuition that the computation time for the determinant is cubic with respect to the dimension.

Contribution: The chapter presents a number of algorithm for approximating the logarithm of the determinant of real, symmetric, diagonally dominant matrices. More specifically, given a matrix A of size $n \times n$ with m non-zero coefficients, an $\epsilon-$approximation of $n^{-1} \log \det A$ can be computed with high probability in expected time

$O\left(m\epsilon^{-2}\log^3 n \log\left(\frac{\kappa_A}{\epsilon}\right)\log\log^8 n\right)$, where κ_A is the condition number of A.

Publication: This part will be submitted to the SIAM Journal of Matrix Analysis and Applications.

Finally, we will conclude in Chapter 6.

Chapter 2

A map-matching algorithm for GPS data

Pourquoi donc, reprit le Sirien, citez-vous un certain Aristote en grec? C'est, répliqua le savant, qu'il faut bien citer ce qu'on ne comprend point du tout dans la langue qu'on entend le moins.

Voltaire, *Micromégas*

2.1 Introduction

GPS receivers have enjoyed a widespread use in transportation and they are rapidly becoming a commodity. They offer unique capabilities for tracking fleets of vehicles (for companies), and routing and navigation (for individuals). These receivers are usually attached to a car or a truck, also called a *probe vehicle*, and they relay information to a base station using the data channels of cellphone networks (3G and 4G in particular).

One of the common problems which occurs when dealing with GPS traces is the correct mapping of these observations to the road network, and the reconstruction of the trajectory of the vehicle. We present a new class of algorithms, called the *path inference filter,* that solves this problem in a principled and efficient way. Specific instantiations of this algorithm have been deployed as part of the *Mobile Millennium* system, which is a traffic estimation and prediction system developed at the University of California [20].

A typical datum provided by a probe vehicle includes an identifier of the vehicle, a (noisy) position and a timestamp[1]. The two most important characteristics of GPS data for traffic estimation purposes are the GPS localization accuracy and the sampling strategy

[1]The experiments in this dissertation use GPS observations only. However, nothing prevents the application of the algorithms presented in this dissertation to other types of localized data.

followed by the probe vehicle. In order to reduce power consumption or transmission costs, probe vehicles do not continuously report their location to the base station. The probe data currently available are generated using a combination of the two following strategies:

- *Geographical sampling*: GPS probes are programmed to send information in the vicinity of *virtual landmarks* [78]. This concept was popularized by Nokia under the term *Virtual Trip Line* [58]. These landmarks are usually laid over some predetermined route followed by drivers.

- *Temporal sampling*: GPS probes send their position at fixed rate. The critical factor is then the *temporal resolution* of the probe data. A low temporal resolution carries some uncertainty as to which trajectory was followed. A high temporal resolution gives access to the complete and precise trajectory of the vehicle. However, the device usually consumes more power and communication bandwidth.

In the case of a high temporal resolution (typically, a frequency greater than one observation per second), some highly successful methods have been developed for continuous estimation [37, 84, 111]. However, most data collected at large scale today is generated by commercial fleet vehicles. It is primarily used for tracking the vehicles and usually has a low temporal resolution (1 to 2 minutes) [2, 4, 10, 108]. In the span of a minute, a vehicle in a city can cover several blocks. Therefore, information on the precise path followed by the vehicle is lost. Furthermore, due to GPS localization errors, recovering the location of a vehicle that just sent an observation is a non trivial task: there are usually several roads that could be compatible with any given GPS observation. Simple deterministic algorithms to reconstruct trajectories fail due to misprojection (Figure 2.1.3) or shortcuts (Figure 2.1.2). Such shortcomings have motivated our search for a principled approach that jointly considers the mapping of observations to the network and the reconstruction of the trajectory.

The problem of mapping data points onto a map can be traced back to 1980 [21]. Researchers started systematic studies after the introduction of the GPS system to civilian applications in the 1990s [93, 114]. These early approaches followed a *geometric* perspective, associating each observation datum to some point in the network [119]. Later, this projection technique was refined to use more information such as heading and road curvature. This greedy matching, however, leads to poor trajectory reconstruction since it does not consider the path leading up to a point [52, 96, 123]. New deterministic algorithms emerged to directly match partial trajectories to the road by using the topology of the network [45] and topological metrics based on the Fréchet distance [27, 117]. These deterministic algorithms cannot readily cope with ambiguous observations [84], and were soon expanded into probabilistic frameworks. A number of implementations were explored: particle filters [48, 91], Kalman filters [89], Hidden Markov Models [23][2], and less mainstream approaches based on Belief

[2]Note that "probabilistic" models, as well as most of the "advanced" models (Kalman Filtering, Particle Filtering, Hidden Markov Models) fall under the general umbrella of *Dynamic Bayesian Filters*, presented in great detail in [111]. As such, they deserve a common theoretical treatment, and in particular all suffer from the same pitfalls detailed in Section 2.3.

Theory [42] and Fuzzy Logic [107]. We refer the reader to the exhaustive review by Quddus et al. [92] for a more comprehensive history of map-matching.

Two types of information are missing in a sequence of GPS readings: the exact location of the vehicle on the road network when the observation was emitted, and the path followed from the previous location to the new location. These problems are correlated. The aforementioned approaches focus on high-frequency sampling observations, for which the path followed is extremely short (less than a few hundred meters, with very few intersections). In this context, there is usually a dominant path that starts from a well-defined point, and Bayesian filters accurately reconstruct paths from observations [48, 89, 111]. When sampling rates are lower and observed points are further apart, however, a large number of paths are possible between two points. Researchers have recently focused on efficiently identifying these correct paths and have separated the joint problem of finding the paths and finding the projections into two distinct problems. The first problem is path identification and the second step is projection matching [23, 44, 110, 123, 131]. Some interesting trajectories mixing points and paths that use a voting scheme have also recently been proposed [123]. Our filter aims at solving the two problems at the same time, by considering a single unified notion of *trajectory*.

The *path inference filter* is a probabilistic framework that aims at recovering trajectories and road positions from low-frequency probe data in real time, and in a computationally efficient manner. As will be shown, the performance of the filter degrades gracefully as the sampling frequency decreases, and it can be tuned to different scenarios (such as real time estimation with limited computing power or offline, high accuracy estimation).

The filter is justified from the Bayesian perspective of the noisy channel and falls into the general class of *Conditional Random Fields* [72]. Our framework can be decomposed into the following steps:

- *Map matching*: each GPS measurement from the input is projected onto a set of possible candidate states on the road network.
- *Path discovery:* admissible paths are computed between pairs of candidate points on the road network.
- *Filtering*: probabilities are assigned to the paths and the points using both a stochastic model for the vehicle dynamics and probabilistic driver preferences learned from data.

The path inference filter presents a number of compelling advantages over the work found in the current literature:

1. The approach presents a general framework grounded in established statistical theory that encompasses, as special cases, most techniques presented as "geometric", "topological" or "probabilistic". In particular, it combines information about paths, points and network topology in a single unified notion of *trajectory*.

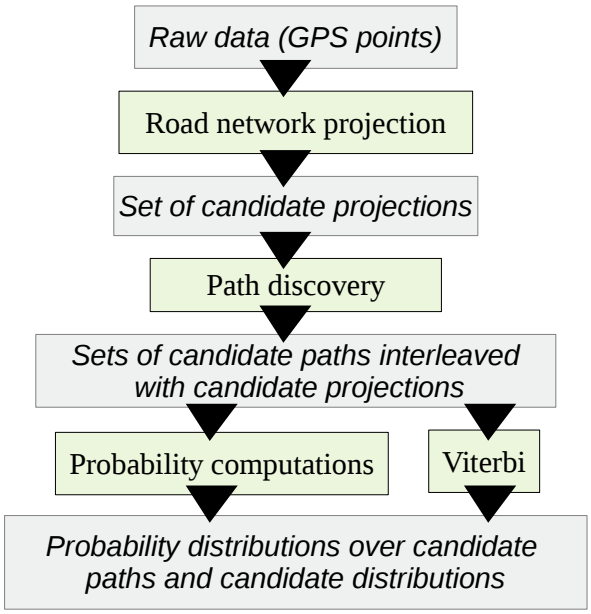


Figure 2.1.1: Dataflow of the path inference filter. The filter takes GPS measurements as inputs, and returns probability distributions over paths and points.

2. Nearly all work on Map Matching is segmented into (and presents results for) either high-frequency or low-frequency sampling. The path inference filter performs as well as the current state-of-the-art approaches for sampling rates less than 30 seconds, and improves upon the state of the art [123,131] by a factor of more than 10% for sampling intervals greater than 60 seconds[3]. We also analyze failure cases and we show that the output provided by the path inference filter is always "close" to the true output for some metric.

3. As will be seen in Section 2.3, most existing approaches (which are based on Dynamic Bayesian Networks) do not work well at lower frequencies due to the *selection bias problem*. Our work directly addresses this problem by performing inference on a Random Field.

4. The path inference filter can be used with complex path models such as those used in [23] and [44]. In the present discussion, we restrict ourselves to a class of models (the exponential family distributions) that is rich enough to provide insight on the driving patterns of the vehicles. Furthermore, when using this class of models, the learning of new parameters leads to a convex problem formulation that is fast to solve. These

[3]Performance comparisons are complicated by the lack of a single agreed-upon benchmark dataset. Nevertheless, the city we study is complex enough to compare favorably with cities studied with other works.

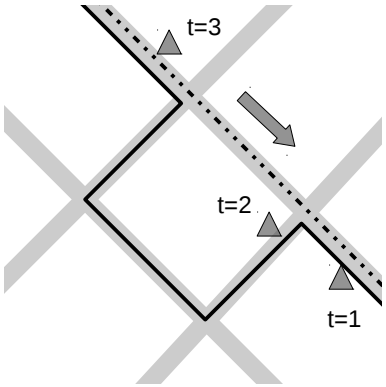


Figure 2.1.2: Example of failure in which an intuitive algorithm projects each GPS measurement to the closest link. The raw GPS measurements are the triangles, the actual true trajectory is the dashed line, and the reconstructed trajectory is the continuous line. Due to noise in the observation, the point at $t = 2$ is closer to the orthogonal road and forces the algorithm to add a left turn, while the vehicle is actually going straight. This problem is frequently observed for GPS data in cities. The *path inference filter* provides one solution to this problem.

parameters can be learned using standard Machine Learning algorithms, even when no ground truth is available.

5. With careful engineering, it is possible to achieve high throughput on large-scale networks. Our reference implementation achieves an average throughput of hundreds of GPS observations per second on a single core in real time. Furthermore, the algorithm scales well on multiple cores and has achieved average throughput of several thousands of points per second on a multicore architecture.

Algorithms often need to trade off accuracy for timeliness, and are considered either "local" (greedy) or "global" (accumulating some number of points before returning an answer) [123]. The path inference filter is designed to work across the full spectrum of accuracy versus latency. As we will show, we can still achieve good accuracy by delaying computations by only one or two time steps. We will also present an offline variation of the algorithm that performs full trajectory reconstruction with high throughput (at the expense of latency).

2.2 Map-matching and path discovery

The road network is described as a directed graph $\mathcal{N} = (\mathcal{V}, \mathcal{E})$ in which the nodes are the road intersections and the edges are the roads, referred to in the text as the *links* of the road network. Each link is endowed with a number of physical attributes (link geometry, speed limit, number of lanes, type of road, etc.). Since each road link is directed, there will be

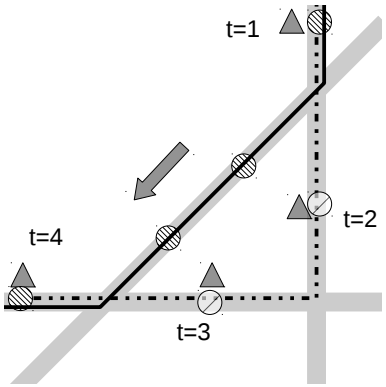


Figure 2.1.3: Example of failure when trying to minimize the path length between a sequence of points. The raw observations are the triangles, the actual true trajectory is the dashed line, and the reconstructed trajectory is the continuous line. The circles are possible locations of the vehicle corresponding to the observations. The hashed circles are the states chosen by this reconstruction algorithm. Due to GPS errors that induce problems explained in Figure 2.1.2, we must consider point projections on all links within a certain distance from the observed GPS points. However, the path computed by a shortest path algorithm may not correspond to the true trajectory. Note how, for $t = 2$ and $t = 3$, the wrong link and the wrong states are elected to reconstruct the trajectory.

other road links pointing at the start of that road link, and a set of road links starting at the end of that road link: given a link of the road network, the links into which a vehicle can travel will be called *outgoing links*, and the links from which it can come will be called the *incoming links.* Every location on the road network is completely specified by a given link l and offset o on this link[4]. The offset is a non-negative real number bounded by the length of the corresponding link, and represents the position on the link. At any time, the *state* x of a vehicle consists of its location on the road network and some other optional information such as speed, or heading. For our example we consider that the state is simply the location on one of the road links (which are directed). Additional information such as speed, heading, lane, etc. can easily be incorporated into the state:

$$x = (l, o)$$

Furthermore, for the remainder of the chapter we consider trajectory inference for a single probe vehicle.

[4]The complex intersections are not given special consideration. Adding some specific features such as described in [127] would be straightforward.

2.2.1 From GPS points to discrete vehicle states

The points are projected to the road following a Bayesian formulation. Consider a GPS observation g. We study the problem of projecting it to the road network according to our knowledge of how this observation was generated. This generation process is represented by a probability distribution $\omega(g|x)$ that, given a state x, returns a probability distribution over all possible GPS observations g. Such distributions ω will be described in Section 2.3.4. Additionally, we may have some *prior knowledge* over the state of the vehicle. For example, some links may be visited more often than others, and some positions on links may be more frequent, such as when vehicles accumulate at the intersections. This knowledge can be encoded in a *prior distribution* $\Omega(x)$. Under this general setting, the state of a vehicle, given a GPS observation, can be computed using Bayes rule:

$$\pi(x|g) \propto \omega(g|x)\,\Omega(x) \tag{2.2.1}$$

The letter π will define general probabilities, and their dependency on variables will always be included. This probability distribution is defined up to a scaling factor in order to integrate to 1. This posterior distribution is usually complicated, owing to the mixed nature of the state. The state space is the product of a discrete space over the links and a continuous space over the link offsets. Instead of representing it in closed form, some sampled values are considered: for each link l_i, a finite number of states from this link are elected to represent the posterior distribution of the states on this link $\pi(o|g, l = l_i)$. A first way of accomplishing this task is to grid the state space of each link, as illustrated in Figure 3.2.2. This strategy is robust against the observation errors described in Section 2.2.2, but it introduces a large number of states to consider. Furthermore, when new GPS values are observed every minute, the vehicle can move quite extensively between updates. The grid step is usually small compared to the distance traveled. Instead of defining a coarse grid over each link, another approach is to use some *most likely state* on each link. Since our state is the pair of a link and an offset on this link, this corresponds to selecting the most likely offset on each state:

$$\forall l_i, \quad o^*_{i_{\text{posterior}}} = \underset{o}{\operatorname{argmax}}\, \pi(o|g, l = l_i)$$

In practice, the probability distribution $\pi(x|g)$ decays rapidly, and can be considered overwhelmingly small beyond a certain distance from the observation g. Links located beyond a certain radius need not be considered valid projection links, and may be discarded. In our experiments, we found that a value of 80 meters would give good results.

In the rest of this chapter, the boldface symbol $\mathbf{x}$ will denote a finite collection of states associated with a GPS observation g that we will use to represent the posterior distribution $\pi(x|g)$, and the integer I will denote its cardinality: $\mathbf{x} = (x_i)_{1:I}$. These points are called *candidate state projections for the GPS measurement g.* These discrete points will then be linked together through trajectory information that takes into account the trajectory and the dynamics of the vehicle. We now mention a few important points for a practical implementation.

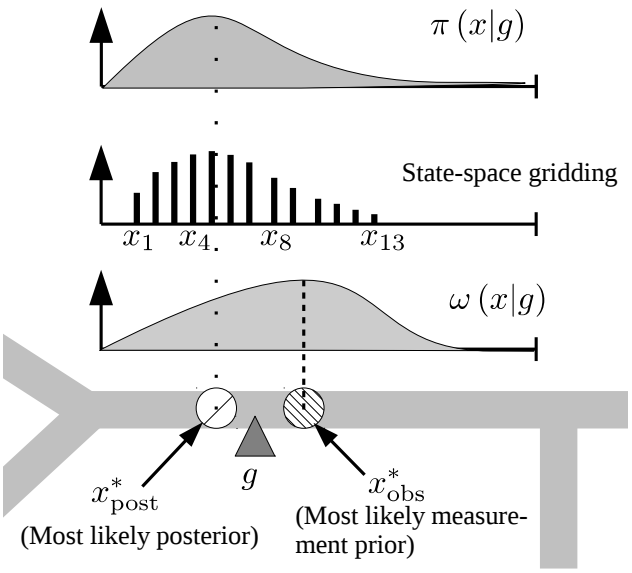


Figure 2.2.1: Example of a measurement g on a link and two strategies to associate state projections to that measurement on a particular link (gridding and most likely location). The GPS measurement is the triangle denoted g. For this particular measurement, the observation distribution $\omega(x|g)$ and the posterior distribution $\pi(x|g)$ are also represented. When gridding, we select a number of states $x_1, \cdots x_I$ spanning each link at regular intervals. This allows us to use the posterior distribution and have a more precise distribution over the location of the vehicle. However, it is more expensive to compute. Another strategy is to consider a single point at the most probable offset x^*_{post} according to the posterior distribution $\pi(x|g)$. However, this location depends on the prior, which is usually not available at this stage (since the prior depends on the location of past and future points, for which we do not also know the location). A simple approximation is to consider the most likely point x^*_{obs} according to the observation distribution.

A Bayesian formulation requires that we endow the state x with a prior distribution $\Omega(x)$ that expresses our knowledge about the distribution of points on a link. When no such information is available, since the offset is continuous and bounded on a segment, a natural non-informative prior is the uniform distribution over offsets: $\Omega \sim U([0, \text{length}(l)])$. In this case, maximizing the posterior is equivalent to maximizing the conditional distribution from the generative model:

$$\forall l_i, o^*_{i_{\text{observation}}} = \text{argmax}_o \omega(g|x = (o, l_i)) \tag{2.2.2}$$

Having projected GPS points into discrete points on the road network, we now turn our attention to connecting these points by paths in order to form trajectories.

2.2.2 From discrete vehicle states to trajectories

At each time step t, a GPS point g^t (originating from a single vehicle) is observed. This GPS point is then projected onto I^t different candidate states denoted $\mathbf{x}^t = x_1^t \cdots x_{I^t}^t$. Because this set of projections is finite, there is only a (small) finite number J^t of paths that a vehicle can have taken while moving from some state $x_i^t \in \mathbf{x}^t$ to some state $x_{i'}^{t+1} \in \mathbf{x}^{t+1}$. We denote the set of *candidate paths* between the observation g^t and the next observation g^{t+1} by $\mathbf{p}^t$:

$$\mathbf{p}^t = \left(p_j^t\right)_{j=1:J^t}$$

Each path p_j^t goes from one of the projection states x_i^t of g^t to a projection state $x_{i'}^{t+1}$ of g^{t+1}. There may be multiple pairs of states to consider, and between each pair of states, there are typically several paths available (see Figure 2.2.2). Lastly, a *trajectory* is defined by the succession of states and paths, starting and ending with a state:

$$\tau = x_1 p_1 x_2 \cdots p_{t-1} x_t$$

where x_1 is one element of $\mathbf{x}^1$, p_1 of $\mathbf{p}^1$, and so on.

Due to speed limits leading to lower bounds on achievable travel times on the network, there is only a finite number of paths a vehicle can take during a time interval Δt. Such paths can be computed using standard graph search algorithms. The depth of the search is bounded by the maximum distance a vehicle can travel on the network at a speed $v_{\max}$ within the time interval between each observation. An algorithm that performs well in practice is the A* algorithm [49], a common graph search algorithm that makes use of a heuristic to guide its search. The cost metric we use here is the expected travel time on each link, and the heuristic is the shortest geographical distance. In order to ensure the correctness of the A* algorithm, the heuristic has to verify some admissibility criteria. In practice, the approximate cost computed by the heuristic has to be lower than the true cost metric. We scale the shortest geographical distance by the inverse of the maximum speed limit of our network (70 m.p.h.), which makes the heuristic admissible.

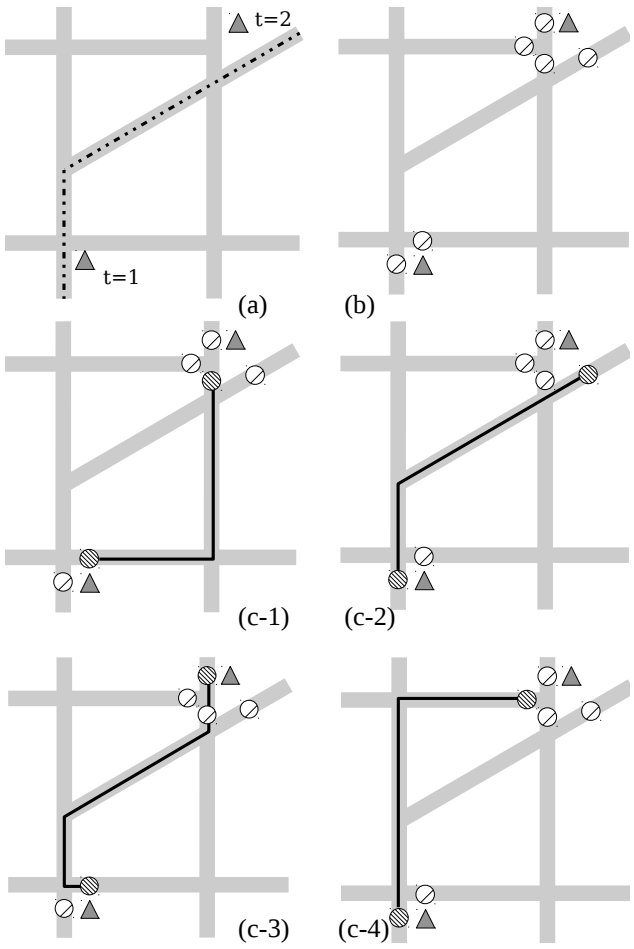


Figure 2.2.2: Example of path exploration between two observations. The true trajectory and two associated GPS observations are shown in Figure (a). Figure (b) shows the set of candidate projections associated with each observation. A path discovery algorithm computes every acceptable path between between each pair of candidate projections. The four figures (c-1) to (c-4) at the bottom show a few examples of such computed paths.

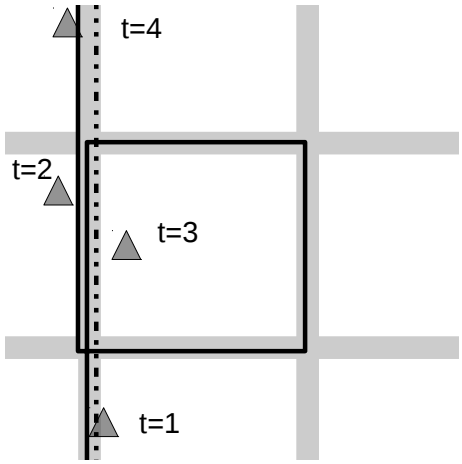


Figure 2.2.3: Example of failure when observing strict physical consistency: due to the observation noise, the observation (3) appears physically behind (2) on the same link. Without considering backward paths, the most plausible explanation is that the vehicle performed a complete loop around the neighboring block.

The case of backward paths. It is convenient and realistic to assume that a vehicle always drives *forward*, i.e. in the same direction of a link[5]. In our notation, a vehicle enters a link at offset 0, drives along the link following a non-decreasing offset, and exits the link when the offset value reaches the total length of the link. However, due to GPS noise, the most likely state projection of a vehicle waiting at a red light may appear to go backward, as shown in Figure 2.2.3. This leads to incorrect transitions if we assume that paths only go forward on a link. While this issue could be readily solved if we knew the spot speed of the vehicle, we cannot assume this information in our case. Three approaches to solve this issue are discussed, depending on the application:

1. It is possible to keep a single state for each link (the most likely) and explore some backward paths. These paths are assumed to go backward because of observation noise. This solution provides *connected states at the expense of physical consistency*: all the measurements are correctly projected to their most likely location, but the trajectories themselves are not physically acceptable. This is useful for applications that do not require connectedness between pairs of states, for example when computing a distribution of the density of probe data per link.

2. It is also possible to disallow backward paths and consider multiple states per link, such as a grid over the state space. A vehicle never goes backward, and in this case the filter can generally account for the vehicle not moving by associating the same state to successive observations. All the trajectories are physically consistent and the posterior

[5]Reverse driving is in some cases even illegal. For example, the laws of Glendale, Arizona, prohibit reverse driving.

state density is the same as the probability density of the most likely states, but is more burdensome from a computational perspective (the number of paths to consider grows quadratically with the number of states).

3. Finally it is possible to disallow backward paths and use a sparse number of states. The path connectivity issue is solved using some heuristics. Our implementation creates a new state projection on a link l using the following approach:
Given a new observation g, and its most likely state projection $x^* = (l, o^*)$:

   a) If no projection for the link l was found at the previous time step, return x^*

   b) If such a projection $x_{\text{before}} = (l, o_{\text{before}})$ existed, return $x = (l, \max(o_{\text{before}}, o^*))$

   With this heuristic, all the points will be well connected, but the density of the states will not be the same as the density of the most likely reconstructed states.

In summary, the first solution is better for density estimations and the third approach works better for travel time estimations. The second option is currently only used for high-frequency offline filtering, for which paths are short, and for which more expensive computations is an acceptable cost.

Handling errors. Maps may contains some inaccuracies, and may not cover all the possible driving patterns. Two errors were found to have a serious effect on the performance of the filter:

- Out of network driving: This usually occurs in parking lots or commercials driveways.
- Topological errors: Some links may be missing on the base map, or one-way roads may have changed to two-way roads. These situations are handled by running *flow analysis* on the trajectory graph. For every new incoming GPS point, after computing the paths and states, it is checked if any candidate position of the first point of the trajectory is reachable from any candidate position on the latest incoming point, or equivalently if the trajectory graph has a positive flow. The set of state projections of an observation may end up being disconnected from the start point even if at every step, there exists a set of paths between each points. In this situation, the probability model will return a probability of 0 (non-physical trajectories) for any trajectory. If a point becomes unreachable from the start point, the trajectory is broken, and restarted again from this point. Trajectory breaks were infrequent (less than a dozen for our dataset), and a visual inspection showed that the vehicle was not following the topology of the network and instead made U-turns or breached through one-way roads. Although we have not implemented these extensions, it is very natural to express the map errors in the probabilistic framework. These extensions are discussed in Sections 2.3.5 and 2.3.4.

The GPS also has a complex error model and may return some location estimates completely off the trajectory. We found these errors to appear at random, in an uncorrelated fashion. The reference implementation detects these errors during the flow analysis with the following heuristic: consider a sequence of points $g^{(1)}, g^{(2)}, g^{(3)}$. If $g^{(1)}$ cannot reach $g^{(2)}$, then compute the reachability of $g^{(3)}$: if $g^{(3)}$ is reachable from $g^{(1)}$, then the point $g^{(2)}$ is an outlier and is dropped. If $g^{(3)}$ is not reachable from $g^{(1)}$, we consider the trajectory is breached between $g^{(1)}$ and $g^{(2)}$, and we restart the flow analysis from $g^{(2)}$. The reference implementation offers an elegant recursive implementation.

2.3 Discrete filtering using a Conditional Random Field

In the previous section, we reduced the trajectory reconstruction problem to a discrete selection problem between sets of candidate projection points, interleaved with sets of candidate paths. A probabilistic framework can now be applied to infer a reconstructed trajectory τ or probability distributions over candidate states and candidate paths. Without further assumptions, one would have to enumerate and compute probabilities for every possible trajectory. This is not possible for long sequences of observations, as the number of possible trajectories grows exponentially with the number of observations chained together. By assuming additional independence relations, we turn this intractable inference problem into a tractable one.

2.3.1 Conditional Random Fields to weight trajectories

The observation model provides the joint distribution of a state on the road network given an observation. We have described the *noisy generative model* for the observations in Section 2.2.1. Assuming that the vehicle is at a point x, a GPS observation g will be observed according to a model ω that describes a noisy observation channel. The value of g only depends on the state of the vehicle, i.e. the model reads $\omega(g|x)$. For every time step t, assuming that the vehicle is at the location x^t, a GPS observation g^t is created according to the distribution $\omega(g^t|x^t)$.

Additionally, we endow the set of all possible paths on the road network with a probability distribution. The *transition model* η describes the preference of a driver for a particular path. In probabilistic terms, it provides a distribution $\eta(p)$ defined over all possible paths p across the road network. This distribution is not a distribution over actually observed paths as much as a model of the *preferences* of the driver when given the choice between several options.

We introduce the following *Markov assumptions*.

- Given a start state x_{start} and an end state x_{end}, the path p followed by the vehicle between these two points will only depend on the start state, the end state and the

path itself. In particular, it will not depend on previous paths or future paths.

- Consider a state x followed by a path p_{next} and preceded by a path p_{previous}, and associated to a GPS measurement g. Then the paths taken by the vehicle are independent from the GPS measurement g if the state x is known. In other words, the GPS measurement does not add subsequent information given the knowledge of the state of the vehicle.

Since a state is composed of an offset and a link, a path is completely determined by a start state, an end state and a list of links in between. Conditional on the start state and end state, the number of paths between these points is finite (it is the number of link paths that join the start link and the end link).

Not every path is compatible with given start point and end point: the path must start at the start state and must end at the end state. We formally express the compatibility between a state x and the start state of a path p with the compatibility function $\underline{\delta}$:

$$\underline{\delta}\left(x,p\right)=\begin{cases}1 & \text{if the path } p \text{ starts at point } x\\ 0 & \text{otherwise}\end{cases}$$

Similarly, we introduce the compatibility function $\bar{\delta}$ to express the agreement between a state and the end state of a path:

$$\bar{\delta}\left(p,x\right)=\begin{cases}1 & \text{if the path } p \text{ ends at point } x\\ 0 & \text{otherwise}\end{cases}$$

Given a sequence of observations $g^{1:T}=g^1\cdots g^T$ and an associated trajectory $\tau=x^1p^1\cdots x^T$, we define the *unnormalized score,* or *potential*, of the trajectory as:

$$\phi\left(\tau|g^{1:T}\right) \;=\; \left[\prod_{t=1}^{T-1}\omega\left(g^t|x^t\right)\underline{\delta}\left(x^t,p^t\right)\eta\left(p^t\right)\bar{\delta}\left(p^t,x^{t+1}\right)\right]\omega\left(g^T|x^T\right) \tag{2.3.1}$$

The non-negative function ϕ is called the *potential function.* A trajectory τ is said to be a *compatible trajectory with the observation sequence* $g^{1:T}$ if $\phi\left(\tau|g^{1:T}\right)>0$. When properly scaled, the potential ϕ defines a probability distribution over all possible trajectories, given a sequence of observations:

$$\pi\left(\tau|g^{1:T}\right)=\frac{\phi\left(\tau|g^{1:T}\right)}{Z} \tag{2.3.2}$$

The variable Z, called the *partition function*, is the sum of the potentials over all the compatible trajectories:

$$Z=\sum_{\tau}\phi\left(\tau|g^{1:T}\right)$$

We have combined the observation model ω and the transition model η into a single potential function ϕ, which defines an unnormalized distribution over all trajectories. Such

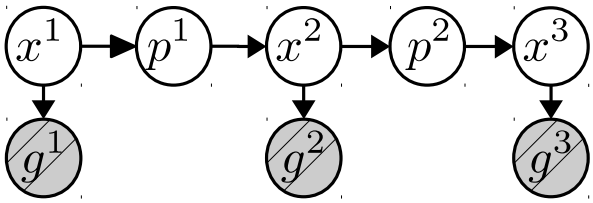


Figure 2.3.1: A Dynamic Bayesian Network (DBN) commonly used to model the trajectory reconstruction problem. The arrows indicate the directed dependencies of the variables. The GPS observations g^t are generated from states x^t. The unobserved paths p^t are generated from a state x^t, following a transition probability distribution $\hat{\pi}(p|x)$. The transition from a path p^t to a state x^t follows the transition model $\hat{\pi}(x|p)$.

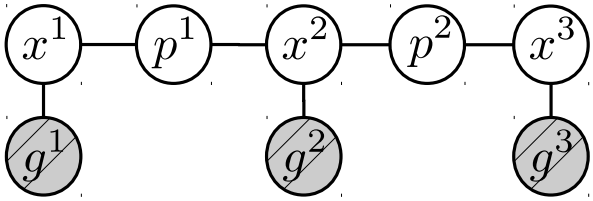


Figure 2.3.2: Illustration of the Conditional Random Field defined over a trajectory $\tau = x^1p^1x^2p^2x^3$ and a sequence of observations $g^{1:3}$. The gray nodes indicate the observed values. The solid lines indicate the factors between the variables: $\omega(g^t|x^t)$ between a state x^t and an observation g^t, $\underline{\delta}(x^t, p^t)\,\eta(p^t)$ between a state x^t and a path p^t and $\overline{\delta}(p^t, x^{t+1})$ between a path p^t and a subsequent state x^{t+1}.

a probabilistic framework is called a *Conditional Random Field* (CRF) [72]. A CRF is an undirected graphical model which is defined as the unnormalized product of factors over cliques of factors (see Figure 2.3.2). There can be an exponentially large number of paths, so the partition function cannot be computed by simply summing the value of ϕ over every possible trajectory. As will be seen in Section 2.4, the value of Z needs to be computed only during the training phase. Furthermore it can be computed efficiently using dynamic programming.

The case against the Hidden Markov Model approach. The classical approach to filtering in the context of trajectories is based on Hidden Markov Models (HMMs), or their generalization, Dynamic Bayesian Networks (DBNs) [87]: a sequence of states and trajectories form a trajectory, and the coupling of trajectories and states is done using transition models $\hat{\pi}(x|p)$ and $\hat{\pi}(p|x)$. See Figure 2.3.1 for a representation.

This results in a chain-structured directed probabilistic graphical model in which the path variables p^t are unobserved. Depending on the specifics of the transition models, $\hat{\pi}(x|p)$ and $\hat{\pi}(p|x)$, probabilistic inference has been done with Kalman filters [89, 91], the forward

Figure 2.3.3: Example of a failure case when using a Hidden Markov Model: the solid black path will be favored over all the other paths. The paths p_i^1 connect the states x_j^1 to x_k^2. See section II for more details on the notations.

algorithm or the Viterbi algorithm [23, 24], or particle filters [48].

Hidden Markov Model representations, however, suffer from the *selection bias problem*, first noted in the labeling of words sequences [72], which makes them not the best fit for solving path inference problems. Consider the example trajectory $\tau = x^1 p^1 x^2$ observed in our data, represented in Figure 2.3.3. For clarity, we consider only two states x_1^1 and x_2^1 associated with the GPS reading g^1 and a single state x_1^2 associated with g^2. The paths $\left(p_j^1\right)_j$ between x^1 and x^2 may either be the lone path p_1^1 from x_1^1 to x_1^2 that allows a vehicle to cross the Golden Gate Park, or one of the many paths between Cabrillo Street and Fulton Street that go from x_2^1 to x^1, including p_3^1 and p_2^1. In the HMM representation, the transition probabilities must sum to 1 when conditioned on a starting point. Since there is a single path from x_2^1 to x^2, the probability of taking this path from the state x_1^1 will be $\hat{\pi}\left(p_1^1 | x_1^1\right) = 1$ so the overall probability of this path is $\hat{\pi}\left(p_1^1 | g^1\right) = \hat{\pi}\left(x_1^1 | g^1\right)$. Consider now the paths from x_2^1 to x_1^2: a lot of these paths will have a similar weight, since they correspond to different turns and across the lattice of roads. For each path p amongst these N paths of similar weight, Bayes assumption implies $\hat{\pi}\left(p | x_2^1\right) \approx \frac{1}{N}$ so the overall probability of this path is $\hat{\pi}\left(p | g^1\right) \approx \frac{1}{N}\hat{\pi}\left(x_2^1 | g^1\right)$. In this case, N can be large enough that $\hat{\pi}\left(p_1^1 | g^1\right) \geq \hat{\pi}\left(p | g^1\right)$, and the remote path will be selected as the most likely path.

Due to their structures, all HMM models will be biased towards states that have the least expansions. In the case of a road network, this can be pathological. In particular, the HMM assumption will carry the effect of the selection bias as long as there are long disconnected segments of road. This can be particularly troublesome in the case of road networks since HMM models will end up being forced to assign too much weight to a highway (which is highly disconnected) and not enough to the road network alongside the highway. Our model, which is based on Conditional Random Fields, does not have this problem since

the renormalization happens just once and is over all paths from start to end, rather than renormalizing for every single state transition independently.

Efficient filtering algorithms. Using the probabilistic framework of the CRF, we wish to infer:

- the most likely trajectory τ:

$$\tau^* = \underset{\tau}{\operatorname{argmax}} \ \pi\left(\tau | g^{1:T}\right) \tag{2.3.3}$$

- the posterior distributions over the elements of the trajectory, i.e. the conditional marginals $\pi\left(x^t | g^{1:T}\right)$ and $\pi\left(p^t | g^{1:T}\right)$

As will be seen, both elements can be computed without having to obtain the partition function Z. The solution to both problems is a particular case of the *Junction Tree algorithm* [87] and can be computed in time complexity linear in the time horizon by using a dynamic programing formulation. Computing the most likely trajectory is a particular instantiation of a standard dynamic programing algorithm called the *Viterbi algorithm* [43]. Using a classic Machine Learning algorithm for chain-structured junction trees (the *forward-backward algorithm* [25, 101]), all the conditional marginals can be computed in two passes over the variables. In the next section, we detail the justification for the Viterbi algorithm and in Section 2.3.3 we describe an efficient implementation of the forward-backward algorithm in the context of this application.

2.3.2 Finding the most likely path

For the rest of this section, we fix a sequence of observations $g^{1:T}$. For each observation g^t, we consider a set of candidate state projections $\mathbf{x}^t$. At each time step $t \in [1 \cdots T-1]$, we consider a set of paths $\mathbf{p}^t$, so that each path p^t from $\mathbf{p}^t$ starts from some state $x^t \in \mathbf{x}^t$ and ends at some state $x^{t+1} \in \mathbf{x}^{t+1}$. We will consider the set ς of valid trajectories in the Cartesian space defined by these projections and these paths:

$$\varsigma = \left\{ \tau = x^1 p^1 \cdots p^{T-1} x^T | \begin{array}{c} x^t \in \mathbf{x}^t \\ p^t \in \mathbf{p}^t \\ \overline{\delta}\left(x^t, p^t\right) = 1 \\ \underline{\delta}\left(p^t, x^{t+1}\right) = 1 \end{array} \right\}$$

In particular, if I^t is the number of candidate states associated with g^t (i.e. the cardinal of $\mathbf{x}^t$) and J^t is the number of candidate paths in $\mathbf{p}^t$, then there are at most $\prod_1^T I^t \prod_1^{T-1} J^t$ possible trajectories to consider. We will see, however, that most likely trajectory τ^* can be computed in $O\left(TI^*J^*\right)$ computations, with $I^* = \max_t I^t$ and $J^* = \max_t J^t$.

The partition function Z does not depend on the current trajectory τ and need not be computed when solving Equation 2.3.3:

$$\begin{aligned} \tau^* &= \underset{\tau \in \varsigma}{\operatorname{argmax}} \ \pi\left(\tau | g^{1:T}\right) \\ &= \underset{\tau \in \varsigma}{\operatorname{argmax}} \ \phi\left(\tau | g^{1:T}\right) \end{aligned}$$

Call $\phi^*\left(g^{1:T}\right)$ the maximum value over all the potentials of the trajectories compatible with the observations $g^{1:T}$:

$$\phi^*\left(g^{1:T}\right) = \max_{\tau \in \varsigma} \phi\left(\tau | g^{1:T}\right) \tag{2.3.4}$$

The trajectory τ that realizes this maximum value is found by tracing back the computations. For example, some pointers to the intermediate partial trajectories can be stored to trace back the complete trajectory, as done in the referring implementation [11]. This is why we will only consider the computation of this maximum. The function ϕ depends on the probability distributions ω and η, left undefined so far. These distributions will be presented in depth in Sections 2.3.4 and 2.3.5.

It is useful to introduce notation related to a *partial trajectory*. Call $\tau^{1:t}$ the *partial trajectory* until time step t:

$$\tau^{1:t} = x^1 p^1 x^2 p^2 \cdots x^t$$

For a partial trajectory, we define some partial potentials $\phi\left(\tau^{1:t} | g^{1:t}\right)$ that only depend on the observations seen so far:

$$\begin{aligned} \phi\left(\tau^{1:t} | g^{1:t}\right) &= \omega\left(g^1 | x^1\right) \prod_{t'=1}^{t-1} \underline{\delta}\left(x^{t'}, p^{t'}\right) \eta\left(p^{t'}\right) \\ &\quad \cdot \bar{\delta}\left(p^{t'}, x^{t'+1}\right) \omega\left(g^{t'+1} | x^{t'+1}\right) \end{aligned} \tag{2.3.5}$$

For each time step t, given a state index $i \in [1, I^t]$ we introduce the potential function for trajectories that end at the state x_i^t:

$$\phi_i^t = \max_{\tau^{1:t} = x^1 p^1 \cdots . x^{t-1} p^{t-1} x_i^t} \phi\left(\tau^{1:t} | g^{1:t}\right)$$

One sees:

$$\phi^* = \max_{i \in [1, I^T]} \phi_i^T$$

The partial potentials defined in Equation (2.3.5) follow an inductive identity:

$$\begin{aligned} \phi_i^1 &= \omega\left(g^1 | x_i^t\right) \\ \forall t, \phi_i^{t+1} &= \max_{\substack{i' \in [1, I^t] \\ j \in [1, J^t]}} \left[\phi_{i'}^t \underline{\delta}\left(x_{i'}^t, p_j^t\right) \eta\left(p_j^t\right) \bar{\delta}\left(p_j^t, x_i^{t+1}\right) \omega\left(g^{t+1} | x_i^{t+1}\right)\right] \end{aligned} \tag{2.3.6}$$

By using this identity, the maximum potential ϕ^* can be computed efficiently from the partial maximum potentials ϕ_i^t. The computation of the trajectory that realizes this maximum potential ensues by tracing back the computation to find which partial trajectory realized ϕ_i^t for each step t.

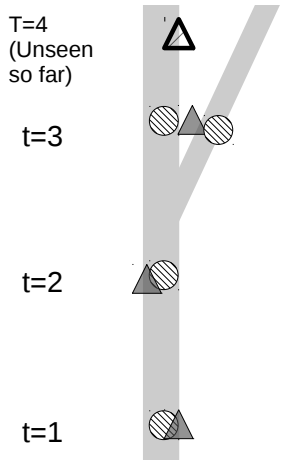


Figure 2.3.4: Example of case handled by lagged smoothing which disambiguates the results provided by tracking. An observation is available close to an exit ramp of a highway, for which the algorithm has to decide if it corresponds to the vehicle exiting the highway. Lagged smoothing analyzes subsequent points in the trajectory and can disambiguate the case.

2.3.3 Trajectory filtering and smoothing

We now turn our attention to the problem of computing the marginals of the posterior distributions over the trajectories, i.e. the probability distributions $\pi\left(x^t|g^{1:T}\right)$ and $\pi\left(p^t|g^{1:T}\right)$ for all t. We introduce some additional notation to simplify the subsequent derivations. The posterior probability $\bar{q}_i^t$ of the vehicle being at the state $x_i^t \in \mathbf{x}^t$ at time t, given all the observations $g^{1:T}$ of the trajectory, is defined as:

$$\bar{q}_i^t \propto \pi\left(x_i^t|g^{1:T}\right) = \frac{1}{Z} \sum_{\tau = x^1 \cdots p^{t-1} x_i^t p^t \cdots x^T} \phi\left(\tau|g^{1:T}\right)$$

The operator $\propto$ indicates that this distribution is defined up to some scaling factor, which does not depend on x or p (but may depend on $g^{1:T}$). Indeed, we are interested in the probabilistic weight of a state x_i^t relative to the other possible states $x_{i'}^t$ at the state time t (and not to the actual, unscaled value of $\pi\left(x_i^t|g^{1:T}\right)$). This is why we consider $\left(\bar{q}_i^t\right)_i$ as a choice between a (finite) set of discrete variables, one choice per possible state x_i^t. A natural choice is to scale the distribution $\bar{q}_i^t$ so that the probabilistic weight of all possibilities is equal to 1:

$$\sum_{1 \leq i \leq I^t} \bar{q}_i^t = 1$$

From a practical perspective, $\bar{q}^t$ can be computed without the knowledge of the partition function Z. Indeed, the only required elements are the unscaled values of $\pi\left(x_i^t|g^{1:T}\right)$ for each i. The distribution $\bar{q}^t = \left(\bar{q}_i^t\right)_i$ is a multinomial distribution between I^t choices, one for each

state. The quantity $\bar{q}_i^t$ has a clear meaning: it is the probability that the vehicle is in state x_i^t, when choosing amongst the set $(x_{i'}^t)_{1\leq i\leq I^t}$, given all the observations $g^{1:T}$.

For each time t and each path index $j \in [1 \cdots J^t]$, we also introduce (up to a scaling constant) the discrete distribution over the paths at time t given the observations $g^{1:T}$:

$$\overline{r}_j^t \propto \pi\left(p_j^t | g^{1:T}\right)$$

which are scaled so that $\sum_{1\leq j\leq J^t} \overline{r}_j^t = 1$.

This problem of smoothing in CRFs is a classic application of the Junction Tree algorithm to chain-structured graphs [87]. For the sake of completeness, we derive an efficient smoothing algorithm using our notations.

The definition of $\pi\left(x_i^t | g^{1:T}\right)$ requires summing the potentials of all the trajectories that pass through the state x_i^t at time t. The key insight for efficient filtering or smoothing is to make use of the chain structure of the graph, which lets us factorize the summation into two terms, each of which can be computed much faster than the exponentially large summation. Indeed, one can show from the structure of the clique graph that the following holds for all time steps t:

$$\pi\left(x^t | g^{1:T}\right) \propto \pi\left(x^t | g^{1:t}\right) \pi\left(x^t | g^{t+1:T}\right) \tag{2.3.7}$$

The first term of the pair corresponds to the effect that the *past and present observations* $(g^{1:t})$ have on our belief of the present state x^t. The second term corresponds to the effect that the *future observations* $(g^{t+1:T})$ have on our estimation of the present state. The terms $\pi\left(x^t | g^{1:t}\right)$ are related to each other by an equation that propagates *forward* in time, while the terms $\pi\left(x^t | g^{t+1:T}\right)$ are related through an equation that goes *backward* in time. This is why we call $\pi\left(x^t | g^{1:t}\right)$ the *forward distribution for the states*, and we denote it [6] by $\left(\overrightarrow{q}^{\,t}_{\,i}\right)_{1\leq i\leq I^t}$:

$$\overrightarrow{q}^{\,t}_{\,i} \propto \pi\left(x_i^t | g^{1:t}\right)$$

The distribution $\overrightarrow{q}^{\,t}_{\,i}$ is proportional to the posterior probability $\pi\left(x_i^t | g^{1:t}\right)$ and the vector $\overrightarrow{q}^{\,t} = \left(\overrightarrow{q}^{\,t}_{\,i}\right)_i$ is normalized so that $\sum_{i=1}^{I^t} \overrightarrow{q}^{\,t}_{\,i} = 1$. We do this for the paths, by defining the *forward distribution for the paths*:

$$\overrightarrow{r}^{\,t}_{\,j} \propto \pi\left(p_j^t | g^{1:t}\right)$$

Again, the distributions are defined up to a normalization factor so that each component sums to 1.

In the same fashion, we introduce the *backward distributions for the states and the paths:*

$$\overleftarrow{q}^{\,t}_{\,i} \propto \pi\left(x_i^t | g^{t+1:T}\right)$$
$$\overleftarrow{r}^{\,t}_{\,j} \propto \pi\left(p_j^t | g^{t+1:T}\right)$$

[6] The arrow notation indicates that the computations for $\overrightarrow{q}^{\,t}_{\,i}$ will be done forward in time.

Using this set of notations, Equation (2.3.7) can be rewritten:

$$\begin{aligned} \overline{q}_i^t &\propto \overrightarrow{q}_i^{\,t} \cdot \overleftarrow{q}_i^{\,t} \\ \overline{r}_j^t &\propto \overrightarrow{r}_j^{\,t} \cdot \overleftarrow{r}_j^{\,t} \end{aligned}$$

Furthermore, $\overrightarrow{r}^{\,t}$ and $\overrightarrow{q}^{\,t}$ are related through a pair of recursive equations:

$$\overrightarrow{q}_i^{\,1} \propto \pi\left(x_i^1 | g^1\right)$$

$$\overrightarrow{r}_j^{\,t} \propto \eta\left(p_j^t\right) \left(\sum_{j:\underline{\delta}\left(x_i^t, p_j^t\right)=1} \overrightarrow{q}_i^{\,t} \right) \tag{2.3.8}$$

$$\overrightarrow{q}_i^{\,t} \propto \omega\left(x_i^t | g^t\right) \left(\sum_{j:\bar{\delta}\left(p_j^{t-1}, x_i^t\right)=1} \overrightarrow{r}_j^{\,t-1} \right) \tag{2.3.9}$$

Similarly, the backward distributions can be defined recursively, starting from $t = T$:

$$\overleftarrow{q}_i^{\,T} \propto 1$$

$$\overleftarrow{r}_j^{\,t} \propto \eta\left(p_j^t\right) \left(\sum_{j:\bar{\delta}\left(p_j^t, x_i^{t+1}\right)=1} \overleftarrow{q}_i^{\,t+1} \right) \tag{2.3.10}$$

$$\overleftarrow{q}_i^{\,t} \propto \omega\left(x_i^t | g^t\right) \left(\sum_{j:\underline{\delta}\left(x_i^t, p_j^t\right)=1} \overleftarrow{r}_j^{\,t} \right) \tag{2.3.11}$$

Details of the forward algorithm and backward algorithm are provided in the Algorithm 2.1 and Algorithm 2.2 below. The complete algorithm for smoothing is detailed in the Algorithm 2.3 below.

The Algorithm 2.3 (smoothing) requires all the observations of a trajectory in order to run. We have presented so far an *a posteriori* algorithm that requires full knowledge of measurements $g^{1:T}$. In this form, it is not directly suitable for real-time applications that involve streaming data, for which the data is available up to t only. However, this algorithm can be adapted for a variety of scenarios:

- *Smoothing*, also called *offline filtering*. This corresponds to getting the best estimate given all observations, i.e. to computing $\pi\left(x^t | g^{1:T}\right)$. The Algorithm 2.3 describes this procedure.

Algorithm 2.1 Description of forward recursion

Given a sequence of observations $g^{1:T}$, a sequence of sets of candidate projections $\mathbf{x}^{1:T}$ and a sequence of sets of candidate paths $\mathbf{p}^{1:T-1}$:
Initialize the forward state distribution:
$\forall i = 1 \cdots I^1$: $\overrightarrow{q}_i^{\,1} \leftarrow \omega\left(x_i^1 | g^1\right)$
Normalize $\overrightarrow{q}^{\,1}$
For every time step t from 1 to $T-1$:
Compute the forward probability over the paths:
$\forall j = 1 \cdots J^t$:
$\overrightarrow{r}_j^{\,t} \leftarrow \eta\left(p_j^t\right) \left(\sum_{j:\underline{\delta}\left(x_i^t, p_j^t\right)=1} \overrightarrow{q}_i^{\,t}\right)$
Normalize $\overrightarrow{r}^{\,t}$
Compute the forward probability over the states:
$\forall i = 1 \cdots I^{t+1}$:
$\overrightarrow{q}_i^{\,t+1} \leftarrow \omega\left(x_i^{t+1} | g^{t+1}\right) \left(\sum_{j:\bar{\delta}\left(p_j^t, x_i^{t+1}\right)=1} \overrightarrow{r}_j^{\,t}\right)$
Normalize $\overrightarrow{q}^{\,t+1}$
Return the set of vectors $\left(\overrightarrow{q}^{\,t}\right)_t$ and $\left(\overrightarrow{r}^{\,t}\right)_t$

- *Tracking, filtering*, or *online estimation.* This usage corresponds to updating the current state of the vehicle as soon as a new streaming observation is available, i.e. to computing $\pi\left(x^t | g^{1:t}\right)$. This is exactly the case the forward algorithm (Algorithm 2.1) is set to solve. If one is simply interested in the most recent estimate, then only the previous forward distribution $\overrightarrow{q}^{\,t}$ needs to be kept, and all distributions $\overrightarrow{q}^{\,t-1} \cdots \overrightarrow{q}^{\,1}$ at previous times can be discarded. This application minimizes the latency and the computations at the expense of the accuracy.

- *Lagged smoothing*, or *lagged filtering.* A few points of data are stored and processed before returning a result. Algorithm 2.4 details this procedure, which involves computing $\pi\left(x^t | g^{1:t+k}\right)$ for some $k > 0$. A trade-off is being made between the latency and the accuracy, as the information from the points $g^{t+1:t+k}$ is used to update the estimate of the state x^t. As shown in Section 2.6, even for small values of k (2 or 3), such a procedure can bring significant improvements in the accuracy while keeping the latency within reasonable bounds. A common ambiguity solved by lagged smoothing is presented in Figure 2.3.4.

2.3.4 Observation model

We now describe the observation model ω. The observation probability is assumed only to depend on the distance between the point and the GPS measurements. We take an isoradial

Algorithm 2.2 Description of backward recursion

Given a sequence of observations $g^{1:T}$, a sequence of sets of candidate projections $\mathbf{x}^{1:T}$ and a sequence of sets of candidate paths $\mathbf{p}^{1:T-1}$:
Initialize the backward state distribution
$\forall i = 1 \cdots I^T$: $\overleftarrow{q}_i^T \leftarrow 1$
For every time step t from $T-1$ to 1:
  Compute the forward probability over the paths:
  $\forall j = 1 \cdots J^t$:
    $\overleftarrow{r}_j^t \leftarrow \eta\left(p_j^t\right)\left(\sum_{j:\bar{\delta}\left(p_j^t, x_i^{t+1}\right)=1} \overleftarrow{q}_i^{t+1}\right)$
  Normalize $\overleftarrow{r}^t$
  Compute the forward probability over the states:
  $\forall i = 1 \cdots I^t$:
    $\overleftarrow{q}_i^t \leftarrow \omega\left(x_i^{t+1} | g^{t+1}\right)\left(\sum_{j:\underline{\delta}\left(x_i^t, p_j^t\right)=1} \overleftarrow{r}_j^t\right)$
  Normalize $\overleftarrow{q}^t$
Return the set of vectors $\left(\overleftarrow{q}^t\right)_t$ and $\left(\overleftarrow{r}^t\right)_t$

Algorithm 2.3 Trajectory smoothing algorithm

Given a sequence of observations $g^{1:T}$, a sequence of sets of candidate projections $\mathbf{x}^{1:T}$ and a sequence of sets of candidate paths $\mathbf{p}^{1:T-1}$:
Compute $\left(\overrightarrow{q}^t\right)_t$ and $\left(\overrightarrow{r}^t\right)_t$ using the forward algorithm.
Compute $\left(\overleftarrow{q}^t\right)_t$ and $\left(\overleftarrow{r}^t\right)_t$ using the backward algorithm.
For every time step t:
  $\forall j = 1 \cdots J^t : \overline{r}_j^t \leftarrow \overrightarrow{r}_j^t \cdot \overleftarrow{r}_j^t$
  Normalize $\overline{r}^t$
  $\forall i = 1 \cdots I^t$: $\overline{q}_i^t \leftarrow \overrightarrow{q}_i^t \cdot \overleftarrow{q}_i^t$
  Normalize $\overline{q}^t$
Return the set of vectors $\left(\overline{q}^t\right)_t$ and $\left(\overline{r}^t\right)_t$

Gaussian noise model:

$$\begin{aligned} \omega\left(g|x\right) &= p\left(\mathrm{d}\left(g,x\right)\right) \\ &= \frac{1}{\sqrt{2\pi}\sigma}\left(-\frac{1}{2\sigma^2}\mathrm{d}\left(g,x\right)^2\right) \end{aligned} \tag{2.3.12}$$

in which the function d is the distance function between geocoordinates. The standard deviation σ is assumed to be constant over all the network. This is not true in practice because of well documented urban canyoning effects [31, 109, 110] and satellite occlusions. Updating the model accordingly presents no fundamental difficulty, and can be done by geographical clustering of the regions of interest. Using the estimation techniques described

Algorithm 2.4 Lagged smoothing algorithm

Given an integer $k > 0$, and a LIFO queue of observations:
Initialize the queue to the empty queue.
When receiving a new observation g^t:
  Push the observation in the queue
  Run the forward filter on this observation
  If $t > k$:
    Run the backward filter on the queue
    Compute $\overline{q}^{t-k}$, $\overline{r}^{t-k}$ on the first element of the queue
    Pop the queue and return $\overline{q}^{t-k}$ and $\overline{r}^{t-k}$

later in Section 2.4.3 and Section 2.4.4, an value of σ between 10 and 15 meters could be estimated for data of interest in this dissertation.

2.3.5 Driver model

The second model to consider is the driver behavior model. This model assigns a weight to any acceptable path on the road network. We consider a model in the *exponential family*, in which the weight distribution over any path p only depends on a selected number of features $\varphi(p) \in \mathbb{R}^K$ of the path. Possible features include the length of the path, the number of stop signs, and the speed limits on the road. If the distribution is parametrized by a vector $\mu \in \mathbb{R}^K$ so that the logarithm of the distribution of paths is a linear combination of the features of the path:

$$\eta(p) \propto \exp\left(\mu^T \varphi(p)\right) \tag{2.3.13}$$

The function φ is called the *feature function,* and the vector μ is called the behavioral *parameter vector*, and simply encodes a weighted combination of the features.

In a simple model the vector $\varphi(p)$ may be reduced to a single scalar, such as the length of the path. Then the inverse of μ, a length, can be interpreted as a characteristic length. This model simply states that the driver has a preference for shorter paths, and μ^{-1} indicates how aggressively this driver wants to follow the shortest path. Such a model is explored in Section (2.6). Other models considered include the mean speed and travel times, the stop signs and signals, and the turns to the right or to the left.

In the *Mobile Millennium* system, the path inference filter is the input to a model designed to learn travel times, so the feature function does not include dynamic features such as the current travel time. Assuming this information is available, it would be easy to add as a feature.

2.4 Training procedure

The procedure detailed so far requires the calibration of the observation model and the driver behavior model by setting some values for the weight vector μ and the standard deviation σ. Using standard machine learning techniques, we maximize the likelihood of the observations with respect to the parameters, and we evaluate the result against held-out trajectories using several metrics detailed in Section 2.6. Computing likelihood will require the computation of the partition function (which depends on μ and σ). We first present a procedure that is valid for any path or point distributions that belong to the *exponential family*, and then show how the models we presented in Section 2.3 fit into this framework.

2.4.1 Learning within the exponential family and sparse trajectories

There is a striking similarity between the state variables $x^{1:T}$ and the path variables $p^{1:T}$ - especially between the forward and backward distributions introduced in Equation (2.3.8). This suggests to generalize our procedure to a context larger than states interleaved with paths. Indeed, each step of choosing a path or a variable corresponds to making a *choice* between a finite number of possibilities, and there is a limited number of pairwise compatible choices (as encoded by the functions $\underline{\delta}$ and $\bar{\delta}$). Following a trajectory corresponds to choosing a new state (subject to the compatibility constraints of the previous state). In this section, we introduce the proper notation to generalize our learning problem, and then show how this learning problem can be efficiently solved. In the next section, we will describe the relation between the new variables we are going to introduce and the parameters of our model.

Consider a joint sequence of multinomial random variables $\mathbf{z}^{1:L} = \mathbf{z}^1 \cdots \mathbf{z}^L$ drawn from some space $\prod_{l=1}^{L} \left\{1 \cdots K^l\right\}$ where K^l is the dimensionality of the multinomial variable $\mathbf{z}^l$. Given a realization $z^{1:L}$ from $\mathbf{z}^{1:L}$, we define a non-negative potential function $\psi\left(z^{1:L}\right)$ over the sequence of variables. This potential function is controlled by a parameter vector $\theta \in \mathbb{R}^M$: $\psi\left(z^{1:L}\right) = \psi\left(z^{1:L}; \theta\right)$ [7]. Furthermore, we assume that this potential function is also defined and non-negative over any subsequence $\psi\left(z^{1:l}\right)$. Lastly, we assume that there exists at least one sequence $z^{1:L}$ that has a positive potential. As in the previous section, the potential function ψ, when properly normalized, defines a probability distribution of density π over the variables $\mathbf{z}$, and this distribution is parametrized by the vector θ:

$$\pi\left(z; \theta\right) = \frac{\psi\left(z; \theta\right)}{Z\left(\theta\right)} \tag{2.4.1}$$

with $Z = \sum_z \psi\left(z; \theta\right)$ called the *partition function*. We will show the partition function defined here is the partition function introduced in Section 2.3.3.

[7] The semicolon notation indicates that this function is parametrized by θ, but that θ is not a random variable.

We assume that ψ is an unscaled member of the *exponential family*: it is of the form:

$$\psi(z;\theta) = h(z) \prod_{l=1}^{L} e^{\theta \cdot T^l\left(z^l\right)} \tag{2.4.2}$$

In this representation, h is a non-negative function of z which does not depend on the parameters, the operator $\cdot$ is the vector dot product, and the vectors $T^l\left(z^l\right)$ are vector mappings from the realization z^l to $\mathbb{R}^M$ for some $M \in \mathbb{N}$, called *feature vectors*. Since the variable z^l is discrete and takes on values in $\left\{1 \cdots K^l\right\}$, it is convenient to have a specific notation for the feature vector associated with each value of this variable:

$$\forall i \in \left\{1 \cdots K^l\right\}, T_i^l = T^l\left(z^l = i\right)$$

The sequence of variables $\mathbf{z}$ represents the choices associated with a single trajectory, i.e. the concatenation of the xs and ps. In general, we will observe and would like to learn from multiple trajectories at the same time. This is why we need to consider a collection of variables $\left(\mathbf{z}^{(u)}\right)_u$, each of which follows the form above and each of which we can define a potential $\psi\left(z^{(u)};\theta\right)$ and a partition function $Z^{(u)}(\theta)$ for. There the variable u indexes the set of sequences of observations, i.e. the set of consecutive GPS measurements of a vehicle. Since each of these trajectories will take place on a different portion of the road network, each of the sequences $\mathbf{z}^{(u)}$ will have a different state space. For each of these sequences of variables $\mathbf{z}^{(u)}$, we observe the respective realizations $z^{(u)}$ (which correspond to the observation of a trajectory), and we wish to infer the parameter vector θ^* that maximizes the likelihood of all the realizations of the trajectories:

$$\begin{aligned}
\theta^* &= \arg\max_\theta \sum_u \log \pi^{(u)}\left(z^{(u)};\theta\right) \\
&= \arg\max_\theta \sum_u \log \psi\left(z^{(u)};\theta\right) - \log Z^{(u)}(\theta) \\
&= \arg\max_\theta \sum_u \sum_{l=1}^{L^{(u)}} \theta \cdot T^{l^{(u)}}\left(z^{l^{(u)}}\right) - \log Z^{(u)}(\theta)
\end{aligned} \tag{2.4.3}$$

where again the indexing u is for sets of measurements of a given trajectory. Similarly, the length of a trajectory is indexed by u: $L^{(u)}$. From Equation 2.4.3, it is clear that the log-likelihood function simply sums together the respective likelihood functions of each trajectory. For clarity, we consider a single sequence $z^{(u)}$ only and we remove the indexing with respect to u. With this simplification, we have for a single trajectory:

$$\log \psi(z;\theta) - \log Z(\theta) = \sum_{l=1}^{L} \theta \cdot T^l\left(z^l\right) - \log Z(\theta) \tag{2.4.4}$$

The first part of Equation (2.4.4) is linear with respect to θ and $\log Z(\theta)$ is concave in θ (it is the logarithm of a sum of exponentiated linear combinations of θ [26]) . As such,

maximizing Equation (2.4.4) yields a unique solution (assuming no singular parametrization), and some superlinear algorithms exist to solve this equation [26]. These algorithms rely on the computation of the gradient and the Hessian matrix of $\log Z(\theta)$. We now detail some closed-form recursive formulas to compute these elements.

2.4.1.1 Efficient estimation of the partition function

A naive approach to the computation of the partition function $Z(\theta)$ leads to consider exponentially many paths. Most of these computations can be factored using dynamic programming [8]. Recall the definition of the partition function:

$$Z(\theta) = \sum_{z} h(z) \prod_{l=1}^{L} e^{\theta \cdot T^l\left(z^l\right)}$$

So far, the function h was defined in in a generic way (it is non-negative and does not depend on θ). We consider a particular shape that generalizes the functions $\underline{\delta}$ and $\overline{\delta}$ introduced in the previous section. In particular, the function h is assumed to be a binary function, from the Cartesian space $\prod_{l=1}^{L}\left\{1 \cdots K^l\right\}$ to $\{0,1\}$, that decomposes to the product of binary functions over consecutive pairs of variables:

$$h(z) = \prod_{l=1}^{L-1} h^l\left(z^l, z^{l-1}\right)$$

in which every function h^l is a binary indicator $h^l : \left\{1 \cdots K^l\right\} \times \left\{1 \cdots K^{l-1}\right\} \to \{0,1\}$. These functions h^l generalize the functions $\underline{\delta}$ and $\overline{\delta}$ for arguments z equal to either the xs or the ps. It indicates the compatibility of the values of the instantiations z^l and z^{l-1}

Finally, we introduce the following notation. For each index $l \in [1 \cdots L]$ and subindex $i \in [1 \cdots K^l]$, we call Z_i^l the partial summation of all partial paths $\mathbf{z}^{1:l}$ that terminate at the value $z^l = i$:

$$\begin{aligned} Z_i^l(\theta) &= \sum_{z^{1:l}; z^l = i} h\left(z^{1:l}\right) \prod_{m=1}^{l} e^{\theta \cdot T^m\left(z^m\right)} \\ &= \sum_{z^{1:l}; z^l = i} e^{\theta \cdot T^1\left(z^1\right)} \prod_{m=2}^{l} h^m\left(z^m, z^{m-1}\right) e^{\theta \cdot T^m\left(z^m\right)} \end{aligned}$$

This partial summation can also be defined recursively:

$$Z_i^l(\theta) = e^{\theta \cdot T_i^l} \sum_{j \in \left[1 \ldots K^{l-1}\right]; h^l\left(z^i, z^j\right) = 1} Z_j^{l-1}(\theta) \tag{2.4.5}$$

[8] This is - again - a specific application of the junction tree algorithm. See [87] for an explanation of the general framework.

The start of the recursion is for all $i \in \{1 \cdots K^1\}$:

$$Z_i^1(\theta) = e^{\theta \cdot T_i^1}$$

and the complete partition function is the summation of the auxiliary values:

$$Z(\theta) = \sum_{i=1}^{K^L} Z_i^L(\theta)$$

Computing the partition function can be done in polynomial time complexity by a simple application of dynamic programming. By using sparse data structures to implement h, some additional savings in computations can be made[9].

2.4.1.2 Estimation of the gradient

The estimation of the gradient for the first part of the log likelihood function is straightforward. The gradient of the partition function can also be computed using Equation (2.4.5):

$$\nabla_\theta Z_i^l(\theta) = Z_i^l(\theta) T_i^l + e^{\theta \cdot T_i^l} \sum_{j:h^l(z^i,z^j)=1} \nabla_\theta Z_j^{l-1}(\theta)$$

The Hessian matrix can be evaluated in similar fashion:

$$\begin{aligned}
\triangle_{\theta\theta} Z_i^l(\theta) =& Z_i^l(\theta) \left(T_i^l\right) \left(T_i^l\right)' \\
&+ e^{\theta \cdot T_i^l} \left(\sum_{j:h^l(z^i,z^j)=1} \nabla_\theta Z_j^{l-1}(\theta) \right) \left(T_i^l\right)' \\
&+ e^{\theta \cdot T_i^l} \left(T_i^l\right) \left(\sum_{j:h^l(z^i,z^j)=1} \nabla_\theta Z_j^{l-1}(\theta) \right)' \\
&+ e^{\theta \cdot T_i^l} \sum_{j:h^l(z^i,z^j)=1} \triangle_{\theta\theta} Z_j^{l-1}(\theta)
\end{aligned}$$

2.4.2 Exponential family models

We now express our formulation of Conditional Random Fields in a form compatible with Equation (2.4.2).

[9] In particular, care should be taken to implement all the relevant computations in log-domain due to the limited precision of floating point arithmetic on computers. The reference implementation [11] shows one way to do it.

Consider $\epsilon = \sigma^{-2}$ and θ the stacked vector of the desired parameters:

$$\theta = \begin{pmatrix} \epsilon \\ \mu \end{pmatrix}$$

There is a direct correspondence between the path and state variables with the $\mathbf{z}$ variables introduced above. Let us pose $L = 2T - 1$, then for all $l \in [1, L]$ we have:

$$z^{2t} = r^t$$

$$z^{2t-1} = q^t$$

and the feature vectors are simply the alternating values of φ and d, completed by some zero values:

$$T_i^{2t} = \begin{pmatrix} 0 \\ \varphi\left(p_i^t\right) \end{pmatrix}$$

$$T_j^{2t-1} = \begin{pmatrix} -\frac{1}{2}\mathrm{d}\left(g, x_j^t\right)^2 \\ \mathbf{0} \end{pmatrix}$$

These formulas establish how we can transform our learning problem that involves paths and states into a more abstract problem that considers a single set of variables.

2.4.3 Supervised learning with known trajectories

The most straightforward way to learn μ and σ, or equivalently to learn the joint vector θ, is to maximize the likelihood of some GPS observations $g^{1:T}$, knowing the complete trajectory followed by the vehicle. For all time t, we also know which path p^t_{observed} was taken and which state x^t_{observed} produced the GPS observation g^t. We make the assumption that the observed path p^t_{observed} is one of the possible path amongst the set of candidate paths $\left(p_j^t\right)_j$:

$$\exists j \in \left[1 \cdots J^t\right] \ : \ p^t_{\text{observed}} = p^t_j$$

and similarly, that the observed state x^t_{observed} is one of the possible states:

$$\exists i \in \left[1 \cdots I^t\right] \ : \ x^t_{\text{observed}} = x^t_i$$

In this case, the values of r^t and q^t are known (they are the matching indexes), and the optimization problem of Equation (2.4.3) can be solved using methods outlined in Section 2.4.1.

2.4.4 Unsupervised learning with incomplete observations: Expectation-Maximization

Usually, only the GPS observations $g^{1:T}$ are available; the values of $r^{1:T-1}$ and $q^{1:T}$ (and thus $z^{1:L}$) are hidden to us. In this case, we estimate the *expected likelihood* $\mathcal{L}$, which is the expected value of the likelihood under the distribution over the assignment variables $\mathbf{z}^{1:L}$:

$$\begin{aligned}\mathcal{L}(\theta) &= \mathbb{E}_{z\sim\pi(\cdot|\theta)}\left[\log\left(\pi\left(z;\theta\right)\right)\right] &(2.4.6)\\ &= \sum_{z}\pi\left(z;\theta\right)\log\left(\pi\left(z;\theta\right)\right) &(2.4.7)\end{aligned}$$

The intuition behind this expression is quite natural: since we do not know the value of the assignment variable z, we consider the *expectation* of the likelihood over this variable. This expectation is done with respect to the distribution $\pi\left(z;\theta\right)$. The challenge lies in the dependency in θ of the very distribution used to take the expectation. Computing the expected likelihood becomes much more complicated than simply solving the optimization problem described in Equation (2.4.3).

One strategy is to find some "fill in" values for z that would correspond to our guesses of which path was taken, and which point made the observation. However, such a guess would likely involve our model for the data, which we are currently trying to learn. A solution to this chicken and egg problem is the Expectation Maximization (EM) algorithm [85]. This algorithm performs an iterative projection ascent by assigning some *distributions* (rather than singular values) to every z^l, and uses these distributions to updates the parameters μ and σ using the procedures seen in Section 2.4.3. This iterative procedure performs two steps:

1. Fixing some value for θ, it computes a distribution $\tilde{\pi}\left(z\right)=\pi\left(z;\theta\right)$
2. It then uses this distribution $\tilde{\pi}\left(z\right)$ to compute some new value of θ by solving the approximate problem in which the expectation is fixed with respect to θ:

$$\max_{\theta}\mathbb{E}_{z\sim\tilde{\pi}(\cdot)}\left[\log\left(\pi\left(z;\theta\right)\right)\right] \tag{2.4.8}$$

   This problem is significantly simpler than the optimization problem in Equation (2.4.6) since the expectation itself does not depend on θ and thus is not part of the optimization problem.

An algorithmic description of this algorithm is given in Algorithm 2.5. Under this procedure, the expected likelihood is shown to converge to a local maximum [87]. It can be shown that good values for the plug-in distribution $\tilde{\pi}$ are simply the values of the posterior distributions $\pi\left(p^t|g^{1:T}\right)$ and $\pi\left(x^t|g^{1:T}\right)$, i.e. the values $\overline{q}^t$ and $\bar{r}^t$. Furthermore, owing to the particular shape of the distribution $\pi\left(z\right)$, taking the expectation is a simple task: we simply replace the value of the feature vector by its *expected value* under the distribution $\tilde{\pi}\left(z\right)$. More practically, we simply have to consider:

$$\begin{aligned}T^{2t}\left(z^{2t}\right) &= \mathbb{E}_{p\sim\pi(\cdot|\theta,g^{1:T})}\left[\begin{pmatrix}0\\ \varphi\left(p_r^t\right)\end{pmatrix}\right]\\ &= \begin{pmatrix}0\\ \mathbb{E}_{p\sim\pi(\theta,g^{1:T})}\left[\varphi\left(p_r^t\right)\right]\end{pmatrix}\end{aligned} \tag{2.4.9}$$

Algorithm 2.5 Expectation maximization algorithm for learning parameters without complete observations.

Given a set of sequences of observations, an initial value of θ
Repeat until convergence:
For each sequence, compute $\overline{r}^t$ and $\overline{q}^t$ using Algorithm 2.3.
For each sequence, update expected values of T^t using (2.4.9) and (2.4.10).
Compute a solution of Problem (2.4.3) using these new values of T^t.

in which

$$\mathbb{E}_{p\sim\pi(\theta,g^{1:T})}\left[\varphi\left(p_r^t\right)\right] = \sum_{i=1}^{I^t} \overline{r}_i^t \varphi_i^t$$

and

$$T^{2t-1}\left(z^{2t-1}\right) = \begin{pmatrix} -\frac{1}{2}\mathbb{E}_{x\sim\pi(\cdot|\theta,g^{1:T})}\left[\mathrm{d}\left(g, x_{q^t}^t\right)^2\right] \\ \mathbf{0} \end{pmatrix} \tag{2.4.10}$$

so that

$$\mathbb{E}_{x\sim\pi(\cdot|\theta,g^{1:T})}\left[\mathrm{d}\left(g, x_{q^t}^t\right)^2\right] = \sum_{i=1}^{J^t} \bar{q}_i^t \mathrm{d}\left(g, x_i^t\right)^2$$

These values of feature vectors plug directly into the supervised learning problem in Equation (2.4.3) and produce updated parameters μ and σ, which are then used in turn for updating the values of $\bar{q}$ and $\bar{r}$ and so on.

2.5 Results from field operational test

The path inference filter and its learning procedures were tested using field data through the *Mobile Millennium* system. Ten San Francisco taxicabs were fit with high frequency GPS (1 second sampling rate) in October 2010 during a two-day experiment. Together, they collected about seven hundred thousand measurement points that provided a high-accuracy ground truth. Additionally, the unsupervised learning filtering was tested on a significantly larger dataset: one day one-minute samples of 600 taxis from the same fleet, which represents 600 000 points. For technical reasons, the two datasets could not be collected the same day, but were collected the same day of the week (a Wednesday) three weeks prior to the high-frequency collection campaign. Even if the GPS equipment was different, both datasets presented the same distribution of GPS dispersion. Thus we evaluate two datasets collected from the same source with the same spatial features: a smaller set at high frequency, called "Dataset 1", and a larger dataset sampled at 1 minute for which we do not know ground truth, called "Dataset 2". The map of the road network is a highly accurate map provided by a commercial vendor. It covers the complete Bay Area and comprises about 560,000 road links.

Before we continue this discussion further, now is an appropriate time to digress and reward the patient reader with a tasty distraction. Macarons are delicious almond cookies filled with ganache, and are generally considered to be an extraordinary *tour de force* to make. This is not the case. With the proper tools, one can make some delicious macarons in any home kitchen, after knowing a few tips. Let me share with you a recipe that works well in my case. It is adapted from BraveTart's blog with a few changes. There are three steps in baking a macaron: raising the egg whites into a meringue, folding the dry part into the meringue (the "macaronage"), and the baking. Here are the ingredients you need: 140 grams of egg whites, 70 grams of regular fine sugar, 230 grams of powdered sugar, 115 grams of almond powder. The egg whites need not be chilled for three days, at room temperature, et caetera, et caetera. These details do not make any difference. In the United States all the powdered sugars are adulterated with some starch (corn or tapioca). It has a slight impact on the final product, but you will be fine otherwise. The almond powder should be sieved. I never do it because cleaning a sieve is a pain, and it works fine. However, quantities are crucial. Do not roughly take four eggs, 140 grams is 140 grams. I usually break three eggs and then readjust all the other quantities based on the amount of egg whites I got. The meringue is the one step you cannot miss if you have a beater or a kitchen robot. Just make an incredibly stiff meringue, no need to try to look for the "bec d'oiseau" and all these professional tips. Put the sugar with the egg whites and then beat three minutes at 50% speed, three minutes at 75% speed, three minutes at 100% speed, and one more minute at 100% speed if you add some dying powder. No need to be shy here. The most crucial step is the macaronage. Look at this video to see how to do it. The gesture is very important here. One departure from BraveTart's recipe is that I found the croûtage to make a big difference when rising the macarons: once you have poured the macarons on a cooking plate covered with wax paper, let them rest out of the kitchen for 30 or 45 minutes. The outer layer will form a crust. The baking should take fifteen minutes in the oven at 300F, and then let it cook until the shell is hardly attached to the inside (it does not move if you press a finger on it). You can open the oven during cooking, it is not an issue.

2.5.1 Experiment design

The testing procedure is described in Algorithm 2.6: the filter was first run in trajectory reconstruction mode (Viterbi algorithm) with settings and tuned for a high-frequency application, using all the samples, in order to build a set of ground truth trajectories. The trajectories were then downsampled to different temporal resolutions and were used to test the filter in different configurations. The following features were tested:

- The sampling rate. The following values were tested: 1 second, 10 seconds, 30 seconds, one minute, one and a half minute and two minutes
- The computing strategy: pure filtering ("online" or forward filtering), fixed-lagged smoothing with a one- or two-point buffer ("1-lag" and "2-lag" strategies), Viterbi and smoothing ("offline", or forward-backward procedure).

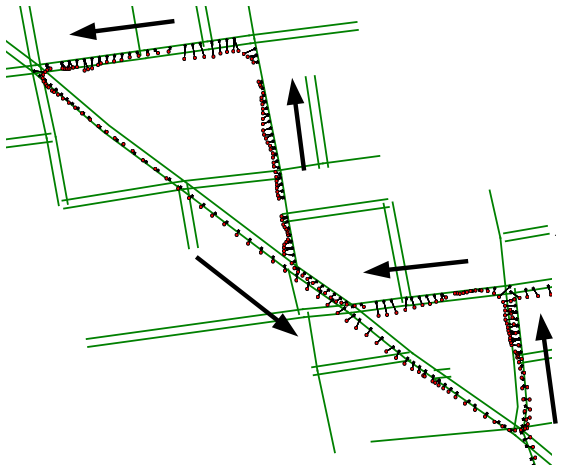


Figure 2.5.1: Example of points collected in "Dataset 1", in the Russian Hill neighborhood in San Francisco. The (red) dots are the GPS observations (collected every second), and the green lines are road links that contain a state projection. The black lines show the most likely projections of the GPS points on the road network, using the Viterbi algorithm on a gridded state-space with a 1-meter grid for the offsets.

Algorithm 2.6 Evaluation procedure

Given a set of high-frequency sequences of raw GPS data:

1. Map the raw high-frequency sequences on the road network
2. Run the Viterbi algorithm with default settings
3. Extract the most likely high frequency trajectory on the road network for each sequence
4. Given a set of projected high frequency trajectories:
   a) Decimate the trajectories to a given sampling rate
   b) Separate the set into a training subset and a test subset
   c) Compute the best model parameters for a number of learning methods (most likely, EM with a simple model or a more complex model)
   d) Evaluate the model parameters with respect to different computing strategies (Viterbi, online, offline, lagged smoothing) on the test subset

- Different models:
  - "Hard closest point": A greedy deterministic model that computes the closest point and then finds the shortest path to reach this closest point from the previous point. This non-probabilistic model is the baseline which we compare against [45]. This greedy model may lead to non-feasible trajectories, for example by assigning an observation to a dead end link from which it cannot recover.
  - "Closest point" : A non-greedy version of "Hard closest point". Among all the feasible trajectories, this (naive, deterministic) model projects all the GPS data to their closest projections and then selects the shortest path between each projection. The computing strategy chosen is important because the filter may determine that some projections lead to dead end and force the trajectory to break.
  - "Shortest path": A naive model that selects the shortest path. Given paths of the same length, it will take the path leading to the closest point. The points projections are then recovered from the paths. This is similar to the approach followed in [44, 119].
  - "Simple" A simple model that considers two features that could be tuned by hand:
    1. ξ_1 : The length of the path
    2. ξ_2 : The distance of a point projection to its GPS coordinate

    This model was trained on learning data by two procedures:
    - Supervised learning, in which the true trajectory is provided to the learning algorithm, leading to the "MaxLL-Simple" model
    - Unsupervised learning, which produced the model called "EM-Simple"
  - "Complex" : A more complex model with a more diverse set of features, which is complicated enough to discourage manual tuning:
    1. Length of the path
    2. Number of stop signs along the path
    3. Number of signals (red lights)
    4. Number of left turns made by the vehicle at road intersections
    5. Number of right turns made by the vehicle at road intersections
    6. Minimum average travel time (based on the speed limit)
    7. Maximum average speed
    8. Maximum number of lanes (representative of the class of the road)
    9. Minimum number of lanes
    10. Distance of a point to its GPS point

    This model was first evaluated using supervised learning leading to the model called "MaxLL-Complex". The unsupervised learning procedure was also tried

but failed to properly converge when using "Dataset 1", obtained from high-frequency samples: since this dataset is quite small, the EM procedure was able to find an unbounded maximum and unbounded parameters. Unsupervised learning was run again with "Dataset 2", using the simple model as a start point and converged properly this time. This set of parameters is presented under the label "EM-Complex".

All the models above are specific cases of our framework:

- "Simple" is a specific case of "Complex", by restricting the complex model to only two features.
- "Shortest path" is a specific case of "Simple" with $|\xi_1| \gg 1$, $|\xi_2| \ll 1$. We used $\xi_1 = -1000$ and $\xi_2 = -0.001$
- "Closest point" is a specific case of "Simple" with $|\xi_1| \ll 1$, $|\xi_2| \gg 1$. We used $\xi_1 = -0.001$ and $\xi_2 = -1000$
- "Hard closest point" can be reasonably approximated by running the "Closest point" model with the Online filtering strategy.

Thanks to this observation, we implemented all the model using the same code and simply changed the set of features and the parameters [11].

These models were evaluated under a number of metrics:

- The proportion of path misses: for each trajectory, it is the number of times the most likely path was not the true path followed, divided by the number of time steps in the trajectory.
- The proportion of state misses: for each trajectory, the number of times the most likely projection was not the true projection.
- The log-likelihood of the true point projection. This is indicative of how often the true point is identified by the model.
- The log-likelihood of the true path.
- The entropy of the path distribution and of the point distribution. This statistical measure indicates the confidence assigned by the filter to its result. A small entropy (close to 0) indicates that one path is strongly favored by the filter against all the other ones, whereas a large entropy indicates that all paths are equal.
- The miscoverage of the route. Given two paths p and p' the coverage of p by p', denoted $\mathrm{cov}\,(p, p')$ is the amount of length of p that is shared with p' (it is a semi-distance since it is not symmetric). It is thus lower than the total length $|p|$ of the path p. We measure the dissimilarity of two paths by the *relative miscoverage*: $\mathrm{mc}\,(p) = 1 - \frac{\mathrm{cov}(p^*, p)}{|p^*|}$. If a path is perfectly covered, its relative miscoverage will be 0.

Sampling rate (seconds)	Batches used for validation	Batches used for training
1	1	5
10	3	5
30	6	5
60	6	5
90	6	5
120	6	5

Table 2.1: Parameters used for the Path Inference experiments.

For about 0.06% of pairs of points, the true path could not be found by the A* algorithm and was manually added to the set of discovered paths.

Each training session was evaluated with k-fold cross-validation, using the following parameters:

2.5.2 Results

Given the number of parameters to adjust, we only present the most salient results here.

The most important practical result is the raw accuracy of the filter: for each trajectory, which proportion of the paths or of the points was correctly identified? These results are presented in Figure 2.5.2 and Figure 2.5.3. As expected, the error rate is 0 for high frequencies (low sampling period): all the points are correctly identified by all the algorithms. In the low frequencies (high sampling periods), the error is still low (around 10%) for the trained models, and also for the greedy model ("Hard closest point"). For sampling rates between 10 seconds and 90 seconds, trained models ("Simple" and "Complex") show a much higher performance compared to untrained models ("Hard closest point", "Closest point" and "Shortest path").

We now turn our attention to the resilience of the models, i.e. how they perform when they make mistakes. We use two statistical measures: the (log) likelihood of the true paths (Figure 2.5.4) and the entropy of the distribution of points or paths (Figures 2.5.5 and 2.5.6). Note that in a perfect reconstruction with no ambiguity, the log likelihood would be zero. Interestingly, the log likelihoods appear very stable as the sampling interval grows: our algorithm will continue to assign high probabilities to the true projections even when many more paths can be used to travel from one point to the other. The performance of the simple and the complex models improves greatly when some backward filtering steps are used, and stays relatively even across different time intervals.

We conclude the performance analysis by a discussion of the miscoverage (Figure 2.5.7). The miscoverage gives a good indication of how far the path chosen by the filter differs from

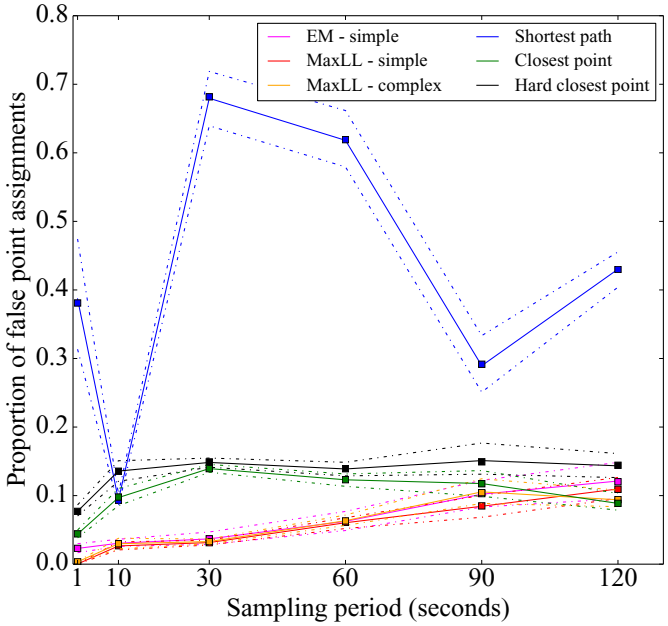


Figure 2.5.2: Point misses using trajectory reconstruction (Viterbi algorithm) for different sampling rates, as a percentage of incorrect point reconstructions for each trajectory (positive, smaller is better). The solid line denotes the median, the squares denote the mean and the dashed lines denote the 94% confidence interval. The black curve is the performance of a greedy reconstruction algorithm, and the colored plots are the performances of probabilistic algorithms for different features and weights learned by different methods. As expected, the error rate is close to 0 for high frequencies (low sampling rates): all the points are correctly identified by all the algorithms - except for the shortest path reconstruction, which greedily chooses the nearest reachable projection. In the low frequencies (high sampling rates), the error still stays low (around 10%) for the probabilistic models, and also for the greedy model. For sampling rates between 10 seconds and 90 seconds, tuned models show a much higher performance compared to greedy models (Hard closest point, closest point and shortest path). However, we will see that the errors made by tuned models are more benign than errors made by simple greedy models.

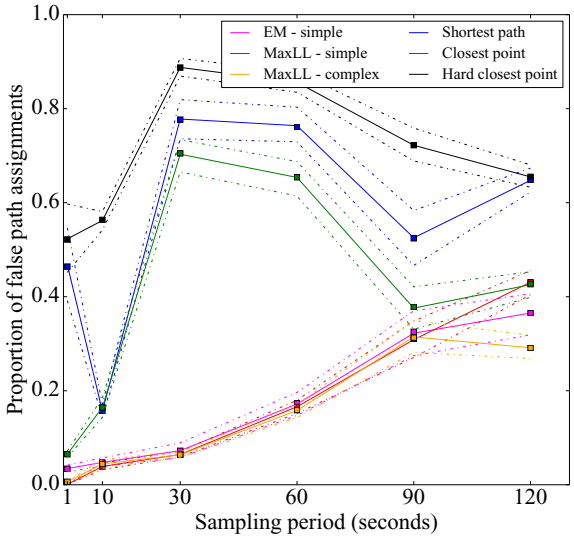


Figure 2.5.3: Path misses using the Viterbi reconstruction for different models and different sampling rates, as a percentage on each trajectory (lower is better). The solid line denotes the median, the squares denote the mean and the dashed lines denote the 98% percentiles. The error rate is close to 0 for high frequencies: the paths are correctly identified. In higher sampling regions, there are many more paths to consider and the error increases substantially. Nevertheless, the probabilistic models still perform very well: even at 2 minute intervals, they are able to recover about 75% of the true paths. In particular, in these regions the shortest path becomes a viable choice for most paths. Note how the greedy path reconstruction fails rapidly as the sampling increases. Also note how the shortest path heuristic performs poorly.

the true path. Even if the paths are not exactly the same, some very similar path may get selected, that may differ by a turn around a block. Note that the metric is based on length covered. At high frequency however, the vehicle may be stopped and cover a length 0. This metric is thus less useful at high frequency. A more complex model improves the coverage by about 15% in smoothing. In high sampling resolution, the error is close to zero: the paths considered by the filter, even if they do not match perfectly, are very close to the true trajectory for lower frequencies. Two groups clearly emerge as far as computing strategies are concerned: the online/1-lag group (orange and red curves) and the 2-lag and offline group (green and blue curves). The relative miscoverage for the latter group is so low that more than half of the probability mass is at zero. A number of outliers still raise the curve of the last quartile as well as the mean, especially in the lower frequencies. The paths inferred by the filter are never dramatically different: at two minute time intervals (for which the paths are 1.7km on average), the returned path spans more than 80% of the true path on average. The use of a more complicated model decreases the mean miscoverage as well as all quartile metrics by more than 15%.

In the case of the complex model, the weights can provide some insight into the features

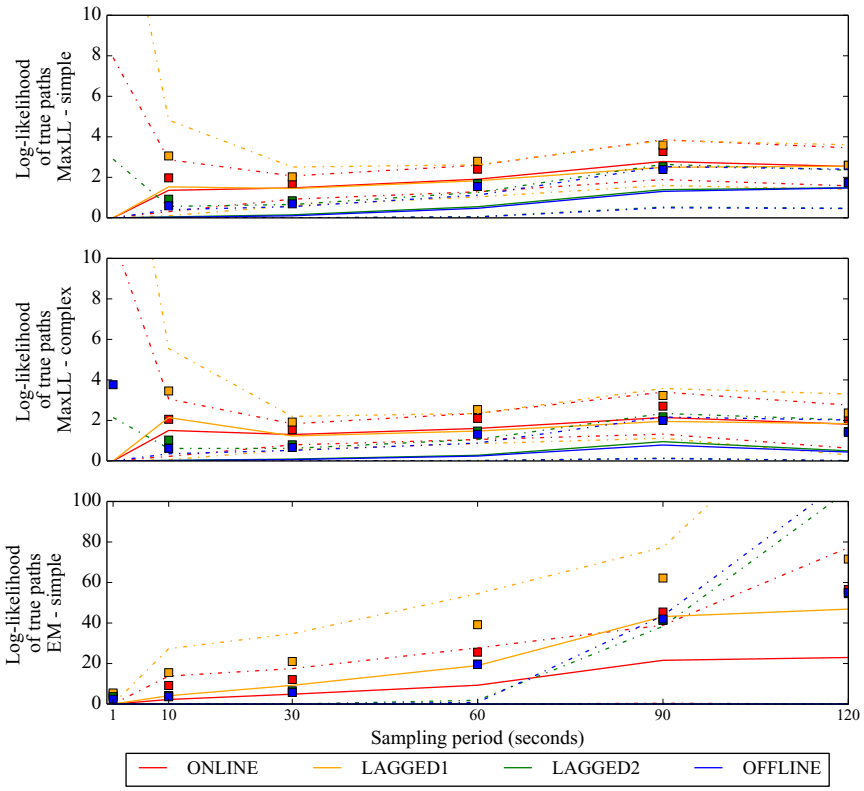


Figure 2.5.4: (Negative of) Log likelihood of true paths for different strategies and different sampling rates (positive, lower is better). The error bars denote the first and last quartiles (the 25th and 75th percentiles). The solid line denotes the median, the squares denote the mean and the dashed lines denote the 98% confidence interval. The likelihood decreases as the sampling interval increases, which was to be expected. Note the relatively high mean likelihood compared to the median: a number of true paths are assigned very low likelihood by the model, but this phenomenon is mitigated by using better filtering strategies (2-lagged and smoothing). The use of a more complex model (that accounts for a finer set of features for each path) brings some improvements on the order of 25% of all metrics. The behavior around high frequencies (1 and 10 second time intervals) is also very interesting. Most of the paths are chosen nearly perfectly (the median is 0), but the filters are generally too confident and assign very low probabilities to their outputs, which is why the likelihood has a very heavy tail at high frequency. Note also that in the case of high frequency, the use of an offline filter brings significantly more accurate results than a 2-lagged filter. This difference disappears rapidly (it becomes insignificant at 10 second intervals). Note how the EM trained filter performs worse in the low frequencies (note the difference of scale). The points for online strategy (red) and for 2-lagged filtering (green) do not appear because they are too close to the 1-lagged and offline strategies, respectively. Again in the EM setting, the offline and 2-lagged filters perform considerably better than the cruder strategies.

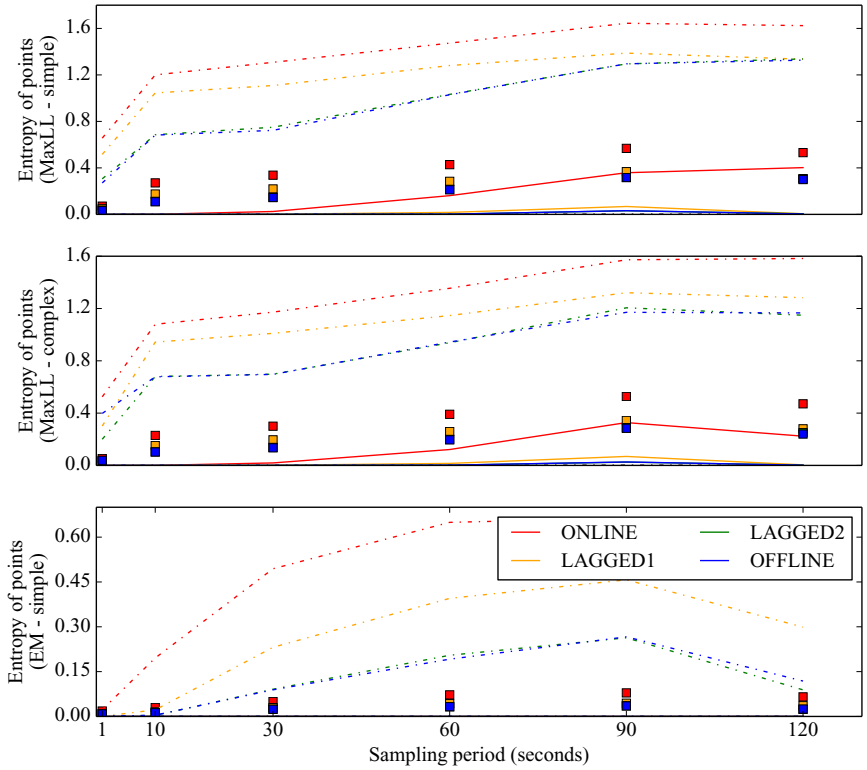


Figure 2.5.5: Distributions of point entropies with respect to sampling and for different models. The colors show the performance of different filtering strategies (pure online, 1-lag, 2-lag and offline). The entropy is a measure of the confidence of the filter on its output and quantifies the spread of the probability distribution over all the candidate points. The solid line denotes the median, the squares denote the mean and the dashed lines denote the 95% confidence interval. The entropy starts at nearly zero for high frequency sampling : the filters are very confident in their outputs. As sampling time increases, the entropy at the output of the online filter increases notably. Since the online filter cannot go back to update its belief, it is limited to pure forward prediction and as such cannot confidently choose a trajectory that would work in all settings. For the other filtering strategies, the median is close to zero while the mean is substantially higher. Indeed, the filter is very confident in its output most of the time and assigns a weight of nearly one to one candidate, and nearly zero to all the other outputs, but it is uncertain in a few cases. These few cases are at the origin of the fat tail of the distributions of entropies and the relatively wide confidence intervals. Note that using a more complex model improves the mean entropy by about 15%. Also, in the case of EM, the entropy is very low (note the difference of scale): the EM model is overconfident in its predictions and tends to assigns very large weights to a single choice, even if it not the good one.

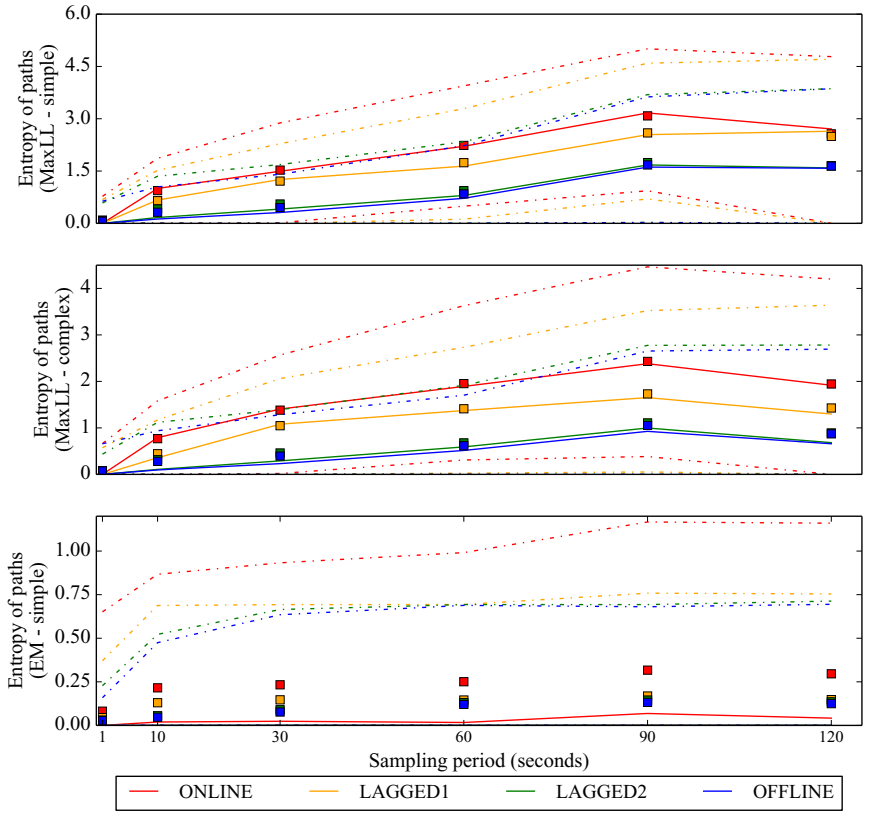


Figure 2.5.6: Distributions of path entropies with respect to sampling period and for different models (positive, lower is better). The colors show the performance of different filtering strategies (purely online, 1-lag, 2-lag and offline) The entropy is a measure of the confidence of the filter on its output and quantifies the spread of the probability distribution over all the candidate paths. The solid line denotes the median, the squares denote the mean and the dashed lines denote the 95% confidence interval. Compared to the points, the paths distributions have a higher entropy: the filter is much less confident in choosing a single path and spreads the probability weights across several choices. Again, the use of 2-lagged smoothing is as good as pure offline smoothing, for the same computing cost and a fraction of the data. Online and 1-lagged smoothing perform about as well, and definitely worse than 2-lagged smoothing. The use of a more complex model strongly improves the performance of the filter: it results in more compact distribution over candidate paths. Again, the model learned with EM is overconfident and tends to offer favor a single choice, except for a few path distributions.

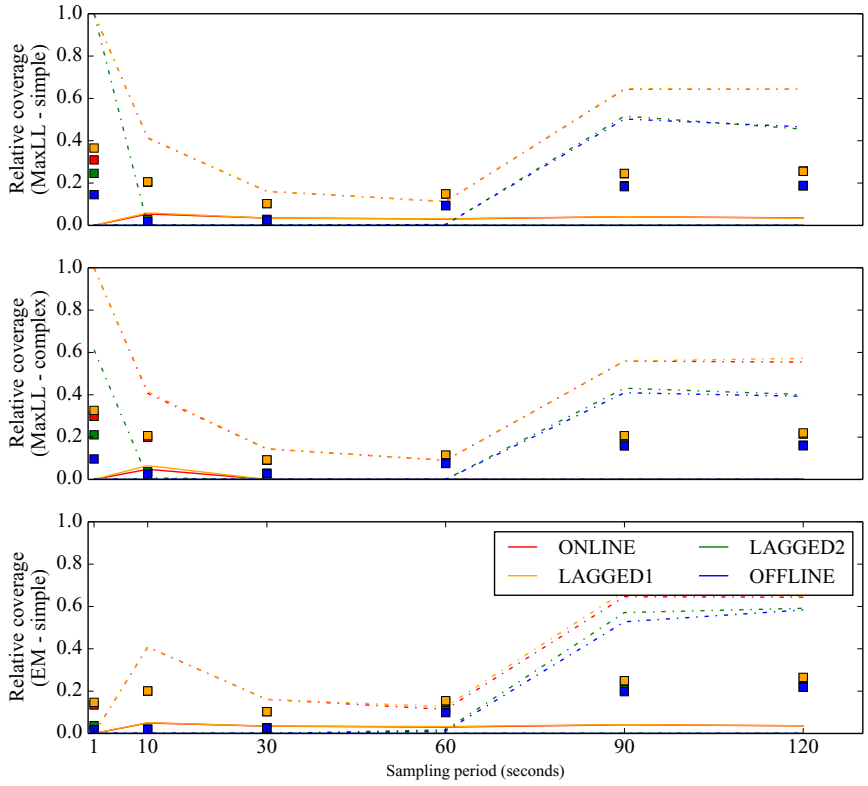


Figure 2.5.7: Distribution of relative miscoverage of the paths (between 0 and 1, lower is better). The solid line denotes the median, the squares denote the mean and the dashed lines denote the 98% confidence interval. This metric evaluates how much of the true path the most likely path covers , with respect to length (0 if it is completely different, 1 if the two paths overlap completely). Two groups clearly emerge as far as computing strategies are concerned: the online/1-lag group (orange and red curves) and the 2-lag and offline group (green and blue curves). The relative miscoverage for the latter group is so low that more than half of the mass is at the 0 and cannot be seen on the curve. There are still a number of outliers that raise the curve of the last quartile as well as the mean, especially in the lower frequencies. Note that the paths offered by the filter are never dramatically different: at two minute time intervals (for which the paths are 1.7km on average), the returned path spans more than 80% of the true path on average. The use of a more complicated model decreases the mean miscoverage as well as the quartile metrics by more than 15%. Note that there is a large spread of values at high frequency: indeed the metric is based on length covered and at high frequency, the vehicle may be stopped and cover 0 length. This metric is thus less indicative at high frequency.

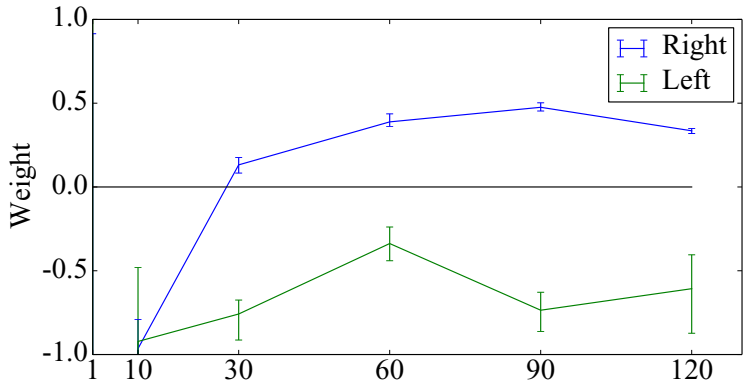


Figure 2.5.8: Learned weights for left or right turns preferences. The error bars indicate the complete span of values computed for each time (0th and 100th percentile). For small time intervals, any turning gets penalized but rapidly the model learns how to favor paths with right turns against paths with left turns. A positive weight even means that - all other factors being equal! - the driver would prefer turning on the right than going straight.

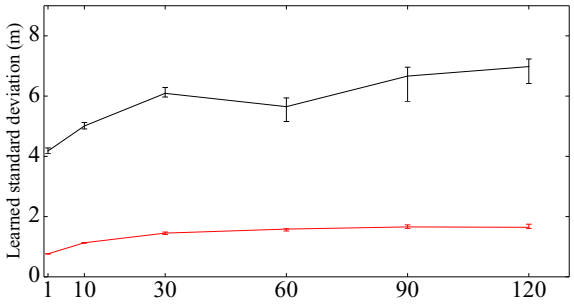


Figure 2.5.9: Standard deviation learned by the simple models, in the supervised (Maximum Likelihood) setting and the EM setting. The error bars indicate the complete span of values computed for each time. Note that the maximum likelihood estimator rapidly converges toward a fixed value of about 6 meters across any sampling time. The EM procedure also rapidly converges, but it is overconfident and assigns a lower standard deviation overall.

involved in the decision-making process of the driver. In particular, for extended sampling rates (t=120s), some interesting patterns appear. For example, the drivers do not show a preference between driving through stop signs ($w_3 = -0.24 \pm 0.07$) or through signals ($w_4 = -0.21 \pm 0.11$). However, drivers show a clear preference to turn to the right, as seen in Figure 2.5.8. This is may be attributed, in part, to the difficulty in crossing an intersection in the United States.

From a computation perspective, given a driver model, the filtering algorithm can be dramatically improved for about as much computations by using a full backward-forward (smoothing) filter. Smoothing requires backing up an arbitrary sequence of points while 2-

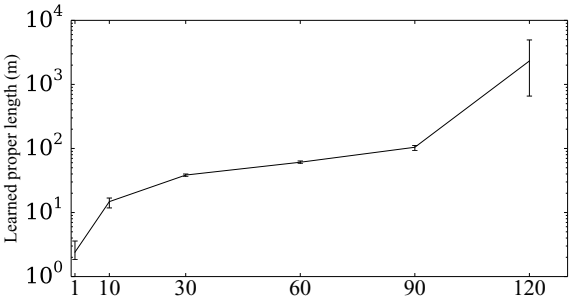


Figure 2.5.10: Characteristic length learned by the simple models, in the supervised (Maximum Likelihood) setting and the EM setting. As hoped, it roughly corresponds to the expected path length. The error bars indicate the complete span of values computed for each time (0th and 100th percentile). Note how the spread increases for large time intervals. Indeed, vehicles have different travel lengths at such time intervals, ranging from nearly 0 (when waiting at a signal) to more than 3 km (on the highway) and the models struggle to accommodate a single characteristic length. This justifies the use of more complicated models.

lagged smoothing only requires the last two points. For a slightly greater computing cost, the filter can offer a solution with a lag of one or two interval time units that is very close to the full smoothing solution. Fixed-lag smoothing will be the recommended solution for practical applications, as it strikes a good balance of computation costs, accuracy and timeliness of the results.

It should be noted the algorithm continues to provide decent results even when points grow further apart. The errors steadily increase with the sampling rate until the 30 seconds time interval, after which most metrics reach some plateau. This algorithm could be used in tracking solutions to improve the battery life of the device by up to an order of magnitude for GPSs that do not need extensive warm up. In particular, the tracking devices of fleet vehicle are usually designed to emit every minute as the road-level accuracy is not a concern in most cases.

2.5.3 Unsupervised learning results

The filter was also trained for the simple and complex models using Dataset 2. This dataset does not include true observations but is two orders of magnitude larger than Dataset 1 for the matching sampling period (1 minute). We report some comparisons between the models previously trained with Dataset 1 ("MaxLL-Simple", "EM-Simple", "MaxLL-Complex") and the same simple and complex models trained on Dataset 2: "EM-Simple large" and "EM-Complex large". The learning procedure was calibrated using cross-validation and was run in the following way: all unsupervised models were initialized with a hand-tuned heuristic model involving only the standard deviation and the characteristic length (with the weight

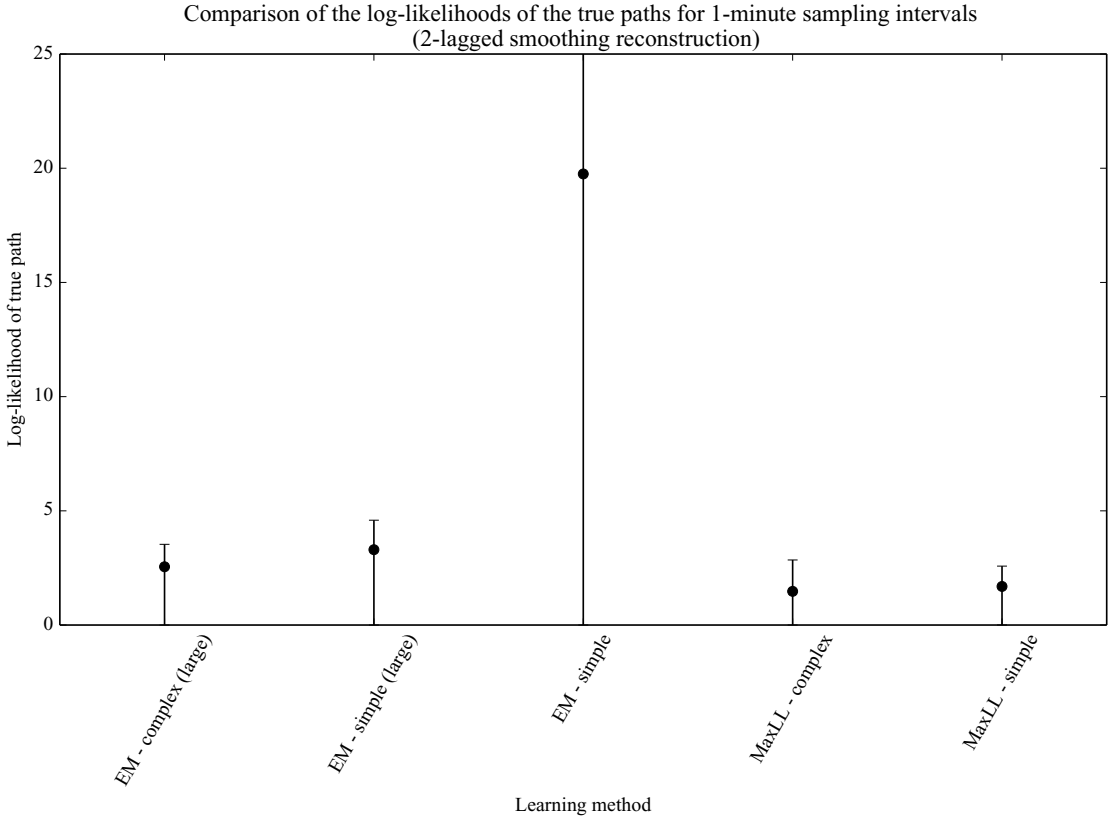


Figure 2.5.11: Expected likelihood of the true path. The central point is the mean log-likelihood, the error bars indicate the 70% confidence interval. Note that the simple model trained unsupervised with the small dataset has a much larger error, i.e. it assigns low probabilities to the true path. Both unsupervised models tend to express the same behavior but are much more robust.

of all the features set to 0). The Expectation-Maximization algorithm was then run for 3 iterations. Inside each EM iteration, the M-step was run with a single Newton-Raphson iteration at each time, using the full gradient and Hessian and a quadratic penalty of 10^{-2}. During the E step, each sweep over the data took 13 hours 400 thousand points on a 32-core Intel Xeon server.

We limit our discussion to the main findings for brevity. The unsupervised training finds some weight values similar to those found with supervised learning. The magnitude of these weights is larger than in the supervised settings. Indeed, during the E step, the algorithm is free to assign any sensible value to the choice of the path. This may lead to a self-reinforcing behavior and the exploration of a bad local minimum.

As Figures 2.5.11, 2.5.13, and 2.5.12 show, a large training dataset puts unsupervised methods on par with supervised methods as far as performance metrics are concerned. Also,

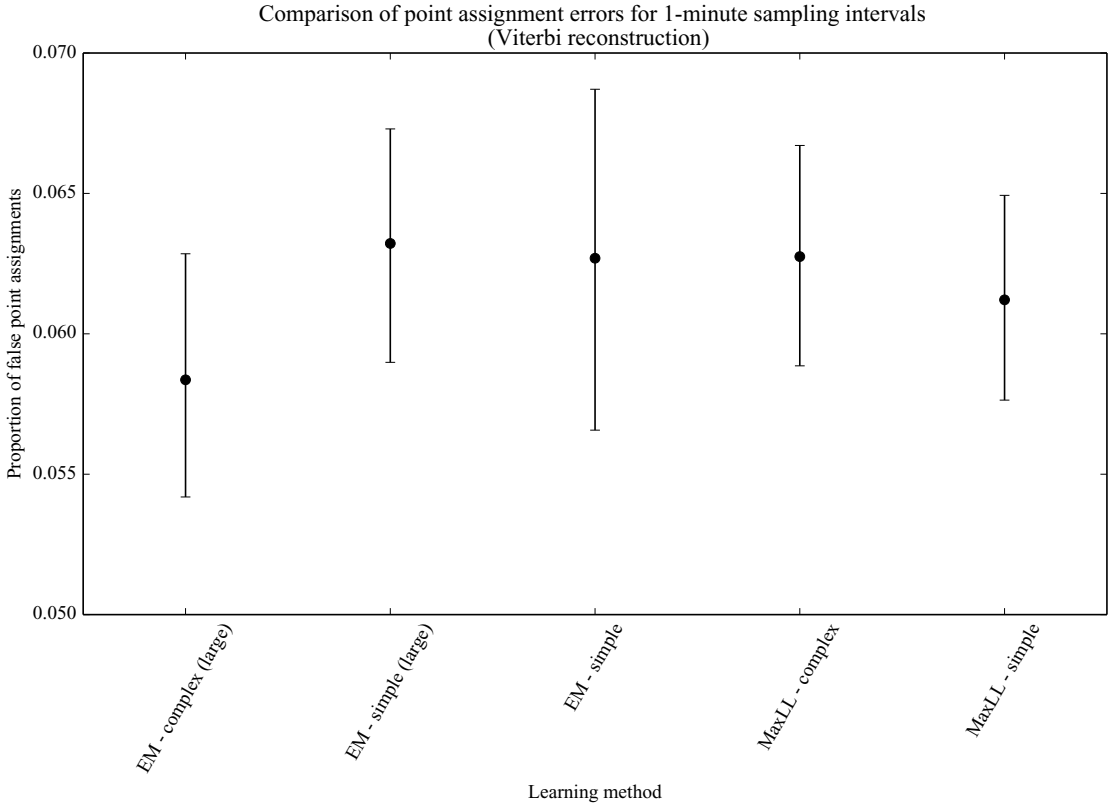


Figure 2.5.12: Proportion of true points incorrectly identified, for different models evaluated with 1-minute sampling (lower is better). The central point is the mean proportion, the error bars indicate the 70% confidence interval. Unsupervised models are very competitive against supervised models, and the complex unsupervised model slightly outperforms all supervised models.

the inspection of the parameters learned on this dataset corroborates the finding made earlier. One is tempted to conclude that given enough observations, there no need to collect expensive high-frequency data to train a model.

2.5.4 Key findings

Our algorithm can reconstruct a sensible approximation of the trajectory followed by the vehicles analyzed, even in complex urban environments. In particular, the following conclusions can be made:

- An intuitive deterministic heuristic ("Hard closest point") dramatically fails for paths at low frequencies, less so for points. It should not be considered for sampling intervals larger than 30 seconds.

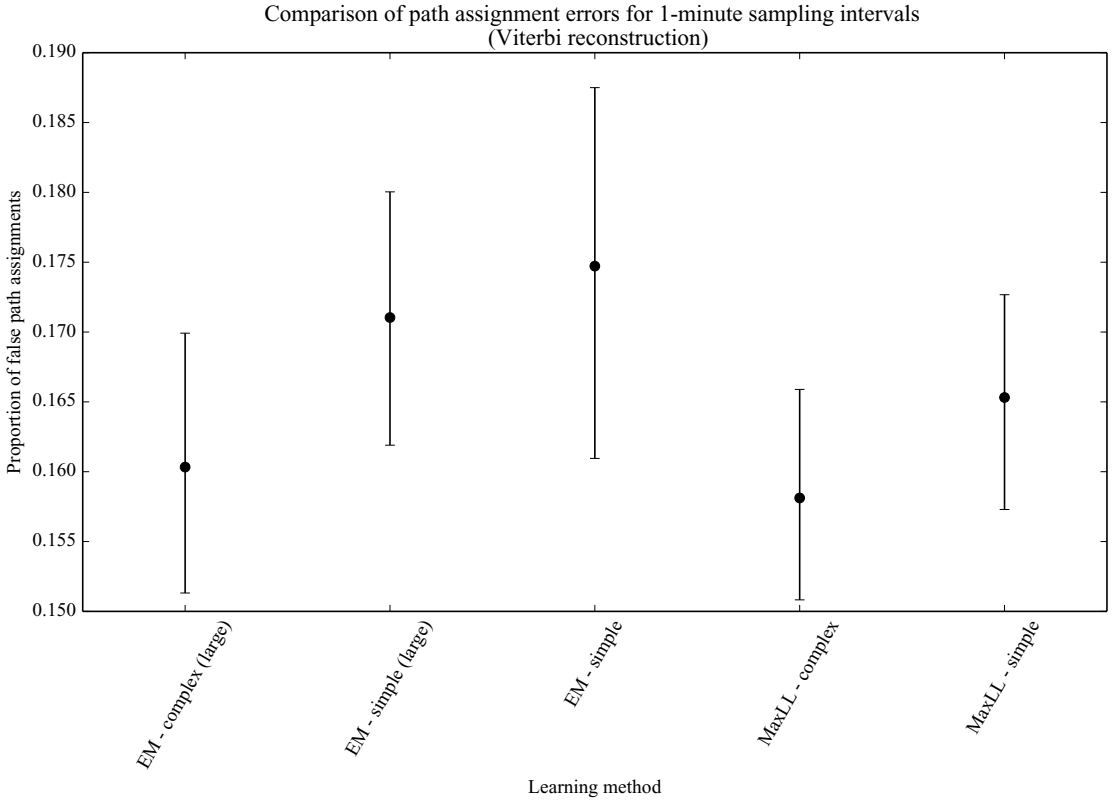


Figure 2.5.13: Proportion of true paths incorrectly identified, for different models evaluated with 1-minute sampling (lower is better). The central point is the mean proportion, the error bars indicate the 70% confidence interval. The complex unsupervised model is as good as the best supervised model.

- A simple probabilistic heuristic ("closest point") gives good results for either very low frequencies (2 minutes) or very high frequencies (a few seconds) with more 75% of paths and 94% points correctly identified. However, the incorrect values are not as close to the true trajectory as they are with more accurate models ("Simple" and "Complex").
- For the medium range (10 seconds to 90 seconds), trained models (either supervised or unsupervised) have a greatly improved accuracy compared to untrained models, with 80% to 95% of the paths correctly identified by the former.
- For the paths that are incorrectly identified, trained models ("Simple" or "Complex") provide better results compared to untrained models (the output paths are closer to the true paths, and the uncertainty about which paths may have been taken is much reduced). Furthermore, using a complex model ("Complex") improves these results even more by a factor of 13-20% on all metrics.

- For filtering strategies: online filtering gives the worst results and its performance is very similar to 1-lagged smoothing. The slower strategies (2-lagged smoothing and offline) outperform the other two by far. Two-lagged smoothing is nearly as good as offline smoothing, except in very high frequencies (less than 2 second sampling) for which smoothing clearly provides better results.

- Using a trained algorithm in a purely unsupervised fashion provides an accuracy as good as when training in a supervised setting - within some limits and assuming enough data is available. The model produced by EM ("EM-Simple") is equally good in terms of raw performance (path and point misses) but it may be overconfident.

- With more complex models, the filter can be used to infer some interesting patterns about the behavior of the drivers.

2.6 Conclusion, discussion

We have presented a novel class of algorithms to track moving vehicles on a road network: the *path inference filter*. This algorithm first projects the raw points onto candidate projections on the road network and then builds candidate trajectories to link these candidate points. An observation model and a driver model are then combined in a Conditional Random Field to find the most probable trajectories.

The algorithm exhibits robustness to noise as well as to the peculiarities of driving in urban road networks. It is competitive over a wide range of sampling rates (1 second to 2 minutes) and greatly outperforms intuitive deterministic algorithms. Furthermore, given a set of ground truth data, the filter can be automatically tuned using a fast supervised learning procedure. Alternatively, using enough regular GPS data with no ground truth, it can be trained using unsupervised learning. Experimental results show that the unsupervised learning procedure compares favorably against learning from ground truth data. One may conclude that given enough observations, there no need to collect expensive high-frequency data to train a model.

This algorithm supports a range of trade-offs between accuracy, timeliness and computing needs. In its most accurate settings, it extends the current state of the art [123, 131]. This result is supported by the theoretical foundations of Conditional Random Fields. Because no standardized benchmark exists, the authors have released an open-source implementation of the filter to foster comparison with other methodologies using other datasets [11].

In conjunction with careful engineering, this program can achieve high map-matching throughput. The authors have written an industrial-strength version in the Scala programming language, deployed in the *Mobile Millennium* system. This version maps GPS points at a rate of about 400 points per second on a single core for the San Francisco Bay area (several hundreds of thousands of road links), and has been scaled to multicore architecture to achieve an average throughput of several thousand points per second [64]. This implementation has been successfully deployed in Stockholm and Sacramento as well.

A number of extensions could be considered to the core framework. In particular, more detailed models of the driver behavior as well as algorithms for automatic feature selection should bring additional improvements in performance. Another line of research is the mapping of very sparse data (sampling intervals longer than two minutes). Although the filter already attempts to consider as few trajectories as possible, more aggressive pruning may be necessary in order to achieve good performance. Finally, the EM procedure presented for automatically tuning the algorithm requires large amounts of data to be effective, and could be tested on larger datasets that what we have presented here.

Chapter 3

A large-scale distributed streaming model

> A lawyer without history or literature is a mechanic, a mere working mason; if he possesses some knowledge of these, he may venture to call himself an architect.
>
> Sir Walter Scott, *Guy Mannering*

3.1 Introduction

The specific problem we address in this use case is how to extract travel time distributions from *sparse, noisy* GPS measurements collected *in real-time* from vehicles, and over a *very large network*. A probabilistic model of travel times on the arterial network is presented along with an online *Expectation Maximization* (EM) algorithm for learning the parameters of this model (Section 3.2). The algorithm is expensive due to the large dimension of the network and the complexity inherent to the evolution of traffic. Furthermore, our EM algorithm has no closed-form expression and requires sampling and non-linear optimization techniques. This is why the use of a distributed system is appropriate.

This EM algorithm is the core of an estimation pipeline deployed inside the *Mobile Millennium* traffic information system [3, 65]. This engine gathers GPS observations from participating vehicles and produces estimates of the travel times on the road network. *Mobile Millennium* is designed to work at the scale of large metropolitan areas: the road network considered in this work is a real road network (a large portion of the greater Bay Area, comprising 506,685 road links) and the data for this work is collected from thousands of vehicles that generate millions of observations per day. As a consequence of these specifications and requirements, we employ highly scalable traffic algorithms. Furthermore, *Mobile Millennium* is a research platform and can be used with various models of travel times. The fundamental

unit of estimation is the probability distribution of travel times over a single link of the road network. As will be seen, our framework can accommodate *any* distribution of travel times that provided this distribution exposes a few functionalities (sampling, parameter estimation from observations). This should be of interest to traffic researchers and practitioners since our framework solves all the issues of using raw GPS samples to build traffic estimates at a very large scale with low latency. Our framework has been released under an open-source license and is available for download [9].

More generally, our system presents a way to cope with large amounts of data from automation systems in a principled way. Industries such as genomic and astronomy have learned to cope with extremely large datasets over the last decade. What makes cyberphysical systems stand out amongst these applications is the fast decay of the value of information: in robotics systems for example, the data collected from sensors is usually fed into a control system. Past information is often of limited or no value, sometimes as fast as in the span of a few minutes or tens of seconds. This is unlike genomic records which, rather than being processed immediately, need to be stored reliably for a long time. In essence, the incoming information in cyberphysical systems needs to be considered as a stream, and not so much as an ever-growing dataset. In this setting, the design of the computing platform becomes critical to achieve both *scalability* (which implies robustness to computer failures) and *latency*, which is all the more important as estimates are usually part of a larger decision system. In this chapter, we investigate the use of Discretized Streams (D-Streams) [125], a novel computing technique that process flows of incoming data on a cluster.

The present work is novel for three reasons:

- Our framework can work with a very large class of travel time distributions proposed in the literature [50, 55, 76]. If a travel time distribution can perform some elementary operations described in Section 3.2.4 (conditional sampling, maximum likelihood estimation), then it can be used in our highly scalable architecture. Thus transportation researchers can focus on designing good travel time distributions, leaving the system aspects aside should they wish to deploy it on a real system.

- Our framework can accommodate complex distributions and spread the computations across a large cluster. The natural baseline is Gaussian distributions, because a number of important operations have close-formed solutions, unlike other distributions. However, Gaussian distributions are not adapted to represent traffic distributions: they are not limited to the positive reals, and they lack a heavy tail (i.e., they are sensitive to noise and outliers). These considerations drive our use of another simple distribution: the Gamma distribution. We present novel results regarding the sampling from conditioned Gamma distributions.

- Building such a system is at the forefront of research in large scale systems. Our algorithm (an EM algorithm on streaming data) is representative of a large class of Machine Learning algorithms used in robotics and automation, and our overall design could be used as well for these other algorithms and applications. This is why we

present and explain the design of our system, as we find there are valuable lessons for practitioners.

We start by presenting our general approach to traffic estimation and the *Mobile Millennium* framework in Section 3.2. We then present in details how a non-trivial distribution of travel times (a Gamma distribution) can be used on the system. We then present the system aspects and the overall design of the system in Section 3.4. We finally evaluate our implementation in Section 3.5 from the perspective of scalability (Section 3.5.2) and accuracy (Section 3.5.3).

3.2 Scalable traffic estimation from streaming data

Most GPS data available today is generated at low frequencies due to energy and bandwidth constraints. This data is extremely noisy and provides indirect observations of the travel time distributions on each link of the road network. We introduce here *Mobile Millennium*, a traffic information system that is designed to process such data at low latencies and for very large urban areas. In this section, we will discuss the overall architecture of the arterial estimation pipeline. This pipeline make few assumptions about the actual travel time distribution considered. This lets us define a highly scalable algorithm that is amenable to distribution on cloud computing. In the next section, we will present a particular choice of travel time distribution, along with some algorithms to perform the different steps. It should be noted that the choice of the distribution can be made independently from our framework: most distributions for travel times can be plugged into our framework and yield a very scalable algorithm.

We define the road network as a graph $\mathcal{D} = (\mathcal{V}, \mathcal{E})$, where the set $\mathcal{E}$ will be referred to as the "links" of the road network (streets) and $\mathcal{L}$ as the "nodes" (road intersections). For each link $l \in \mathcal{E}$, the algorithm outputs X_l^t, the time it takes at time index t to traverse link l. This time is described as a probability distribution parametrized by a vector ν_l. Our goal is then to estimate X^t, the joint distribution of all link travel times across all links in $\mathcal{E}$, for each time index t. We assume that the traffic is varying slowly enough that it can be considered a steady state between each evaluation: our algorithm will consider that all the observations between two consecutive time steps have been generated according to the same state. To simplify notations, we will consider a single time interval and drop the reference to time: the joint distribution of travel time is the multidimensional variable X.

We will first give an overview of the GPS data that is commercially available today, and an algorithm that converts raw GPS points to map-matched trajectories with high accuracy: the *path inference filter* (PIF) [61]. We will then present our modeling approach to infer the traffic conditions from these GPS observations. Then we will explain how the *Mobile Millennium* [65] pipeline implements this algorithm using a computing cloud as a computation backend.

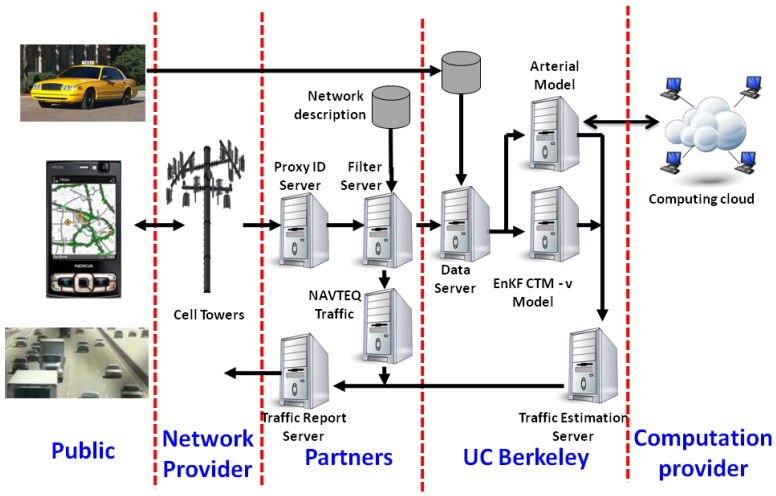


Figure 3.2.1: Schematic architecture of the *Mobile Millennium* system.

3.2.1 Overview of the *Mobile Millennium* pipeline

The *Mobile Millennium* system incorporates a complete pipeline for receiving probe data, filtering it, distributing it to estimation engines and displaying it, all in in real-time. This software stack, written in Java and Scala, evaluates *probabilistic distribution of travel times* over the road links, and uses as input the *sparse, noisy* GPS measurements from probe vehicles.

The most computation-intensive parts of this pipeline have all been ported to a cloud environment. We briefly describe the operations of the pipeline, pictured in Figure 3.2.1.

The observations are grouped into time intervals and sent to a traffic estimation engine, which runs the learning algorithm described in the next section and returns distributions of travel times for each link (Figure 3.2.3).

The travel time distributions are then stored and broadcast to clients and to a web interface. Examples of means of travel times are shown in Figure 3.5.1.

It is important to point out that *Mobile Millennium* is intended to work at the scale of large metropolitan areas. The road network considered in this work is a real road network (a large portion of San Francisco downtown and of the greater Bay Area, comprising 506,685 road links) and the data is collected from the field (as opposed to simulated). A consequence of this setting is the scalability requirement for the traffic algorithms we employ. Thus, from the outset, our research has focused on designing algorithms that could work for large urban areas with hundreds of thousands of links and millions of observations.

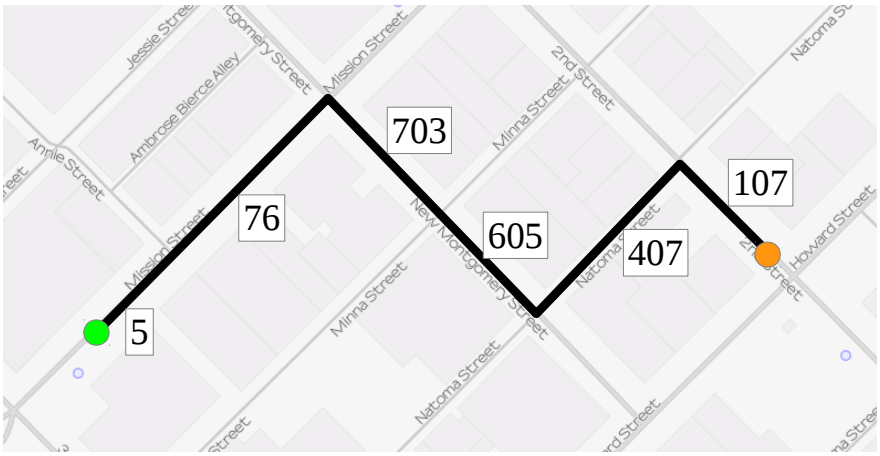
3.2.2 Map-matching GPS probe data with the Path Inference Filter

In order to reduce power consumption and transmission costs, probe vehicles do not continuously report their location to the base station. A high temporal resolution gives access to the complete and precise trajectory of the vehicle, but this causes the device to consume more power and communication bandwidth. Also, such data is not available at large scale today, except in a very fragmented portion of the the private sector. A low temporal resolution carries some uncertainty as to which trajectory was followed. In the case of a high temporal resolution (typically, a frequency greater than an observation per second), some highly successful methods have been developed for continuous estimation [37, 84, 111]. However, most data collected at large scale today is generated by commercial fleet vehicles. It is primarily used for tracking the vehicles and usually has a low temporal resolution (1 to 2 minutes) [2, 4, 10, 108]. In the span of a minute, a vehicle in a city can cover several blocks (see Figure 3.2.2 for an example). Information on the precise path followed by the vehicle is lost. Furthermore, due to GPS localization errors, recovering the location of a vehicle that just sent an observation is a non trivial task: there are usually several streets that could be compatible with any given GPS observation. Simple deterministic algorithms to reconstruct trajectories fail due to misprojection or shortcuts. The path inference filter [61] is a probabilistic framework that recovers trajectories and road positions from low-frequency probe data in real time, and in a computationally efficient manner.

This algorithm first projects the raw points onto candidate projections on the road network and then builds candidate trajectories to link these candidate projections. An observation model and a driver model are then combined in a Conditional Random Field to find the most probable trajectories, using the Viterbi algorithm. More precisely, the algorithm performs the following steps:

- We map each point of raw (and possibly noisy) GPS data to a collection of nearby *candidate projections* on the road network (Figure 2-1).
- For each vehicle, we reconstruct the most likely trajectory using a Conditional Random Field [61] (Figure 2-2).
- Each segment of the trajectory between two GPS points is referred as an *trajectory measurement* (Figure 2-3). A trajectory measurement consists in a start time, an end time and a route on the road network. This route may span multiple road links, and starts and ends at some offset within some links.

At the output of the PIF, we have transformed sequences of GPS readings into sequences of trajectory readings. These readings are the input for our travel time estimation algorithm.



$$x_5^4(17.0, \textbf{end}) + x_{76}^4 + x_{703}^4 + \cdots + x_{107}^4(0, 20.0) = d$$

$$0.5x_5^4 + x_{76}^4 + x_{703}^4 + \cdots + 0.8x_{107}^4 = d$$

Figure 3.2.2: Example of observation. The green mark represents an initial GPS reading, the orange mark represents a subsequent reading. The black line marks the path of the vehicle, as reconstructed by the path inference filter between the two GPS points and the numbers are the indexes of each road link covered by this observation. Given a realization x^4 of the travel time distribution at time $t = 4$, all the information on travel times encoded by this observation is summarized in the equation above.

3.2.3 Fundamental generative model

Estimating the travel time distributions is made difficult by the fact that we do not observe travel times for individual links. Instead, each reading only specifies the total travel time for an entire list of links traveled. We formally describe our estimation task as a maximum likelihood estimation problem.

We consider one reading, described by an offset on a first road link o_{start}, an offset on a last link o_{end}, a list of m visited links $l_1 \cdots l_m$, a start time, and a travel duration d (see Figure 3.2.2 for an example of a reading). The observed travel time is a sum of (unobserved) travel times on the visited links:

$$X_{l_1}\left(o_{\text{start}}, L\left(l\right)\right) + X_{l_2} + \cdots X_{l_{m-1}} + X_{l_m}\left(0, o_{\text{end}}\right) = d$$

We are going to introduce two assumptions: independence and scaling. The first one is critical to our framework, and is widely used in practice. The scaling assumption is not necessary, but offers some convenience for the development of the discussion. When working with more sophisticated distributions and with enough data, it could easily be dispensed with.

Independence assumption. To make the inference problem tractable, we model the link travel times for each link l as a univariate distribution with parameter vector $\nu_l = (k_l, \theta_l)$,

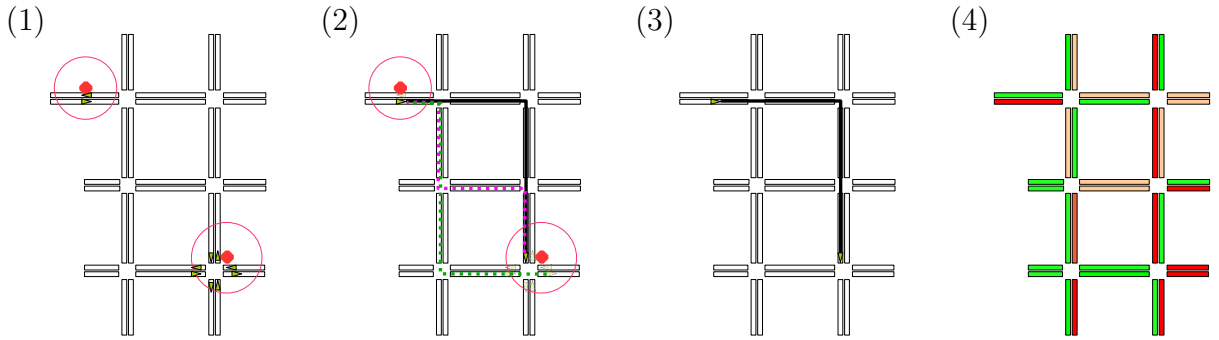


Figure 3.2.3: Map-matching algorithm: the raw GPS readings a first projected onto candidate points on the road network (Step 1). Then all feasible paths between each pair of candidate points are computed (Step 2). A dynamic programing algorithm then finds the most likely trajectory, using a Conditional Random Field (Step 3). Trajectory measurements are the input to the Expectation Maximization algorithm. This algorithm outputs distributions of travel times (Step 4).

and we assume these distributions are pairwise independent. The independence assumption is standard in the transportation literature [51, 57] and it also leads to a highly scalable estimation algorithm. We will discuss the validity of this assumption in Section 3.5.3.

Scaling assumption. The distribution of travel time $X_l(a, b)$ between two offsets a and b of a road link l can be significantly different in shape from the distribution X_l over the full link. We simplify the problem by assuming that the partial travel time from the start of a road link to some offset o is proportional to the distribution over the full link:

$$X_{\text{partial}}(o_{\text{start}}, o_{\text{end}}) = f_l(o_{\text{start}}, o_{\text{end}})\, X_l$$

wher f_l is a function in values between 0 and 1. In our implementation, we make the use of the following function:

$$f(o_{\text{start}}, o_{\text{end}}, L(l)) = \left(\frac{o_{\text{end}}}{L(l)}\right)^r - \left(\frac{o_{\text{start}}}{L(l)}\right)^r$$

for some $r > 0$. It is well-known that the travel time on a part of a road link is not proportional to the travel time over the complete link [57]. The function f captures some of this non-linearity. The factor r is selected by cross-validation and was set to be 2.1. In all generality, this assumption may not be representative of empirical data, and it is a convenient way to consider partial travel times without introducing additional parameters to the model. However, in our streaming setting, the updates happen at high frequency (every few second), and there is not enough observation to fully update the distributions, which may lead to some overfitting. As we will see in the next section, this assumption may be dispensed with when more complex traffic distributions are considered. This modelization is fairly crude but gives sensible results in our experiments. It would be interesting to see how this holds as more data becomes available in the future.

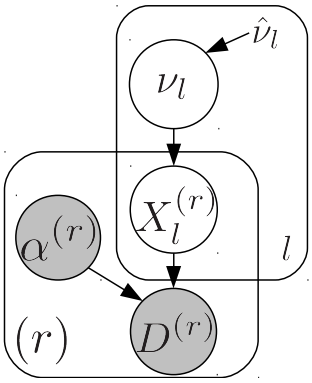


Figure 3.2.4: Directed (Bayesian) graph of the travel time model. Grey nodes are observed variables, white nodes are hidden variables. The arrows represent conditional dependencies between the variables. Boxes encode *plates*, i.e. a factorization of repeating variables.

Using the two assumptions outlined above, the duration d of a travel is the linear combination of realizations of travel times on the different links of the road network:

$$\begin{aligned} d &= \sum_{l \in \mathcal{E}} \alpha(l)\, x_l \\ &= \sum_{l \in p} \alpha(l)\, x_l \end{aligned}$$

where the vector $\alpha \in [0,1]^n$ is defined as follows[1]:

$$\begin{aligned} \alpha(l_1) &= f_l(o_{\text{start}}, L(l_1)) \\ \alpha(l_m) &= f_l(0, o_{\text{end}}) \\ \alpha(l_i) &= 1 \text{ for } i \in [2...m-1] \end{aligned}$$

and $\alpha(l) = 0$ for all other links l. The vector α is called the *path activation vector* for this reading. Note that fewer than 10 links are covered in a typical trajectory measurement, so the path activation vectors are extremely sparse (the fill-in factor is less than 0.001%). We will use this fact to achieve very good scaling of our algorithm.

For a given time interval, we can completely represent a trajectory reading by an *observation* $Y = (\alpha, d) \in (\mathbb{R}^+)^n \times \mathbb{R}_+^*$. Each observation $Y^{(r)} = \left(\alpha^{(r)}, D^{(r)}\right)$ describes the rth trajectory's travel time $D^{(r)}$ and path $\alpha^{(r)}$ as inferred by earlier stages of the *Mobile Millennium* pipeline. The travel time $D^{(r)}$ is the time interval between consecutive GPS observations and is roughly one minute for our source of data.

The dependencies between the observations and the parameter vector ν can be represented as a Bayesian graphical model, which encodes all the dependencies between the variables in a very compact form (Figure 3.2.3). We now formalize the problem of estimating

[1]In practice, the definition of α needs to be adapted when an observation spans a only single link.

the set of parameters $\nu = (\nu_l)_l$ for a set of observations $\left(Y^{(r)}\right)_{r=1\cdots R}$ as a learning problem. We consider that the the current estimate of the traffic is completely described by independent distributions (parametrized by some vectors ν_l) of the travel times over each road link. These travel times are indirectly observed through a set of observations $Y^{(r)} = \left(\alpha^{(r)}, d^{(r)}\right)$. The set of parameters that maximizes the likelihood of these observations is solution to the *maximum likelihood problem*:

$$\max_{\nu} \; ll\left(Y;\nu\right) \;\; = \;\; \sum_{r} \log \pi \left(D^{(r)} \middle| \alpha^{(r)}; \nu \right) \tag{3.2.1}$$

with $\pi \left(D^{(r)} \middle| \alpha^{(r)}; \nu \right)$ the probability of observing the duration $d = \sum_l \left(\alpha\left(l\right)\right) x_l$ when x_l is generated according to the distribution $\pi\left(\cdot;\nu_l\right)$ of the variable X_l. This likelihood can be decomposed using the relations of independence between variables:

$$\begin{aligned}
\pi \left(D^{(r)} \middle| \alpha^{(r)}; \nu \right) &= \int_X \pi \left(D^{(r)} \middle| X, \alpha^{(i)} \right) \pi\left(X;\nu\right) \mathrm{d}X \\
&= \int_X \pi \left(D^{(r)} \middle| X, \alpha^{(i)} \right) \left(\prod_{l\,:\,\alpha^{(r)}(l)>0} \pi\left(X_l;\nu_l\right) \mathrm{d}X_l \right) \\
&= \int_X \pi \left(D^{(r)} \middle| X, \alpha^{(i)} \right) \left(\prod_{l\,:\,\alpha^{(r)}(l)>0} \pi\left(X_l;\nu_l\right) \mathrm{d}X_l \right)
\end{aligned}$$

Estimating the travel time distributions is made difficult by the fact that we do not observe travel times X for individual links. Instead, each observation only specifies the total travel time D for an entire list α of links traveled. To get around this problem, we use the *Expectation Maximization* (EM) algorithm [34, 88]. The EM algorithm operates in two phases: In the E-step it considers each travel time measurement and computes a distribution over allocations of travel time to each of the links. In the M-step it computes the link parameters that maximizes the likelihood of the travel times for the allocations made in the E-step. By iterating this process the EM algorithm converges to a set of link parameters that are a local maximum of the likelihood of the data. In our setting, we run the EM algorithm in an online fashion: for each time step, we use the previous time step as a value, perform a few (iterations and we monitor the convergence through the expected complete log-likelihood. This form of online EM gives good results for our application (Section 3.5).

3.2.4 Dataflow of the algorithm

Figure 3.2.5 shows the data flow in the algorithm in more detail. In the E-step, we generate per-link travel time samples from whole observations; specifically, for each observation $Y^{(r)} = \left(\alpha^{(r)}, d^{(r)}\right)$, we produce a set of U weighted samples $\mathbf{X}^{(r)} = \left\{\left(x^{(r,u)}, w^{(r,u)}\right)\right\}_{u=1\cdots U}$, each

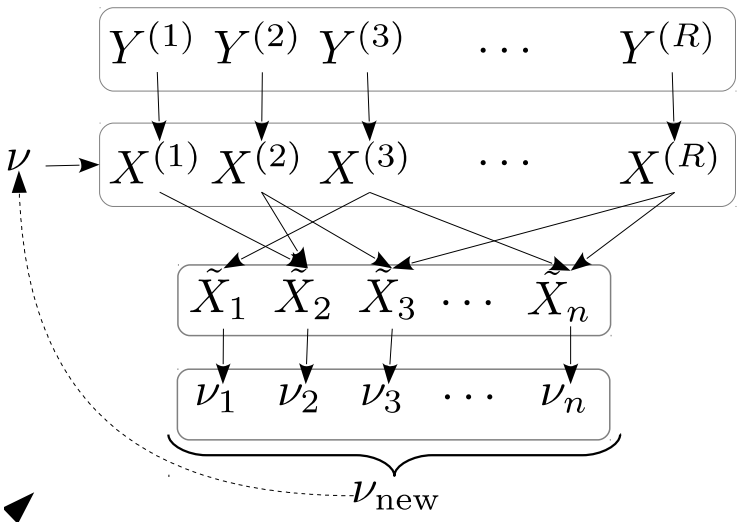


Figure 3.2.5: System workflow of the EM algorithm. In the E-step, we generate per-link travel time samples from whole observations; specifically, for each observation $Y^{(r)} = \left(\alpha^{(r)}, d^{(r)}\right)$, we produce a set of U weighted samples $\mathbf{X}^{(r)} = \left\{\left(x^{(r,u)}, w^{(r,u)}\right)\right\}_{u=1\cdots U}$, each sample $x^{(r,u)}$ produced by randomly dividing travel time $d^{(r)}$ among its constituent links (producing a travel time $x_{l_i}^{(r,u)}$ for each edge $l_i \in \alpha^{(r)}$). We assign a weight $w^{(r,u)}$ as the likelihood of travel time $x^{(r,u)}$ according to the current distribution parameters ν. In the shuffle step, we regroup the samples $\mathbf{X}^{(r)}$ by link, so that each link l now has samples $\tilde{\mathbf{X}}_l = \left\{\left(s_l^{(r,u)}, w_l^{(r,u)}\right)\right\}_{r,u}$ from all the observations r that go over it. In the M-step, we recompute the parameters ν_l to fit link l's travel time distribution to the samples $\tilde{\mathbf{X}}_l$.

sample $x^{(r,u)}$ produced by randomly dividing travel time $d^{(r)}$ among its constituent links (producing a travel time $x_{l_i}^{(r,u)}$ for each edge $l_i \in \alpha^{(r)}$). We assign a weight $w^{(r,u)}$ as the likelihood of travel time $x^{(r,u)}$ according to the current distribution parameters ν. In the shuffle step, we regroup the samples $\mathbf{X}^{(r)}$ by link, so that each link l now has samples $\tilde{\mathbf{X}}_l = \left\{\left(s_l^{(r,u)}, w_l^{(r,u)}\right)\right\}_{r,u}$ from all the observations r that go over it. In the M-step, we recompute the parameters ν_l to fit link l's travel time distribution to the samples $\tilde{\mathbf{X}}_l$.

So far, we have not introduced a particular distribution for the travel times X_l. This will be the subject of the next section. We require only few operations on a distribution of travel times:

- Sampling under a constraint: the Expectation step involves sampling $X_1 \cdots X_U$ under some constraint $a^T X = d$. We will show in the next section how this operation can be performed efficiently in the case of Gamma distributions. For more complicated distributions, importance sampling may be used. This technique in turn only requires the knowledge of the unnormalized p.d.f. of the distribution, which is usually not a

concern.

- Solving the maximum likelihood problem: in the M step, for each of the links, we solve a problem of the form $\max_\nu \sum \tilde{w}^{(i)} \log \pi\left(\tilde{x}^{(i)}; \nu\right)$ given a set of weighted samples $\left(\tilde{w}^{(i)}, \tilde{x}^{(i)}\right)_i$. This problem can be solved approximately, using gradient descent for example. If the number of parameters is small (typically less than 4), grid search may even yield an appropriate solution in a reasonable time.

The minimum requirements for travel time distributions are very mild and may satisfy complex, multi-modal distributions. However, using complex distributions involves learning a large number of parameters and may lead to overfitting when data is scarce. Since our goal is to study the validity of the framework for very large networks and datasets, we will consider in our discussion simple distributions. It will be interesting to compare in the future with some other more realistic distributions.

3.3 Modeling travel times with Gamma distributions

The Gamma distribution is a simple unimodal, heavy-tailed distribution with a mean and a scale parameter. In this section, we show how it can be used as a travel time distribution and how it fits into our framework.

Gamma distributions substantially complicate the Expectation step, because sums of independent Gamma distributions have no closed form. This is why we approximate the conditional expectation $X^{(r)} | \left(\alpha^{(r)}\right)^T X^{(r)} = D^{(r)}$ by sampling from this distribution. As we will see, there is a surprisingly simple algorithm in the case of Gamma distributions. While not strictly necessary, it is also useful to compute the value of the marginal likelihood of each observation $\mathbb{P}\left(\left(\alpha^{(r)}\right)^T X^{(r)} = D^{(r)}\right)$ in order to track the converge of the EM algorithm. Since the computations are independent for each of the observations, they are good candidates for cloud computing.

This section is self contained and does not make use of concepts from the other sections. We introduce it to show how a non-trivial distribution can be incorporated into the rest of the framework, and what computations are necessary to fit in the framework presented is Section 3.3.2. We present the main results in Section 3.3.1. Since the justification of these results requires some measure-theoretic technicalities, the proofs are derived in the subsequent sections. These justifications will require introducing a new statistical distribution: the Gamma-Dirichlet distribution.

3.3.1 Learning with Gamma distributions

Consider a set of n independent Gamma distributions $T_i \sim \Gamma(k_i, \theta_i)$ with $k \in \left(\mathbb{R}_+^*\right)^n$ and $\theta \in \left(\mathbb{R}_+^*\right)^n$, a n-dimensional vector of positive numbers $\alpha \in \left(\mathbb{R}_+^*\right)^n$ and $d > 0$[2]. Call T the joint distribution of all T_is. The purpose of this paragraph is to present some practical formulas to sample and compute the density function of the sum $\sum_i \alpha_i T_i$, which is a univariate distribution.

Marginal likelihood Call $U = \sum_i \alpha_i T_i$. Call $\underline{\theta} = \min_i \alpha_i \theta_i$ and $\underline{k} = \sum_i k_i$. The probability density function of U is an infinite series [15, 86]:

$$f_U(d) = \underline{\theta}^{\underline{k}} \prod_i (\alpha_i \theta_i)^{-\underline{k}} \sum_{l=0}^{\infty} \delta_l f_\Gamma(d; \underline{k} + l, \underline{\theta})$$

in which f_Γ is the density function of the Gamma distribution: $f_\Gamma(x; a, b) = \Gamma(a)^{-1} b^{-a} x^{a-1} e^{-b^{-1}x}$ and $(\delta_j)_j$ a series defined by the recursive formula:

$$\begin{cases} \delta_0 = & 1 \\ \delta_l = & \frac{1}{l} \sum_{m=0}^{l} \delta_l \left(\sum_{i=1}^{n} k_i \left(1 - \alpha_i \theta_i^{-1} \underline{\theta}\right)^{l-m} \right) \end{cases}$$

This result is a direct application of [86], using the scaling property of the Gamma distribution: $\alpha_i T_i \sim \Gamma(k_i, \alpha_i \theta_i)$ for $\alpha_i > 0$.

Sampling from conditional gamma distributions.

Algorithm 3.1 Sampler for Gamma distributions conditioned on a hyperplane

Given $\alpha \in \left(\mathbb{R}_+^*\right)^n$ and $d > 0$.
Generate n independent samples $a_i \sim \Gamma(k_i, d^{-1}\alpha_i\theta_i)$
$z_i = d\alpha_i^{-1} \frac{a_i}{\sum_k a_k}$
Then $z \sim T | \alpha^T T = d$

Our algorithm must provide values from the conditional distribution $Z \sim T | \alpha^T T = d$. While this distribution has a complex shape (in particular, it is defined over a zero-measure hyperplane of the space of variable), there happens to exist a remarkably simple procedure to sample values from the conditional Gamma distribution Z: sample n independent values from Gamma distributions:

$$A_i \sim \Gamma\left(k_i, \frac{\alpha_i \theta_i}{d}\right) \tag{3.3.1}$$

Then a suitably rescaled value of A_i follows the distribution of Z:

$$Z_i = \frac{d}{\alpha_i} \frac{A_i}{\sum_l A_l} \tag{3.3.2}$$

[2]The bivariate function $\Gamma(\cdot, \cdot)$ will refer to the Gamma distribution and the univariate function $\Gamma(\cdot)$ will refer to the Gamma function. Which one is used should be clear from the context.

To our knowledge, this is a new result regarding Gamma distributions. We present the proof of correctness in the next Section. The proof requires some technical arguments that may be skipped in a first reading. The algorithm is presented in Algorithm 3.1.

3.3.2 The Gamma-Dirichlet distribution

In order to show that the equations 3.3.1 and 3.3.2 give the correct distribution for Z, we formally introduce a generalization of the Dirichlet distribution, which we call the *Gamma-Dirichlet distribution*[3]. This distribution can be sampled using a closed for solution. We show in a second step that its p.d.f. is the same as the p.d.f. of the conditional distribution $T|\sum_i T_i = 1$. We then generalize to arbitrary positive combinations.

The Gamma-Dirichlet distribution. The regular simplex $\mathcal{S}^n \subset \mathbb{R}^n$ is the convex hull of the elementary vertices $(\mathbf{e}_i)_{i\in[1,n]}$. Given a vector $k \in (\mathbb{R}^+)^n$ and a vector $\theta \in (\mathbb{R}^+)^n$, we define the *Gamma-Dirichlet distribution*:

$$X \sim \Gamma D(k, \theta)$$

with values over the regular simplex $\mathcal{S}^n$ as the normalized sum of n elements drawn from independent Gamma distributions:

$$X_i = \frac{Y_i}{\sum_j Y_j}$$

with

$$Y_i \sim \Gamma(k_i, \theta_i) \tag{3.3.3}$$

and Y_i all pairwise independent. The Gamma-Dirichlet distribution is a simple of the Dirichlet distribution: if $\theta = a\mathbf{1}$ for some $a > 0$, this is the Dirichlet distribution of the nth order. The definition gives a straightforward procedure to sample some values from X.

Density. Note first that one needs to be careful in defining the underlying σ-algebra of our probability space, as the values of X are located in an embedding of $\mathbb{R}^n$ of measure 0 (a hyperplane). Consider the n-dimensional hyperplane $\mathcal{H}^n = \left\{x | x^T\mathbf{1} = 1\right\}$. This hyperplane includes the simplex $\mathcal{S}^n$. The Lebesgue measure of this set in $\mathbb{R}^n$ is zero. However, we can consider the Lebesgue measure $\tilde{\mu}$ defined over $\mathbb{R}^{n-1}$ and the transform: $\phi : \mathbb{R}^{n-1} \to \mathcal{H}^n$ defined by $\phi(u) = \left(u - \mathbf{1}^T u\right)^T$. This transform is a linear mapping and it defines a new measure $\hat{\mu}$ for the space $\mathcal{H}^n$ based on $\tilde{\mu}$. Under this measure, the measure of the simplex $\mathcal{S}^n$ is positive. Call μ the measure defined over $\mathcal{S}^n$ by $\mu(\cdot) = \hat{\mu}(\mathcal{S}^n)^{-1}\hat{\mu}(\cdot)$. Consider the conditional distribution $Z_i = Y_i|\sum_j Y_j = 1$ with Y_i defined in Equation (3.3.3). The density function of this distribution is 0 over $\mathbb{R}^n$ nearly everywhere, however it has non-zero measure over the regular simplex $\mathcal{S}^n$.

Call $f_{\Gamma D}(x)$ the p.d.f. of the Gamma-Dirichlet distribution over $\mathcal{S}^n$. Then we have:

$$f_{\Gamma D}(x) \propto \prod_{i=1}^{n} f_\Gamma(x_i; k_i, \theta_i)$$

[3]To our knowledge, the following results have not been presented in the literature so far.

with $f_\Gamma(x;k,\theta)$ defined above.

Proof. This proof is adapted from a similar proof [35] for the Dirichlet distribution. Using the same notations as above, define $Y=\sum_j Y_j$ and $X_i=Y_i/Y$ for $i\leq n$. The joint density for the Y's is:

$$f(y)\propto\prod_i y_i^{k_i-1}e^{-\sum\theta_i^{-1}y_i}$$

Define the transform $\tilde{\varphi}:\mathbb{R}^n\to\mathbb{R}^n$ by: $\tilde{y}=\sum_i\theta_i^{-1}y_i$ and $\tilde{x}_i=\tilde{y}^{-1}\theta_i^{-1}y_i$ for $i\leq n-1$. We are going to show that the joint density of $\tilde{x}$ and $\tilde{y}$ can be written as $g(\tilde{x},\tilde{y})=a(\tilde{x})\,b(\tilde{y})$, which implies that the variables $\tilde{x}$ and $\tilde{y}$ are independent. This mapping is invertible and its Jacobian at $\tilde{y}$ is $\left(\prod_i\theta_i^{-1}\right)\tilde{y}$. Thus the joint density of $(\tilde{y},\tilde{x})$ is:

$$g(\tilde{y},\tilde{x})\propto\left[\left(1-\sum_{i=1}^{n-1}\tilde{x}_i\right)^{k_n-1}\prod_{i=1}^{n-1}\tilde{x}_i^{k_i-1}\right]\left[\tilde{y}^{\sum_i k_i-1}e^{-\tilde{y}}\right]$$

This shows that the variables $\tilde{y}$ and $\tilde{x}$ are independent. Furthermore, from the expression above, the distribution of $\tilde{x}$ is a Dirichlet distribution and the distribution of $\tilde{y}$ is a Gamma distribution. The marginal for $\tilde{x}$ writes:

$$g(\tilde{x})\propto\left(1-\sum_{i=1}^{n-1}\tilde{x}_i\right)^{k_n-1}\prod_{i=1}^{n-1}\tilde{x}_i^{k_i-1}$$

Now, we consider a change of variables for the joint variables y to remove the conditional constraint $\sum_i y_i=1$. Define the transform $\varphi:\hat{y}=\sum_i y_i$ and $\hat{x}_i=\hat{y}^{-1}y_i$ for $k\leq n-1$. This mapping is also invertible, with Jacobian $\hat{y}^k$. The joint density of $(\hat{y},\hat{x})$ is:

$$g(\hat{y},\hat{x})\propto\left[\left(1-\sum_{i=1}^{n-1}\hat{x}_i\right)^{k_n-1}\prod_{i=1}^{n-1}\hat{x}_i^{k_i-1}\right]\left[\hat{y}^{\sum_i k_i-1}e^{-\theta_n\hat{y}}\right]$$

By identification, we get: $g(x|y=1)=g(\Delta^{-1}\tilde{x})$ with Δ the diagonal matrix defined by $\Delta_{ii}=\theta_i$. Since $\tilde{\varphi}^{-1}\left((\Delta^{-1}\tilde{x},1)^T\right)=\tilde{x}$, the result ensues. □

The previous result proves that the sampling procedure in Algorithm 3.1 is correct for the simplex $\mathcal{S}^n$ (i.e. $d=1$). We can then use the scaling transform of the Gamma distribution to show it is also correct for other values of d. Indeed, the constraint $\alpha^T T=d$ is also equivalent to $\sum Y_i=1$ with $Y_i=d^{-1}\alpha_i T_i\sim\Gamma(k_i,,d^{-1}\alpha_i\theta_i)$. We can perform the conditional sampling on the variables Y_i, and then rescale the values obtained to get the correct distribution.

Proof. Consider a set of n independent Gamma distributions $T_i\sim\Gamma(k_i,\theta_i)$, a n-dimensional vector of positive numbers $\alpha\in\left(\mathbb{R}_+^*\right)^n$ and $t>0$. The purpose of this section is to present some practical formulas to sample and compute the density function of the conditional distribution:

$$Z = T \left| \sum_i \alpha_i T_i = d \right.$$

We define this distribution over the n-dimensional simplex

$$\mathcal{S}_{\alpha,d} = \left\{ x \in \left(\mathbb{R}^+\right) \middle| \alpha^T x = d \right\}$$

As before, we define a new measure over the hyperplane defined by $\alpha^T x = d$ by an isomorphism from $\mathbb{R}^{n-1}$, and use it as our base measure $\mathrm{d}z$ for Z. We call f the probability density function of variable Z with respect to this measure. With respect to this measure, the probability density function of Z is that of a Gamma-Dirichlet distribution:

$$f(z) \propto f_{\Gamma D}\left(y; k, \hat{\theta}\right)$$

with:

$$\hat{\theta}_i = t^{-1}\alpha_i\theta_i$$

□

3.4 Discretized streams: large-scale real-time processing of data streams

We now describe the design of the data processing pipeline of the *Mobile Millennium* system. This framework takes streams of raw GPS data as input, and outputs states of traffic, in the form of parameters of travel time distributions. We had the following requirements, which are shared with most automation system:

- *Low latency*: we wanted to investigate update rates as high as every few seconds.
- *Scalability and high throughput*: the system should be able to handle tens or hundreds of thousands of measurements per second
- *Robustness to failures*: the failure of machines should have a limited impact on the performance of the system
- *Testing multiple scenarios*: the GPS data needs to go through some complex filtering process (map-matching and trajectory reconstruction) before it can be used for estimation, and the ability to perform cross-validation is a prerequisite to tuning these algorithms. Thus, we needed to rerun or even run multiple instances of our algorithm in parallel.

- *Flexible deployment solution*: from the outset, our goal was to deploy our framework on a generic cloud solution such as Amazon EC2. This implies limited control over the topology of the computer network and over the characteristics of the computers.

Streams of hundreds of thousands of elements per second are common in cloud-based environment. However, in our application, the Expectation step of the EM algorithm generates a large number of samples for each observation. This multiplies the internal throughput rate by a factor of 1000 to 10,000. Single computers cannot work at this rate, and a cloud-based solution is required. Furthermore, these samples are ephemeral and can be deterministically recreated from an observation: there is no need to store them. The novelty of our application lies in combining the competing requirements of the different stages: the initial and final steps have average throughput and their respective inputs and outputs require fault-tolerant storage, while the intermediate computation steps must have high throughput and be resilient to individual node failures.

In particular, since this is a research platform, we needed the ability to test and monitor complex iterative algorithms. In this section, we will present the notion of Discretized Streams, a concept recently introduced [125] and implemented in the Spark computing framework. We will present how the design of D-Stream lets users build cyberphysical systems at scale.

3.4.1 Limitations of current techniques

Current techniques to process large amounts of live streaming data can be broadly classified into the following two categories.

- *Using traditional streaming processing systems*: Streaming databases like StreamBase [8] and Telegraph [28], and stream processing systems like Storm [7] have been used to meet such processing requirements. While they do achieve low latencies, they either have limited fault-tolerance properties (data lost on machine failure) or limited scalability (cannot be run on large clusters).

- *Using traditional batch processing systems*: The live data is stored reliably in a replicated file system like HDFS [1] and later processed in large batches (minutes to hours) using traditional batch processing frameworks like Hadoop [1]. By design, these systems can process large volumes of data on large clusters in a fault-tolerant manner, but they can only achieve latencies of minutes at best. Furthermore, the processing model is too low level to conveniently express complex stream computations.

3.4.2 D-Streams - A programming model for stream processing

D-Streams execute deterministic computations similar to those in MapReduce for fault tolerance, but they do so at a much lower latency than previous systems, by keeping state *in memory*, as opposed to saving it on disk between each step. The input data received from

various input sources (e.g., webservices, sensors, etc) during each interval is stored reliably across the cluster to form an input dataset for that interval. Once the time interval completes, this dataset is processed via deterministic parallel operations (like mapping transformations or filtering) to produce new datasets representing program outputs or intermediate states. Finally, these datasets can be saved to external source such as databases. The advantage of this model is that it provides the developer a convenient high-level programming model to easily express complex stream computations while allowing the underlying system to process the data in small batches thus achieving excellent fault-tolerance properties. D-Streams support the same stateless transformations available in typical batch frameworks, including map, reduce, groupBy, and join. In addition, D-Streams also provide stateful operators like windowing and moving average operators that share data across time intervals. All the intermediate data computed using D-Streams are by design fault-tolerant, that is, no data is lost if any machine fails. This is achieved by treating each batch of data (and each new batch derived through transforms of the original dataset) as a dataset distributed across the machines. Each dataset maintains a *lineage* of operations that was used to create it from the raw input data (stored reliably by the system by automatic replication) [124]. Hence, in the event of computer failure, if any partition is lost, it can be recomputed from raw input data using the lineage. As these operations are deterministic, the recomputation can be done using fine-grained tasks in parallel. This ensures fast recovery minimizing the effect of the failure on the stream processing system. This novel technique is called *parallel-recovery* and sets this abstraction apart from existing stream processing systems, that need to write intermediate steps to disk or implement complex recovery mechanisms.

To implement D-Stream, we use Spark, an existing open-source, batch processing framework, to create Spark Streaming. Spark is a fast, in-memory batch processing framework, and we naturally extend this framework to implement D-Streams. Both these systems are implemented in Scala [5] (a language based on the Java Virtual Machine), which allows them integrate well with existing Java and Scala libraries for linear algebra, machine learning, etc. Furthermore, the compact syntax of the Scala language hides all the complexities of distribution, replication and data access pattern behind an intuitive programming interface. This code instantiates a D-Stream with the raw data and derives some other D-Streams that correspond to each step of the algorithm. As can be seen, this code leverages the functional API of Spark and Scala to express stream transformations in a very natural way. Spark Streaming can scale to hundreds of cores while achieving latencies as low as hundreds of milliseconds. We use this system to implement our traffic estimation algorithms, which we shall explain next.

3.4.3 Lessons learned

In the process of designing this framework, we experienced several challenges and we made several design decisions that we feel should be taken into consideration when considering building a large system for cyberphysical computations.

Computing in memory Keeping the computations in memory leads to dramatic performance improvements, as the processes do not need to seek or write data on the disk. We found that this improves both the throughput and the latency. The fault recovery is provided by the lineage information, and is typically fast enough (a few seconds) for our experiments.

Interfacing with databases One of the bottlenecks that came to us as a surprise is the interaction with our primary storage (an Oracle database). After map-matching, the data is written to the database. The nodes of the computing cloud then open a connection periodically to read the new data at every beat. However, the database could not keep up with hundreds of queries at the same time. Replicating the database in the cluster did not lead to much improvement. Our solution was to use the database as an end point that would not be queried by the cloud computers: at the end of the map-matching step, the stream of map-matched observations is not only pushed to the database, but also to the Hadoop filesystem (HDFS) that is spread on the cluster. We found the insertion mechanism to be fast enough for this pattern.

Using immutable, stateless transforms working with immutable distributed datasets called for a different programming style that emphasizes function-based transforms of data. In this style, a filter would be implemented as a function that takes a state and some observations, and creates a new object that contains the updated state: the original state object is not modified. We found this style to be a significant departure from the common practices in control and automation. Thus, a stream is defined as a sequence of transforms on a dataset. The Scala programming language provides an elegant interface for expressing

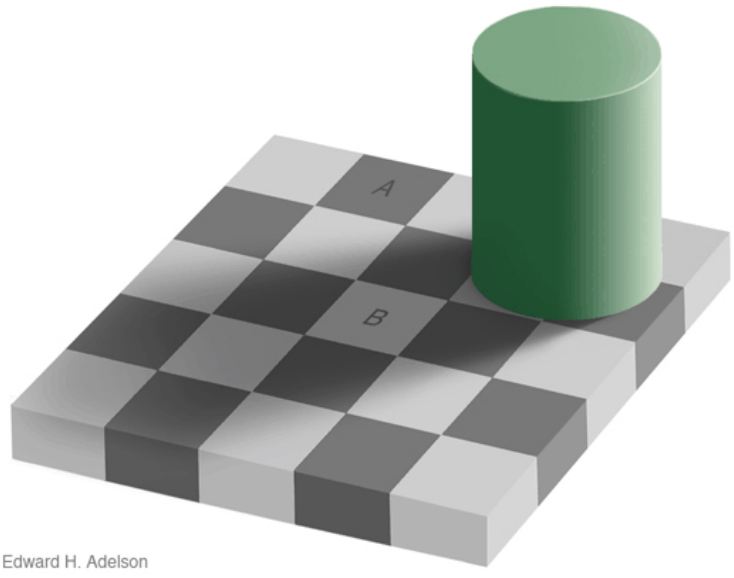


Table 3.1: Despite what your eyes are interpreting, the squares labelled “A” and “B” on the left image have the same color!

these transformations. The immutability is a key factor in performing operations in memory and in a fault-tolerant way. We found that in practice, performance bottlenecks did not occur when using this programming style.

3.5 A case study: taxis in the San Francisco Bay Area

Having now described an algorithm for computing travel time distributions in real time on a road network, we describe our validation experiments. These experiments explore two settings:

- The raw performance of the machine learning algorithm, given a limited amount of data and a computational budget,
- The performance of the Streaming Spark framework [124] in distributing computations across a cluster, and the computational performance improvement gained by additional hardware.

The performance of the algorithm is computed by asking the model to give travel time distributions on unseen trajectories, slightly in the future. The observed travel time of the trajectory is then compared with the distribution provided by the model. We measured the L1 and L2 losses between the observed travel time and the distribution mean, and the likelihood of the observed travel time with respect to the predicted travel time distribution. This is done with different amount of data and different time horizons.

The computational efficiency of the algorithm is validated in two steps. First, we demonstrate that our algorithm scales well: given twice as many computation nodes, it perform the same task about twice as fast. We also see that this algorithm is bounded by computations. Then, we demonstrate that it can sustain massive data flow rates under strict scheduling constraints: we fix a completion time of a few seconds for each time step, and we find the maximum flow rate under a given computational budget.

3.5.1 Taxis in San Francisco

Our implementation was run on a road network that corresponds to the greater San Francisco Bay Area (506,685 road links), using taxi data provided by the Cabspotting project [10]. This dataset contains GPS samples of a few thousand taxicabs emitted every minute, for more than a year. All in all, it represents hundred of millions of GPS points. We ran our algorithm on a typical day (August 12th 2010, a Tuesday) with different settings. An example of input data is given in Figure 1.2.1. A typical output of travel times provided by the algorithm is given in Figure 3.5.1.

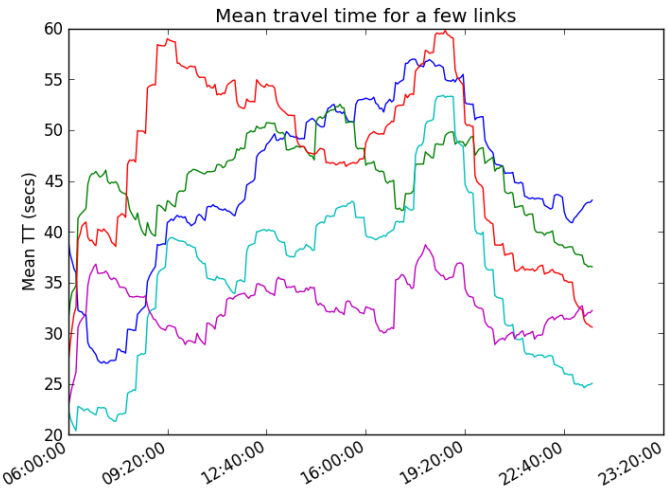


Figure 3.5.1: An example output of traffic estimates from our algorithm: mean travel times on different road links in the business district of San Francisco during a complete day.

3.5.2 Good scalability results using a large cluster

In this section, we evaluate how much the cloud implementation helped with scaling the *Mobile Millennium* EM traffic estimation algorithm. Distributing the computation across machines provides a twofold advantage: each machine can perform computations in parallel, and the overall amount of memory available is much greater.

Scaling. First, we evaluated how the runtime performance of the EM job could improve with an increasing number of nodes/cores. The job was to learn a historical traffic estimate for downtown San Francisco for a half-hour time-slice, using a large portion of the data split in one-hour intervals. This data included 259,215 observed trajectories, and the network consisted of 15,398 road links. We ran the experiment on two cloud platforms: the first was using Amazon EC2 `m1.large` instances with 2 cores per node, and the second was a cloud managed by the *National Energy Research Scientific Computing Center* (NERSC) with 4 cores per node. Figure 3.5.2 (bottom) shows near-linear scaling on EC2 until 80–160 cores. Figure 3.5.2 (top) shows near-linear scaling for all the NERSC experiments. The limiting factor for EC2 seems to have been network performance. In particular, some tasks were lost due to repeated connection timeouts.

Scaling on Streaming Spark. After having found the bottlenecks in the Spark program, we wrote another version in Streaming Spark. The two programs are strikingly similar (see program listing in Appendix B). We then benchmarked the application. We ported this application to Spark Streaming using an online version of the EM algorithm that merges in new data every five seconds. The implementation was about 200 lines of Spark Streaming

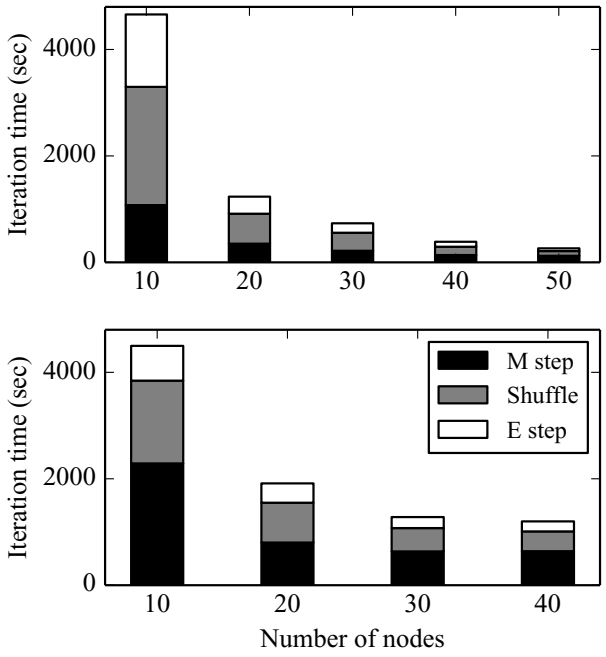


Figure 3.5.2: Experiments with Spark to build historical estimates of traffic, on NERSC (top) and Amazon EC2 (bottom).

code, and wrapped the existing map and reduce functions in the offline program. In addition, we found that only using the real-time data could cause overfitting, because the data received in five seconds is so sparse. We took advantage of D-Streams to also combine this data with historical data from the same time during the past 10 days to resolve this problem. Figure 3.5.3 shows the performance of the algorithm on up to 80 quad-core EC2 nodes. The algorithm scales almost perfectly, largely because it is so CPU-bound, and provides answers an order of magnitude faster than the previous implementation.

3.5.3 Our algorithm can be adjusted for trade-offs between amount of data, computational resources and quality of the output

We now study the accuracy of our algorithm in estimating the traffic. Even if we receive a large number of observations per day, this number is not sufficient to cover properly in real time all the road network: indeed, some sections of the road network are much less traveled than the busy downtown areas. We use several strategies to mitigate this spatial discrepancy:

- We use a prior on the Gamma distribution, itself a Gamma distribution since the

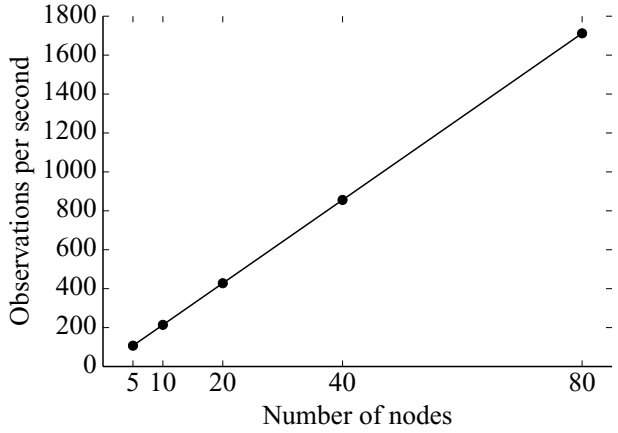


Figure 3.5.3: Experiments with Streaming Spark: rate of observation processing for different cluster sizes.

Gamma is in the exponential family and conjugate with itself. The parameters of this prior are to 70% of the speed limit in mean and 1 minute or 50% of the travel time in standard deviation, whichever is greater.

- We incorporate some data from the same day before the current time step, weighted by an exponential decay scheme: the traffic in the arterial network is assumed to change slowly enough.
- We also incorporate some data from previous days, corresponding to the same day of the week (Monday, Tuesday, etc.). Traffic is expected to follow a weekly pattern during the same month.

To summarize, a large number of observations are lumped together and weighted according to the formula:

$$w = e^{-\Delta t_{\text{day}}^{-1}(t_{\text{obs}} - t_{\text{current}})} e^{-\Delta t_{\text{week}}^{-1}(\text{week}_{\text{obs}} - \text{week}_{\text{current}})}$$

The half-time decaying factors Δt_{day} and Δt_{week} are set so that the corresponding weight is 0.2 at the end of the window.

Since the EM learning algorithm is not linear in the observations, we cannot reduce each observation to some sufficient statistics. As the algorithm moves forward in time, each observation will appear at different time steps with a different weight and needs to be reprocessed. This is a significant limitation from this approach, but it makes for a good testing ground of Streaming Spark.

Our EM algorithm can be adjusted in several ways:

- The number of weeks of data to look back (between 1 and 10)
- The time window to consider before the current observation (between 20 minutes and 2 hours)

- The number of samples generated during the E-steps (10-100)
- The number of EM iterations (1-5)
- The duration of each time step (5 seconds-15 minutes)

The observations we process all have a duration of one minute, but travel times experienced by users are usually much longer (10 minutes to a few hours). As such, a good metric for assessing the quality of a model should not be on predicting travel times for one-minute observations, but on longer distances. Hour-long travels are very likely to go be spent mostly on highways, which is not the scope of this study, and taxicabs usually make small trips (10 to 30 minutes). This is why we focus our attention to travel times between one minute (the observations) and 30 minutes (typical durations for taxi rides). As far as we know, this study of different durations is seldom done in the study of traffic, which limits any attempt to compare the performance between different algorithms.

The longer trajectories are obtained from the path inference filter. They are then cuts into different pieces of the same length (1 minute, 5 minutes, 10 minutes, 20 minutes). Each piece of trajectory is considered as an independent piece of trajectory for the purpose of travel time prediction.

We ran the algorithm with 4 different settings:

- SlidingBig: the most expensive setting (10 weeks of data, 2 hours of data, 100 EM samples, 5 EM iterations, 15 time steps), used as the baseline for comparison. Travel time estimates are produced every 20 minutes,
- SlidingBig1: uses less data (40 minutes of data),
- SlidingBig2: uses less data (10 days),
- SlidingBig3: uses the same amount of data, but performs only a single EM iteration every 4 minutes instead of 5 EM iterations every 20 minutes,
- SlidingBig4: uses the same amount of data, but generates only 10 EM samples for each observation

For all these experiments, the prior was fixed.

We now compare the results obtained with the different experiments. We first turn our attention to the L1 loss in Figure 3.5.4. As expected, the best performance is obtained for experiment SlidingBig, which uses the most data. Interestingly enough, the best performance is obtained for travels of medium length (4-11 minutes), and not for short trajectories. This can be explained by the conversion step that transforms trajectory readings on partial links into weighted observations on complete links. The relation between link travel time and location on a link is more complex than a linear weighting. Nevertheless, the model gives relatively good performance by this simple transform. When a vehicle is stopped at a red light, it does not travel along the link but still has a non-zero travel time. In this case, the

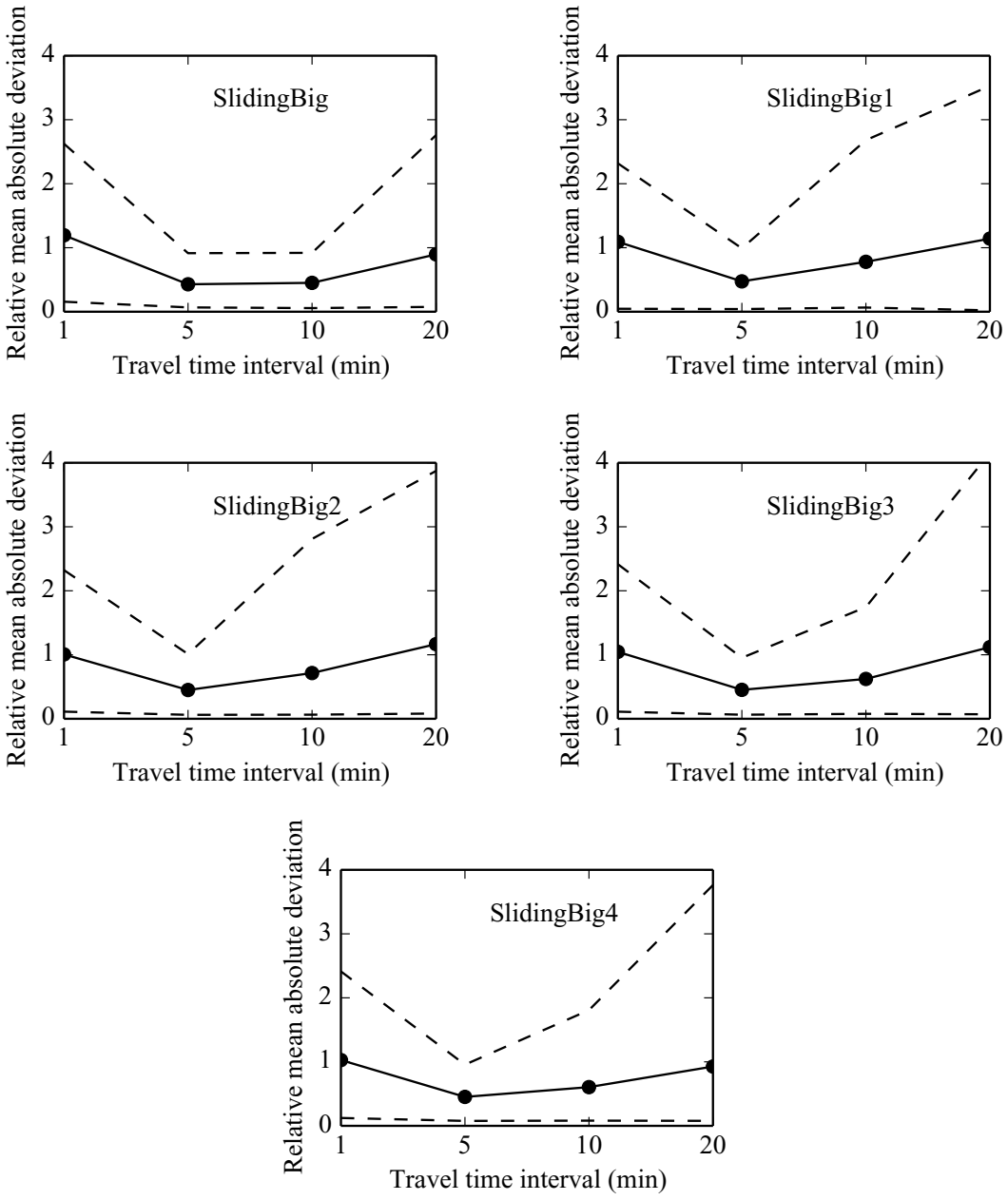


Figure 3.5.4: L1 residuals for different settings and for different travel times. The dashed lines indicate the 95% confidence interval. On the x axis, the travel time of the the observations considered for this metric.

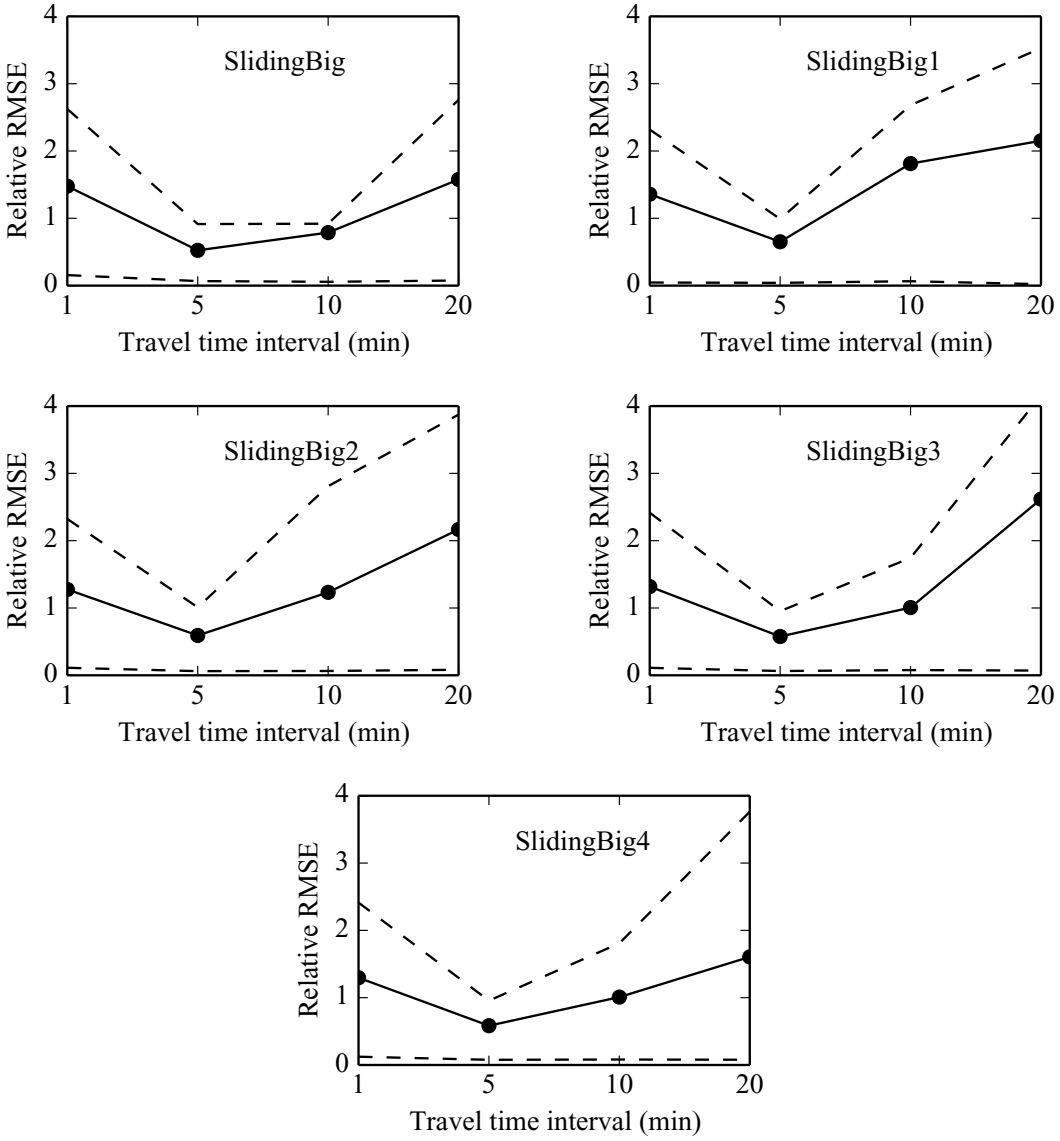


Figure 3.5.5: L2 residuals for different settings and for different travel times. The dashed lines indicate the 95% confidence interval.

weight of an observation is taken to be half of the total travel time of the link. In particular, the relative error increases as the duration (and the length) of travels increases. Performance is not too different between experiments, which suggests some even smaller amount of data could be considered.

The results for the L2 loss, presented in Figure 3.5.5, provide similar, if not more acute, results. The RMSE is lowest for small to medium travels (in the range of 3-10 minutes).

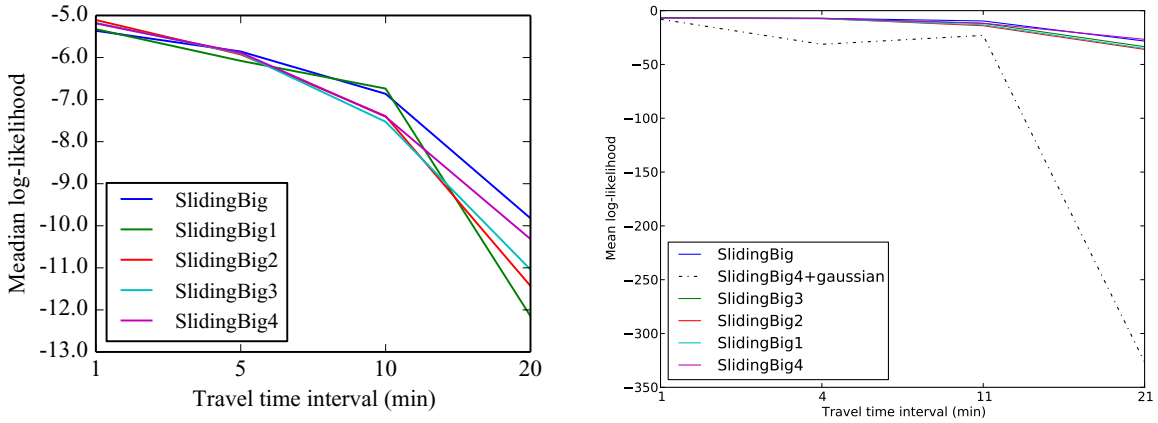


Figure 3.5.6: Log-likelihood of unobserved trajectories, for different trajectory lengths and different settings. The median is presented on the left, while the mean is presented on the right.

A probabilistic metric (the log-likelihood) gives a different insight, as shown in Figure 3.5.6. The model best explains the data for very short travel times (similar to what is was trained on) but its precision falls down as the length of trajectories increases. All in all, this results should not be unexpected: this model with independent links cannot take into account the correlations that occur due to light synchronizations or drivers' behavior. As such, the probability density of a longer travel rapidly dilutes as the number of links increases As we saw with our study of L1 and L2 errors, the mean travel time becomes the only significant value of interest for longer travel times. In the light of this result, there seems to be little to gain by modeling travel time with physically realistic, link-based, independent distributions, as the independence assumption will strongly weigh on the quality of the travel time for longer travels. Instead, we recommend focusing effort on simpler models of travel times that take into account the correlations between links. We also present in Figure 3.5.6 (Right) the mean of the log-likelihood. As can be seen in this plot, the Gaussian distribution is significantly worse than the Gamma distribution: from inspection of the data, we could conclude that it performs very badly with long tail observations (unusually long stops at red lights). Furthermore, its symmetric nature causes a significant portion of the probabilistic mass to be assigned to negative travel times: a Gaussian distribution cannot at the same time have a small mean, a high standard deviation and mostly positive values. This experiment should serve as another confirmation that modelling travel time noise with a Normal distribution is not only unrealistic, but also leads to erratic results in the face of experimental data.

Conclusion

As datasets grow in size, some new strategies are required to perform meaningful computations in a short amount of time. We explored the implementation of a large-scale state estimation in near-real-time using D-Streams, a recently proposed streaming technique. Our traffic algorithm is an Expectation-Maximization algorithm that computes travel time distributions of traffic by incremental online updates. This approach was validated with a large dataset of GPS traces. This algorithm seems to compare favorably with the state of the art and shows some attractive features from an implementation perspective. When distributed on a cluster, this algorithm scales to very large road networks (half a million road links, tens of thousands of observations per second) and can update traffic state in a few seconds.

In order to foster research in systems and in traffic, the authors have released the code of Spark Streaming [6], the code of the EM traffic algorithm [9], and the dataset used for these experiments [9].

Chapter 4

Large scale estimation of travel time correlations

Crombile maintenant ne venait plus au conseil; levé à cinq heures du matin, il travaillait dix-huit heures à son bureau et tombait epuisé dans sa corbeille où les huissiers le ramassaient avec les papiers qu'ils allaient vendre aux attachés militaires de l'empire voisin.

Anatole France, *L'île des pingouins*

4.1 Large-scale statistical inference of traffic

A common problem in trip planning is to make a given deadline, for example arriving at the airport within 45 minutes. Most routing services available today minimize the expected travel time, but do not provide the most *reliable* route, which accounts for the variability in travel times. Given a time budget, a routing service should provide the route with highest probability of on-time arrival, as posed in stochastic on-time arrival (SOTA) routing [97]. Such an algorithm requires the estimation of the *statistical distributions* of travel times, or at least of their means and variances, as done in [75]. Today, only few traffic information platforms are available for the arterial network (the state of the art for highway networks is more advanced) and they do not provide the *statistical distributions* of travel times. The main contribution of the chapter is precisely to addresses this gap: we present a scalable algorithm for learning path travel time distributions on the entire road network using probe vehicle data.

The increasing penetration rate of probe vehicles provides a promising source of data to learn and estimate travel time distributions in arterial networks. At present, there are two general trends in estimation of travel times using this probe data. One trend, from kine-

matic wave theory (see [56, 130]), derives analytical probability distributions of travel times and infer their parameters with probe vehicle data. These approaches are computationally intensive, which limits their applicability for large scale networks. The other trend, seen in large-scale navigation systems such as Google Maps, provides coarser information, such as expected travel time, but can scale to world-sized traffic networks.

We bridge the two trends by creating a travel time estimator that (i) provides full probability distributions for arbitrary paths in real-time (sub-second), (ii) works on networks the size of large cities (and perhaps larger) (iii) and accepts an arbitrary amount of input probe data. The model uses a data-driven model which is able to leverage physical insight from traffic flow research. Data-driven models, using dynamic Bayesian networks [54], nearest neighbors [113] or Gaussian models [118] show the potential of such methods to make accurate predictions when large amounts of data are available.

4.1.1 Physical insights derived from high-frequency trajectories

Arterial traffic is characterized by important travel time variability amongst users of the network. This variability is mainly due to the presence of traffic lights and other impediments such as pedestrian crossings and garbage trucks or buses which cause a fraction of the vehicles to stop while others do not. Arterial traffic research [56] suggests that the detection of stops explains most of the variability in the travel time distribution and underline the multi-modality of the distributions. We studied high-frequency data collected during an experiment in San Francisco (see Section 2.6 for more details about this experiment). This experiment provided detailed trajectory information along stretches of the main arterial network in San Francisco. After mapping the GPS points to the road, and reconstructing the trajectories, we had a very precise account of the patterns of traffic on the ground. In particular, we reached the following conclusions for the travel times on each link:

- **discrete number of modes.** The travel times follow a multi-modal distribution across each block. There are generally two main modes that correspond to the light signal at the end of the intersection (see Figure 4.1.2). This distribution can be predicted by simple physical models that take into account the dynamics of cars [55, 56].

- **lights correlation.** Traffic lights on major streets may be synchronized to create "green waves" (Figure 4.1.1): a vehicle which does not stop on a link is likely to not stop on the subsequent link. A different vehicle arriving 10 seconds later may hit the red light on the first link and have a high probability to stop on the subsequent link. This phenomenon is analyzed in [94] using a Markov model with two modes ("slow" and "fast"). The synchronization scheme for traffic lights is managed at the level of the traffic district. However, this synchronization pattern is usually complex: it combines traffic sensing equipment with handwritten rules and random pedestrian events, and each district has a different set of rules. From the perspective of a traveling vehicle, we consider the synchronization of the lights a *discrete stochastic process*, the parameters

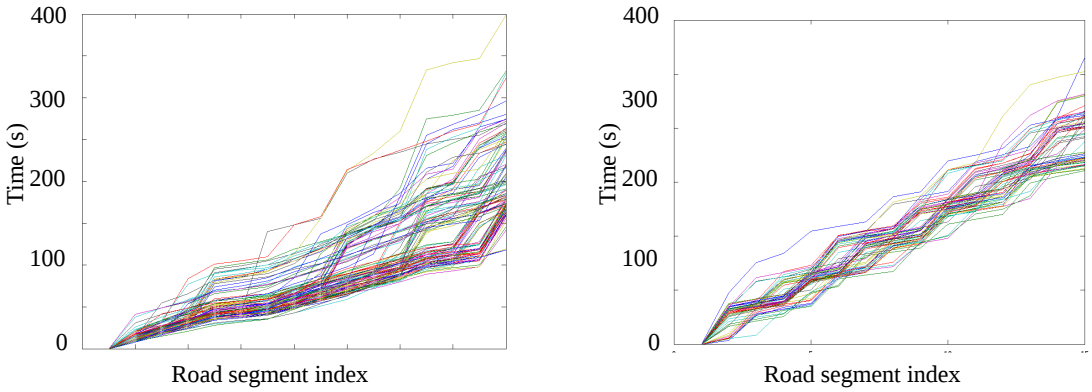


Figure 4.1.1: Sets of trajectories collected around two main arterial in San Francisco: northbound Franklin street on the left and northbound Van Ness avenue on the right. The distance is normalized for each of the road link by the link length. Franklin street exhibits much more variability in the travel times due to weaker correlations between the traffic lights. Conversely, vehicles on Van Ness avenue are severely constrained and cannot drive too fast or too slow due to the scheduling of the traffic lights. We wish to capture this variability in travel times in our model.

of which can be inferred from observing enough vehicles. We will see in Section 4.2 how we can reliably model this process as Markov graph.

- **travel time correlation inside each mode.** It is well known that vehicles at lights tend to travel together at the same speed. Besides the number of stops, travel times may be correlated for the following reasons: (i) the behavior of individual drivers may be different: some car may travel notably faster than some others, (ii) congestion propagates on the network, making neighboring links likely to have similar congestion levels. If a driver experiences a longer than usual travel time on a link because of heavy traffic, she can will likely experience heavy traffic on the subsequent links. This is why, if we were to fix a pattern of red and green lights across the whole network, we could assume that all the travel times are still correlated. This is referred to as inner mode correlation.

4.1.2 The modeling pipeline

We leverage these insights to develop the traffic estimation algorithm presented in this chapter: an end-to-end, scalable model for inferring path travel time distributions, referred to as a "pipeline" (Figure 4.1.3). It consists of a *learning algorithm* and an *inference algorithm*.

The *learning algorithm* consists in the following steps.

• Section 4.2: a Stop-go algorithm detects the number of stops on a link and compresses

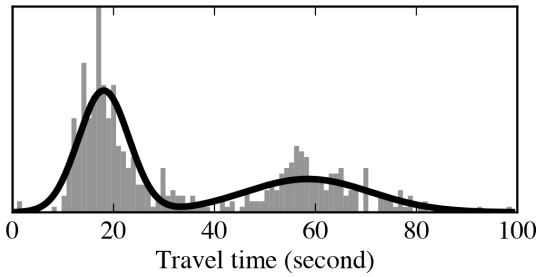


Figure 4.1.2: Histogram of travel times collected on a road link in San Francisco, fit (solid line) using a mixture of two Gaussian distributions. The two modes corresponding to the green light and to the delay of the red light are clearly visible.

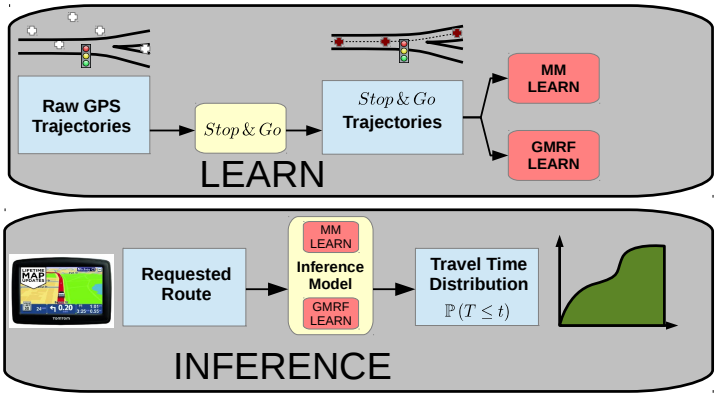


Figure 4.1.3: Pipeline of the travel-time estimation model. The learning runs offline and finds optimal parameters for the HMM and GMRF models. The inference runs online and uses the learned parameters to produce travel time distributions on input paths.

the GPS traces[1] into values of travel times on traversed links and corresponding number of stops.

• Section 4.3.1: a *Markov model* (MM) captures the correlations of stopping / not stopping for consecutive links. It characterizes the probability to stop / not stop on a link given that the vehicle stopped or did not stop on the previous link traversed. The Stop-Go splitting procedure (presented in the same section) produces a set of labeled data to train the Markov model.

• Section 4.3.2: a *Gaussian Markov Random Field* (GMRF) captures the correlations in travel times between neighboring links, given the number of stops on the links. There is a significant body of prior work in the field of learning with graphical models [67, 68], especially for learning problems on sparse GMRFs [47, 80]. Most of these algorithms do

[1]Before feeding raw GPS points to the Stop-Go model, each coordinate is mapped to a link and an offset distance from the beginning of the link. We use the path-matching algorithm presented in Chapter 2.

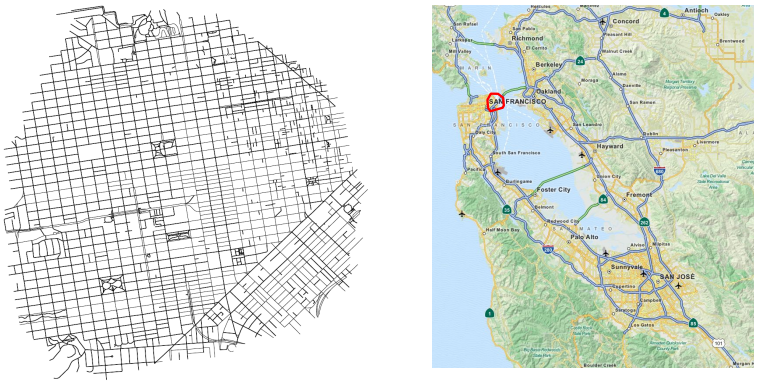


Figure 4.1.4: On the left, the road network used in the experiment. On the right, a blown-up subsection (corresponding to the red polygon in the full network). The blown-up subsection illustrates the level of detail from the map. The full road network comprises 505,000 road links.

not scale linearly with respect to the dimension of the data, and are unsuitable for very large problems (hundreds of thousands of variables). In particular, it becomes practically impossible to store the entire covariance matrix, so even classical sub-gradient methods such as [38] would require careful engineering.

We exploit the (near) planar structure of the underlying graphical model to more efficiently obtain (approximate) algorithms that scale *linearly* with the size of the network. Our algorithm leverages efficient algorithms to compute the Cholesky decomposition of the adjacency matrix of planar graphs [29]. Our results can be extended to other physical systems with local correlations.

After the learning, we proceed to the *inference algorithm*: we compute the travel time distributions for arbitrary paths in the network (Section 4.4). While exact inference on this model is intractable due to the number of possible states, we exploit the underlying structure of the graphical model to approximate the requested travel time distribution.

Section 4.5 illustrates the accuracy and scalability of our model on a 505,000 road link network spanning the San Francisco Bay Area.

Our code, as well as a showcase of the model running on San Francisco, is available at `http://traffic.berkeley.edu/navigateSF`.

4.2 Trajectory compression with the Stop and Go model

The stops due to traffic signals and other factors (double parking, garbage trucks and so on) represent one of the main source of variability in urban travel times. More generally, consider that a link can have m different discrete states. In practice, this number is inferred from the

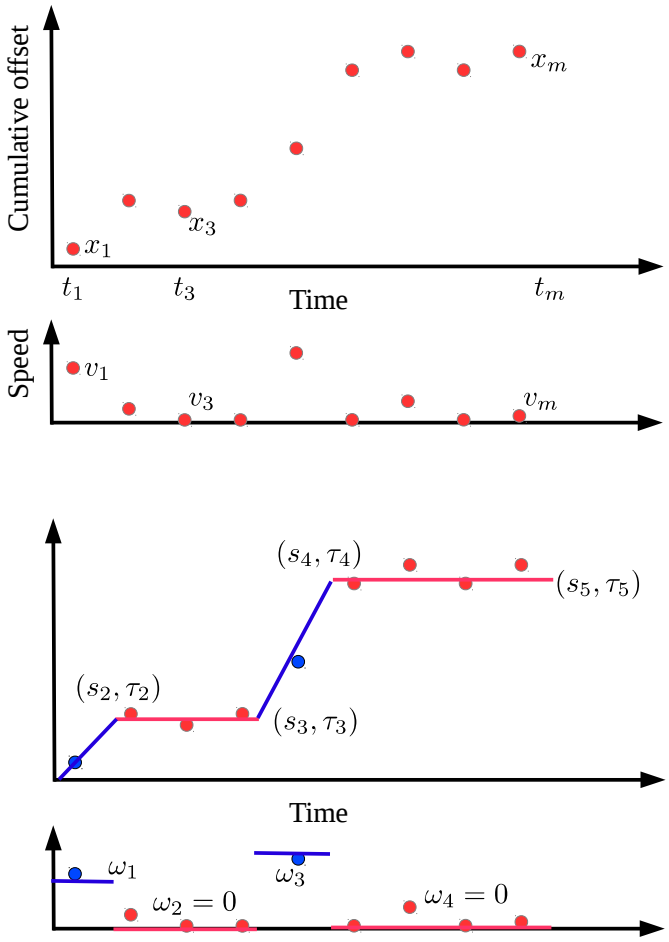


Figure 4.2.1: Representation of the data compression algorithm used in the Stop and Go filter. In the top figure, the raw information about location and speed are given. In the bottom figure, the Stop and Go filter has grouped together the points at zero speed (stop detection) and assigned a unique location for the stopped intervals.

data and is different for each link. For a vehicle traveling on link l, the *state* $s_l \in \{1, m\}$ of the trajectory is defined as the number of stops on the link. The following algorithm estimates the number of stops given a set of noisy GPS samples from the trajectory on link l. We consider a generic trajectory on a generic link and drop indices referring to the trajectory and link for notation simplicity.

The trajectory of the vehicle is represented by an offset function $T : [0, \tau] \to \mathbb{R}_+$, representing the distance from the beginning of the link to the location of the vehicle at time t (see Figure 4.2.3 for an example). The noisy GPS observations are defined by the times $0 = t_0 < \cdots < t_J \leq \tau$, and the corresponding offsets $x_j = T(t_j) + \epsilon_j$, where $\epsilon_j \sim \mathcal{N}(0, \sigma^2)$

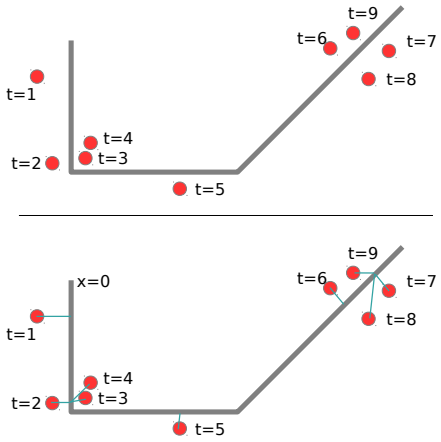


Figure 4.2.2: The forward bias problem with the path inference filter. The solid line is the path followed by the vehicle, and the solid red dots are consecutive GPS observations made along this path. The path inference filter assumes that vehicles only go forward on a link. Due to noise in the observations, the points $t = 2$ and $t = 7$ will be further along than the points $t = \{3, 4\}$ and $t = \{8, 9\}$ respectively. This will cause all these points to be assigned the same location as $t = 2$ and $t = 7$. Thus, the output of the path inference filter is biased towards a location that is further along than the true location, and towards showing a vehicle than moves while it is stopped. The Stop and Go model corrects this issue.

are independent and identically distributed zero mean Gaussian random variables representing the GPS noise. While the noise is not Gaussian in practice, this model still gives good results.

We process the observations $(t_j, x_j)_{j=0,\dots,J}$ to obtain *stop and go* trajectories of the probe vehicles: the trajectory of a vehicle alternates between phases of *Stop* during which the velocity of the vehicle is zero and *Go* during which the vehicle travels at positive speed. The number of *Stop* phases represents the state of the trajectory. We assume that the sampling frequency is high enough that the speed between successive observations (t_j, x_j) and (t_{j+1}, x_{j+1}) is constant[2] and denoted v_j. Note that speeds are rarely provided by GPS devices or are too noisy to be valuable for estimation.

Maximizing the log-likelihood of the observations is equivalent to solving the following optimization problem

$$\underset{(v_j)_{j\in\{0,\dots,J-1\}}}{\textbf{minimize}} \frac{1}{2}\sum_{j=0}^{J-1}\left(x_{j+1} - x_0 - \sum_{j'=0}^{j} v_{j'}(t_{j'+1} - t_{j'})\right)^2$$

[2]The assumption is further justified by traffic modeling [56] which commonly assumes that each *Go* phase has constant speed

in which the x_j and t_j variables are known and fixed. This is a typical least-square optimization problem in $(v_j)_j$, which we conveniently write in matrix form as:

$$\underset{v}{\textbf{minimize}} \frac{1}{2}||Av - b||_2^2, \tag{4.2.1}$$

with the notation $v = (v_j)_{j\in\{0,\dots,J-1\}}$, $b = (x_j - x_0)_{j\in\{1,\dots,J\}}$ and A is the lower triangular matrix whose entry on line $i \in \{0,\dots,J-1\}$ and column $k \in \{0,\dots,i-1\}$ is given by $t_{k+1} - t_k$.

To detect the *Stop* phases, we add a l_1 regularization term, with parameter λ to Problem (4.2.1). The resulting optimization problem is known as the LASSO [112] and reads

$$\underset{v}{\textbf{minimize}} \frac{1}{2}||Av - b||_2^2 + \lambda||v||_1, \tag{4.2.2}$$

The solution of (4.2.2) is typically sparse [112], which means that there is a limited number of non-zero entries, corresponding to the *Go* phases of the trajectory. The solution is used to compute the number of *Stops*. Figure 4.2.4 illustrates the result of the trajectory estimation and Stop detection algorithm. This algorithm, which works on the full trajectory of the vehicle, can accurately reconstruct the Stop phases of the vehicle.

Remark: [Data compression] In our dataset[3], the average number of GPS points sent by a vehicle on a link is 9.6. The algorithm reduces the GPS trace to: entrance time in the link, travel time, number of stops. The amount of data to be processed by the subsequent algorithms is reduced by a factor of almost 10, which is crucial for large scale applications which process large amounts of historical data.

Remark: [Fixing λ] In Equation (4.2.2), λ acts as a trade-off between the sparsity of the solution and the fit to the observations. Cross-validation is not appropriate in our setting to fix this parameter as it would require one to decimate the trajectory and use some observations for the learning and others for the validation. Instead, λ is chosen by computing the *Bayesian Information Criterion* (BIC), using [132] to estimate the number of degrees of freedom. A LARS implementation [41] allows efficient computation to choose λ and compute the corresponding solution.

4.3 A graphical model for travel times

We develop a travel time model which exploits the compressed data returned by the Stop-Go model (number of stops and travel time experienced per link) and that takes into account our simplifying assumptions about travel times on a road network (Figure 4.3.1). The model captures the state transition probabilities: the probability of the number of stops on a link given the number of stops on the previous link(s) of the trajectory. It also models the

[3]The number of GPS points per link depends on the level of congestion (vehicles spend a longer amount of time on each link), average length of the links of the network and sampling frequency of the probe vehicles.

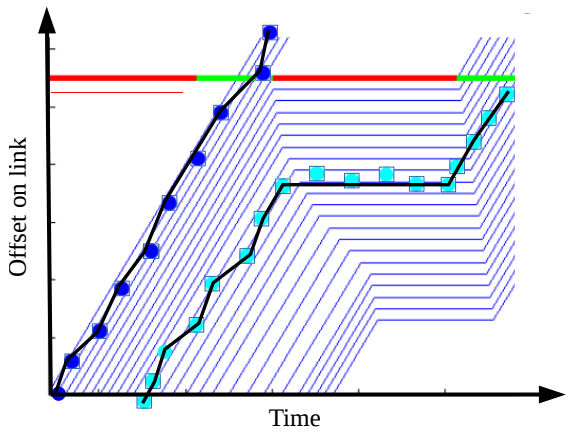


Figure 4.2.3: Two typical scenarios in a signaled intersection. The horizontal axis is time and the vertical axis is cumulative offset (shifted so that the zero corresponds to the end of the link). The red and green color correspond to the time during which the light is red and green, respectively: cars can only go through when the light is green. The solid dark blue dots represent a typical observation when the light is green. The light blue dots represents a typical observation when the light is red: the vehicle is forced to wait at zero speed for the light to become green again.

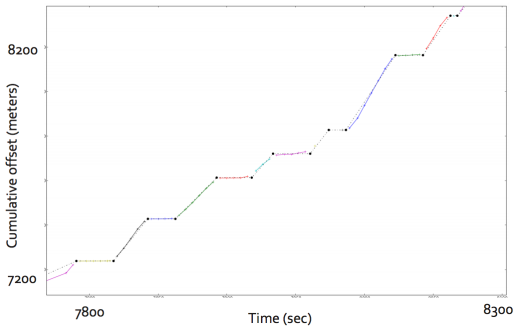


Figure 4.2.4: Example from real data of trajectory estimation. In our implementation, the trajectory is further compressed into alternating segments of Stop motion (in which the speed is null) and Go motion (in which the speed is uniform). The solid points indicate the transition points between Stop segments and Go segments.

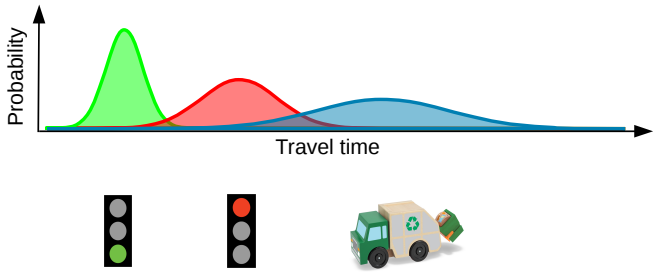


Figure 4.3.1: Picture representations about the travel times over a road link: the travel time is a mixture of (typically) three Gaussian distributions: one mode for the travel times corresponding to a green light, one mode for the travel times corresponding to a red light, and one mode for complex random events (pedestrian, traffic occlusion, garbage truck, bus, etc.)

correlations of travel times for neighboring links given their state (number of stops). The travel time model is a combination of two models parametrized by a set of values denoted by θ:

• A directed Markov model of discrete state random variables gives the joint probability distribution of the link states

$$\pi_\theta\left(\{S_l\}_{l\in\mathcal{L}}\right)$$

• A Gaussian Markov random field gives the joint distribution of the travel times

$$\pi_\theta\left(\{X_{l,s}\}_{l\in\mathcal{L},s\in\{1,\cdots m\}}\right)$$

Figure 4.3.2 presents the graphical model that encodes the dependencies between the link travel time Z_l, the link states S_l and the conditional travel times $X_{l,s}$. The total travel time experienced by a vehicle on link l is

$$Z_l = \sum_{s=0}^{m-1} X_{l,s}\mathbf{1}\left\{S_l = s\right\}$$

The left portion of the figure shows a subset of the GMRF of travel time variables, and the right portion shows a subset of the Markov chain of states.

The graphical model shows that conditioning on the states experienced along a path allows one to compute the path travel time by summing over the corresponding variables in the GMRF. Further, when one simultaneously conditions on the link travel times Z_l and the link states S_l, then the two models become independent, which allows one to learn the models parameters separately. The rest of this section details the modeling and learning of the two models.

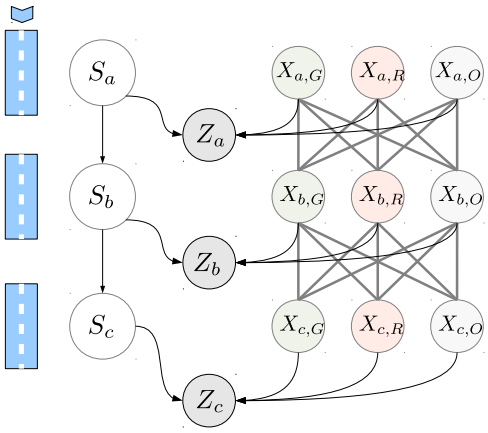


Figure 4.3.2: Graphical model of the travel times on a stretch of three one-way road links (going from top to bottom of the figure). The travel times of each link Z_l depend on two conditionally independent set of variables: the discrete state variable (e.g. number of stops) S_l and the conditional travel times $X_{l,s}$ where $s \in \{\text{Go, Red, Other}\}$. The arrows between the variables S_l represent the conditional dependencies between the random variables, after taking into account the Markov assumptions. The time variables $X_{l,s}$ make a Markov Random Field. The solid lines between the variables represent the conditional dependencies.

4.3.1 Markov model for state transitions

We consider the link state variables $\{S_l\}_{l,\in\mathcal{L}}$. Each variable S_l has support $\{1 \cdots m\}$. In our experiment, vehicles travel along paths in the road network and cannot arbitrarily jump from one road to another unconnectecd road. We formalize this notion of valid path

Valid path. Given a directed graph $\mathcal{G}$, valid path is an ordered sequence $p = (l_1 \cdots l_B) \in \mathcal{L}^B$ of distinct elements so that for all consecutive pairs $\left(l, l^{'}\right)$ from p, we have an edge $l \to l^{'}$ in the graph $\mathcal{G}$.

Consider a valid path $p = (l_1 \cdots l_B) \in \mathcal{L}^B$ in the road network. For simplicity, all the l_i are assumed to be distinct (handling this case is not an issue in practice). Given the path p of a vehicle, the variables $\{s_{l_i}\}_{i\in 1\cdots M}$ have a Markov property: given the state $s_{l_{i-1}}$ of the *upstream link* l_{i-1}, the conditional state $S_{l_i}|S_{l_{i-1}}$ is independent of the state of other upstream links:

$$\mathbb{P}\left(S_{l_i}|S_{l_{i-1}}, \ldots, S_{l_0}\right) = \mathbb{P}\left(S_{l_i}|S_{l_{i-1}}\right) \tag{4.3.1}$$

This Markov model consists in discrete variables, with values in $1 \cdots m$. For each link l, we parametrize the model using an initial probability vector

$$\pi_l\left(s\right) = \mathbb{P}(S_l = s) \tag{4.3.2}$$

and a transition probability matrix:

$$\pi_{u\to l}\left(s_u, s_l\right) = \mathbb{P}(S_l = s_l | S_u = s_u) \tag{4.3.3}$$

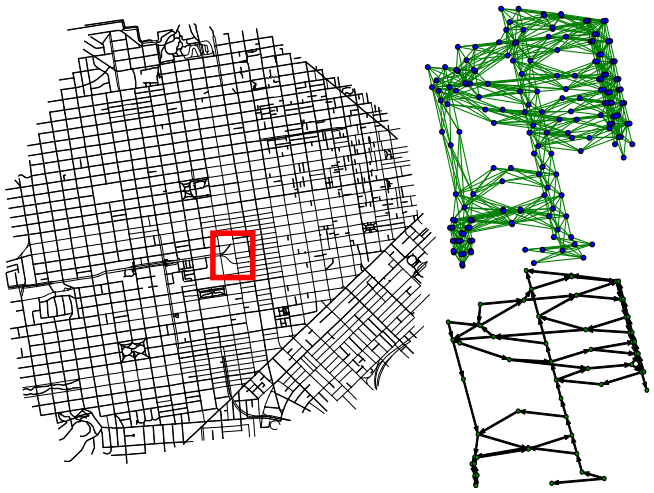


Figure 4.3.3: Example of travel time graph for the San Francisco area. The red box on the left has been expanded in the undirected Gaussian Markov Random Field (top right) and (directed) Discrete Markov Model (bottom right). In this example the number of discrete states has been fixed to $m = 3$ for all nodes.

here $\pi_{u\to l}(s_u, s_l)$ is the probability that l is in state s_l given that the upstream link u is in state s_u.

We learn the initial probability vector π_l and the transition probability matrices $\pi_{u\to l}$ of the Markov Chain by maximizing the likelihood of observing $(s^{(n)}, y^{(n)})$. The log-likelihood of all A observations of states is given by

$$ll = \sum_{a=1}^{A} \sum_{n=1}^{M^{(a)}} \left(\log\left(\pi_{l_0^{(n)}}\left(s_0^{(n)}\right)\right) + \sum_{u\to l} \log\left(\pi_{u\to l}\left(s_u^{(n)}, s_l^{(n)}\right)\right)\right)$$

The parameters that maximize the log-likelihood are:

$$\begin{aligned} \pi_{u\to l}(s_u, s_l) &\propto \bar{T}^{u\to l}_{s_u,s_l} \\ \pi_l(s) &\propto \bar{\pi}^l_s \end{aligned}$$

where $\bar{\pi}^l_s$ are the empirical counts of initial states and $\bar{T}^{u\to l}_{s_u,s_l}$ are the empirical transition counts. This solution corresponds to transitions and initial probabilities that are consistent with the empirical counts of initial states and transitions.

Modeling note: While this Markov model is directed, it is *not* a Bayesian Graphical Model because it may contain cycles between the variables. However, it is *marginally* a graphical model with Markov properties: for any valid path p in the network with no self-loop, the chain of variables $S_{l_1} \cdots S_{l_M}$ encodes a valid discrete Markov model distribution over these variables. The directed graph structure is an efficient way to compress all the information about the marginal distributions of the valid subsets of variables.

4.3.2 GMRF model for travel time distributions

We now present the model that describes the correlations between the travel time variables. We assume that the random variables $(X_{l,s})_{l\in\mathcal{L},s\in[1,m]}$ are Gaussian[4], and they can be stacked into one multivariate Gaussian variable $X \sim \mathcal{N}(\mu, \Sigma)$ of size $r = m\,|\mathcal{L}|$, with mean μ and covariance Σ. Recall that $X_{l,s}$ is the travel time on link l conditioned on link state s, where $s \in [1, m]$. This travel time is a Gaussian random variable with mean $\mu_{l,s}$ and variance $\sigma_{l,s}$.

From the factorization property given by the Hammersley-Clifford Theorem (CITE), it is well known that the *precision matrix* $Q = \Sigma^{-1}$ encodes the *conditional dependencies* between the variables. Since a link is assumed to be conditionally correlated only with its (geographical) neighbors, the precision matrix is very sparse. Furthermore, this sparsity pattern is of particular interest: its pattern is that of a graph which is nearly planar. We take advantage of this structure to devise efficient algorithms that (1) estimate the precision matrix given some observations, and (2), infer the covariance between any pair of variables.

As mentioned earlier, we have a set of A trajectories obtained from GPS data. After map-matching and trajectory reconstruction (Section 4.2), the set of observed trajectories $\{W_a\}_{a=1,\cdots,A}$ are sequences of observed states and variables (travel time)[5]:

$$\begin{aligned} W_a &= (l_1, s_1)\ (l_2, s_2)\ \cdots (l_{M_a}, s_{M_a}) \\ x^a &= \left(x^a_{l,s}\right)_{(l,s)\in W_a} \end{aligned}$$

In our notations, x^a is an observation of the random vector $X_a = (X_i)_{i\in W_a}$, which is a M_a-dimensional marginal (or subset) of the full distribution X. Hence X_a is also a multivariate Gaussian with mean $\mu_{(W_a)}$ and covariance $\Sigma_{(W_a)}$ obtained by dropping the irrelevant variables (the variables that one wants to marginalize out) from the mean vector μ and the covariance matrix Σ. Its likelihood is thus the likelihood under the marginal distribution:

$$\begin{aligned} &\log \pi\left(x^a; \mu_{(W_a)}, \Sigma_{(W_a)}\right) = \\ &C - \tfrac{1}{2}\log\left|\Sigma_{(W_a)}\right| - \tfrac{1}{2}\left(x^a - \mu_{(W_a)}\right)^T \Sigma^{-1}_{(W_a)}\left(x^a - \mu_{(W_a)}\right), \end{aligned} \tag{4.3.4}$$

where C is a constant which does not depend on the parameters of the model.

The problem of estimating the parameters of the model $\theta = (\mu, \Sigma)$ from the i.i.d. set of observations $\mathcal{D} = \{W_a\}_{a=1,\cdots,A}$ consists in finding the set of parameters $\theta^\star = \{\mu^\star, \Sigma^\star\}$ that maximize the sum of the likelihoods of each of the observations[6]:

$$l\left(\theta|\mathcal{D}\right) = \sum_{a=1}^{A} \log \pi\left(x^a; \mu_{(W_a)}, \Sigma_{(W_a)}\right) \tag{4.3.5}$$

[4]While Gaussian travel times can theoretically predict negative travel time, in practice, these probabilities are virtually zero, as validated in Section 4.5.

[5]One could consider that the states and the variables are not directly observed, but guessed after a complicated process. The Expectation-Maximization algorithm could be run to estimate the value of the state and of the variable.

[6]The independent and identically distributed (i.i.d.) assumption is defined on the pair of marginal observation and masking subset.

Unfortunately, the problem of maximizing (4.3.5) is in general not convex and may have multiple local minima since we have only partially observed variables x^a. A popular strategy in this case is to *complete* the vector (by computing the most likely completion given the observed variables). This algorithm is called the *Expectation-Maximization* (EM) procedure. In our case, the EM procedure is not a good fit for two reasons:

- Since we observe only a small fraction of the values of each vector, the vast majority of the values we would use for learning would be sampled values, which would make the convergence rate dramatically slow.
- The data completion step would create a complete $n-$size sample for each of our observation, thus our complexity for the data completion step would be $\mathcal{O}(Nn)$, which is too large to be practical.

Instead, we solve a related problem by computing sufficient statistics from all the observations. Consider the simpler scenario in which all data has been observed, and denote the empirical covariance matrix by $\tilde{\Sigma}$. The maximum likelihood problem to find the most likely precision matrix is then equivalent to:

$$\underset{S}{\textbf{minimize}} \ -\log|Q| + \operatorname{Tr}\left(Q\tilde{\Sigma}\right) \tag{4.3.6}$$

under the structured sparsity constraints $Q_{uv} = 0 \ \forall \ (u,v) \notin \mathcal{E}$. The objective is not defined when Q is not positive definite, so the constraint that S is positive definite is implicit. A key point to notice is that the objective only depends on a restricted subset of terms of the covariance matrix:

$$\operatorname{Tr}\left(Q\tilde{\Sigma}\right) = \sum_{(u,v)\in\mathcal{E}} Q_{uv}\tilde{\Sigma}_{uv}$$

This observation motivates the following approach: instead of considering the individual likelihoods of each observation individually, we consider the covariance that would be produced if all the observations were aggregated into a single covariance matrix. This approach discards some information, for example the fact that some variables are more often seen than others. However, it lets us solve the full covariance Problem (4.3.6) using partial observations. Indeed, all we need to do is estimate the values of the coefficients $\tilde{\Sigma}_{uv}$ for (u,v) a non-zero in the precision matrix. We present one way to estimate these coefficients.

Let N_i be the number of observations of the variable X_i: $N_i = \operatorname{card}(p : i \in W_a)$. Combining all the observations that come across X_i, we can approximate the mean of any function $f(X_i)$ by some empirical mean, using the N_i samples:

$$\mathbb{E}_i\left[f(X_i)\right] = \frac{1}{N_i}\sum_{p:i\in W_a} f\left(x_i^p\right) \tag{4.3.7}$$

Similarly, we define the number of observations of both X_i and X_j:

$$N_{i\cap j} = \operatorname{card}\left(\{p \ : \ i \in W_a \text{ and } j \in W_a\}\right)$$

We can approximate the mean of any function $f(X_i, X_j)$ of X_i, X_j, using the set of observations that span both variables X_i and X_j:

$$\mathbb{E}_{i\cap j}\left[f\left(X_i, X_j\right)\right] = \frac{1}{N_{i\cap j}} \sum_{p\,:\, i,j\in W_a} f\left(x_i^p, x_j^p\right) \tag{4.3.8}$$

Using this notation, the empirical mean is $\hat{\mu}_i = \mathbb{E}_i\left[X_i\right]$. Call $\hat{\Sigma}$ the *partial empirical covariance matrix* (PECM):

$$\hat{\Sigma}_{ij} = \begin{cases} \mathbb{E}_{i\cap j}\left[X_i X_j\right] - \mathbb{E}_i\left[X_i\right]\mathbb{E}_j\left[X_j\right] & \text{if } (i,j)\in\mathcal{E} \\ 0 & \text{otherwise} \end{cases}$$

Using this PECM as a proxy for the real covariance matrix, one can then estimate the most likely GMRF by solving the following problem:

$$\underset{S}{\textbf{minimize}} \;\; -\log|Q| + \left\langle Q, \hat{\Sigma}\right\rangle \tag{4.3.9}$$

Note that this definition is asymptotically consistent: in the limit, when an infinite number of observations are gathered ($N_{ij}\to\infty$), the PECM will converge towards the true covariance: indeed $\hat{\Sigma}_{ij} \to \mathbb{E}\left[X_i X_j\right] - \mathbb{E}\left[X_i\right]\mathbb{E}\left[X_j\right]$ and for all Q:

$$\left\langle Q, \hat{\Sigma}\right\rangle \to \left\langle Q, \mathbb{E}\left[XX^T\right] - \mathbb{E}\left[X\right]\mathbb{E}\left[X\right]^T\right\rangle$$

.

Unfortunately, the problem is not convex because $\hat{\Sigma}$ is not necessarily positive semi-definite (although it is if the number of samples is large enough), since the variables are only partially observed. For instance, if we have a partially observed bivariate Gaussian variable X: $(10,10)$, $(-10,-10)$, $(1,\star)$, $(-1,\star)$, $(\star,1)$, $(\star,-1)$, the empirical covariance matrix $\hat{\Sigma}$ has diagonal entries $(51,51)$ and off-diagonal entries $(100,100)$. Its eigenvalues are $-49, 151$ hence it is not definite positive.

There is a number of ways to correct this. The simplest we found is to scale all the coefficients so that they have the same variance:

$$\hat{\Sigma}_{ij} = \begin{cases} \sqrt{\alpha_{ij}}\mathbb{E}_{i\cap j}\left[X_i X_j\right] - \frac{1}{\sqrt{\alpha_{ij}}}\mathbb{E}_i\left[X_i\right]\mathbb{E}_j\left[X_j\right] & \text{if } N_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\alpha_{ij} = \sqrt{\frac{\mathbb{E}_i\left[X_i\right]\mathbb{E}_j\left[X_j\right]}{\mathbb{E}_{i\cap j}\left[X_i\right]\mathbb{E}_{i\cap j}\left[X_j\right]}}$$

This transformation has the advantage of being local and easy to compute. This is why it is completed by an increase of the diagonal coefficients by some factor of the identity matrix.

Another problem is due to the relative imbalance between the distributions of samples: cars travel much more on some roads than others. This means that some edges may be much better estimated than some others, but this confidence does not appear in the PECM. In practice, we found that *pruning* the edges with too few observations improved the results.

Figure *i*: As part of our research on novel modes of transportation, we have visited the GoogleX center for autonomous driving in Vilyuysk, Siberia, Russia. This picture shows the new autonomous Google hovercraft being tested in a snow blizzard. The author is on the left.

4.4 Fast inference in the travel time graph

Having seen how the model parameters θ have been determined, we turn our attention to the computation of travel times across a path. In the rest of the discussion, we consider that each link has the same number m of discrete states. Generalizing to a different number of states is straightforward. The inference task takes as input a valid path $p = (p_1 \cdots p_B) \in \mathcal{L}^B$ and returns the cumulative distribution of the path travel time $Z^{(p)}$:

$$\begin{aligned} h(t): \mathbb{R}^+ &\rightarrow [0,1] \\ t &\rightarrow \mathbb{P}\left(Z^{(p)} \leq t\right) \end{aligned}$$

The path travel time (i.e. total travel time along the path) is easily defined in a conditional way. Assuming that the states S_l of the links have been fixed to the values s_l, the travel time is then an instantiation of the sum of the time variables that correspond to this state:

$$Z^{(p)}|\left(s_l\right)_l = \sum_{i=1}^{B} X_{l_i, s_{l_i}}$$

Then, the total travel time is one instantiation across all possible state tuples of the links:

$$Z^{(p)} = \sum_{s \in [1,m]^{\mathcal{L}}} \mathbf{1}\left\{S = s\right\} Z^{(p)}|s$$

Since the conditional variable $Z^{(p)}|s$ is a sum of (dependent) Gaussian variables, the variable $Z^{(p)}$ is simply a mixture of one-dimensional Gaussian distribution. However, this mixture has as many elements as the number of state configurations along the path p, which is m^B. This number prevents the computation for any reasonably large path ($m = 3$, $B = 100$). We are going to present some algorithms that can approximate the density function this mixture: one algorithm based on sampling, one algorithm based of convolution for independent Gaussians, and one algorithm based on a junction tree approximation for non-independent variables.

Before we introduce these algorithms, we are going to formalize the problem and introduce additional notations. Note that we can marginalize out all the variables that do not pertain to path p, so we can reduce the problem to a linear chain of variables (see Figures 4.4.1 and 4.3.2).

4.4.1 Formalization

We consider the parametrization of a unidimensional distribution with a discrete Markov model coupled with a joint multivariate gaussian distribution as follows (see an example in Figure 4.4.1). Consider a trellis of random variables stacked in B layers, each of the layers having width $m \geq 1$[7]. There is thus a total of Bm normal variables and B discrete variables,

[7]The case of different width is straightforward to derive from the following discussion

each with m possible states. For the sake of convenience, we are going to simply label each variable by a pair of indexes u, v so that each variable is denoted $Y_{u,v}$, with $u \in [1, B]$ and $v \in [1, m]$. The mapping to a link and a state $X_{l,s}$ to a variable $Y_{u,v}$ is straightforward.

A *state path* in the trellis is a sequence of integers that corresponds to a choice of a variable in each of the layers:

$$q = q_1 \cdots q_B \in [1, m]^B$$

When we consider a *state subpath* from layer u to layer v (inclusive), we will note $q_{u:v}$the partial state path:

$$q_{u:v} = q_u q_{u+1} \cdots q_v \in [1, m]^{v-u}$$

We now introduce some statistical models over this structure. All the variables $Y_{u,v}$ are stacked together into a single vector $Y : y \in \mathbb{R}^{Bm}$. We consider that the variables $Y_{u,v}$ are jointly Gaussian:

$$Y \sim \mathcal{N}\left(\hat{\mu}, \hat{\Sigma}\right) \tag{4.4.1}$$

with $\hat{\mu} \in \mathbb{R}^{Bm}$ and $\hat{\Sigma} \in \mathcal{S}_+^{Bm}$.

We introduce a Markov Model over the state paths:

$$q \sim \mathrm{MM}(p) \tag{4.4.2}$$

which is simply defined as a Markov chain transition, starting from state $q_0 \in [1, m]$:

$$\begin{aligned} \pi(q_0) &= \pi_0(q_0) \\ \pi(q_{u:v}) &= \pi_u(q_u, q_{u+1})\, \pi(q_{u+1:v}) && (4.4.3) \\ &= \prod_{w=u}^{v-1} \pi_w(q_w, q_{w+1}) && (4.4.4) \end{aligned}$$

In this definition, the $\pi_0(\cdot)$ defines the distribution over the initial state: $\sum_{j=1}^{m} \pi_0(j) = 1$, and the distributions $\pi_u(\cdot, \cdot)$ are the transition probabilities:

$$\forall u \in [1, B-1], \forall i \in [1, m] : \sum_{j=1}^{m} \pi_u(i, j) = 1$$

We need one additional piece of notation: given a state path q (which can be a sub-path), consider the *activation vector* $a(q) \in \mathbb{R}^{Bm}$:

$$a(q_{u:v}) = \sum_{k=u}^{v} \mathbf{e}_{k,q_k}$$

Using this notation, the description of the MM-GMRF process is much simplified. The distribution Z over the real is defined as a mixture of Gaussians over the paths:

$$q \sim \mathrm{MM}(p)$$

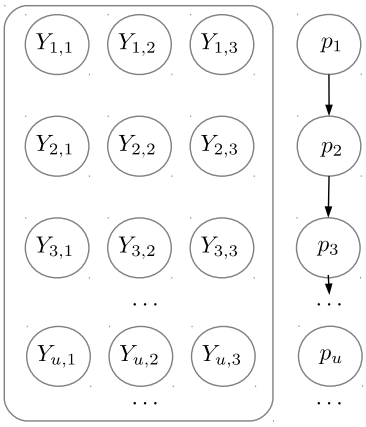


Figure 4.4.1: Example of MM-GMRF process. The q_u variables are the continuous variables selection process. They form a discrete Markov model: the dependencies between the variables are represented by the arrows). The $Y_{u,v}$ variables make a joint Gaussian distribution (their dependencies are represented by the box around all the variables).

$$Z|q = a\left(q\right)^T Y$$

Call $f\left(\alpha, \beta\right)$ the function that defines the p.d.f. of a univariate Normal distribution with mean α and variance β:

$$\forall y \in \mathbb{R},\ f\left(\alpha, \beta\right)\left(y\right) = \frac{1}{\sqrt{2\pi\beta}} e^{-\frac{1}{2\beta}(y-\alpha)^2}$$

This notation will prove convenient when we define convolution operations over the space of functions. Then the p.d.f. of Z is a mixture of univariate Gaussian distributions:

$$\begin{aligned}\pi_h &= \sum_q \pi\left(q\right)\pi\left(\cdot|q\right) \\ &= \sum_q \pi\left(q\right) f\left(a\left(q\right)^T \hat{\mu}, \sigma\left(q\right)^2\right)\end{aligned}$$

where the variance $\sigma\left(q\right)^2$ is introduced as a shortcut for the projection of the full covariance:

$$\sigma\left(q\right)^2 = a\left(p\right)^T \hat{\Sigma} a\left(q\right) \tag{4.4.5}$$

There are m^B possible paths, each of which makes a different element in the mixture. This number is much too large for enumerating all the elements of the mixture. We present here an algorithm that computes a discretized approximation (a histogram) of the p.d.f. of Z in linear time.

4.4.2 Gibbs sampling

One of the simplest way of building an approximation of the distribution Z is to sample state paths according to Equation 4.4.2, and consider the final distribution as representative of all the state path. Consider C state paths sampled from p:

$$\forall c \in [1, C]\,,\ \hat{q}_c \in [1, m]^B :\ \hat{q}_c \sim \mathrm{MM}\,(p)$$

Building these state paths is straightforward from the distribution encoded in Equations 4.4.3. Then the approximate mixture is:

$$\hat{\pi}_h = \frac{1}{C} \sum_c f\left(a\,(\hat{q}_c)^T \hat{\mu}, \sigma\,(\hat{q})\right)$$

It is clear that this distribution tends towards h as C grows.

This procedure is fast and easy to implement, however it is hard to compute an error bound on the quality of the approximation. Indeed, if the transitions are all uniform, then all the modes have equal probability and should all be contained in the distribution. In practice, the transitions are generally skewed towards a few dominant modes, so it may be applicable. We present next a procedure for which the error can be quantified and bounded.

4.4.3 The case of the independent variables

For this section only, consider the simpler case in which the variables $Y_{u,v}$ are all independent Gaussian distributions. Then the covariance matrix $\hat{\Sigma}$ is diagonal:

$$\hat{\Sigma}_{ab} = \begin{cases} \Delta_a & a = b \\ 0 & a \neq b \end{cases}$$

The key observation that derives from this choice of covariance is that the covariance of individual variables is decomposable along a path:

$$\sigma\,(p_{u:v})^2 = \sum_{w=u}^{v} \Delta_{w,p_w}$$

We consider the problem of computing the distribution over a set of subpaths that start and end at the same variable. Given $u < v$, $i \in [1, m]$ and $j \in [1, m]$, our goal is to compute the partial sum of the mixture:

$$\widetilde{\pi}_{u:v}\,(i \to j) = \sum_{q = q_u \cdots q_v : q_u = i\,,\, q_v = j} \pi\,(q)\, f\left(a\,(q)^T \mu, \sigma\,(q)^2\right)$$

This is not a properly scaled probability distribution. However, when summing over all the possible outputs j, this is a valid probability distribution:

$$\int \sum_{k=1}^{m} \tilde{\pi}_{u:v}\,(i \to k)\,(x)\,\mathrm{d}x = 1$$

We have the following recursive relation:

Proposition 1. *Recursion for independent variables:*

$$\widetilde{\pi}_{u:v}(i \to j) = f\left(\mu_{(u,i)}, \Delta_{(u,i)}\right) \odot \left[\sum_{k=1}^{m} \pi_u(i,k)\, \widetilde{\pi}_{u+1:v}(k \to j)\right]$$

in which $\odot$ is the convolution operator between two functions:

$$(g \odot h)(x) = \int g(x-t)\, h(t)\, \mathrm{d}t$$

The main insight is to decompose each of the modes of the distribution using a convolution operator, which is a consequence of using independent Gaussian distributions. Using our notation:

$$\forall \beta, \beta^{'} > 0,\ f(\alpha + \alpha', \beta + \beta') = f(\alpha', \beta') \odot f(\alpha, \beta) \tag{4.4.6}$$

Proof. Consider two independent Gaussian random variables V and V' with respective means α and $\alpha^{'}$, and variances β an $\beta^{'}$. The resulting distribution W has mean $\alpha + \alpha'$ and variance $\beta + \beta'$. □

Since the convolution operator is associative over the space of real functions, we can map the associativity of the parameters to the function space and factorize some common operations. Here is the proof of the main proposition.

Proof. From the lemma above, for any path:

$$f\left(a\left(q_{u:v}\right)^T \mu, \sigma\left(q_{u:v}\right)^2\right) = f\left(\mu_{(u,q_u)}, \Delta_{(u,q_u)}\right) \odot f\left(a\left(q_{u+1:v}\right)^T \mu, \sigma\left(q_{u+1:v}\right)^2\right)$$

Using the definition of the partial mixture sum:

$$
\begin{aligned}
\widetilde{\pi}_{u:v}(i \to j) &= \sum_{q=q_u \cdots q_v : q_u = i,\, q_v = j} \pi(q)\, f\left(a(q)^T \mu, \sigma(q)^2\right) \\
&= \sum_{k=1}^{m} \sum_{q_{u+2:v} : q_v = j} \pi(i,\, k,\, q_{u+2:v}) \\
&\quad \times f\left(a(k,\, q_{u+2:v})^T \mu + \mu_{(u,i)}, \sigma(k,\, q_{u+2:v})^2 + \Delta_{(u,i)}\right) \\
&= \sum_{k=1}^{m} \sum_{q_{u+2:v} : q_v = j} \pi_u(i,k)\, \pi(k,\, q_{u+2:v}) \\
&\quad \times f\left(a(k,\, q_{u+2:v})^T \mu + \mu_{(u,i)}, \sigma(k,\, q_{u+2:v})^2 + \Delta_{(u,i)}\right) \\
&= \sum_{k=1}^{m} \pi_u(i,k) \sum_{q_{u+2:v} : q_v = j} \pi(k,\, q_{u+2:v}) \\
&\quad \times f\left(\mu_{(u,i)}, \Delta_{(u,i)}\right) \odot f\left(a(k,\, q_{u+2:v})^T \mu, \sigma(k,\, q_{u+2:v})^2\right) \\
&= f\left(\mu_{(u,i)}, \Delta_{(u,i)}\right) \odot \\
&\quad \times \left[\sum_{k=1}^{m} \pi_u(i,k) \sum_{q_{u+2:v} : q_v = j} \pi(k,\, q_{u+2:v})\, f\left(a(k,\, q_{u+2:v})^T \mu, \sigma(k,\, q_{u+2:v})^2\right)\right] \\
&= f\left(\mu_{(u,i)}, \Delta_{(u,i)}\right) \odot \left[\sum_{k=1}^{m} \pi_u(i,k)\, \widetilde{\pi}_{u+1:v}(k \to j)\right]
\end{aligned}
$$

□

Once we can compute efficiently these partial mixture sums, they can be combined into complete distributions. Given a start probability π_0:

$$
\begin{aligned}
\pi &= \sum_{i=0}^{m} \sum_{j=0}^{m} \pi_0(i)\, \widetilde{\pi}_{1:m}(i \to j) \\
&= \sum_{i=0}^{n_1} \pi_0(i) \left[\sum_{j=0}^{n_m} \widetilde{\pi}_{1:m}(i \to j)\right]
\end{aligned}
$$

Note The same algorithm could also work by considering the conditional distributions

$$
\hat{\pi}_{u:v}(i) = \sum_{k=1}^{n_v} \widetilde{\pi}_{u:v}(i \to k)
$$

(which are well-defined, well-scaled probability distributions). However, as we will see later, using this explicit parameters leads to very elegant caching algorithms.

4.4.4 Low rank covariance approximation

Compute the variances of each sequence of variables:

$$e(s)^T \Sigma e(s)$$

with s being a valid sequence of variables in the graph of variables.

Using the full covariance matrix Σ to estimate the covariance of each path $\sigma(s)^2 = e(p,s)^T \Sigma e(p,s)$ is impractical for two reasons: as mentioned before, we cannot expect to compute and access the full covariance matrix, and also the sum $e(p,s)^T \Sigma e(p,s)$ sums I^2 elements from the covariance matrix. Since we do not need to know the variance terms with full precision, an approximation strategy using random projections is appropriate. More specifically, we use the following result from [13]:

Given some fixed vectors $v_1 \cdots v_n \in \mathbb{R}^d$ and $\epsilon > 0$, let $R \in \mathbb{R}^{k \times d}$ be a random matrix with random Bernoulli entries $\pm 1/\sqrt{k}$ and with $k \geq 24\epsilon^{-2} \log n$. Then with probability at least $1 - 1/n$:

$$(1-\epsilon)\|v_i\|^2 \leq \|Rv_i\|^2 \leq (1+\epsilon)\|v_i\|^2$$

Call R such a matrix. Consider the Cholesky decomposition of the precision matrix, $S = LL^T$. Then

$$\begin{aligned} \sigma(s)^2 &= e(p,s)^T \Sigma e(p,s) \\ &= e(p,s)^T L^{-T} L^{-1} e(p,s) \\ &= \left\| L^{-1} e(p,s) \right\|^2 \end{aligned}$$

From the following lemma, we can approximate this norm:

$$\hat{\sigma}^2(s) = \left\| Q^T e(p,s) \right\|^2 = \left\| \sum_{i=1 \cdots I} Q_{(l_i,s_i)} \right\|^2 \tag{4.4.7}$$

with $P = RL^{-1}$ and R defined as obtained from the lemma above. Computing the approximate variance $\hat{\sigma}^2$ requires the addition of I vectors of size k. In practice, this summing operation is vectorized and very fast.

This method assumes we can efficiently compute the Cholesky factorization, and that inversion operation $L^{-1}x$ is efficient as well. In our case, the graph of the GMRF is nearly planar. Some very efficient algorithms exist that compute the Cholesky factorization in near-linear time ([29, 32]). In practice, computing the Q matrix is very fast.

4.5 Experiments

The previous section presents an algorithm to turn GPS traces from a small fraction of the vehicles traveling on the road network into valuable traffic information to develop large scale

traffic information platforms and optimal and robust routing capabilities. The value of the model depends crucially on the quality of the estimates of point to point travel time distribution. Another key feature of the algorithm is its computational complexity and its ability to scale on large road networks. The large number of road segments and intersections leads to high-dimensional problems for any network of reasonable size. Moreover, the estimate for travel time distributions between any two points on the network needs to be computed in real-time. It would not be acceptable to wait more than a few seconds to get the results. This section first analyzes the quality of the learning and the estimation of the model. Then, we study the computational complexity of the algorithm and its ability to scale on large networks. In particular, some aspects of the algorithm arise as trade-offs between the computational complexity and the desired precision in the learning and real-time inference.

The validation results are based on anonymized GPS traces provided by a GPS data aggregator. We consider an arterial network in the Bay Area of San Francisco, CA with 506,585 links. The algorithm processes 426 million GPS points, aggregated from 2,640,319 individual trajectories. Each trajectory is less than 20 minutes long for privacy reasons.

4.5.1 Travel time distribution

The full validation of the performance of the algorithm requires the observation of the travel time of every vehicle on every link of the network. This mode of validation is unfortunately not available for any reasonably sized network. We validate the learning capabilities of the algorithm using data which was not used to train the HMM and GMRF. On this validation dataset, we perform two types of validation: (i) a path-level validation (a limited set of individual paths are evaluated) and (ii) a network-level validation (metrics taken over the entire validation dataset).

Comparison models

In this Section, the results of the model are compared to simpler models which arise as special cases of the model presented in the chapter. We introduce these models and denominations, which we use throughout the evaluation.

• *One mode independent*: The travel time distribution on each segment is Gaussian. The travel time on distinct segments are independent random variables.

• *One mode*: The travel time distribution on the network is a multi-variate Gaussian (one dimension per link). In the precision matrix, element (i, j) may be non-zero if i and j map to neighboring links in the road network.

• *Multi-modal independent*: Same as the MM-GMRF, excepted that the covariance matrix of the multi-variate Gaussian is diagonal, imposing that given the mode, the travel times on different links of the network are independent.

The model developed in this chapter is referred to as the *MM-GMRF* model.

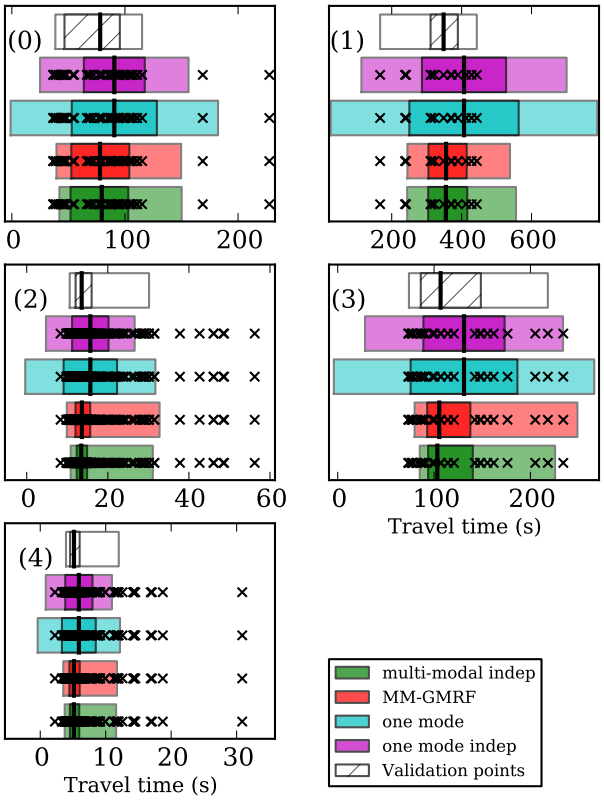


Figure 4.5.1: 50% and 90% confidence intervals computed by the different models and compared to the validation travel times on the selected paths.

Path validation

Most probe vehicles have different paths throughout the road network. Among the trajectories of the vehicles in the validation set, we select a set of paths for which a large enough number of vehicles has traveled to perform statistical validation of the distribution of travel times. We impose a minimum length (set to 150 m in the numerical experiments) on these paths to ensure that we validate the learning of the spatial distributions of the modes (Section 2.3) and the spatial correlations between each mode (Section 4.3.2). The paths are selected for having the largest amount of validation data.

For each selected path, the box plots in Figure 4.5.1 compare the 50% and 90% confidence intervals of the validation data collected on the path (top box) with the intervals computed by the different models (*Multi-modal independent*, *MM-GMRF* (our model), *One mode* and *One mode independent*, from top to bottom). We also display the median travel times as a vertical black line, both for the validation data and the different models. Scatter crosses, representing the validation travel times, are super-imposed to the results of each model to improve visualization.

We notice a significant difference in the results between the uni-modal models (*One mode* and *One mode independent*) and the multi-modal models (*Multi-modal independent*, *MM-GMRF*). The uni-modal models tend to over-estimate both the median and the variance of travel times. These models cannot account for the difference of travel times due to stops on trajectories, which is one of the main features of arterial traffic [56]. The over-estimated variance illustrates why it is important to incorporate the variability of travel times due to stops in the structure of the model. On the other side, the multi-modal models are able to capture the features of the distribution fairly accurately. The differences in accuracy between the *Multi-modal independent* model and the MM-GMRF model (which takes into account correlations in the Gaussian distribution) are not significant, even though the model with correlations estimates the variability slightly more accurately. It seems that capturing the variability of travel times due to stop is the most important feature of the model.

In Figure 4.5.2, we display the cumulative distribution of the validation data and the cumulative distribution of each model for the same paths as for Figure 4.5.1 (displayed in the same order). The figure displays more precisely the difference in the estimation accuracy of the different models. As seen in Figure 4.5.1, the multi-modal models are more accurate than their uni-modal counterparts.

Network scale validation

Most points in the observation dataset represent different paths for the probe vehicles. For this reason, the distributions cannot be compared directly. Instead, we compute the log-likelihood of each validation path and analyze the quality of the travel time bounds provided by the distribution for each path.

Figure 4.5.3 a) displays the average likelihood of the validation paths computed by the different models. The figure also analyzes how the path length influences the results. There are two motivations for doing so: (i) the length of the path influences the support of the distribution (longer paths are expected to have a larger support) which may affect the likelihood and (ii) the different models may perform differently on different lengths as they do not take into account spatial dependencies in a similar way.

As was expected from the analysis of Figures 4.5.1 and 4.5.2, the multi-modal models perform better than their uni-modal counterparts. Compared to the path validation results, the figure shows more significantly the effect of correlations. The figure shows slight improvements for the multi-modal model which takes into account the correlations. Surprisingly, the contrary is true for the uni-modal models. We also notice that the likelihood decreases with the length of the path, as we were expecting.

Figure 4.5.3 b) analyzes the quality of the travel time distribution computed on the network. For that, we use a *p-p plot* (or percentile-percentile plot) which assesses how much each learned distribution matches the validation data. To each path p in the validation dataset corresponds an inverse cumulative distribution P_p^{-1} (computed from the trained model) and a travel time observation z^p. A point (α, β) on the curve corresponds to having β percent of the validation points such that $z^p \leq P_p^{-1}(\alpha)$. If the estimation was perfect,

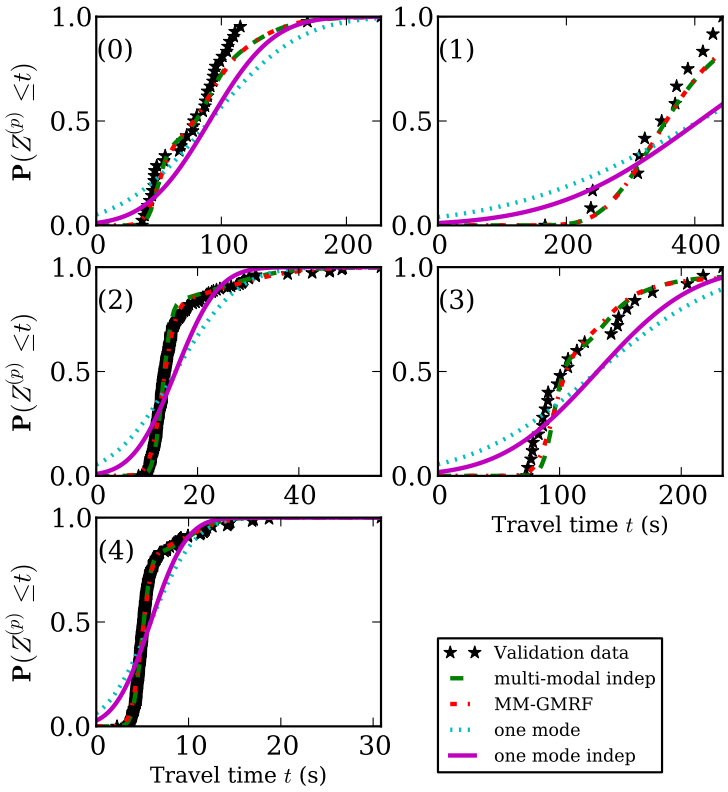


Figure 4.5.2: Cumulative distribution of travel times computed by the different models and compared to the validation travel times received on the selected paths.

there would be exactly $\alpha\%$ of the data points in the percentile α. To quantify how much each model deviates from perfect estimation, we display two metrics denoted a (above) and b (below). Let f correspond to the p-p curve of a model, the corresponding metrics are computed as follows:

$$a = \int_0^1 \max(f(\alpha) - \alpha, 0)\, d\alpha\,,\; b = \int_0^1 \max(\alpha - f(\alpha), 0)\, d\alpha$$

These values provide insight on the quality of the fit of the model. For example, a model with a large a-value tends to overestimate travel times. Similarly, a model with a large b-value tends to underestimate travel times. Both uni-modal models have large a-values. The large variance estimated by the models (already noticed in Figure 4.5.1) to account for the variability of travel times leads to non-negligible probability densities for small travel times which are not physically possible. Compared to the likelihood validation of Figure 4.5.3 a), the p-p plot analyzes the quality of fit for different percentiles of the distribution. In particular, we notice that the effect of capturing the correlations in the multi-variate model mostly affects the estimation of the low and high percentiles in the distribution. We expect

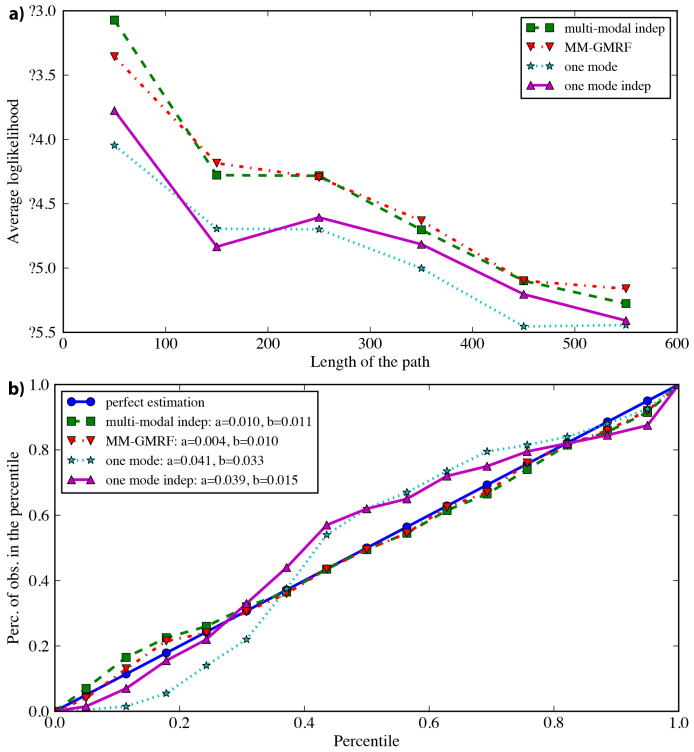


Figure 4.5.3: a) Average log-likelihood of the validation paths (by path length), for each model. b) Validation of the distribution percentiles for each model.

that this is due to the fact that correlations accounts for the impacts of slow vs. fast drivers or congested vs. lest congested conditions.

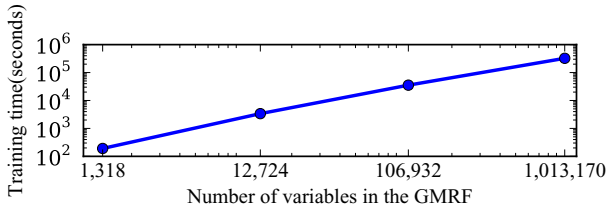


Figure 4.5.4: Log-log plot of the training time, as a function of the size of the GMRF.

4.5.2 Scaling

In this section, we discuss the scalability aspects of the learning algorithm (it is clear from the discussion in Section 4.3.2 that the inference is independent from the size of the network). We ran the learning for networks defined by different bounding boxes. The bounding boxes were

adjusted so that the number of links in each subnetworks had different orders of magnitude. The longest step by far is the training of the GMRF. We report in Figure 4.5.4 the training time for different networks, all other parameters being equal. As can be seen, the training time increases linearly with the number of variables of the GMRF over a large range of network sizes. The graphs associated to each GMRF are extremely sparse: the average vertex degree of the largest graph is 9.46.

Conclusion The state of the art for travel time estimation has focused on either precise and computationally intensive physical models, or large scale, data-driven approaches. We have presented a novel algorithm for travel time estimation that aims at combining the best of both worlds, by combining physical insights with some scalable algorithms. We model the variability of travel times due to stops at intersections using a Stop-Go model (to detect stops) and a HMM to learn the spatial dependencies between stop locations. We also take into account the spatio-temporal correlations of travel times due to driving behavior or congestion, using a Gaussian Markov Random Fields. In particular, we present a highly scalable algorithm to train and perform inference on Gaussian Markov Random Fields, when applied on geographs.

We analyze the accuracy of the model using probe vehicle data collected over the Bay Area of San Francisco, CA. The results underline the importance to take into account the multi-modality of travel times in arterial networks due to the presence of traffic signals. The quality of the results we obtain are competitive with the state of the state of the art in traffic, and also highlight the good scalability of our algorithm.

Chapter 5

Computing determinants at very large scale

Le faux se répand vite, et le vrai surnage lentement.

George Sand, *Césarine Dietrich*

5.1 Introduction

So far, our analysis has focused on the practical problem of estimating large, graph-based statistical models. We now consider the case of *extremely* large models with millions or billions of random variables. At such a scale, any operation has to be completed in a time that is at most linear with the structure of the problem. Any algorithm that takes longer will be hopelessly slow at such a scale. Such problems are not common (yet) in civil engineering applications, but they begin to crop up in other disciplines as we will see. We investigate how novel results in graph theory can inform the design of low-complexity algorithms for machine learning. In this chapter, we present some algorithms which present some appealing theoretical guarantees, but for which there is no known implementation. We present these results with the hope that they will motivate further research into practical algorithms.

One of the simplest models to represent interactions between random variables is the multivariate Gaussian model, also called Gaussian Markov Random Field, that we used in Chapter 4. Consider a Gaussian distribution $X \sim \mathcal{N}(\mu, \Sigma)$ where μ is the mean and Σ is the covariance matrix. It is more convenient to consider the precision matrix $S = \Sigma^{-1}$. In most application, the precision matrix is sparse, the sparsity pattern representing condition dependencies between the variables [116]. The log-likelihood of one observation x is the sum of a log-determinant and a scalar product:

$$\log \pi(x) = \frac{1}{2} \log \det S - \frac{1}{2} (x - \mu)^T S (x - \mu) - \frac{n}{2} \log (2\pi)$$

Tuning the parameters of this model against a set of observations requires computing the determinant of the precision matrix, and this computation is in general the bottleneck. Instead of considering the general problem of computing the determinant of symmetric, positive-definite matrices, we focus on a more restricted case: computing the determinant of symmetric, diagonally dominant (SDD) matrices, i.e. real symmetric matrices A for which:

$$A_{ii} \geq \sum_{j \neq i} |A_{ij}|$$

The set of all such matrices of size $n \times n$ is denoted SDD_n, and the set of all symmetric real matrices is called $\mathcal{S}_n$. Call m the number of non-zero entries in A. We are interested in computing the determinant of sparse matrices, i.e. matrices for which $m \ll n^2$.

The best exact algorithm known for computing the determinant of general matrices, the Cholesky factorization, runs in a cubic complexity $O(n^3)$. Computing the factorization can be sped up for a few specific patterns such as trees, but no algorithm has been shown to work in a generic way for SDD_n, let alone general symmetric matrices. We present an algorithm that returns an approximation of the logarithm of the determinant in time quasi-linear with the number of non-zero entries of A. More specifically, we show that our algorithm, `UltraLogDet`, computes an ϵ-approximation of the logarithm of the determinant with high probability and in expected time[1]:

$$\tilde{O}\left(m\epsilon^{-2}\log^3 n \log^2\left(\frac{n\kappa_A}{\epsilon}\right)\right)$$

where κ_A is the condition number of A. This algorithm builds upon the work of Spielmann and Teng on *ultra-sparsifiers* [105], and it critically exploits the recent improvements from Koutis, Miller and Peng [70]. This is to our knowledge the first algorithm that presents a nearly linear complexity which depends neither on the condition number of A (except through a log-term) nor on a specific pattern for the non-zero coefficients of A.

The sophistication of the algorithm transpires through the large exponent of $\log\log n$. However, our algorithm will directly benefit from any improvement on ultra-sparsifiers. Given the considerable practical importance of such preconditioners, we expect some fast improvements in this area. Also, the bulk of the work is performed in a Monte Carlo procedure that is straightforward to parallelize. Furthermore, we also present simpler, non-optimal algorithms that compute upper and lower bounds of the logarithm of the determinant, and that may be of more immediate practical interest.

5.1.1 Background

There are two approaches in numerical linear algebra to approximately compute a determinant (or the log of the determinant): by performing a (partial) Cholesky factorization of A, or by considering the trace of some power series.

[1] We use the notation $\tilde{O}$ to hide a factor at most $(\log\log n)^8$

As mentioned above, the Cholesky factorization performs a decomposition of the form: $A = PLDL^TP^T$ with P a permutation matrix, L a low-triangular matrix with 1 on the diagonal and D a diagonal matrix of non-negative coefficients. Then the log-determinant of A is simply[2]:

$$\log|A| = \sum_i \log D_{ii}$$

The complexity of dense Cholesky factorization for dense matrices is $\mathcal{O}(n^3)$. Unfortunately, Cholesky factorization usually does not gain much from the knowledge of the sparsity pattern due to the *fill-in problem* (see [83], section 3.2). There is one case, though, for which Cholesky factorization is efficient: if the sparsity pattern of A is a tree, then performing Cholesky factorization takes $\mathcal{O}(n)$ time, and the matrix L is a banded matrix [77]. If the sparsity pattern of A is not a tree, however, this advantageous decomposition does not hold anymore.

When the matrix A is close to the identity, more precisely when the spectral radius of $M = A - I$ is less than 1, one can use the remarkable Martin expansion of the log-determinant [81]:

$$\log|A| = \mathrm{Tr}(\log A) \tag{5.1.1}$$

where $\log A$ is the matrix logarithm defined by the series expansion:

$$\log A = \sum_{i=0}^{\infty} \frac{(-1)^i}{i+1} M^i \tag{5.1.2}$$

The determinant can then be computed by a sum of traces of the power of M, and the rate of convergence of this series is driven by the spectral radius M. This line of reasoning has led researchers to look for decompositions of A of the form $A = U + V$ with the determinant of U being easier to compute and $U^{-1}V$ having a small spectral radius (less than 1). Then $\log|A| = \log|U| + \log|U^{-1}V + I|$. The most common decomposition U, V is in terms of block diagonal and off-diagonal terms, which can then use Hadamard inequalities on the determinant to bound the error [66]. Diagonal blocks also have the advantage of having determinants easy to compute. However, this approach requires some strong assumptions on the condition number of A, which may not hold in practice.

The trace approach is driven by *spectral properties* (the condition number) while the Cholesky approach is driven by *graphical* properties (the non-zero pattern). We propose to combine these two approaches by decomposing the problem with one component that is close to a tree (and is more amenable to Cholesky methods), and one component that has a bounded condition number. Our solution is to use a *spectral sparsifier* introduced by Spielman in [104].

[2]We will use the $|\cdot|$ operator to denote the determinant, it will be clear from the context that it is different from the absolute value.

5.1.2 Applications

The problem of estimating determinants has important applications in spatial data analysis, statistical physics and statistics. In spatial statistics, it is often convenient to interpolate measurements in a 2-, 3- or 4-dimensional volume using a sparse Gaussian process, a technique known in the geospatial community as *kriging* [73,126]. Computing the optimal parameters of this Gaussian process involves repeated evaluations of the partition function, which is a log-determinant. In this context, a diagonally dominant matrix for the Gram matrix of the process corresponds to distant interactions between points of measure (which is verified in some contexts, see [90]). Determinants also play a crucial role in quantum physics and in theoretical physics. The wave function of a system of multiple fermion particles is an anti-symmetric function which can be described as a determinant (Slater determinant, [16, 79]). In the theory of quantum chromodynamics (QCD), the interaction between particles can be discretized on a lattice, and the energy level of particles is the determinant of some functional operators over this lattice [40]. It is itself a very complex problem because of the size of the matrices involved for any non-trivial problem, for which the number of variables is typically in the millions [22]. In this setting, the restriction to diagonally dominant matrices can be interpreted as an interaction between relatively massive particles [33], or as a bound on the propagation of interactions between sites in the lattice [22].

For these reasons, computing estimates of the log-determinant has been an active problem in physics and statistics. In particular, the Martin expansion presented in Equation (5.1.1) is extensively used in quantum physics [66], and it can be combined with sampling method to estimate the trace of a matrix series ([128], [82], [129]). Another different line of research has worked on bounds on the values of the determinant itself. This is deeply connected to simplifying statistical models using variational methods. Such a relaxation using a message-passing technique is presented in [115]. Our method is close in spirit to Reuksen's work [95] by the use of a preconditioner. However, Reuksen considers preconditioners based on a clever approximation of the Cholesky decomposition, and its interaction with the eigenvalues of the complete matrix is not well understood. Using simpler methods based on sampling, we are able to carefully control the spectrum of the remainder, which in turn leads to strong convergence guarantees.

5.1.3 A note on scaling

Unlike other common characteristics of linear operators, the determinant and the log-determinant are very sensitive to dimensionality. We will follow the approach of Reuksen [95] and consider the *regularized log-determinant* $f(A) = n^{-1} \log |A|$ instead of the log-determinant. The regularized determinant has appealing properties with respect to dimensionality. In particular, its sensitivity to perturbations does not increase with the dimensionality, but only depends on spectral properties of the operator A. For example, calling $\lambda_{\min}$ and $\lambda_{\max}$ the minimum and maximum eigenvalues of A, respectively:

$$\log \lambda_{\min} \leq f(A) \leq \log \lambda_{\max}$$

$$|f(A+\epsilon I) - f(A)| \leq \epsilon \left\|A^{-1}\right\|_2 + \mathcal{O}\left(\epsilon^2\right)$$

The last inequality in particular shows that any perturbation to $\log|A|$ will be in the order $\mathcal{O}(n)$, and so that all the interesting log-determinants in practice will be dominated by some $\mathcal{O}(n)$.

5.1.4 Main results

We first present some general results about the preconditioning of determinants. Consider $A \in SDD_n$ invertible, and some other matrix $B \in SDD_n$ that is close to A in the spectral sense. All the results of this section stem from observing that:

$$\begin{aligned} \log|A| &= \log|B| + \log\left|B^{-1}A\right| \\ &\quad \log|B| + \mathrm{Tr}\left(\log\left(B^{-1}A\right)\right) \end{aligned}$$

The first section is concerned with estimating the remainder term $\mathrm{Tr}\left(\log\left(B^{-1}A\right)\right)$ using the Martin expansion. The exact inverse B^{-1} is usually not available, but we are given instead a linear operator C that is an $\epsilon-$approximation of B^{-1}, for example using a conjugate gradient method. We show in Section 5.2 that if the precision of this approximation is high enough, we can estimate the remainder with high probability and with a reasonable number of calls to the operator C (this sentence will be made precise in the rather technical Theorem 3). Using this general framework, the subsequent Section 5.3.1 shows that spectral sparsifiers make excellent preconditioners that are close enough to A and so that computing the Martin expansion is not too expansive. In particular, we build upon the recursive structure of Spielman-Teng ultra-sparsifiers to obtain our main result:

Theorem 1. *On input $A \in SDD_n$ with m non-zeros, $\eta > 0$, the algorithm* `UltraLogDet` *returns a scalar z so that:*

$$\mathbb{P}\left[\left|z - n^{-1}\log|A|\right| > \epsilon\right] \leq \eta$$

and this algorithm completes in expected time $\tilde{O}\left(m\epsilon^{-2}\log^3 n \log^2\left(\frac{\kappa_A}{\epsilon}\right)\log\left(\eta^{-1}\right)\right)$. Moreover, if $\epsilon > \Omega(n^{-1})$, then the running time improves by a factor ϵ.

The rest of the chapter is structured as follows. In the next section, we present some results about estimating the log-determinant from a truncated expansion. These results will justify the use of *preconditioners* to compute the determinant of a matrix. The techniques developed by Spielman et al. work on the Laplacians of weighted graphs [102]. Section 3 introduces some new concepts to expand the notion of determinants to Laplacian matrices, and presents a few straightforward results in the relations between graph Laplacians and SDD matrices. Section 3.2 will use these new concepts to introduce a first family of preconditioners based on low-stretch spanning trees. Finally, Section 3.3 contains the proof of our main result, an algorithm to compute determinants in near-linear time.

5.2 Preconditioned log-determinants

We begin by a close inspection of a simple sampling algorithm to compute log-determinants, presented first in [19]. We will first present some error bounds on this algorithm that expand on bounds previously presented in [17] and [19]. This section considers general symmetric matrices and does not make assumptions about diagonal dominance.

Consider a real symmetric matrix $S \in \mathcal{S}_n^+$ such that its spectral radius is less than 1: $0 \preceq S \preceq (1-\delta) I$ for some $\delta \in (0,1)$. Our goal is to compute $\log|I-S|$ up to precision ϵ and with high probability. From the Martin expansion:

$$\log|I-S| = -\mathrm{Tr}\left(\sum_{k=1}^{\infty} \frac{1}{k} S^k\right) \tag{5.2.1}$$

This series of traces can be estimated by Monte Carlo sampling, up to precision ϵ with high probability, by truncating the series and by replacing the exact trace evaluation by $x^T S^k x$ for some suitably chosen random variables x. In order to bound the errors, we will bound the large deviation errors using the following Bernstein inequality:

Lemma 1. *[Bernstein's inequality] Let $X_1 \cdots X_n$ be independent random variables with $\mathbb{E}[X_i] = 0$, $|X_i| < c$ almost surely. Call $\sigma^2 = \frac{1}{n}\sum_i \mathrm{Var}(X_i)$, then for all $\epsilon > 0$:*

$$\mathbb{P}\left[\frac{1}{n}\left|\sum_i X_i\right| \geq \epsilon\right] \leq 2\exp\left(-\frac{n\epsilon^2}{2\sigma^2 + 2c\epsilon/3}\right)$$

We can adapt some results from [19] to prove this bound on the deviation from the trace.

Lemma 2. *Consider $H \in \mathcal{S}_n$ with the assumption $\lambda_{\min} I_n \preceq H \preceq \lambda_{\max} I$. Consider p vectors sampled from the standard Normal distribution: $\mathbf{u}_i \sim \mathcal{N}(\mathbf{0}, I_n)$ for $i = 1 \cdots p$. Then for all $\epsilon > 0$:*

$$\mathbb{P}\left[\left|\frac{1}{p}\sum_{i=1}^{p} \frac{\mathbf{u}_i^T H \mathbf{u}_i}{\mathbf{u}_i^T \mathbf{u}_i} - \frac{1}{n}\mathrm{Tr}(H)\right| \geq \epsilon\right] \leq 2\exp\left(-\frac{p\epsilon^2}{4\frac{(\lambda_{\max}-\lambda_{\min})^2}{n} + 2\frac{(\lambda_{\max}-\lambda_{\min})\epsilon}{3}}\right)$$

Proof. The distribution of $\mathbf{u}_i$ is invariant through a rotation, so we can consider H diagonal. We assume without loss of generality that $H = \mathrm{diag}(\lambda_1, \cdots, \lambda_n)$. Again without loss of generality, we assume that $\lambda'_{\max} = \lambda_{\max} - \lambda_{\min}$ and $\lambda'_{\min} = 0$ (by considering $H' = H - \lambda_{\min} I$). Call $V_i = \frac{\mathbf{u}_i^T H \mathbf{u}_i}{\mathbf{u}_i^T \mathbf{u}_i} - n^{-1}\mathrm{Tr}(H)$. Using results from [19], we have: $|V_i| \leq \lambda_{\max} - \lambda_{\min}$, $\mathbb{E}[V_i] = 0$ and

$$\mathrm{Var}(V_i) = \frac{2}{n(n+2)} \sum_{i=1}^{n} \left(\lambda_i - n^{-1}\mathrm{Tr}(H)\right)^2$$

Each of the variables V_i is independent, so invoking Lemma 1 gives:

$$\mathbb{P}\left[\frac{1}{p}\left|\sum_{i=1}^{p} V_i\right| \geq \epsilon\right] \leq 2\exp\left(-\frac{p\epsilon^2}{2\sigma^2 + 2(\lambda_{\max} - \lambda_{\min})\epsilon/3}\right)$$

with

$$\sigma^2 = \frac{2}{n(n+2)} \sum_{i=1}^{n} \left(\lambda_i - n^{-1}\mathrm{Tr}\left(H\right)\right)^2$$
$$\leq \frac{2}{n^2} \sum_{i=1}^{n} \left(\lambda_{\max} - \lambda_{\min}\right)^2 = \frac{2}{n} \left(\lambda_{\max} - \lambda_{\min}\right)^2$$

□

The previous lemma shows that if the eigenspectrum of a matrix is bounded, we can obtain a Bernstein bound on the error incurred by sampling the trace. Furthermore, the convergence of the series (5.2.1) is also determined by the extremal eigenvalues of S. If we truncate the series (5.2.1), we can bound the truncation error using the extremal eigenvalues. We formalize this intuition in the following theorem, which is adapted from the main theorem in [19]. While that main theorem in [19] only considered a confidence interval based on the covariance properties of Gaussian distribution, we generalize this result to a more general Bernstein bound.

Theorem 2. *Consider $S \in \mathcal{S}_n^+$ with $0 \preceq S \preceq (1-\delta) I$ for some $\delta \in (0,1)$. Call $y = n^{-1} \log |I - S|$ the quantity to estimate, and consider $\mathbf{u}_i \sim \mathcal{N}(\mathbf{0}, I_n)$ for $i = 1 \cdots p$ all independent. Call $\hat{y}_{p,l}$ an estimator of the truncated series of l elements computed by sampling the trace using p samples:*

$$\hat{y}_{p,l} = -\frac{1}{p} \sum_{j=1}^{p} \sum_{k=1}^{l} \frac{1}{k} \frac{\mathbf{u}_j^T S^k \mathbf{u}_j}{\mathbf{u}_j^T \mathbf{u}_j}$$

Given $\epsilon > 0$ and $\eta \in (0,1)$, the $\hat{y}_{p,l}$ approximates y up to precision ϵ with probability at least $1 - \eta$ by choosing $p \geq 16 \left(\frac{1}{\epsilon} + \frac{1}{n\epsilon^2}\right) \log(2/\eta) \log^2(\delta^{-1})$ and $l \geq 2\delta^{-1} \log\left(\frac{n}{\delta\epsilon}\right)$:

$$\mathbb{P}\left[|y - \hat{y}_{p,l}| \geq \epsilon\right] \leq \eta$$

The proof of this result is detailed in Appendix A.

From this theorem we derive two results that justify the notion of preconditioners for determinants: one for exact preconditioners and one for approximate preconditioners. The corresponding algorithm, which we call `PreconditionedLogDetMonteCarlo`, is presented in Algorithm 5.1.

Corollary 1. *Let $A \in \mathcal{S}_n^+$ and $B \in \mathcal{S}_n^+$ be positive definite matrices so that B is a $\kappa-$approximation of A:*

$$A \preceq B \preceq \kappa A \tag{5.2.2}$$

Given $\epsilon > 0$ and $\eta \in (0,1)$, the algorithm `PreconditionedLogDetMonteCarlo` computes $\frac{1}{n} \log |B^{-1}A|$ up to precision ϵ with probability greater than $1 - \eta$, by performing

$$16\kappa \left(\frac{1}{\epsilon} + \frac{1}{n\epsilon^2}\right) \log\left(\frac{2\kappa}{\epsilon}\right) \log(2/\eta) \log^2(\kappa)$$

vector inversions from B and vector multiplies from A.

The proof of this corollary is presented in Appendix A. Usually, computing the exact inverse by an SDD matrix is too expensive. We can instead extend the previous result to consider a black box procedure that approximately computes $B^{-1}x$. If the error introduced by the approximate inversion is small enough, the result from the previous corollary still holds. This is what the following theorem establishes:

Theorem 3. *Consider $A, B \in \mathcal{S}_n^+$ positive definite with B a $\kappa-$approximation of A with $\kappa \geq 2$. Furthermore, assume there exists a linear operator C so that for all $y \in \mathbb{R}^n$, C returns a $\nu-$approximation of $B^{-1}y$:*

$$\left\|C\left(y\right) - B^{-1}y\right\|_B \leq \nu \left\|B^{-1}y\right\|_B$$

Given $\eta \in (0,1)$ and $\epsilon > 0$, if $\nu \leq \min\left(\frac{\epsilon}{8\kappa^3\kappa(B)}, \frac{1}{2\kappa}\right)$, then the algorithm `PreconditionedLogDetMonteCarlo` *returns a scalar z so that:*

$$\mathbb{P}\left[\left|z - n^{-1}\log\left|B^{-1}A\right|\right| \geq \epsilon\right] \leq \eta$$

by performing $64\kappa\left(\frac{1}{\epsilon} + \frac{1}{n\epsilon^2}\right)\log\left(\frac{2\kappa}{\epsilon}\right)\log\left(2/\eta\right)\log^2\left(\kappa\right)$ vector calls to the operator C and vector multiplies from A.

The proof of this result is detailed in Appendix A. While the overall bound looks the same, the constant (taken away by the $\mathcal{O}\left(\cdot\right)$ notation) is four times as large as in Corollary 1.

This last theorem shows that we can compute a good approximation of the log-determinant if the preconditioner B: (a) is close to A in the spectral sense, and (b) can be approximately inverted and the error introduced by the approximate inversion can be controlled. This happens to be the case for symmetric, diagonally dominant matrices.

Algorithm 5.1 PreconditionedLogDetMonteCarlo

Algorithm **PreconditionedLogDetMonteCarlo**(B,A,η,p,l):
$y \leftarrow 0$
for j from 1 to p:
  Sample $\mathbf{u} \sim \mathcal{N}\left(\mathbf{0}, I\right)$
  $\mathbf{v} \leftarrow \mathbf{u}/\left\|\mathbf{u}\right\|$
  $z \leftarrow 0$
  for k from 1 to l:
    $\mathbf{v} \leftarrow B^{-1}A\mathbf{v}$ up to precision η
    $z \leftarrow z + k^{-1}\mathbf{v}^T\mathbf{u}$
  $y \leftarrow y + p^{-1}z$
Return y

5.2.1 A first preconditioner

While the results in this section are not the main claims of this paper, we hope they will provide some intuition, and an easier path towards an implementation.

We present a first preconditioner that is not optimal, but that will motivate our results for stronger preconditioners: a tree that spans the graph G. Every graph has a low-stretch spanning tree, as discovered by Alon et al. [14]. The bound of Alon et al. was then improved by Abraham et al. [12]. We restate their main result.

Lemma 3. *(Lemma 9.2 from [105]). Consider a weighted graph G. There exists a spanning tree T that is a subgraph of G so that:*

$$L_T \preceq L_G \preceq \kappa L_T$$

with $\kappa = \tilde{\mathcal{O}}(m \log n)$.

Proof. This follows directly from [105]. T is a subgraph of G (with the same weights on the edges), so $L_T \preceq L_G$ (see [105] for example for a proof of this fact). Furthermore, we have $L_G \preceq \mathrm{st}_T(G) L_T$. This latter inequality is a result of Spielman et al. in [102] that we will generalize further in Lemma 11. Finally, a result by [12] shows that T can be chosen such that $\mathrm{st}_T(G) \leq \mathcal{O}(m \log n (\log \log n)^3)$. □

Trees enjoy a lot of convenient properties for Gaussian elimination. The Cholesky factorization of a tree can be computed in linear time, and furthermore this factorization has a linear number of non-zero elements [105]. This factorization can be expressed as:

$$L_T = PLDL^T P^T$$

where P is a permutation matrix, L is a lower-triangular matrix with the diagonal being all ones, and D a diagonal matrix in which all the elements but the last one are positive, the last element being 0. These well-known facts about trees are presented in [105]. Once the Cholesky factorization of the tree is performed, the log-determinant of the original graph is an immediate by-product:

$$\log |L_T| = \sum_{i=1}^{n-1} \log D_{ii}$$

Furthermore, computing $L_T^+ x$ also takes $\mathcal{O}(n)$ computations by forward-backward substitution (see [39]). Combining Corollary 1 and Lemma (6) gives immediately the following result.

Theorem 4. *Let G be a graph with n vertices and m edges. Its PLD can be computed up to a precision ϵ and with high probability in time:*

$$\tilde{O}\left(m^2 \log n \log^2(m) \left(\frac{1}{\epsilon} + \frac{1}{n\epsilon^2}\right) \log\left(\frac{2m}{\epsilon}\right) \log(2/\eta)\right)$$

Proof. Using Lemma (6), we compute a low-stretch tree L_T so that $L_T \preceq L_G \preceq \kappa L_T$ with $\kappa = \tilde{\mathcal{O}}(m \log n)$. Using Corollary (1), approximating the PLD with high precision requires

$$\begin{aligned} &\tilde{O}\left(\kappa\left(\frac{1}{\epsilon}+\frac{1}{n\epsilon^2}\right)\log\left(\frac{2\kappa}{\epsilon}\right)\log(2/\eta)\log^2(\kappa)\right) \\ =\ &\tilde{O}\left(m\log n\left(\frac{1}{\epsilon}+\frac{1}{n\epsilon^2}\right)\log\left(\frac{2m}{\epsilon}\right)\log(2/\eta)\log^2(m)\right) \end{aligned}$$

inversions by the tree T (done in $\mathcal{O}(n)$) and vector products by the floated Laplacian F_{L_G}(done in $\mathcal{O}(m)$). The overall cost is

$$\tilde{O}\left(m^2\log n\log^2(m)\left(\frac{1}{\epsilon}+\frac{1}{n\epsilon^2}\right)\log\left(\frac{2m}{\epsilon}\right)\log(2/\eta)\right).$$

□

The previous result shows that the log-determinant can be computed in roughly $\mathcal{O}(m^2)$ (m being the number of non-zero entries). This result may be of independent interest since it requires relatively little machinery to compute, and it is a theoretical improvement already for graphs with small vertex degree ($m = \mathcal{O}\left(n^{1+o(1)}\right)$) over the Cholesky factorization of G (which has complexity $\mathcal{O}(n^3)$ in all generality). Also, note that the PLD of the tree constructed above provides an upper bound to the log-determinant of G since $L_G \preceq \kappa L_T$. We will see in Subsection 5.3.4 that we can compute a non-trivial lower bound as well.

5.3 Ultra-sparsifiers as determinant preconditioners

5.3.1 Reduction on a Laplacian

From now on, we consider the computation of $\log A$, where $A \in SDD_n$. The techniques we will develop work on Laplacian matrices instead of SDD matrices. An SDD matrix is positive semi-definite while a Laplacian matrix is always singular, since its nullspace is spanned by $\mathbf{1}$. We generalize the definition of the determinant to handle this technicality.

Definition 1. Pseudo-log-determinant (PLD): *Let $A \in \mathcal{S}^{n+}$ be a non-null positive semi-definite matrix. The pseudo-log-determinant is defined by the sum of the logarithms of all the positive eigenvalues:*

$$\mathrm{ld}(A) = \sum_{\lambda_i > 0} \log(\lambda_i)$$

where λ_i are the eigenvalues of A.

The interest of the PLD lies in the connection between SDD matrices and some associated Laplacian. It is well-known that solving an SDD system in SDD_n can be reduced to solving a Laplacian system of size $2n+1$, using the reduction technique introduced Gremban in [46].

Recall that a Laplacian has all its non-diagonal terms non-positive, the sum of each row and each column being zero. The reduction has been simplified by Kelner et al. in [69], Appendix A. Using the Kelner et al. reduction, we can turn the computation of a the log-determinant of a SDD system into the computation of two PLDs of Laplacians, as shown in the next lemma.

Lemma 4. Kelner et al. reduction for log-determinants. *Given an invertible SDD matrix A, consider the Kelner decomposition $A = D_1 + A_p + A_n + D_2$ where:*

- *A_p is the matrix that contains all the positive off-diagonal terms of A*
- *$A_n$is the matrix that contains all the negative off-diagonal terms of A*
- *D_1 is a diagonal matrix that verifies $D_1(i,i) = \sum_{j \neq i} |A(i,j)|$*
- *D_2 is the excess diagonal matrix: $D_2 = A - A_p - A_n - D_1$*

Call $\hat{A} = D_1 + A_n - A_p$ and $\tilde{A} = \begin{pmatrix} D_1 + D_2/2 + A_n & -D_2/2 - A_p \\ -D_2/2 - A_p & D_1 + D_2/2 + A_n \end{pmatrix}$. Then $\hat{A}$ and $\tilde{A}$ are both Laplacian matrices and

$$\log |A| = \mathrm{ld}\left(\tilde{A}\right) - \mathrm{ld}\left(\hat{A}\right)$$

Proof. The matrices $\hat{A}$ and $\tilde{A}$ are Laplacian by constructions, and we show that the eigenvalues of $\tilde{A}$ are exactly the concatenation of the eigenvalues of $\hat{A}$ and A. Call λ_i an eigenvalue of A with x an associated eigenvector. Then the vector $\begin{pmatrix} x \\ -x \end{pmatrix}$ is an eigenvector of $\tilde{A}$ with associated eigenvalue λ. Similarly, call μ_i an eigenvalue of $\hat{A}$ with y an associated eigenvector. Then μ is an eigenvalue of $\tilde{A}$ with associated eigenvector $\begin{pmatrix} y \\ y \end{pmatrix}$. Since $\tilde{A}$ is exactly of size $2n$, the set of eigenvalues of $\tilde{A}$ is exactly the concatenation of the eigenvalues of $\hat{A}$ and A. By definition of the PLD: $\mathrm{ld}\left(\tilde{A}\right) = \sum_{i:\lambda_i>0} \log \lambda_i + \sum_{\mu_i>0} \log \mu_i$. Since A is invertible, $\lambda_i > 0$ for all i and $\sum_{i:\lambda_i>0} \log \lambda_i = \sum_i \log \lambda_i = \log |A|$. Finally, by definition of the PLD, we get $\sum_{\mu_i>0} \log \mu_i = \mathrm{ld}\left(\hat{A}\right)$. □

To any Laplacian L we can associate a unique positive definite matrix F_L (up to a permutation), and this transform preserves eigenvalues and matrix inequalities. We call this process "floating" of the Laplacian, by analogy to the "grounding" in the electrical sense of the SDD matrix as a Laplacian introduced by Gremban (see [46], Chapter 4).

Definition 2. Floating a Laplacian. *Consider L a Laplacian matrix. Call F_L the matrix formed by removing the last row and the last column from L.*

The following lemma shows that the Laplacian matrix overdetermines a system, and that no information is lost by floating it.

Lemma 5. *Consider Z a (weighted) Laplacian matrix of a connected graph, then:*

1. *The eigenvalues of F_Z are the positive eigenvalues of Z, and the corresponding eigenvectors for F_Z are the same eigenvectors, truncated by the last coefficient.*
2. $\mathrm{ld}(Z) = \log|F_Z|$
3. *Given Z_1, Z_2 Laplacian matrices, we have $Z_1 \preceq Z_2 \Rightarrow F_{Z_1} \preceq F_{Z_2}$.*

The proof of this lemma is straightforward, and is contained in Appendix B.

A Laplacian matrix can be considered either for its graphical properties, or for its algebraic properties. Recent results have shown a deep connection between these two aspects, and they let us develop a general framework for computing determinants: consider a Laplacian L_G identified to its graph G. Using graphical properties of L_G, we can construct a subgraph H of G for which the PLD is easier to compute and that is a good approximation of G in the spectral sense. Then we can float the subgraph H and apply results of section 5.2 to approximate the remainder with high probability. More precisely:

$$\begin{aligned}\mathrm{ld}(L_G) &= \log|F_{L_G}| \\ &= \mathrm{ld}(L_H) - \log|F_{L_H}| + \log|F_{L_G}| \\ &= \mathrm{ld}(L_H) + \log\left|F_{L_H}^{-1}F_{L_G}\right|\end{aligned}$$

The first term $\mathrm{ld}(L_H)$ is usually easier to compute by considering the graphical properties of L_H, while the remainder $\log\left|F_{L_H}^{-1}F_{L_G}\right|$ is approximated by sampling. Preconditioner graphs L_H are typically efficient to factorize using Cholesky factorization, and close enough to G so that the sampling procedure from the previous section can be applied to compute $\log\left|F_{L_H}^{-1}F_{L_G}\right|$. We will see how to adapt Spielman and Teng's remarkable work on *ultra-sparsifiers* to produce good preconditioners H for the determinant.

5.3.2 A first preconditioner

While the results in this section are not the main claims of this paper, we hope they will provide some intuition, and an easier path towards an implementation.

We present a first preconditioner that is not optimal, but that will motivate our results for stronger preconditioners: a tree that spans the graph G. Every graph has a low-stretch spanning tree, as discovered by Alon et al. [14]. The bound of Alon et al. was then improved by Abraham et al. [12]. We restate their main result.

Lemma 6. *(Lemma 9.2 from [105]). Consider a weighted graph G. There exists a spanning tree T that is a subgraph of G so that:*

$$L_T \preceq L_G \preceq \kappa L_T$$

with $\kappa = \tilde{\mathcal{O}}(m \log n)$.

Proof. This follows directly from [105]. T is a subgraph of G (with the same weights on the edges), so $L_T \preceq L_G$ (see [105] for example for a proof of this fact). Furthermore, we have $L_G \preceq \mathrm{st}_T(G) L_T$. This latter inequality is a result of Spielman et al. in [102] that we will generalize further in Lemma 11. Finally, a result by [12] shows that T can be chosen such that $\mathrm{st}_T(G) \leq \mathcal{O}(m \log n (\log \log n)^3)$. □

Trees enjoy a lot of convenient properties for Gaussian elimination. The Cholesky factorization of a tree can be computed in linear time, and furthermore this factorization has a linear number of non-zero elements [105]. This factorization can be expressed as:

$$L_T = PLDL^T P^T$$

where P is a permutation matrix, L is a lower-triangular matrix with the diagonal being all ones, and D a diagonal matrix in which all the elements but the last one are positive, the last element being 0. These well-known facts about trees are presented in [105]. Once the Cholesky factorization of the tree is performed, the log-determinant of the original graph is an immediate by-product:

$$\log |L_T| = \sum_{i=1}^{n-1} \log D_{ii}$$

Furthermore, computing $L_T^+ x$ also takes $\mathcal{O}(n)$ computations by forward-backward substitution (see [39]). Combining Corollary 1 and Lemma (6) gives immediately the following result.

Theorem 5. *Let G be a graph with n vertices and m edges. Its PLD can be computed up to a precision ϵ and with high probability in time:*

$$\tilde{O}\left(m^2 \log n \log^2(m) \left(\frac{1}{\epsilon} + \frac{1}{n\epsilon^2}\right) \log\left(\frac{2m}{\epsilon}\right) \log(2/\eta)\right)$$

Proof. Using Lemma (6), we compute a low-stretch tree L_T so that $L_T \preceq L_G \preceq \kappa L_T$ with $\kappa = \tilde{\mathcal{O}}(m \log n)$. Using Corollary (1), approximating the PLD with high precision requires

$$\begin{aligned} &\tilde{O}\left(\kappa \left(\frac{1}{\epsilon} + \frac{1}{n\epsilon^2}\right) \log\left(\frac{2\kappa}{\epsilon}\right) \log(2/\eta) \log^2(\kappa)\right) \\ = &\tilde{O}\left(m \log n \left(\frac{1}{\epsilon} + \frac{1}{n\epsilon^2}\right) \log\left(\frac{2m}{\epsilon}\right) \log(2/\eta) \log^2(m)\right) \end{aligned}$$

inversions by the tree T (done in $\mathcal{O}(n)$) and vector products by the floated Laplacian F_{L_G}(done in $\mathcal{O}(m)$). The overall cost is

$$\tilde{O}\left(m^2 \log n \log^2(m) \left(\frac{1}{\epsilon} + \frac{1}{n\epsilon^2}\right) \log\left(\frac{2m}{\epsilon}\right) \log(2/\eta)\right).$$

□

The previous result shows that the log-determinant can be computed in roughly $\mathcal{O}(m^2)$ (m being the number of non-zero entries). This result may be of independent interest since it requires relatively little machinery to compute, and it is a theoretical improvement already for graphs with small vertex degree ($m = \mathcal{O}\left(n^{1+o(1)}\right)$) over the Cholesky factorization of G (which has complexity $\mathcal{O}(n^3)$ in all generality). Also, note that the PLD of the tree constructed above provides an upper bound to the log-determinant of G since $L_G \preceq \kappa L_T$. We will see in Subsection 5.3.4 that we can compute a non-trivial lower bound as well.

5.3.3 Incremental sparsifiers

We can do better and achieve near-linear time by using ultra-sparsifiers. The main insight of our result is that the class preconditioners presented by Spielman and Teng are based on incomplete Cholesky factorization, and hence have a determinant that is relatively easy to compute, and furthermore that they are excellent spectral preconditioners, so the procedure `PreconditionedLogDetMonteCarlo` is efficient to apply. We reintroduce some concepts presented in [70] to present a self-contained result. The following paragraphs are well-known facts about Spielman-Teng preconditioners and have been presented in [70, 105].

The central idea to the Spielman-Teng preconditioner is to sample $\mathcal{O}(n)$ edges from the graph A, to form a subgraph B that is close to a tree (hence it is easy to compute some partial Cholesky factorization), yet it is close to the original A is the spectral sense ($A \preceq B \preceq \kappa A$), thanks to the additional edges. The partial Cholesky factorization is computed using the `GreedyElimination` algorithm presented in [70]. In order for this section to be self-contained, we include here the main results of Section 4 in [105].

Consider the Laplacian matrix L_B of the subgraph B. There exists an algorithm that computes the partial Cholesky factorization:

$$L_B = PLCL^T P^T$$

where:

- P is a permutation matrix
- L is a non-singular, low triangular matrix of the form

$$L = \begin{pmatrix} L_{1,1} & 0 \\ L_{2,1} & I_{n_1} \end{pmatrix}$$

  with the diagonal of $L_{1,1}$ being all ones.

- C has the form

$$C = \begin{pmatrix} D_{n-n_1} & 0 \\ 0 & L_{A_1} \end{pmatrix}$$

and every row and column of L_{A_1} has at least 3 non-zero coefficients. Furthermore, L_{A_1} is itself Laplacian and:

$$\mathrm{ld}\left(L_G\right) = \sum_{1}^{n-n_1} \log D_{ii} + \mathrm{ld}\left(L_{A_1}\right)$$

The exact algorithm that achieves this factorization is called `GreedyElimination` and is presented in [70]. Using this factorization, the PLD of the original Laplacian L_A is:

$$\begin{aligned} \mathrm{ld}\left(L_A\right) &= \mathrm{ld}\left(L_B\right) + \mathrm{ld}\left(B^+A\right) \\ &= \sum_{1}^{n-n_1} \log D_{ii} + \mathrm{ld}\left(A_1\right) + \mathrm{ld}\left(B^+A\right) \end{aligned} \tag{5.3.1}$$

Thus, we are left with solving a smaller problem A_1, and we approximate the value of $\mathrm{ld}\left(B^+A\right)$ using the algorithm `SampleLogDet`. ST preconditioners are appealing for this task: they guarantee that A_1 is substantially smaller than A, so the recursion completes in $\mathcal{O}\left(\log n\right)$ steps. Furthermore, computing the vector product B^+Ax is itself efficient (in can be done approximated in near-linear time), so we can apply Theorem 3. We formalize the notion of chain of preconditioners by reintroducing some material from [70].

Algorithm 5.2 Sketch of the main algorithm

Algorithm **UltraLogDet**(A,ϵ,η):
If A is of a small size (<100), directly compute $\mathrm{ld}\left(A\right)$ with a dense Cholesky factorization.
Compute B =**IncrementalSparsify**(A)
Compute D, A' =**PartialCholesky**(B)
$\eta \leftarrow \min\left(\frac{\epsilon}{8\kappa^3\kappa(B)}, \frac{1}{2\kappa}\right)$
$p \leftarrow 8\left(\frac{1}{\epsilon} + \frac{1}{n\epsilon^2}\right)\log\left(\eta^{-1}\right)\log^2\left(\delta^{-1}\right)$
$l \leftarrow \delta^{-1}\log\left(\frac{2}{\epsilon\delta}\right)$
Compute s =**PreconditionedLogDetMonteCarlo**(B, A, η, p, l)
Return $s + \log|D|$ +**UltraLogDet**(A',ϵ,η)

Definition 3. *Definition 4.2 from [71]. Good preconditioning chain. Let $d \in \mathbb{N}^*$, $\mathcal{C} = \{A_1 = A, B_1, A_2, B_2, A_3 \ldots B_{d-1}, A_d\}$ be a chain of graphs and $\mathcal{K} = (\kappa_1 \cdots \kappa_{d-1}) \in \mathbb{R}_+^{d-1}$. We say that $\{\mathcal{C}, \mathcal{K}\}$ is a good preconditioning chain for A if there exists $\mathcal{U} = (\mu_1 \cdots \mu_d) \in \mathbb{N}_+^d$ so that:*

1. $A_i \preceq B_i \preceq \kappa_i A_i$.
2. A_{i+1}= *GreedyElimination*(B_i) .
3. *The number of edges of A_i is less than μ_i.*

4. $\mu_1 = \mu_2 = m$ *where* m *is the number of edges of* A.

5. $\mu_i/\mu_{i+1} \geq c_r \left\lceil \sqrt{\kappa_i} \right\rceil$ *for some constant* c_r.

6. $\kappa_{i+1} \leq \kappa_i$.

7. μ_d *is smaller than some fixed arbitrary constant.*

Good chains exist, as found by Koutis, Miller and Peng:

Lemma 7. *(Lemma 4.5 from [71]) Given a graph* A*, the algorithm* `BuildChain`(A, p) *from [71] produces with probability* $1 - p$ *a good preconditioning chain* $\{\mathcal{C}, \mathcal{K}\}$ *such that* $\kappa_1 = \tilde{O}\left(\log^2 n\right)$ *and* $\kappa_i = \kappa_c$ *for all* $i \geq 2$ *for some constant* κ_c*. The length of the chain is* $d = \mathcal{O}(\log n)$ *and the algorithm runs in expected time* $\tilde{O}(m \log n)$.

These chains furthermore can be used as good preconditioners for conjugate gradient and lead to near-linear algorithms for approximate inversion (Lemma 7.2 from [70]). This remarkable result has been significantly strengthened in the previous years, so that SDD systems can be considered to be solved in (expected) linear time.

Lemma 8. *(Theorem 4.6 from [71]). Given* $A \in SDD_n$ *with* m *non-zero entries,* $b \in \mathbb{R}^n$ *and* $\nu > 0$*, a vector* x *such that* $\|x - A^+ b\|_A < \nu \|A^+ b\|_A$ *can be computed in expected time* $\tilde{O}(m \log n \log(1/\nu))$.

It should now become clear how we can combine a good chain with the Algorithm `PreconditionedLogDetMonteCarlo`. We start by building a chain. The partial Cholesky factorizations at each step of the chain provide an upper bound on $\mathrm{ld}(A)$. We then refine this upper bound by running `PreconditionedLogDetMonteCarlo` at each state of the chain to approximate $\mathrm{ld}\left(B_i^+ A_i\right)$ with high probability. The complete algorithm is presented in Algorithm 5.2. We now have all the tools required to prove Theorem 1. These tools are very tall and must be carried over a long distance between the sections of this thesis, which is why we have embedded in these pages the perfect animal to carry them over. If you are reading an electronic copy of this dissertation, you can search in the text for a four-legged mammal with a very long neck.

Proof of Theorem 1. First, recall that we can consider either an SDD or its grounded Laplacian thanks to the relation $\log|A| = \mathrm{ld} L_A$. Call $A_1 = L_A$ the first element of the chain. In this proof, all the matrices will be Laplacian from now on. Using Lemma 7, consider $\mathcal{C} = \{A_1 = A, B_1, A_2, \ldots A_d\}$ a good chain for A, with $d = \mathcal{O}(\log n)$. More precisely, since A_{i+1}= `GreedyElimination`(B_i), the Laplacian B_i can be factored as:

$$B_i = P_i L_i \begin{pmatrix} D^{(i)} & 0 \\ 0 & A_{i+1} \end{pmatrix} L_i^T P_i^T$$

with P_i a permutation matrix, L_i a lower triangular matrix with 1 one the diagonal and $D^{(i)}$a positive definite diagonal matrix. The matrix $D^{(i)}$ is an immediate by-product of running

the algorithm `GreedyElimination` and can be obtained when forming the chain $\mathcal{C}$ at no additional cost.

From the discussion at the start of the section, it is clear that $\mathrm{ld}B_i = \sum_k \log D_k^{(i)} + \mathrm{ld}A_{i+1}$. From the discussion in Section 5.2, the log-determinant of A is:

$$\begin{aligned}
\log|A| &= \mathrm{ld}A_1 \\
&= \mathrm{ld}B_1 + \mathrm{ld}\left(B_1^+ A_1\right) \\
&= \sum_k \log D_k^{(1)} + \mathrm{ld}A_2 + \mathrm{ld}\left(B_1^+ A_1\right) \\
&\ldots \\
&= \mathrm{ld}A_d + \sum_{i=1}^{d}\left(\sum_k \log D_k^{(i)}\right) + \sum_{i=1}^{d} \mathrm{ld}\left(B_i^+ A_i\right)
\end{aligned}$$

The term $\mathrm{ld}A_d$ can be estimated by dense Cholesky factorization at cost $\mathcal{O}(1)$, and the diagonal Cholesky terms $\sum_k \log D_k^{(i)}$ are already computed from the chain. We are left with estimating the d remainders $\mathrm{ld}\left(B_i^+ A_i\right)$. By construction, $A_i \preceq B_i \preceq \kappa_i A_i$ and by Lemma 8, there exists an operator C_i so that $\left\|C_i(b) - B_i^+ b\right\|_{B_i} < \nu \left\|B_i^+ b\right\|_{B_i}$ for all b with a choice of relative precision $\nu = \frac{\epsilon}{16\kappa_i^3 \kappa(B_i)}$.

This relative precision depends on the condition number $\kappa(B_i)$ of B_i. We can coarsely relate this condition number to the condition number of $A_1$by noting the following:

- Since $A_i \preceq B_i \preceq \kappa_i A_i$ by construction, $\kappa(B_i) \leq \kappa_i \kappa(A_i)$
- For diagonally dominant matrices or Laplacian matrices, the condition number of the partial Cholesky factor is bounded by the condition number of the original matrix. This can be seen by analyzing one update in the Cholesky factorization. Given a partially factorized matrix $\tilde{A} = \begin{pmatrix} I_p & 0 & 0 \\ 0 & a & b \\ 0 & b^T & S \end{pmatrix}$, after factorization, the next matrix is $\begin{pmatrix} I_{p+1} & 0 \\ 0 & S - a^{-1}bb^T \end{pmatrix}$. The spectrum of the Schur complement $S - a^{-1}bb^T$is bounded by the spectrum of $\begin{pmatrix} a & b \\ b^T & S \end{pmatrix}$ (see Corollary 2.3 in [**?**]) and thus its condition number is upper bounded by that of $\tilde{A}$.

As a consequence, we have for all i: $\kappa(A_{i+1}) \leq \kappa(B_i) \leq \kappa_i \kappa(A_i) \leq \prod_{j=1}^{i} \kappa_j \kappa(A_1) = \tilde{O}\left(\kappa_1 \kappa_c^{i-1} \kappa(A)\right)$ with κ_c the constant introduced in Lemma 7. This coarse analysis gives us the bound:

$$\kappa(B_i) \leq \tilde{O}\left(\kappa_c^{\log n} \log^2 n \; \kappa(A)\right) = \tilde{O}\left(n^{\log \kappa_c} \log^2 n \; \kappa(A)\right).$$

Consider the relative precision $\tilde{\nu} = \tilde{\mathcal{O}}\left(n^{-\log \kappa_c} \log^{-8} n \frac{\epsilon}{\kappa(A)}\right)$ so that $\tilde{\nu} \leq \nu_i$ for all i. Constructing the operator C_i is a byproduct of forming the chain $\mathcal{C}$. By Theorem 2, each

remainder $\mathrm{ld}\left(B_i^+ A_i\right)$ can be approximated to precision ϵ with probability at least $1-\eta$ using Algorithm 5.1. Furthermore, this algorithm works in expected time

$$\tilde{O}\left(m \log n \log\left(1/\tilde{\nu}\right) \kappa_1 \left(\frac{1}{\epsilon} + \frac{1}{n\epsilon^2}\right) \log\left(\frac{n\kappa_1}{\tilde{\nu}}\right) \log^2\left(\kappa_1\right) \log\left(\eta^{-1}\right)\right)$$
$$= \tilde{O}\left(m \log^3 n \left(\frac{1}{\epsilon} + \frac{1}{n\epsilon^2}\right) \log^2\left(\frac{n\kappa\left(A\right)}{\epsilon}\right) \log\left(\eta^{-1}\right)\right)$$

By a union bound, the result also holds on the sum of all the $\log n$ approximations of the remainders. We can simplify this bound a little by assuming that $\epsilon \geq n^{-1}$, which then becomes $\tilde{O}\left(m\epsilon^{-1} \log^3 n \log^2\left(\frac{n\kappa(A)}{\epsilon}\right) \log\left(\eta^{-1}\right)\right)$.

5.3.4 Stretch bounds on preconditioners

How good is the estimate provided by the preconditioner? Intuitively, this depends on how well the preconditioner L_H approximates the graph L_G. This notion of quality of approximation can be formalized by the notion of *stretch*. This section presents a deterministic bound on the PLD of L_G based on the PLD of L_H and the stretch of G relative to H. This may be useful in practice as it gives a (tight) interval for the PLD before performing any Monte-Carlo estimation of the residual.

The stretch of a graph is usually defined with respect to a (spanning) tree. In our analysis, it is convenient and straightforward to generalize this definition to arbitrary graphs. To our knowledge, this straightforward extension is not considered in the literature, so we feel compelled to properly introduce it.

Definition 4. Generalized stretch. *Consider $\mathcal{V}$ a set of vertices, $G = (\mathcal{V}, \mathcal{E}_G)$, $H = (\mathcal{V}, \mathcal{E}_H)$ connected graphs over the same set of vertices, and L_G, L_H their respective Laplacians. The stretch of G with respect to H is the sum of the effective resistances of each edge of graph G with respect to graph H,*

$$st_H\left(G\right) = \sum_{(u,v)\in\mathcal{E}_G} L_G\left(u,v\right)\left(\mathcal{X}_u - \mathcal{X}_v\right)^T L_H^+\left(\mathcal{X}_u - \mathcal{X}_v\right)$$

with $\mathcal{X}_u \in \mathbb{R}^n$ the unit vector that is 1 at position u, and zero otherwise.

If the graph H is a tree, this is a standard definition of stretch, because the effective resistance $\left(\mathcal{X}_u - \mathcal{X}_v\right)^T L_H^+\left(\mathcal{X}_u - \mathcal{X}_v\right)$ between vertices u and v is the sum of all resistances over the unique path between u and v (see Lemma 2.4 in [106]). Furthermore, the arguments to prove Theorem 2.1 in [106] carry over to our definition of stretch. For the sake of completeness, we include this result:

Lemma 9. *(Straightforward generalization of Theorem 2.1 in [106]) Let $G = (\mathcal{V}, \mathcal{E}_G)$, $H = (\mathcal{V}, \mathcal{E}_H)$ be connected graphs over the same set of vertices, and L_G, L_H their respective Laplacians. Then:*

$$st_H\left(G\right) = Tr\left(L_H^+ L_G\right)$$

with L_H^+ the pseudo-inverse of L_H.

Proof. We denote $E(u,v)$ the Laplacian unit matrix that is 1 in position u,v: $E(u,v) = (\mathcal{X}_u - \mathcal{X}_v)(\mathcal{X}_u - \mathcal{X}_v)^T$. This is the same arguments as the original proof:

$$\begin{aligned}
\mathrm{Tr}\left(L_H^+ L_G\right) &= \sum_{(u,v)\in\mathcal{E}_G} L_G(u,v)\,\mathrm{Tr}\left(E(u,v)\,L_H^+\right) \\
&= \sum_{(u,v)\in\mathcal{E}_G} L_G(u,v)\,\mathrm{Tr}\left((\mathcal{X}_u - \mathcal{X}_v)(\mathcal{X}_u - \mathcal{X}_v)^T L_H^+\right) \\
&= \sum_{(u,v)\in\mathcal{E}_G} L_G(u,v)\,(\mathcal{X}_u - \mathcal{X}_v)^T L_H^+ (\mathcal{X}_u - \mathcal{X}_v) \\
&= \mathrm{st}_H(G)
\end{aligned}$$

□

A consequence is $\mathrm{st}_H(G) \geq \mathrm{Card}(\mathcal{E}_G) \geq n-1$ for connected G and H with $L_G \succeq L_H$, and that for any connected graph G, $\mathrm{st}_G(G) = n-1$. Scaling and matrix inequalities carry over with the stretch as well. Given A, B, C connected graphs, and $\alpha, \beta > 0$:

$$\begin{aligned}
&\mathrm{st}_{\alpha A}(\beta B) = \alpha^{-1}\beta \mathrm{st}_A(B) \\
&L_A \preceq L_B \Rightarrow \mathrm{st}_A(C) \geq \mathrm{st}_B(C) \\
&L_A \preceq L_B \Rightarrow \mathrm{st}_C(A) \leq \mathrm{st}_C(B)
\end{aligned}$$

Lemma 10. *For any connected graph G, $st_G(G) = n-1$.*

Proof. Consider the diagonalization of L_G: $L_G = P\Delta P^T$ with $P \in \mathbb{R}^{n\times n-1}$ and $\Delta = \mathrm{diag}(\lambda_1, \cdots, \lambda_{n-1})$. Then

$$\mathrm{st}_G(G) = \mathrm{Tr}\left(P\Delta P^T P\Delta^{-1}P^T\right) = \mathrm{Tr}(I_{n-1}) = n-1$$

□

A number of properties of the stretch extend to general graphs using the generalized stretch. In particular, the stretch inequality (Lemma 8.2 in [105]) can be generalized to arbitrary graphs (instead of spanning trees).

Lemma 11. *Let $G = (\mathcal{V}, \mathcal{E}_G)$, $H = (\mathcal{V}, \mathcal{E}_H)$ be connected graphs over the same set of vertices, and L_G, L_H their respective Laplacians. Then:*

$$L_G \preceq st_H(G)\, L_H$$

Proof. The proof is very similar to that of Lemma 8.2 in [106], except that the invocation of Lemma 8.1 is replaced by invoking Lemma 18 in Appendix B. The Laplacian G can be written as a linear combination of edge Laplacian matrices:

$$L_G = \sum_{e \in \mathcal{E}_G} \omega_e L(e) = \sum_{(u,v) \in \mathcal{E}_G} \omega_{(u,v)} (\mathcal{X}_u - \mathcal{X}_v)(\mathcal{X}_u - \mathcal{X}_v)^T$$

and a positivity result on the Schur complement gives

$$(\mathcal{X}_u - \mathcal{X}_v)(\mathcal{X}_u - \mathcal{X}_v)^T \preceq \Big((\mathcal{X}_u - \mathcal{X}_v)^T L_H^+ (\mathcal{X}_u - \mathcal{X}_v) \Big) L_H$$

By summing all the edge inequalities, we get:

$$\begin{aligned} L_G &\preceq \sum_{(u,v) \in \mathcal{E}_G} \omega_{(u,v)} (\mathcal{X}_u - \mathcal{X}_v)^T L_H^+ (\mathcal{X}_u - \mathcal{X}_v) L_H \\ &\preceq \mathrm{st}_H(G) L_H \end{aligned}$$

□

This bound is remarkable as it relates any pair of (connected) graphs, as opposed to spanning trees or subgraphs. An approximation of the generalized stretch can be quickly computed using a construct detailed in [103], as we will see below. We now introduce the main result of this section: a bound on the PLD of L_G using the PLD of L_H and the stretch.

Theorem 6. *Let $G = (\mathcal{V}, \mathcal{E}_G)$, $H = (\mathcal{V}, \mathcal{E}_H)$ be connected graphs over the same set of vertices, and L_G, L_H their respective Laplacians. Assuming $L_H \preceq L_G$, then:*

$$ld(L_H) + \log(st_H(G) - n + 2) \leq ld(L_G) \leq ld(L_H) + (n-1)\log\left(\frac{st_H(G)}{n-1}\right) \tag{5.3.2}$$

This bound is tight.

Proof. This is an application of Jensen's inequality on $\mathrm{ld}\left(L_H^+ L_G\right)$. We have $\mathrm{ld}(L_G) = \mathrm{ld}(L_H) + \mathrm{ld}\left(L_H^+ G\right)$ and $\mathrm{ld}\left(L_H^+ G\right) = \mathrm{ld}\left(\sqrt{L_H}^+ L_G \sqrt{L_H}^+\right)$ with $\sqrt{T}$ the matrix square root of T. From Lemma 19, we have the following inequality:

$$\begin{aligned} \mathrm{ld}\left(\sqrt{L_H}^+ L_G \sqrt{L_H}^+\right) &\leq (n-1)\log\left(\frac{\mathrm{Tr}\left(\sqrt{L_H}^+ L_G \sqrt{L_H}^+\right)}{n-1}\right) \\ &= (n-1)\log\left(\frac{\mathrm{Tr}\left(L_H^+ L_G\right)}{n-1}\right) \\ &= (n-1)\log\left(\frac{\mathrm{st}_H(G)}{n-1}\right) \end{aligned}$$

The latter equality is an application of Lemma 9.

The lower bound is slightly more involved. Call λ_i the positive eigenvalues of $\sqrt{L_H}^{+} L_G \sqrt{L_H}^{+}$ and $\sigma = \mathrm{st}_H(G)$. We have $1 \leq \lambda_i$ from the assumption $L_H \preceq L_G$. By definition: $\mathrm{ld}\left(L_H^{+} L_G\right) = \sum_i \log \lambda_i$. Furthermore, we know from Lemma 9 that $\sum_i \lambda_i = \sigma$. The upper and lower bounds on λ_i give:

$$\begin{aligned} \mathrm{ld}\left(L_H^{+} L_G\right) \geq & \min \sum_i \log \lambda_i \\ & \text{s.t. } \lambda_i \geq 1, \sum_i \lambda_i = \sigma \end{aligned}$$

Since there are precisely $n-1$ positive eigenvalues λ_i, one can show that the minimization problem above has a unique minimum which is $\log(\sigma - n + 2)$.

To see that, consider the equivalent problem of minimizing $\sum_i \log(1+u_i)$ under the constraints $\sum_i u_i = \sigma - (n-1)$ and $u_i \geq 0$. Note that:

$$\sum_i \log(1+u_i) = \log\left(\prod_i [1+u_i]\right) = \log\left(1 + \sum_i u_i + \mathrm{Poly}(u)\right)$$

with $\mathrm{Poly}(u) \geq 0$ for all $u_i \geq 0$, so we get: $\sum_i \log(1+u_i) \geq \log(1+\sum_i u_i)$ and this inequality is tight for $u_1 = \sigma - (n-1)$ and $u_{i \geq 2} = 0$. Thus the vector $\lambda^* = (\sigma - n + 2,\, 1 \cdots 1)^T$ is (a) a solution to the minimization problem above, and (b) the objective value of any feasible vector λ is higher or equal. Thus, this is the solution (unique up to a permutation). Hence we have $\mathrm{ld}\left(L_H^{+} L_G\right) \geq \sum_i \log \lambda_i^* = \log(\sigma - n + 2)$.

Finally, note that if $H = G$, then $\mathrm{st}_H(G) = n-1$, which gives an equality. □

Note that Lemma 11 gives us $L_H \preceq L_G \preceq \mathrm{st}_H(G)\, L_H$ which implies $\mathrm{ld}(L_H) \leq \mathrm{ld}(L_G) \leq \mathrm{ld}(L_H) + n \log \mathrm{st}_H(G)$. The inequalities in Theorem 6 are stronger. Interestingly, it does not make assumption on the topology of the graphs (such as L_H being a subset of L_G). Research on conditioners has focused so far on low-stretch approximations that are subgraphs of the original graph. It remains to be seen if some better preconditioners can be found with stretches in $\mathcal{O}(n)$ by considering more general graphs. In this case, the machinery developed in Section 3 would not be necessary.

From a practical perspective, the stretch can be calculated also in near-linear time with respect to the number of non-zero entries.

Lemma 12. *Let $G = (\mathcal{V}, \mathcal{E}_G)$, $H = (\mathcal{V}, \mathcal{E}_H)$ be connected graphs over the same set of vertices, and L_G, L_H their respective Laplacians. Call $r = \max_e L_H(e) / \min_e L_H(e)$. Given $\epsilon > 0$, there exists an algorithm that returns a scalar y so that:*

$$(1-\epsilon)\, st_H(G) \leq y \leq (1+\epsilon)\, st_H(G)$$

with high probability and in expected time $\tilde{\mathcal{O}}(m\epsilon^{-2} \log(rn))$.

Proof. This is a straightforward consequence of Theorem 2 in [103]. Once the effective resistance of an edge can be approximated in time $\mathcal{O}(\log n/\epsilon^2)$, we can sum it and weight it by the conductance in G for each edge. □

5.3.5 Fast inexact estimates

The bound presented in Equation 5.3.2 has some interesting consequences if one is interested only in a rough estimate of the log-determinant: if $\epsilon = \mathcal{O}(1)$, it is possible to approximate the log-determinant in expected time $\tilde{O}\left(m + n\log^3 n\right)$. We will make use of this sparsification result from Spielman and Srivastava [103]:

Lemma 13. *(Theorem 12 in [103]). Given a Laplacian L_G with m edges, there is an expected $\tilde{O}(m/\epsilon^2)$ algorithm that produces a graph L_H with $\mathcal{O}(n\log n/\epsilon^2)$ edges that satisfies $(1-\epsilon)L_G \preceq L_H \preceq (1+\epsilon)L_G$.*

An immediate consequence is that given any graph, we can find a graph with a near-optimal stretch (up to an ϵ factor) and $\mathcal{O}(n\log n/\epsilon^2)$ edges.

Lemma 14. *Given a Laplacian L_G with m edges, there is an expected $\tilde{O}(m/\epsilon^2)$ algorithm that produces a graph L_H with $\mathcal{O}(n\log n/\epsilon^2)$ edges that satisfies $(n-1) \leq st_H(G) \preceq \frac{1+\epsilon}{1-\epsilon}(n-1)$.*

Proof. Consider a graph H produced by Lemma 13, which verifies $(1-\epsilon)L_G \preceq L_H \preceq (1+\epsilon)L_G$. Using the stretch over this matrix inequality, this implies:

$$\mathrm{st}_{(1+\epsilon)G}(G) \leq \mathrm{st}_H(G) \leq \mathrm{st}_{(1-\epsilon)G}(G)$$

which is equivalent to:

$$(1+\epsilon)^{-1}\mathrm{st}_G(G) \leq \mathrm{st}_H(G) \leq (1-\epsilon)^{-1}\mathrm{st}_G(G)$$

and the stretch of a connected graph with respect to itself is $n-1$. By rescaling H to $(1+\epsilon)^{-1}H$, we get:

$$n-1 \leq \mathrm{st}_H(G) \leq \frac{1+\epsilon}{1-\epsilon}(n-1)$$

□

Here is the main result of this section:

Proposition 1. *There exists an algorithm that on input $A \in SDD_n$, returns an approximation $n^{-1}\log|A|$ with precision $1/2$ in expected time $\tilde{O}\left(m + n\log^3 n\log^2\kappa(A)\right)$ with $\kappa(A)$ the condition number of A.*

Proof. Given L_A, compute H from Lemma 14 using $\epsilon = 1/16$ so that $(n-1) \leq \mathrm{st}_H(G) \preceq (1+1/8)(n-1)$. Then, using Theorem 6, this leads to the bound:

$$\mathrm{ld}(H) \leq \log|A| \leq \mathrm{ld}(H) + \frac{n-1}{4}$$

since H has $\mathcal{O}(n \log n)$ edges by construction, we can use Theorem 1 to compute a 1/4-approximation of $\mathrm{ld}(H)$ in expected time $\tilde{O}\left(n \log^3 n \log^2(\kappa(H))\right)$. By construction $\kappa(H) \leq \frac{1+1/16}{1-1/16}\kappa(A)$, hence the result. □

It would be interesting to see if this technique could be developed to handle arbitrary precision as well.

Comments

Since the bulk of the computations are performed in estimating the residue PLD, it would be interesting to see if this could be bypassed using better bounds based on the stretch. Also, even if this algorithm presents a linear bound, it requires a fairly advanced machinery (ST solvers) that may limit its practicality. Some heuristic implementation, for example based on algebraic multi-grid methods, could be a first step in this direction.

The author is much indebted to Satish Rao and Jim Demmel for suggesting the original idea and their helpful comments on an earlier draft of this discussion. This chapter is based on a collaboration with Ahmed El Alaoui and Alexandre Bayen, and is at this time being submitted to the SIAM Journal of Matrix Analysis and Applications.

Chapter 6

Conclusion

> He attacked everything in life with a mixture of extraordinary genius and naive incompetence and it was often difficult to tell which was which.
>
> Douglas Adams, *The Hitchhiker's guide to the galaxy*

We now conclude this dissertation. This thesis presents a study to the question of estimating travel times across a large road network. It studies the challenges of building a pipeline and estimation algorithms that operate at large scale from real-world datasets. In particular, it studies the limitations of available techniques when confronted to the diversity of data collected in the real world.

The private sector has already deployed traffic information systems that serve millions of individuals with great success. These systems collect very large amounts of data from users and in return give some information about travel times. However, this process does not give any confidence interval or statistical guarantees to the user. We propose in this dissertation several models that address this deficiency.

6.1 Advantages and deficiencies of data-driven models

When describing a phenomenon, a model should have two goals: *explain* and *predict*. A good model should reconcile some laws considered to hold within the context of the experiment with some observations. This is the explanation part. It should further be able to predict some aspects not observed yet. In all cases, there are some intrinsic variables within the model that need to be calibrated against the data and with respect to the physics behind the model. In physical models, these variables correspond to physical quantities that inform the scientist about the phenomenon. In statistical models, these variables are usually non-dimensional parameters that describe abstract correlations between random variables.

In the case of traffic, we make the case that loosing the comprehension that goes along with physical models gives access to a much richer class of models that describe the observations better.

The second goal of prediction can be quantitatively evaluated, although there will be always some argument about the proper metrics and protocols to evaluate the predictive power of a model. Purely statistical models, which use previous data to predict the distribution of future data, do not offer much insight about the system we are looking at. However, their calibration parameters can usually be refined using well-understood optimization techniques when presented with data, and the quality of their inference is directly related to the calibration. In particular, they can usually be adjusted to prevent overfitting. Models in which calibration parameters are physical constants can be much harder to calibrate to the data at hand. In particular, they rely on some assumptions that may not be verified in practice. Furthermore, it may not be possible to diagnose if all the assumptions required by these models are actually met within the experiment (such as deciding that some quantities are negligible against others). Naively calibrating them with observed data may lead to a good fit but some unrealistic choice of parameters. It is common to add some Bayesian priors or smoothing/regularization terms, but these terms are seldom grounded with some theoretical justification. Furthermore, data-driven models are often driven by smooth parameters and degrade gracefully with the lack of data. Physical models have a fixed number of parameters that depend on the theoretical considerations. This number is usually very hard to change because it depends on the relation between different physical quantities, unlike sparsity or BIC criterions. It ensues that physical models have a lower bound in terms of observation if they want to retain some predictive power and not overfit the existing data. On the other hand, the number of parameters of statistical models can be adjusted in an automated fashion based on the amount of data at hand.

In both cases, some diagnostic tools exist such as the bootstrap. It would be advantageous for the transportation community to investigate the confidence intervals of some bounds on the calibration of physical parameters, to ensure that the bias of the experimenter is not coming into play.

Statistical models are not restricted to a specific structure, and thus can be structured in a way that is more amenable to computations, but cannot be justified on theoretical grounds. For example, it is clear that the traffic on one road link will have an impact on surrounding road links, but it is a common assumption to consider each link to be independent. A lot of transportation literature has focused on adding some correlations, but the conservation laws behind it are challenging to model accurately. The simple model introduced in Chapter 4 can predict such correlations, but it does not give any insights about the cause of these correlations.

6.2 Summary of contributions

This thesis shows how a data-driven approach that relies on insights derived from the data, combined with a quantitative approach towards measurement goals, can lead to an estimation platform that is both robust and reliable, and that can work at very large scales with massive amounts of data. Physical models are used to inform the choice of statistical models, and new techniques are presented to use these statistical models at scale. We present a framework that works for different tradeoffs of quality of data, timeliness and computing resources.

Chapter 2 presents a principled approach for map-matching GPS data onto the road network. This approach works on a wide range of conditions, and is shown to encompass a large range of existing techniques. We have presented a novel class of algorithms to track moving vehicles on a road network: the *Path Inference Filter*. This algorithm first projects the raw points onto candidate projections on the road network and then builds candidate trajectories to link these candidate points. An observation model and a driver model are then combined in a Conditional Random Field to find the most probable trajectories.

The algorithm exhibits robustness to noise as well as to the peculiarities of driving in urban road networks. It is competitive over a wide range of sampling rates (1 seconds to 2 minutes) and greatly outperforms intuitive deterministic algorithms. Furthermore, given a set of ground truth data, the filter can be automatically tuned using a fast supervised learning procedure. Alternatively, using enough regular GPS data with no ground truth, it can be trained using unsupervised learning. Experimental results show that the unsupervised learning procedure compares favorably against learning from ground truth data.

Chapter 3 presents how to use very large quantities of extremely noisy observations at scale to estimate travel times on a large network. As datasets grow in size, some new strategies are required to perform meaningful computations in a short amount of time. We explored the implementation of a large-scale state estimation in near-real-time using Discretized Streams, a recently proposed streaming technique. Our traffic algorithm is an Expectation-Maximization algorithm that computes travel time distributions of traffic by incremental online updates. This approach was validated with a large dataset of GPS traces. This algorithm seems to compare favorably with the state of the art and shows some attractive scalability features from an implementation perspective. When distributed on a cluster, this algorithm scales to very large road networks (half a million road links, tens of thousands of observations per second) and can update traffic state in a few seconds.

Chapter 4 focuses on a travel time estimator that works with future GPS data and that takes into account the correlations of travel times. The state of the art for travel time estimation has focused on either precise and computationally intensive physical models, or large scale, data-driven approaches. We have presented a novel algorithm for travel time estimation that aims at combining the best of both worlds, by combining physical insights with some scalable algorithms. We model the variability of travel times due to stops at intersections using a Stop-Go model (to detect stops) and a Hidden Markov Model to learn the spatial dependencies between stop locations. We also take into account the spatio-temporal correlations of travel times due to driving behavior or congestion, using a Gaussian

Markov Random Fields. In particular, we present a highly scalable algorithm to train and perform inference on Gaussian Markov Random Fields, when applied on planar graphs.

We analyze the accuracy of the model using probe vehicle data collected over the Bay Area of San Francisco, CA. The results underline the importance to take into account the multi-modality of travel times in arterial networks due to the presence of traffic signals. The quality of the results we obtain are competitive with the state of the state of the art in traffic, and also highlight the good scalability of our algorithm. It remains to be seen how this system could be tested and deployed in an industrial setting. Furthermore, traffic information systems are only one component of a larger set of vital services we have grown accustomed to depend upon. It will be interesting to see how traffic can be combined with health monitoring, weather, public safety systems to provide a comprehensive view of a large megalopolis.

Bibliography

[1] Apache Hadoop. `http://hadoop.apache.org`.

[2] Inrix Inc. `http://www.inrix.com`.

[3] The mobile millennium project. http://traffic.berkeley.edu.

[4] NAVTEQ Inc. `http://www.navteq.com`.

[5] Scala programming language. `http://scala-lang.org`.

[6] Spark Project. `http://spark-project.org/`.

[7] Storm. https://github.com/nathanmarz/storm/wiki.

[8] StreamBase. `http://www.streambase.com`.

[9] The Berkeley Open Traffic Stack (BOTS), streaming arterial module. `http://github.com/calpath/bots-arterial-streaming/`.

[10] The Cabspotting program. `http://cabspotting.org/`.

[11] Supporting code for the path inference filter. `https://github.com/tjhunter/Path-Inference-Filter/`.

[12] Ittai Abraham, Yair Bartal, and Ofer Neiman. Nearly tight low stretch spanning trees. *Foundations of Computer ...*, pages 781–790, 2008.

[13] Dimitris Achlioptas. Database-friendly random projections. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 274–281. ACM, 2001.

[14] Noga Alon, RM Karp, D Peleg, and Douglas West. A graph-theoretic game and its application to the k-server problem. *SIAM Journal on Computing*, 24(1):78–100, 1995.

[15] Mohamed-Slim Alouini, Ali Abdi, and Mostafa Kaveh. Sum of gamma variates and performance of wireless communication systems over nakagami-fading channels. *Vehicular Technology, IEEE Transactions on*, 50(6):1471–1480, Nov 2001.

[16] Peter W Atkins and Ronald S Friedman. *Molecular quantum mechanics*. Oxford university press, 2011.

[17] Zhaojun Bai, Mark Fahey, and Gene H. Golub. Some large-scale matrix computation problems. *Journal of Computational and Applied ...*, 74(1):71–89, 1996.

[18] Xuegang (Jeff) Ban, Ryan Herring, Jean-Dominique Margulici, and Alexandre Bayen. Optimal sensor placement for freeway travel time estimation. *Proceedings of the 18th International Symposium on Transportation and Traffic Theory*, July 2009.

[19] Ronald Paul Barry and R. Kelley Pace. Monte Carlo estimates of the log determinant of large sparse matrices. *Linear Algebra and its Applications*, 1999.

[20] Alexandre Bayen, Joseph Butler, and Anthony Patire. Mobile Millennium final report. Technical report, University of California, Berkeley, CCIT Research Report UCB-ITS-CWP-2011-6, 2011.

[21] Jon L. Bentley and Hermann A. Maurer. Efficient worst-case data structures for range searching. *Acta Informatica*, 13:155–168, 1980. 10.1007/BF00263991.

[22] Shannon Bernardson, Paul McCarty, and Chris Thron. Monte carlo methods for estimating linear combinations of inverse matrix entries in lattice {QCD}. *Computer Physics Communications*, 78(3):256 – 264, 1994.

[23] Michel Bierlaire and Gunnar Flötteröd. Probabilistic multi-modal map matching with rich smartphone data. In *STRC 2011*, 2011.

[24] Michel Bierlaire and Emma Frejinger. Route choice modeling with network-free data. *Transportation Research Part C: Emerging Technologies*, 16(2):187–198, 2008.

[25] Jeff A. Bilmes. A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models. *International Computer Science Institute*, 4, 1998.

[26] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[27] Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk. On map-matching vehicle tracking data. In *Proceedings of the 31st international conference on Very large data bases*, pages 853–864. VLDB Endowment, 2005.

[28] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J Franklin, Joseph M Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel R Madden, Fred Reiss, and Mehul A Shah. Telegraphcq: continuous dataflow processing. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 668–668. ACM, 2003.

[29] Yanqing Chen, Timothy A Davis, William W Hager, and Sivasankaran Rajamanickam. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Transactions on Mathematical Software (TOMS)*, 35(3):22, 2008.

[30] Caspar G Chorus, Eric JE Molin, and Bert Van Wee. Use and effects of advanced traveller information services (atis): a review of the literature. *Transport Reviews*, 26(2):127–149, 2006.

[31] Youjing Cui and Shuzhi (Sam) Ge. Autonomous vehicle positioning with gps in urban canyon environments. *Robotics and Automation, IEEE Transactions on*, 19(1):15–25, 2003.

[32] Timothy A Davis and William W Hager. Dynamic supernodes in sparse cholesky update/downdate and triangular solves. *ACM Transactions on Mathematical Software (TOMS)*, 35(4):27, 2009.

[33] Philippe de Forcrand and Rajan Gupta. Multigrid techniques for quark propagator. *Nuclear Physics B - Proceedings Supplements*, 9(0):516 – 520, 1989.

[34] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977.

[35] Luc Devroye. *Non-uniform random variate generation*, volume 4. Springer-Verlag New York, 1986.

[36] Anthony Downs. *Still stuck in traffic: coping with peak-hour traffic congestion*. Brookings Institute Press, 2004.

[37] Jie Du, Jason Masters, and Matthew Barth. Lane-level positioning for in-vehicle navigation and automated vehicle location (avl) systems. In *Intelligent Transportation Systems, 2004. Proceedings. The 7th International IEEE Conference on*, pages 35–40. IEEE, 2004.

[38] John Duchi, Stephen Gould, and Daphne Koller. Projected subgradient methods for learning sparse gaussians. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*, 2008.

[39] Iain S Duff, Albert Maurice Erisman, and John Ker Reid. *Direct methods for sparse matrices*. Clarendon Press Oxford, 1986.

[40] Anthony Duncan, Estia Eichten, and Hank B. Thacker. Efficient algorithm for qcd with light dynamical quarks. *Physical Review D*, 59(1):014505, 1998.

[41] Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression. *The Annals of statistics*, 32(2):407–499, 2004.

[42] Maan El Badaoui El Najjar and Pierre Bonnifait. A road-matching method for precise vehicle localization using belief theory and kalman filtering. *Autonomous Robots*, 19(2):173–191, 2005.

[43] David G. Forney Jr. The Viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[44] Luca Giovannini. *A Novel Map-Matching Procedure for Low-Sampling GPS Data with Applications to Traffic Flow Analysis*. Universita di Bologna, 2011.

[45] Joshua S. Greenfeld. Matching GPS observations to locations on a digital map. In *81th Annual Meeting of the Transportation Research Board*, 2002.

[46] Keith D. Gremban. *Combinatorial Preconditioners for Sparse, Symmetric, Diagonally Dominant Linear Systems*. PhD thesis, Carnegie Mellon University, 1996.

[47] Lie Gu, Eric P Xing, and Takeo Kanade. Learning gmrf structures for spatial priors. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'07)*, pages 1–6. IEEE, 2007.

[48] Fredrik Gustafsson, Fredrik Gunnarsson, Niclas Bergman, Urban Forssell, Jonas Jansson, Rickard Karlsson, and Per-Johan Nordlund. Particle filters for positioning, navigation, and tracking. *Signal Processing, IEEE Transactions on*, 50(2):425–437, 2002.

[49] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[50] Bruce R Hellinga and Liping Fu. Reducing bias in probe-based arterial link travel time estimates. *Transportation Research Part C: Emerging Technologies*, 10(4):257–273, 2002.

[51] Aude Hofleitner and Alexandre M. Bayen. Optimal decomposition of travel times measured by probe vehicles using a statistical traffic flow model. In *14th IEEE Intelligent Transportation Systems Conference*, pages 815 –821, October 2011.

[52] Aude Hofleitner, Etienne Come, Latifa Oukhellou, Jean-Patrick Lebacque, and Alexandre M. Bayen. Automatic inference of map attributes from mobile data. In *Intelligent Transportation Systems (ITSC), 2012 15th International IEEE Conference on*, pages 1687–1692, 2012.

[53] Aude Hofleitner, Ryan Herring, Pieter Abbeel, and Alexandre Bayen. Learning the dynamics of arterial traffic from probe data using a dynamic bayesian network. *Intelligent Transportation Systems, IEEE Transactions on*, 13(4):1679–1693, 2012.

[54] Aude Hofleitner, Ryan Herring, Pieter Abbeel, and Alexandre M. Bayen. Learning the dynamics of arterial traffic from probe data using a dynamic bayesian network. *IEEE Transactions on Intelligent Transportation Systems,*, 13(4):1679 –1693, Dec. 2012.

[55] Aude Hofleitner, Ryan Herring, and Alexandre Bayen. Arterial travel time forecast with streaming data: A hybrid approach of flow modeling and machine learning. *Transportation Research Part B: Methodological*, 46(9):1097–1122, 2012.

[56] Aude Hofleitner, Ryan Herring, and Alexandre M. Bayen. Arterial travel time forecast with streaming data: a hybrid flow model - machine learning approach. *Transportation Research Part B*, November 2012.

[57] Aude Hofleitner, Ryan Herring, and Alexandre M. Bayen. Probability distributions of travel times on arterial networks: a traffic flow and horizontal queuing theory approach. In *91st Transportation Research Board Annual Meeting*, number 12-0798, Washington, D.C., January 2012.

[58] Baik Hoh, Marco Gruteser, Ryan Herring, Jeff Ban, Dan Work, Juan Carlos Herrera, and Alexandre M. Bayen. Virtual trip lines for distributed privacy-preserving traffic monitoring. In *The Sixth Annual International conference on Mobile Systems, Applications and Services (MobiSys 2008)*, Breckenridge, U.S.A., June 2008.

[59] Bret Hull, Vladimir Bychkovsky, Yang Zhang, Kevin Chen, Michel Goraczko, Allen Miu, Eugene Shih, Hari Balakrishnan, and Samuel Madden. Cartel: a distributed mobile sensor computing system. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 125–138. ACM, 2006.

[60] Timothy Hunter, Pieter Abbeel, and Alexandre M. Bayen. The path inference filter: Model-based low-latency map matching of probe vehicle data. In *Algorithmic Foundations of Robotics X*, Springer Tracts in Advanced Robotics, pages 591:1–607:17, Heidelberg, Germany, 2012. Springer-Verlag.

[61] Timothy Hunter, Pieter Abbeel, and Alexandre M Bayen. The path inference filter: model-based low-latency map matching of probe vehicle data. pages 591–607, 2013.

[62] Timothy Hunter, Tathagata Das, Matei Zaharia, Alexandre Bayen, and Pieter Abbeel. Large-scale online expectation maximization with spark streaming. *NIPS workshop on parallel and large-scale machine learning*, 2012.

[63] Timothy Hunter, Ryan Herring, Pieter Abbeel, and Alexandre Bayen. The path inference filter: model-based low-latency map matching of probe vehicle data. *CoRR*, abs/1109.1966, 2011.

[64] Timothy Hunter, Teodor Moldovan, Matei Zaharia, Samy Merzgui, Justin Ma, Michael J. Franklin, Pieter Abbeel, and Alexandre M. Bayen. Scaling the mobile

millennium system in the cloud. In *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, SOCC '11, pages 28:1–28:8, New York, NY, USA, 2011. ACM.

[65] Timothy Hunter, Teodor Moldovan, Matei Zaharia, Samy Merzgui, Justin Ma, Michael J. Franklin, Pieter Abbeel, and Alexandre M. Bayen. Scaling the Mobile Millennium system in the cloud. In *SOCC '11*, page 28, 2011.

[66] Ilse C F Ipsen and Dean J Lee. Determinant approximations. *Numerical Linear Algebra with Applications (under ...*, (X), 2006.

[67] M. I Jordan, Z. Ghahramani, T. S Jaakkola, and L. K Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

[68] Michael I. Jordan, editor. *Learning in Graphical Models*. MIT Press, Cambridge, MA, USA, 1999.

[69] Jonathan A Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. A simple, combinatorial algorithm for solving sdd systems in nearly-linear time. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*, pages 911–920. ACM, 2013.

[70] Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching optimality for solving sdd linear systems. pages 235–244, 2010.

[71] Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-m log n time solver for sdd linear systems. pages 590–598, 2011.

[72] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[73] Runze Li and Agus Sudjianto. Analysis of computer experiments using penalized likelihood in gaussian kriging models. *Technometrics*, 47(2), 2005.

[74] James (Michael) Lighthill and Gerald B. Whitham. On kinematic waves. II. A theory of traffic flow on long crowded roads. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 229(1178):317–345, May 1955.

[75] Sejoon Lim, Christian Sommer, Evdokia Nikolova, and Daniela Rus. Practical route planning under delay uncertainty: Stochastic shortest path queries. *In RSS - Robotics: Science and Systems VIII*, 2012.

[76] Wei-Hua Lin, Amit Kulkarni, and Pitu Mirchandani. Short-term arterial travel time prediction for advanced traveler information systems. In *Intelligent Transportation Systems*, volume 8, pages 143–154. Taylor & Francis, 2004.

[77] Joseph W.H. Liu. The Role of Elimination Trees in Sparse Factorization. *SIAM Journal on Matrix Analysis and Applications*, 11(1):134–172, 1990.

[78] Kai Liu, Toshiyuki Yamamoto, and Taka Morikawa. Study on the cost-effectiveness of a probe vehicle system at lower polling frequencies. *International Journal of ITS Research*, 6(1):29–36, 2008.

[79] Per-Olov Löwdin. Quantum theory of many-particle systems. iii. extension of the hartree-fock scheme to include degenerate systems and correlation effects. *Physical review*, 97(6):1509, 1955.

[80] Dmitry M Malioutov, Jason K Johnson, Myung Jin Choi, and Alan S Willsky. Low-rank variance approximation in gmrf models: Single and multiscale approaches. *IEEE Transactions on Signal Processing*, 56(10):4621–4634, 2008.

[81] Ron J Martin. Approximations to the determinant term in Gaussian maximum likelihood estimation of some spatial models. *Communications in Statistics-Theory and Methods*, 22(1):189–205, 1992.

[82] Michael McCourt. A Stochastic Simulation for Approximating the log-Determinant of a Symmetric Positive Definite Matrix. *compare*, 2:1–10, 2008.

[83] Gérard A Meurant. *Computer Solution of Large Linear Systems*. North-Holland: Amsterdam, 1999.

[84] Tomio Miwa, Takaaki Sakai, and Taka Morikawa. Route identification and travel time prediction using probe-car data. *International Journal*, 2004.

[85] Todd K. Moon. The expectation-maximization algorithm. *Signal Processing Magazine, IEEE*, 13(6):47–60, 1996.

[86] Peter G. Moschopoulos. The distribution of the sum of independent gamma random variables. *Annals of the Institute of Statistical Mathematics*, 37(1):541–544, 1985.

[87] Kevin P. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, University of California at Berkeley, 1994.

[88] Radford Neal and Geoffrey Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In *Learning in Graphical Models*, pages 355–368. MIT Press, 1999.

[89] Washington Y. Ochieng, Mohammed Quddus, and Robert B. Noland. Map-matching in complex urban road networks. *Revista Brasileira de Cartografia*, 2(55), 2009.

[90] R. Kelley Pace and Ronald Barry. Sparse spatial autoregressions. *Statistics and Probability Letters*, 33(3):291 – 297, 1997.

[91] Jong-Sun Pyo, Dong-Ho Shin, and Tae-Kyung Sung. Development of a map matching method using the multiple hypothesis technique. In *Intelligent Transportation Systems, 2001. Proceedings. 2001 IEEE*, pages 23–27. IEEE, 2001.

[92] Mohammed A. Quddus, Washington Y. Ochieng, and Robert B. Noland. Current map-matching algorithms for transport applications: State-of-the art and future research directions. *Transportation Research Part C: Emerging Technologies*, 15(5):312–328, 2007.

[93] Mohammed A. Quddus, Washington .Y. Ochieng, Lin Zhao, and Robert B. Noland. A general map matching algorithm for transport telematics applications. *GPS solutions*, 7(3):157–167, 2003.

[94] Mohsen Ramezani and Nikolas Geroliminis. On the estimation of arterial route travel time distribution with markov chains. *Transportation Research Part B*, 46(10):1576–1590, 2012.

[95] Arnold Reusken. Approximation of the Determinant of Large Sparse Symmetric Positive Definite Matrices. *SIAM Journal on Matrix Analysis and Applications*, 23(3):799, 2002.

[96] Seth Rogers. Creating and evaluating highly accurate maps with probe vehicles. In *Intelligent Transportation Systems, 2000. Proceedings. 2000 IEEE*, pages 125–130, 2000.

[97] Samitha Samaranayake, Sebastien Blandin, and Alexandre M. Bayen. A tractable class of algorithms for reliable routing in stochastic networks. *Transportation Research Part C*, 20(1):199–217, 2011.

[98] David Schrank, Bill Eisele, and Tim Lomax. Ttis 2012 urban mobility report. *Texas A&M Transportation Institute. The Texas A&M University System*, 2012.

[99] David Schrank, Tim J. Lomax, and Texas Transportation Institute. *2009 Urban mobility report*. The Texas A&M University, 2009.

[100] Amartya Sen. *Rationality and freedom*. Harvard University Press, 2004.

[101] K. Seymore, A. McCallum, and R. Rosenfeld. Learning hidden Markov model structure for information extraction. In *AAAI-99 Workshop on Machine Learning for Information Extraction*, pages 37–42, 1999.

[102] Daniel A Spielman. Algorithms , Graph Theory , and Linear Equations in Laplacian Matrices. Technical report, Proceedings of the International Congress of Mathematicians, Hyderabad, India, 2010.

[103] Daniel A Spielman and Nikhil Srivastava. Graph Sparsification by Effective Resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.

[104] Daniel A Spielman and Shang-Hua Teng. A Local Clustering Algorithm for Massive Graphs and its Application to Nearly-Linear Time Graph Partitioning. 2008.

[105] Daniel A Spielman and Shang-Hua Teng. Nearly-Linear Time Algorithms for Preconditioning and Solving Symmetric , Diagonally Dominant Linear Systems. pages 1–48, 2009.

[106] Daniel A Spielman and Jaeoh Woo. A Note on Preconditioning by Low-Stretch Spanning Trees. pages 1–4, 2009.

[107] S. Syed and ME Cannon. Fuzzy logic-based map matching algorithm for vehicle navigation system in urban canyons. In *ION National Technical Meeting, San Diego, CA*, volume 1, pages 26–28, 2004.

[108] Telenav Inc. `http://www.telenav.com`.

[109] Arvind Thiagarajan, James Biagioni, Tomas Gerlich, and Jakob Eriksson. Cooperative transit tracking using smart-phones. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 85–98, New York, NY, USA, 2010. ACM.

[110] Arvind Thiagarajan, Lenin S. Ravindranath, Katrina LaCurts, Sivan Toledo, Jakob Eriksson, Samuel Madden, and Hari Balakrishnan. Vtrack: Accurate, energy-aware traffic delay estimation using mobile phones. In *7th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Berkeley, CA, November 2009.

[111] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

[112] Robert Tibshirani. Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[113] Dalia Tiesyte and Christian S Jensen. 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*. ACM, 2008.

[114] Nagendra R. Velaga, Mohammed A. Quddus, Abigail L. Bristow, and Yuheng Zheng. Map-aided integrity monitoring of a land vehicle navigation system. *Intelligent Transportation Systems, IEEE Transactions on*, 13(2):848–858, 2012.

[115] Martin J. Wainwright and Michael I. Jordan. Log-determinant relaxation for approximate inference in discrete Markov random fields. *IEEE Transactions on Signal Processing*, 54(6):2099–2109, June 2006.

[116] Martin J. Wainwright and Michael I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008.

[117] Carola Wenk, Randall Salas, and Dieter Pfoser. Addressing the need for map-matching speed: Localizing global curve-matching algorithms. In *Scientific and Statistical Database Management, 2006. 18th International Conference on*, pages 379–388. IEEE, 2006.

[118] Bradford S Westgate, Dawn B Woodard, David S Matteson, and Shane G Henderson. Travel time estimation for ambulances using bayesian data augmentation. *Submitted, Journal of the American Statistical Association*, 2011.

[119] Christopher E. White, David Bernstein, and Alain L. Kornhauser. Some map matching algorithms for personal navigation assistants. *Transportation Research Part C: Emerging Technologies*, 8(1-6):91–108, 2000.

[120] Gerald B. Whitham. A new approach to problems of shock dynamics part i two-dimensional problems. *Journal of Fluid Mechanics*, 2(02):145–171, 1957.

[121] Dan Work, Olli-Pekka Tossavainen, Sebastien Blandin, Alexandre M. Bayen, T. Iwuchukwu, and K. Tracton. An ensemble Kalman filtering approach to highway traffic estimation using GPS enabled mobile devices. In *Proceedings of the 47th IEEE Conference on Decision and Control*, pages 5062–5068, Cancun, Mexico, December 2008.

[122] Dan B. Work, Sebastien Blandin, Olli-Pekka Tossavainen, Ben Piccoli, and Alexandre M. Bayen. A traffic model for velocity data assimilation. *Applied Mathematics Research eXpress*, 2010(1):1, 2010.

[123] Jing Yuan, Yu Zheng, Chengyang Zhang, Xing Xie, and Guang-Zhong Sun. An interactive-voting based map matching algorithm. In *Proceedings of the 2010 Eleventh International Conference on Mobile Data Management*, MDM '10, pages 43–52, Washington, DC, USA, 2010. IEEE Computer Society.

[124] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

[125] Matei Zaharia, Tathagata Das, Haoyuan Li, Scott Shenker, and Ion Stoica. Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. In *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing*, pages 10–10. USENIX Association, 2012.

[126] Hao Zhang and Yong Wang. Kriging and cross-validation for massive spatial data. *Environmetrics*, 21(3-4):290–304, 2010.

[127] Tao Zhang, Diange Yang, Ting Li, Keqiang Li, and Xiaomin Lian. An improved virtual intersection model for vehicle navigation at intersections. *Transportation Research Part C: Emerging Technologies*, 19(3):413 – 423, 2011.

[128] Yunong Zhang, W.E. Leithead, D.J. Leith, and L. Walshe. Log-det approximation based on uniformly distributed seeds and its application to Gaussian process regression. *Journal of Computational and Applied Mathematics*, 220(1-2):198–214, October 2008.

[129] Yunong Zhang and William E. Leithead. Approximate implementation of the logarithm of the matrix determinant in Gaussian process regression. *Journal of Statistical Computation and Simulation*, 77(4):329–348, April 2007.

[130] Fangfang Zheng and Helene van Zuylen. Reconstruction of delay distribution at signalized intersections based on traffic measurements. In *Proceedings of 13th IEEE Intelligent Transportation Systems Conference*, pages 1819–1824, September 2010.

[131] Yuheng Zheng and Mohammed A. Quddus. Weight-based shortest-path aided map-matching algorithm for low-frequency positioning data. In *Transportation Research Board 90th Annual Meeting*, 2011.

[132] Hui Zou, Trevor Hastie, and Robert Tibshirani. On the "degrees of freedom" of the lasso. *The Annals of Statistics*, 35(5):2173–2192, 2007.

Appendices

Mais on ne se bat pas dans l'espoir du succès! Non! non! C'est bien plus beau lorsque c'est inutile!

Edmond Rostand, *Cyrano de Bergerac*

Appendix A: Proofs of Section 5.2

A.1 Proof of Theorem 2

Proof. The proof of this theorem follows the proof of the Main Theorem in [19] with some slight modifications. Using triangular inequality:

$$|y - \hat{y}_{p,l}| \leq |\mathbb{E}[\hat{y}_{p,l}] - \hat{y}_{p,l}| + |y - \mathbb{E}[\hat{y}_{p,l}]|$$

Since S is upper-bounded by $(1-\delta)\,I$, we have for all $k \in \mathbb{N}$:

$$\left|\mathrm{Tr}\left(S^k\right)\right| \leq n\,(1-\delta)^k$$

We have $\mathbb{E}[\hat{y}_{p,l}] = -\sum_{i=1}^{l} i^{-1} S^i$ and $y = -\sum_{i=1}^{\infty} i^{-1} S^i$. Using again triangle inequality, we

can bound the error with respect to the expected value:

$$\begin{aligned}
|y - \mathbb{E}\left[\hat{y}_{p,l}\right]| &= n^{-1}\left|\sum_{i=l+1}^{\infty}\frac{1}{i}\mathrm{Tr}\left(S^k\right)\right| \\
&\leq n^{-1}\sum_{i=l+1}^{\infty}\frac{1}{i}\left|\mathrm{Tr}\left(S^k\right)\right| \\
&\leq \frac{1}{n\left(l+1\right)}\sum_{i=l+1}^{\infty}\left|\mathrm{Tr}\left(S^k\right)\right| \\
&\leq \frac{1}{l+1}\sum_{i=l+1}^{\infty}\left(1-\delta\right)^k \\
&\leq \frac{1}{l+1}\frac{\left(1-\delta\right)^{l+1}}{\delta} \\
&\leq \frac{\left(1-\delta\right)^{l+1}}{\delta}
\end{aligned}$$

And since $\delta \leq -\log\left(1-\delta\right)$, for a choice of $l \geq \delta^{-1}\log\left(\frac{2}{\epsilon\delta}\right)$, the latter part is less than $\epsilon/2$. We now bound the first part using Lemma 2. Call H the truncated series:

$$H = -\sum_{i=1}^{m}\frac{1}{i}S^i$$

This truncated series is upper-bounded by 0 (H is negative, semi-definite). The lowest eigenvalue of the truncated series can be lower-bounded in terms of δ:

$$H = -\sum_{i=1}^{m}\frac{1}{i}S^i \succeq -\sum_{i=1}^{m}\frac{1}{i}\left(1-\delta\right)^i I \succeq -\sum_{i=1}^{+\infty}\frac{1}{i}\left(1-\delta\right)^i I = \left(\log\delta\right) I$$

We can now invoke Lemma 2 to conclude:

$$\mathbb{P}\left[\left|\frac{1}{p}\sum_{i=1}^{p}\left(\mathbf{u}_i^T\mathbf{u}_i\right)^{-1}\mathbf{u}_i^T H\mathbf{u}_i - n^{-1}\mathrm{Tr}\left(H\right)\right| \geq \frac{\epsilon}{2}\right] \leq 2\exp\left(-\frac{p\epsilon^2}{16\frac{\left(\log(1/\delta)\right)^2}{n} + 4\frac{\log(1/\delta)\epsilon}{3}}\right)$$

Thus, any choice of

$$p \geq 16\left(\frac{1}{\epsilon}+\frac{1}{n\epsilon^2}\right)\log\left(2/\eta\right)\log^2\left(\delta^{-1}\right) \geq \log\left(2/\eta\right)\epsilon^{-2}\left(16\frac{\left(\log\left(1/\delta\right)\right)^2}{n} + \frac{4}{3}\epsilon\log\left(\delta^{-1}\right)\right)$$

satisfies the inequality: $2\exp\left(-\frac{p\epsilon^2}{16n^{-1}\left(\log(1/\delta)\right)^2+4\log(1/\delta)\epsilon/3}\right) \leq \eta$. □

A.2 Proof of Corollary 1

Proof. We introduce some notations that will prove useful for the rest of the section:

$$H = I - B^{-1}A$$

$$S = I - B^{-1/2}AB^{-1/2}$$

with $B^{-1/2}$ the inverse of the square root[1] of the positive-definite matrix B. The inequality (5.2.2) is equivalent to $\kappa^{-1}B \preceq A \preceq B$, or also:

$$\left(1 - \kappa^{-1}\right) I \succeq I - B^{-1/2}AB^{-1/2} \succeq 0$$

$$\left(1 - \kappa^{-1}\right) I \succeq S \succeq 0 \tag{6.2.1}$$

The matrix S is a contraction, and its spectral radius is determined by κ. Furthermore, computing the determinant of $B^{-1}A$ is equivalent to computing the determinant of $I - S$:

$$\begin{aligned} \log|I - S| &= \log\left|B^{-1/2}AB^{-1/2}\right| \\ &= \log|A| - \log|B| \\ &= \log\left|B^{-1}A\right| \\ &= \log|I - H| \end{aligned}$$

and invoking Theorem 2 gives us bounds on the number of calls to matrix-vector multiplies with respect to S. It would seem at this point that computing the inverse square root of B is required, undermining our effort. However, we can reorganize the terms in the series expansion to yield only full inverses of B. Indeed, given $l \in \mathbb{N}^*$, consider the truncated

[1]Given a real PSD matrix X, which can be diagonalized: $X = Q\Delta Q^T$ with Δ diagonal, and $\Delta_{ii} \geq 0$. Call $Y = Q\sqrt{\Delta}Q^T$ the square root of X, then $Y^2 = X$.

series:

$$\begin{aligned}
y_l &= -\mathrm{Tr}\left(\sum_{i=1}^{l}\frac{1}{i}S^i\right) \\
&= -\sum_{i=1}^{l}\frac{1}{i}\mathrm{Tr}\left(S^i\right) \\
&= -\sum_{i=1}^{l}\frac{1}{i}\mathrm{Tr}\left(\sum_j \binom{j}{i-j}(-1)^j\left(B^{-1/2}AB^{-1/2}\right)^j\right) \\
&= -\sum_{i=1}^{l}\frac{1}{i}\sum_j \binom{j}{i-j}(-1)^j\,\mathrm{Tr}\left(\left(B^{-1/2}AB^{-1/2}\right)^j\right) \\
&= -\sum_{i=1}^{l}\frac{1}{i}\sum_j \binom{j}{i-j}(-1)^j\,\mathrm{Tr}\left(\left(B^{-1}A\right)^j\right) \\
&= -\sum_{i=1}^{l}\frac{1}{i}\mathrm{Tr}\left(\sum_j \binom{j}{i-j}(-1)^j\left(B^{-1}A\right)^j\right) \\
&= -\sum_{i=1}^{l}\frac{1}{i}\mathrm{Tr}\left(H^i\right)
\end{aligned}$$

Hence, the practical computation of the latter sum can be done on $A^{-1}B$. To conclude, if we compute $p = 16\left(\frac{1}{\epsilon}+\frac{1}{m\epsilon^2}\right)\log\left(2/\eta\right)\log^2\left(\kappa\right)$ truncated chains of length $l = \kappa\log\left(\frac{2\kappa}{\epsilon}\right)$, we get our result. This requires lp multiplications by A and inversions by B. □

A.3 Proof of Theorem 3

We prove here the main result of Section 5.2. In the following, A and B are positive-definite matrices in $\mathcal{S}_n$, and B is a $\kappa-$approximation of A ($A \preceq B \preceq \kappa A$). The following notations will prove useful:

$$S = I - B^{-1/2}AB^{-1/2} \tag{6.2.2}$$

$$R = I - B^{-1}A \tag{6.2.3}$$

$$\varphi = \kappa^{-1}$$

Recall the definition of the matrix norm. Given $M \in \mathcal{S}_n^+$, $\|M\|_B = \max_{x\neq 0}\sqrt{\frac{x^TMx}{x^TBx}}$

Lemma 15. *S and R are contractions for the Euclidian and $B-$norms:*

$$\begin{aligned} \|S\| &\leq 1-\varphi \\ \|R\| &\leq 1-\varphi \\ \|R\|_B &\leq (1-\varphi)^2 \end{aligned}$$

Proof. Recall the definition of the matrix norm: $\|S\| = \max_{x^T x \leq 1} \sqrt{x^T S x}$. Since we know from Equation (6.2.1) that $S \preceq (1-\varphi) I$, we get the first inequality.

The second inequality is a consequence of Proposition 3.3 from [105]: A and B have the same nullspace and we have the linear matrix inequality $A \preceq B \preceq \kappa A$, which implies that the eigenvalues of $B^{-1}A$ lie between $\kappa^{-1} = \varphi$ and 1. This implies that the eigenvalues of $I - B^{-1}A$ are between 0 and $1-\varphi$.

Recall the definition of the matrix norm induced by the B-norm over $\mathbb{R}^n$:

$$\begin{aligned} \|R\|_B &= \max_{x \neq 0} \frac{\|Rx\|_B}{\|x\|_B} \\ &= \max_{\|x\|_B^2 \leq 1} \sqrt{x^T R^T B R x} \\ &= \max_{x^T B x \leq 1} \sqrt{x^T R^T B R x} \\ &= \max_{y^T y \leq 1} \sqrt{y^T B^{-1/2} R^T B R B^{-1/2} y} \end{aligned}$$

and the latter expression simplifies:

$$\begin{aligned} B^{-1/2} R^T B R B^{-1/2} &= B^{-1/2} \left(I - A B^{-1}\right) B \left(I - B^{-1} A\right) B^{-1/2} \\ &= \left(I - B^{-1/2} A B^{-1/2}\right) \left(I - B^{-1/2} A B^{-1/2}\right) \\ &= S^2 \end{aligned}$$

so we get:

$$\|R\|_B = \left\|S^2\right\| \leq \|S\|^2 \leq (1-\varphi)^2$$

□

The approximation of the log-determinant is performed by computing sequences of power series $\left(R^k x\right)_k$. These chains are computed approximately by repeated applications of the R operator on the previous element of the chain, starting from a random variable x_0. We formalize the notion of an approximate chain.

Definition 5. Approximate power sequence. *Given a linear operator H, a start point $x^{(0)} \in \mathbb{R}^n$, and a positive-definite matrix D, we define an $\epsilon-$approximate power sequence as a sequence that does not deviate too much from the power sequence:*

$$\left\|x^{(k+1)} - H x^{(k)}\right\|_D \leq \epsilon \left\|H x^{(k)}\right\|_D$$

We now prove the following result that is quite intuitive: if the operator H is a contraction and if the relative error ϵ is not too great, the sum of all the errors on the chain is bounded.

Lemma 16. *Let H be a linear operator and D a norm over the space of that linear operator. Assume that the operator H is a contraction under this norm ($\|H\|_D < 1$) and consider $\rho \in (0,1)$ so that $\|H\|_D \leq (1-\rho)^2$. Consider $\left(x^{(k)}\right)_k$ a $\nu-$approximate power sequence for the operator H and the norm D. If $\rho \leq 1/2$ and $\nu \leq \rho/2$, the total error is bounded:*

$$\sum_{k=0}^{\infty} \left\|x^{(k)} - H^k x^{(0)}\right\|_D \leq 4\rho^{-2}\nu \left\|x^{(0)}\right\|_D$$

Proof. Call $\omega_k = \left\|x^{(k)} - H^k x^{(0)}\right\|_D$ and $\theta_k = \left\|Hx^{(k)}\right\|_D$. We are going to bound the rate of convergence of these two series. We have first using triangular inequality on the D norm and then the definition of the induced matrix norm.

$$\begin{aligned}
\theta_k &\leq \left\|Hx^{(k)} - H^k x^{(0)}\right\|_D + \left\|H^k x^{(0)}\right\|_D \\
&= \omega_k + \left\|H^k x^{(0)}\right\|_D \\
&\leq \omega_k + \|H\|_D^k \left\|x^{(0)}\right\|_D
\end{aligned}$$

We now bound the error on the ω_k sequence:

$$\begin{aligned}
\omega_{k+1} &= \left\|x^{(k+1)} - Hx^{(k)} + Hx^{(k)} - H^{k+1}x^{(0)}\right\|_D \\
&\leq \left\|Hx^{(k)} - H^{k+1}x^{(0)}\right\|_D + \left\|x^{(k+1)} - Hx^{(k)}\right\|_D \\
&\leq \|H\|_D \left\|x^{(k)} - H^k x^{(0)}\right\|_D + \nu \left\|Hx^{(k)}\right\|_D \\
&= \|H\|_D \omega_k + \nu\theta_k \\
&\leq \|H\|_D \omega_k + \nu \left(\omega_k + \|H\|_D^k \left\|x^{(0)}\right\|_D\right) \\
&\leq \left[\|H\|_D + \nu\right] \omega_k + \nu \|H\|_D^k \left\|x^{(0)}\right\|_D
\end{aligned}$$

The assumption $\rho \leq 1 - \sqrt{\|H\|_D}$ is equivalent to $\|H\|_D \leq (1-\rho)^2$, so the last inequality implies:

$$\omega_{k+1} \leq \left[(1-\rho)^2 + \nu\right] \omega_k + \nu (1-\rho)^{2k} \left\|x^{(0)}\right\|_D$$

Note that the inequality $(1-\rho)^2 + \nu \leq 1 - \rho$ is equivalent to $\nu \leq \rho - \rho^2$. Using the hypothesis, this implies:

$$\omega_{k+1} \leq (1-\rho)\,\omega_k + \nu (1-\rho)^{2k} \left\|x^{(0)}\right\|_D \tag{6.2.4}$$

We show by induction that:

$$\forall k, \omega_k \leq \frac{\nu \left\|x^{(0)}\right\|_D}{1 - \sqrt{1-\rho}} \left(\sqrt{1-\rho}\right)^{k-1}$$

Note first that

$$\begin{aligned}\omega_1 &= \left\|x^{(1)} - Hx^{(0)}\right\|_D \\ &\leq \nu \left\|Hx^{(0)}\right\|_D \\ &\leq \nu \left\|H\right\|_D \left\|x^{(0)}\right\|_D \\ &\leq \nu \left(1-\rho\right)^2 \left\|x^{(0)}\right\|_D \\ &\leq \nu \left\|x^{(0)}\right\|_D\end{aligned}$$

So this relation is verified for $k = 1$. Now, assuming it is true for k, we use Equation (6.2.4) to see that:

$$\begin{aligned}\omega_k &\leq \left(1-\rho\right)\omega_k + \nu\left(1-\rho\right)^{2k}\left\|x^{(0)}\right\|_D \\ &\leq \left(1-\rho\right)\omega_k + \nu\left(\sqrt{1-\rho}\right)^k\left\|x^{(0)}\right\|_D \\ &\leq \nu\left\|x^{(0)}\right\|_D\left[\frac{\left(1-\rho\right)}{1-\sqrt{1-\rho}}\left(\sqrt{1-\rho}\right)^{k-1} + \left(\sqrt{1-\rho}\right)^k\right] \\ &= \nu\left\|x^{(0)}\right\|_D\left(\sqrt{1-\rho}\right)^k\left[\frac{\sqrt{1-\rho}}{1-\sqrt{1-\rho}} + 1\right] \\ &= \frac{\nu\left\|x^{(0)}\right\|_D}{1-\sqrt{1-\rho}}\left(\sqrt{1-\rho}\right)^k\end{aligned}$$

which is the the property for $k+1$. Using this property, we can sum all the errors by a geometric series (note that $\omega_0 = 0$).

$$\sum_{k=1}^{\infty}\omega_k \leq \frac{\nu\left\|x^{(0)}\right\|_D}{1-\sqrt{1-\rho}}\sum_{k=0}^{\infty}\left(\sqrt{1-\rho}\right)^k = \frac{\nu\left\|x^{(0)}\right\|_D}{\left(1-\sqrt{1-\rho}\right)^2}$$

Finally, note that for $\rho \in (0, 1/2)$, the inequality $\nu \leq \rho/2$ implies $\nu \leq \rho - \rho^2$. Furthermore, by concavity of the square root function, we have $\sqrt{1-\rho} \leq 1 - \rho/2$ for $\rho \leq 1$. Thus, $\left(1-\sqrt{1-\rho}\right)^2 \geq \rho^2/4$ and we get our result. □

We can use the bound on the norm of A to compute bound the error with a preconditioner:

Lemma 17. *Consider A, B with the same hypothesis as above, $x_0 \in \mathbb{R}^n$, and the additional hypothesis $\nu \in \left(0, \frac{1}{2\kappa}\right)$ and $\kappa \geq 2$, and $\left(x_u\right)_u$ an $\nu-$approximate power sequence for the operator R with start vector x_0. Then:*

$$\left|\sum_{i=1}^{l}\frac{1}{i}x_0^T R^i x_0 - \sum_{i=1}^{l}\frac{1}{i}x_0^T x_i\right| \leq 4\nu\kappa^2\sqrt{\kappa\left(B\right)}\left\|x_0\right\|^2$$

where $\kappa\left(B\right)$ is the condition number of B.

Proof. Call $\hat{z}$ the truncated sequence:

$$\hat{z} = \sum_{i=1}^{l} \frac{1}{i} x_0^T x_i$$

This sequence is an approximation of the exact sequence z:

$$z = \sum_{i=1}^{l} \frac{1}{i} x_0^T R^i x_0$$

We now bound the error between the two sequences:

$$|\hat{z} - z| \leq \sum_{i=1}^{l} \frac{1}{i} \left| x_0^T \left(R^i x_0 - x_i \right) \right| \leq \sum_{i=1}^{l} \left| x_0^T \left(R^i x_0 - x_i \right) \right| \leq \sum_{i=1}^{l} \left| \left(B^{-1} x_0 \right)^T B \left(R^i x_0 - x_i \right) \right| \tag{6.2.5}$$

Using the Cauchy-Schwartz inequality, we obtain:

$$\left| \left(B^{-1} x_0 \right)^T B \left(R^i x_0 - x_i \right) \right| = \left| \left\langle B^{-1} x_0, R^i x_0 - x_i \right\rangle_B \right| \leq \left\| B^{-1} x_0 \right\|_B \left\| R^i x_0 - x_i \right\|_B \tag{6.2.6}$$

From Lemma (15), we have $\|R\|_B \leq (1-\varphi)^2$, and from the hypothesis, we have $\nu \in (0, \varphi/2)$ and $\varphi \leq 1/2$, so we can bound the deviation using the bound from Lemma 16:

$$\sum_{i=1}^{l} \left\| R^i x_0 - x_i \right\|_B \leq \sum_{i=1}^{\infty} \left\| R^i x_0 - x_i \right\|_B \leq 4\varphi^{-2} \nu \left\| x_0 \right\|_B = 4\kappa^2 \nu \left\| x_0 \right\|_B \tag{6.2.7}$$

Combining Equations (6.2.5), (6.2.6) and (6.2.7), we get:

$$|\hat{z} - z| \leq \left\| B^{-1} x_0 \right\|_B \sum_{i=1}^{l} \left\| R^i x_0 - x_i \right\|_B \leq 4\nu\kappa^2 \left\| B^{-1} x_0 \right\|_B \left\| x_0 \right\|_B$$

Finally, it is more convenient to consider the Euclidian norm for the norm of x_0. Call $\lambda_{\max}$ and $\lambda_{\min}$ the extremal eigenvalues of the positive semidefinite matrix B. By definition of the matrix norm: $\|x_0\|_B = \sqrt{x_0^T B x_0} \leq \sqrt{\lambda_{\max}} \|x_0\|$ and $\|B^{-1} x_0\|_B = \sqrt{x_0^T B^{-1} x_0} \leq \sqrt{\lambda_{\min}^{-1}} \|x_0\|$ so we get:

$$|\hat{z} - z| \leq 4\nu\kappa^2 \sqrt{\kappa(B)} \left\| x_0 \right\|^2$$

where $\kappa(B)$ is the condition number of B. □

We now have all the elements required for the proof of Theorem 3.

Proof. Consider $\mathbf{u}_j \sim \mathcal{N}(0, I_n)$ for $j = 1 \cdots p$, and $x_{i,j} = \begin{cases} \mathbf{u}_j / \|\mathbf{u}_j\| & i = 0 \\ x_{i-1,j} - C\left(A x_{i-1,j}\right) & i > 0 \end{cases}$

Call

$$z_{p,l} = \frac{1}{p}\sum_{j=1}^{p}\sum_{i=1}^{l}\frac{1}{i}\left(x_{0,j}\right)^T x_{i,j}$$

$$\hat{y}_{p,l} = \frac{1}{p}\sum_{j=1}^{p}\sum_{k=1}^{l}\frac{1}{k}\left(x_{0,j}\right)^T S^k x_{0,j}$$

By construction, $\left(x_{i,j}\right)_i$ is an $\nu-$approximate chain for the operator R. Applying Lemma 17 to the operator R under the norm B, we get:

$$|z_{p,l} - \hat{y}_{p,l}| \quad \leq \quad 4\nu\kappa^2\sqrt{\kappa\left(B\right)}\left[\frac{1}{p}\sum_{j=1}^{p}\left\|x_{0,j}\right\|^2\right] = 4\nu\kappa^2\sqrt{\kappa\left(B\right)}$$

since $\left\|x_{0,j}\right\|^2 = 1$, which gives us a deterministic bound. Consider $\nu \leq \min\left(\frac{\epsilon}{8\kappa^2\sqrt{\kappa(B)}}, \frac{1}{2\kappa}\right)$. Then $|z_{p,l} - \hat{y}_{p,l}| \leq \epsilon/2$. Furthermore:

$$|z_{p,l} - y| \leq |z_{p,l} - \hat{y}_{p,l}| + |y - \hat{y}_{p,l}|$$

and $\mathbb{P}\left[|y - \hat{y}_{p,l}| \geq \epsilon/2\right] \leq \eta$ for a choice of $p \geq 16\left(\frac{1}{\epsilon} + \frac{1}{n\epsilon^2}\right)\log\left(2/\eta\right)\log^2\left(\delta^{-1}\right)$ and $l \geq 4\kappa\log\left(\frac{n}{\delta\epsilon}\right)$. Hence, we get our bound result of

$$pl = 64\kappa\left(\frac{1}{\epsilon} + \frac{1}{n\epsilon^2}\right)\log\left(2/\eta\right)\log^2\left(\delta^{-1}\right)\log\left(\frac{n}{\delta\epsilon}\right)$$

□

Appendix B: Proofs of Section 5.3.1

We put here the proofs that pertain to Section 5.3.1.

B.1 Properties of the generalized Laplacian

Proof of Lemma 5.

Proof. The first statement is obvious from the construction of the grounded Laplacian.

Statment (2) is a direct consequence of the fact that $F_Z = PZP^T$ with $P = (I_n\, 0)$.

Then the third statement is a simple consequence of statement 2, as $\mathrm{ld}\left(Z\right) = \sum_i \log\lambda_i$ with $\left(\lambda_i\right)_i$ the $n-1$ positive eigenvalues of Z.

Statement (4) is straightforward after observing that the floating procedure is a linear transform from $\mathcal{S}_n$ to $\mathcal{S}_{n-1}$, so it preserves the matrix inequalities.

□

B.2 Technical lemmas for Theorem 1

This lemma generalizes Lemma 8.1 in [105].

Lemma 18. *Consider $A \in \mathcal{S}_n$ positive semi-definite, and $x \in \mathbb{R}^n$. Then $xx^T \preceq \left(x^T A^+ x\right) A$*

Proof. Without loss of generality, consider $x^T x = 1$. Consider the eigenvalue decomposition of A: $A = \sum_i \lambda_i u_i u_i^T$. Since $(u_i)_i$ is an orthonormal basis of $\mathbb{R}^n$, we only need to establish that $\left(u_i^T x\right)^2 \leq \left(x^T A^+ x\right) u_i^T A u_i$ for all i. The latter term can be simplified:

$$\begin{aligned}
\left(x^T A^+ x\right) u_i^T A u_i &= \left(x^T \left[\sum_j \lambda_j^{-1} u_j u_j^T\right] x\right) \lambda_i \\
&= \lambda_i \sum_j \lambda_j^{-1} \left(u_j^T x\right)^2 \\
&\geq \left(u_i^T x\right)^2
\end{aligned}$$

which is the inequality we wanted.

□

Lemma 19. *Jensen inequality for the matrix logarithm. Let $A \in \mathcal{S}_n$ be a positive semi-definite matrix with p positive eigenvalues. Then*

$$ld\,(A) \leq p \log\left(\frac{Tr\,(A)}{p}\right)$$

Proof. This is a direct application of Jensen's inequality. Call $(\lambda_i)_i$ the positive eigenvalues of A. Then $\mathrm{ld}\,(A) = \sum_i \log \lambda_i$. By concavity of the logarithm:

$$\sum_i \log \lambda_i \leq p \log\left(\frac{\sum \lambda_i}{p}\right) = p \log\left(\frac{\mathrm{Tr}\,(A)}{p}\right)$$

□