# The Path Inference Filter: Model-Based Low-Latency Map Matching of Probe Vehicle Data

Timothy Hunter, Pieter Abbeel, and Alexandre Bayen

***Abstract*—We consider the problem of reconstructing vehicle trajectories from sparse sequences of GPS points, for which the sampling interval is between 1 s and 2 min. We introduce a new class of algorithms, which are altogether called the *path inference filter* (PIF), that maps GPS data in real time, for a variety of tradeoffs and scenarios and with a high throughput. Numerous prior approaches in map matching can be shown to be special cases of the PIF presented in this paper. We present an efficient procedure for automatically training the filter on new data, with or without ground-truth observations. The framework is evaluated on a large San Francisco taxi data set and is shown to improve upon the current state of the art. This filter also provides insights about driving patterns of drivers. The PIF has been deployed at an industrial scale inside the *Mobile Millennium* traffic information system, and is used to map fleets of data in San Francisco and Sacramento, CA, USA; Stockholm, Sweden; and Porto, Portugal.**

***Index Terms*—Expectation-maximization algorithms, filtering, GPS data, Markov random fields, maximum likelihood estimation, network-free data, path observation generation, probabilistic map matching, route choice modeling.**

## NOTATION

| | |
|---|---|
| $\underline{\delta}(x,\, p)$ | Compatibility function between state $x$ and the start state of path $p$. |
| $\bar{\delta}(p,\, x)$ | Compatibility function between end state $x$ and the end state of path $p$. |
| $\epsilon = \sigma^{-2}$ | Stacked inverse variance. |
| $\eta = \eta(p\|x)$ | Transition model. |
| $\theta$ | Stacked vector of parameters. |
| $\mu$ | Weight vector. |
| $\xi_1, \xi_2$ | Simple features (the path length and distance of a point projection to its GPS measurement). |
| $\pi$ | Probability distribution. The variables are always indicated to disambiguate which variables are involved. |
| $\hat{\pi}$ | Probability distribution in the case of a dynamic Bayesian network (DBN). The variables are always indicated to disambiguate which variables are involved. |
| $\widetilde{\pi}$ | Expected plug-in distribution. |
| $\varsigma$ | Set of valid trajectories. |
| $\sigma$ | Standard deviation. |
| $\tau$ | Trajectory of a vehicle. |
| $\tau^*$ | Most likely trajectory given model $(\omega, \eta)$ and GPS track $g^{1:T}$. |
| $\phi(\tau\|g^{1:T})$ | Potential or unnormalized score of a trajectory. |
| $\phi_i^t$ | Maximum of all the potentials of the partial trajectories that end in state $x_i^t$. |
| $\phi^*$ | Maximum value over all the potentials of the trajectories compatible with $g^{1:T}$. |
| $\varphi(p)$ | Feature function. |
| $\psi(z^{1:L})$ | Generalized potential function. |
| $\omega = \omega(g\|x)$ | Observation model. |
| $\Omega(x)$ | Prior distribution over states $x$. |
| $g$ | GPS measurement (pair of latitude and longitude). |
| $(g^t)^{1:T}$ | Sequence of all $T$ GPS observations of a GPS track. |
| $I^t$ | Number of projected states of the GPS point at time index $t$ onto the road network. |
| $I$ | Number of mappings of the GPS point onto the road network. |
| $J$ | Number of all candidate trajectories between mappings $\mathbf{x}$ and $\mathbf{x}'$. |
| $J^t$ | Number of all trajectories between the mappings at time $t$ (i.e., $\mathbf{x}^t$) and the mappings at time $t+1$ $\mathbf{x}^{t+1}$. |
| $(l, o)$ | Location in the road network defined by a pair of road link $l$ and offset position $o$ on this link. |
| $L = 2T - 1$ | Complete length of a trajectory. |
| $\mathcal{L}$ | Expected likelihood. |
| $\mathcal{N} = (\mathcal{V},\, \mathcal{E})$ | Road network comprising some vertices (nodes) $\mathcal{N}$ and edges (roads) $\mathcal{E}$. |
| $x = (l, o)$ | State of the vehicle (typically a location on the road network). |
| $p$ | Path between one mapping $x$ and one subsequent mapping $x'$. |
| $\mathbf{p} = (p_j)_{1:J}$ | Collection of all $J$ candidate trajectories between a set of candidate states $\mathbf{x}$ and subsequent set $\mathbf{x}'$. |
| $\mathbf{p}^t = (p_j^t)_{1:J^t}$ | Collection of all $J$ candidate trajectories between the set of candidate states at time $t$ $\mathbf{x}^t$ and a subsequent set of states $\mathbf{x}^{t+1}$. |

Manuscript received May 14, 2013; revised July 12, 2013; accepted September 2, 2013. Date of publication October 21, 2013; date of current version March 28, 2014. This paper was supported in part by grants from Google Inc., SAP AG, Amazon Web Services, Inc., Cloudera, Ericsson, Huawei Technologies Company, Ltd., IBM Corporation, Intel Corporation, MarkLogic, Microsoft Corporation, NEC Laboratories America Inc., NetApp Inc., Oracle Corporation, Splunk, and VMWare, Inc.; by the U.S. Department of Transportation; by the California Department of Transportation; and by Nokia Corporation and NAVTEQ under the Mobile Millennium Project. The Associate Editor for this paper was Z. Li.

The authors are with the Department of Electrical Engineering and Computer Science, College of Engineering, University of California at Berkeley, Berkeley, CA 94720-1776 USA (e-mail: tjhunter@eecs.berkeley.edu; pabbeel@cs.berkeley.edu; bayen@berkeley.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TITS.2013.2282352

| | |
|---|---|
| $\bar{q}_i^t$ | Probability that the vehicle is in the discrete state $x_i^t$ at time $t$ given all observations. |
| $\overrightarrow{q}_i^t$ | Probability that the vehicle is in the discrete state $x_i^t$ at time $t$ given all observations up to time $t$. |
| $\overleftarrow{q}_i^t$ | Probability that the vehicle is in the discrete state $x_i^t$ at time $t$ given all observations after time $t+1$. |
| $\bar{r}_j^t$ | Probability that the vehicle uses the (discrete) path $p_j^t$ at time $t$ given all observations. |
| $\overrightarrow{r}_j^t$ | Probability that the vehicle uses the (discrete) path $p_j^t$ at time $t$ given all observations up to time $t$. |
| $\overleftarrow{r}_j^t$ | Probability that the vehicle uses the (discrete) path $p_j^t$ at time $t$ given all observations after time $t+1$. |
| $T$ | Number of GPS observations for a track. |
| $T^l(z^l)$ | Generalized feature vector. |
| $Z$ | Partition function. |

## I. Introduction

AMONGST the modern man-made plagues, traffic congestion is a universally recognized challenge [11]. Building reliable and cost-effective traffic monitoring systems is a prerequisite to addressing this phenomenon. 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 [45]. The estimation problem is more challenging in the case of the secondary road network, which is also called the *arterial network*, due to the cost of deploying a wide network of sensors in large metropolitan areas. The most promising source of data is the GPS receiver in personal smartphones and commercial fleet vehicles [20]. According to some studies [35], devices with a data connection and a GPS will represent 80% of the cellular phone market by 2015. GPS observations in cities are noisy [10] and are usually provided at low sampling rates (on the order of 1 min) [9]. One of the common problems that 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, which is called the *path inference filter* (PIF), 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, Berkeley, CA, USA [2]. Mobile Millennium infers real-time traffic conditions using GPS measurements from drivers running cellular phone applications, taxis, 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, CA, USA; Stockholm, Sweden; and Porto, Portugal.

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 for routing and navigation (for individuals). These receivers are usually attached to a car or a truck, which is also called a *probe vehicle*, and they relay information to a base station using the data channels of cellular phone networks (third- or fourth-generation networks). Typical data provided by a probe vehicle include an identifier of the vehicle, a (noisy) position, and a timestamp.[1] Fig. 1 graphically shows a subset of probe data collected by *Mobile Millennium*. In addition to these geolocalization attributes, data points contain other attributes such as heading, spot speed, etc. It is shown in Fig. 2 how this additional information can be integrated into the rest of the framework presented in this paper.

Fig. 1. Example of a data set available to *Mobile Millennium* and processed by the PIF: Taxis in San Francisco from the Cabspotting program [9]. Large red circles show the position of the taxis at a given time, and small black dots show past positions (during the last 5 h) of the fleet. The position of each vehicle is observed every minute.

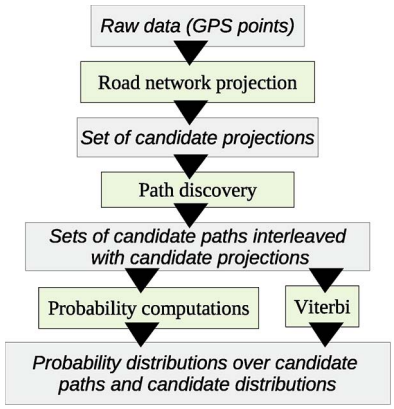


Fig. 2. Data flow of the PIF. The filter takes GPS measurements as inputs and returns probability distributions over paths and points.

The two most important characteristics of GPS data for traffic estimation purposes are the GPS localization accuracy and the sampling strategy followed by the probe vehicle. To reduce power consumption or transmission costs, probe vehicles do not continuously report their location to the base station. The probe

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

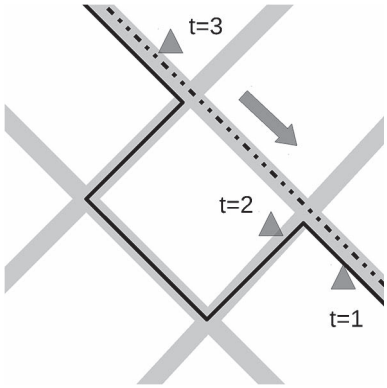


Fig. 3. Example of a failure case in which an intuitive algorithm projects each GPS measurement to the closest link. The raw GPS measurements are the triangles, the actual true trajectory is the dashed line, and the reconstructed trajectory is the continuous line. Due to noise in the observation, the point at $t = 2$ is closer to the orthogonal road and forces the algorithm to add a left turn, although the vehicle is actually going straight. This problem is frequently observed for GPS data in cities. The PIF provides one solution to this problem.

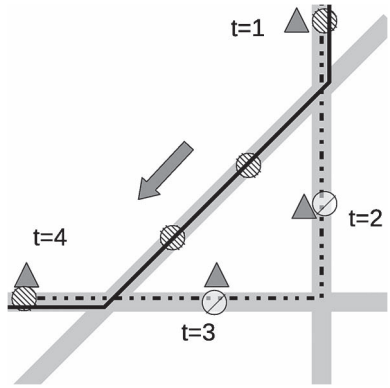


Fig. 4. Example of a failure case when trying to minimize the path length between a sequence of points. The raw observations are the triangles, the actual true trajectory is the dashed line, and the reconstructed trajectory is the continuous line. The circles are possible locations of the vehicle corresponding to the observations. The hashed circles are the states chosen by this reconstruction algorithm. Due to GPS errors that induce problems explained in Fig. 3, we must consider point projections on all links within a certain distance from the observed GPS points. However, the path computed by the 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.

data currently available are generated using a combination of the two following strategies.

- *Geographical sampling.* GPS probes are programmed to send information near *virtual landmarks* [25]. This concept was popularized by Nokia under the term *Virtual Trip Line* [21]. These landmarks are usually laid over some predetermined route followed by drivers.
- *Temporal sampling*. GPS probes send their position at a 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 an observation per second), some highly successful methods have been developed for continuous estimation [12], [26], [41]. However, most data collected at large scale today are generated by commercial fleet vehicles. It is primarily used for tracking the vehicles and usually has a low temporal resolution (1 to 2 min) [9], [23], [29], [38]. 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 nontrivial task; there are usually several roads that could be compatible with any given GPS observation. Simple deterministic algorithms to reconstruct trajectories fail due to shortcuts (see Fig. 3) or misprojection (see Fig. 4). These 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 [3]. Researchers started systematic studies after the introduction of the GPS system to civilian applications in the 1990s [33], [42]. These early approaches followed a *geometric* perspective, associating each observation data to some point in the network [44]. Later, this projection technique was refined to use more information such as heading and road curvature. This greedy matching, however, led to poor trajectory reconstruction since it did not consider the path leading up to a point [19], [34], [46]. New deterministic algorithms emerged to directly match partial trajectories to the road by using the topology of the network [16] and topological metrics based on the Fréchet distance [8], [43]. These deterministic algorithms could not readily cope with ambiguous observations [26] and were soon expanded into probabilistic frameworks. A number of implementations were explored: particle filters [17], [31], Kalman filters [30], hidden Markov models (HMMs) [4],[2] and less mainstream approaches based on belief theory [13] and fuzzy logic [37]. (See the very exhaustive review by Quddus *et al.* [32] 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 [17], [30], [41]. When sampling rates are lower and observed points are farther 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

[2]Note that "probabilistic" models, as well as most of the "advanced" models (Kalman filtering, particle filtering, and HMMs) fall under the general umbrella of *dynamic Bayesian filters*, which are presented in great detail in [41]. As such, they deserve a common theoretical treatment, and in particular, all suffer from the same pitfalls detailed in Section III.

two distinct problems. The first problem is path identification, and the second problem is projection matching [4], [15], [40], [46], [48]. Some interesting trajectories mixing points and paths that use a voting scheme have been also recently proposed [46]. Our filter aims at solving the two problems at the same time, by considering a single unified notion of *trajectory*.

The PIF 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 gracefully degrades 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* (CRFs) [24]. 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 PIF 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.*
2) Nearly all work on map matching is segmented into (and presents results for) either high- or low-frequency sampling. The PIF performs, as well as the current state-of-the-art approaches, for sampling rates less than 30 s, and it improves upon the state of the art [46], [48] by a factor of more than 10% for sampling intervals greater than 60 s.[3] We also analyze failure cases, and we show that the output provided by the PIF is always "close" to the true output for some metric.
3) As will be shown in Section III, most existing approaches (which are based on DBNs) do not work well at lower frequencies due to the *selection bias problem*. This paper directly addresses this problem by performing inference on a random field.
4) The PIF can be used with complex path models such as those used in [4] and [15]. In this paper, we restrict ourselves to a class of models (the exponential family distributions) that is rich enough to provide insights 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 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 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) [46]. The PIF is designed to work across the full spectrum of accuracy versus latency. As we will show in the following, 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).

## II. 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, which are referred to in this paper as the *links* of the road network. Each link is endowed with a number of physical attributes (link geometry, speed limits, number of lanes, type of road, etc.). Since each road link is directed, there will be other road links pointing at the start of that road link, and a set of road link 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 nonnegative 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 be easily incorporated into the following state:

$$x = (l, o).$$

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

### A. 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

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

[4]The complex intersections are not given special consideration. Adding some specific features such as those described in [47] would be straightforward.

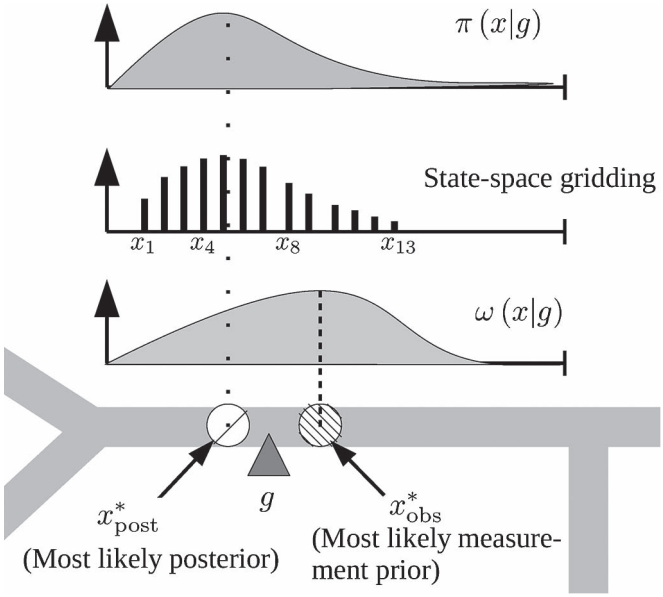


Fig. 5. Example of measurement $g$ on a link and two strategies to associate state projections to that measurement on a particular link (gridding and most likely location). The GPS measurement is the triangle denoted $g$. For this particular measurement, observation distribution $\omega(x|g)$ and posterior distribution $\pi(x|g)$ are also represented. When gridding, we select a number of states $x_1, \ldots, x_I$ 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 $x^*_{\text{post}}$ according to posterior distribution $\pi(x|g)$. However, this location depends on the prior distribution, which is usually not available at this stage (since the prior distribution 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 $x^*_{\text{obs}}$ according to the observation distribution.

generation process is represented by probability distribution $\omega(g|x)$ that, given state $x$, returns a probability distribution over all possible GPS observations $g$. Such distributions $\omega$ will be described in Section III-D. Additionally, we may have some *prior knowledge* over the state of the vehicle. For example, some links may be visited more often than the other links, and some positions on the 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, i.e.,

$$\pi(x|g) \propto \omega(g|x)\Omega(x). \tag{1}$$

$\pi$ will define general probabilities, and their dependence on variables will always be included. This probability distribution is defined up to a scaling factor 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 $\pi(o|g, l = l_i)$. The first way to accomplish this task is to grid the state space of each link, as shown in Fig. 5. This strategy is robust against the observation errors described in Section II-B, 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 with 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, i.e.,

$$o^*_{i_{\text{posterior}}} = \arg\max_o \pi(o|g, l = l_i) \qquad \forall\, l_i.$$

In practice, probability distribution $\pi(x|g)$ rapidly decays 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 300 m would give good results.

In the remainder of this paper, the boldface symbol $\mathbf{x}$ will denote a finite collection of states associated with a GPS observation $g$ that we will use to represent posterior distribution $\pi(x|g)$, and integer $I$ will denote its cardinality, i.e., $\mathbf{x} = (x_i)_{1:I}$. These points are called *candidate state projections for the GPS measurement* $g$. These discrete points will be then 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.

A Bayesian formulation requires that we endow state $x$ with 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 noninformative prior distribution is the uniform distribution over offsets, i.e., $\Omega \sim U([0, \text{length}(l)])$. In this case, maximizing the posterior distribution is equivalent to maximizing the conditional distribution from the generative model, i.e.,

$$o^*_{i_{\text{observation}}} = \arg\max_o \omega\left(g|x = (o, l_i)\right) \qquad \forall\, l_i. \tag{2}$$

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

### B. From Discrete Vehicle States to Trajectories

At each time step $t$, GPS point $g^t$ (originating from a single vehicle) is observed. This GPS point is then projected onto $I^t$ different candidate states denoted as $\mathbf{x}^t = x^t_1, \ldots, x^t_{It}$. 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 $x^t_i \in \mathbf{x}^t$ to some state $x^{t+1}_{i'} \in \mathbf{x}^{t+1}$. We denote the set of *candidate paths* between the observation $g^t$ and the next observation $g^{t+1}$ by $\mathbf{p}^t$, i.e.,

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

Each path $p^t_j$ goes from one of the projection states $x^t_i$ of $g^t$ to a projection state $x^{t+1}_{i'}$ of $g^{t+1}$. There may be multiple pairs of states to consider, and between each pair of states,

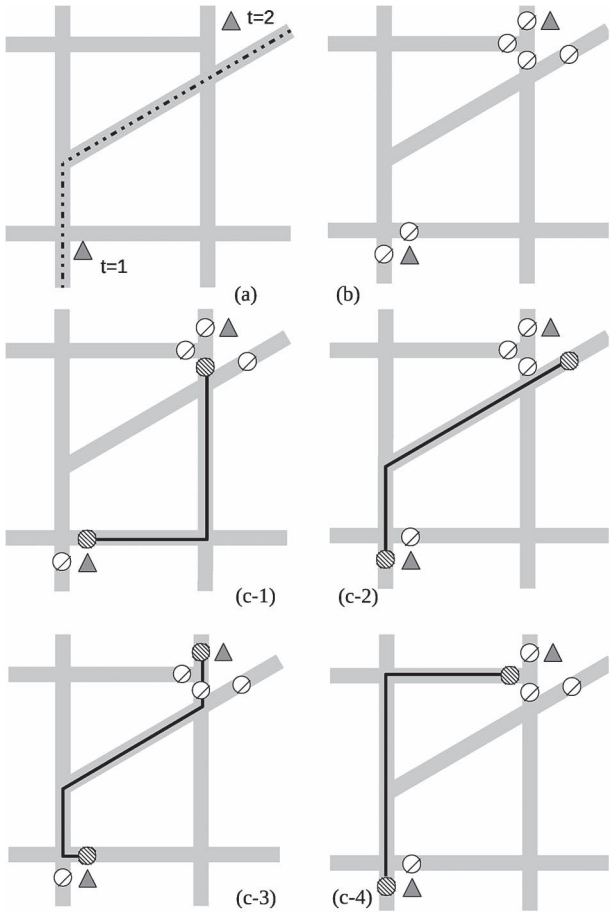


Fig. 6. Example of path exploration between two observations. (a) True trajectory and two associated GPS observations. (b) Set of candidate projections associated with each observation. A path discovery algorithm computes every acceptable path between each pair of candidate projections. (c-1)–(c-4) Examples of the computed paths.

there are typically several paths available (see Fig. 6). Finally, a *trajectory* is defined by the succession of states and paths, starting and ending with a state, as

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

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

Due to the speed limits leading to lower bounds on achievable travel times on the network, there is only a finite number of paths that a vehicle can take during time interval $\Delta t$. 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 speed $v_{\max}$ within the time interval between each observation. An algorithm that performs well in practice is the A$^*$ algorithm [18], which is 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. 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 mi/h), which makes the heuristic admissible.

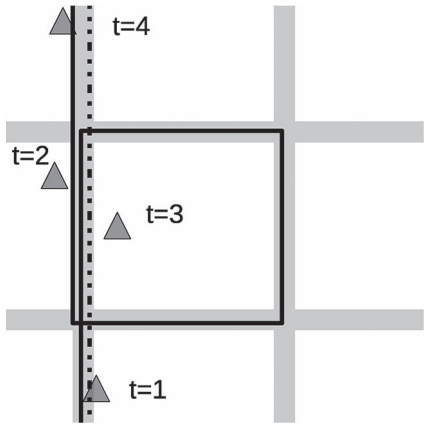


Fig. 7. Example of a failure case when observing strict physical consistency. Due to the observation noise, the observation (3) appears physically behind (2) on the same link. Without considering backward paths, the most plausible explanation is that the vehicle performed a complete loop around the neighboring 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 nondecreasing 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 Fig. 7. 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, e.g., 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 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

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

issue is solved using some heuristics. Our implementation creates a new state projection on 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 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 contain 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 point. In this situation, the probability model will return a probability of 0 (nonphysical trajectories) for any trajectory. If a point becomes unreachable from the start point, the trajectory is broken and is restarted from this point. Trajectory breaks were few (less than a dozen for our data set), 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 Section III-D and E.

The GPS also has a complex error model and may return some location estimates that are completely off the trajectory. We found these errors to appear at random and 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 that 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.

## III. Discrete Filtering Using a CRF

Earlier, we reduced the trajectory reconstruction problem to a discrete selection problem between sets of candidate projection points, which were interleaved with sets of candidate paths. A probabilistic framework can be now applied to infer a reconstructed trajectory $\tau$ or probability distributions over candidate states and 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.

### A. CRFs 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 II-A. Assuming that the vehicle is at point $x$, a GPS observation $g$ will be observed according to model $\omega$ 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 location $x^t$, GPS observation $g^t$ is created according to distribution $\omega(g^t|x^t)$.

Additionally, we endow the set of all possible paths on the road network with a probability distribution. *Transition model* $\eta$ describes the preference of a driver for a particular path. In probabilistic terms, it provides 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 start state $x_{\text{start}}$ and end state $x_{\text{end}}$, 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 path $p_{\text{next}}$ and preceded by a path $p_{\text{previous}}$, and associated to GPS measurement $g$. Then, the paths taken by the vehicle are independent from GPS measurement $g$ if 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. Being conditional on the start state and the 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 the 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

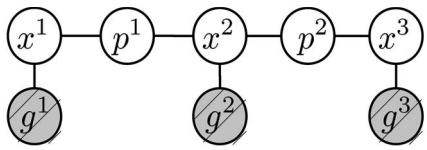


Fig. 8. CRF defined over a trajectory $\tau = x^1p^1x^2p^2x^3$ and a sequence of observations $g^{1:3}$. The gray nodes indicate the observed values. The solid lines indicate the factors between the variables: $\omega(g^t|x^t)$ between state $x^t$ and observation $g^t$, $\underline{\delta}(x^t, p^t)\eta(p^t)$ between state $x^t$ and path $p^t$, and $\bar{\delta}(p^t, x^{t+1})$ between path $p^t$ and subsequent state $x^{t+1}$.

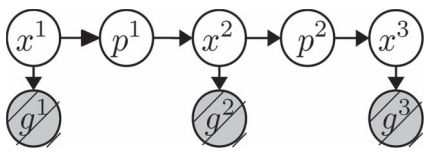


Fig. 9. DBN commonly used to model the trajectory reconstruction problem. The arrows indicate the directed dependence of the variables. GPS observations $g^t$ are generated from states $x^t$. The unobserved paths $p^t$ are generated from state $x^t$, following a transition probability distribution $\hat{\pi}(p|x)$. The transition from path $p^t$ to state $x^t$ follows the transition model $\hat{\pi}(x|p)$.

state $x$ and the start state of path $p$ with the compatibility function $\underline{\delta}$, i.e.,

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

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

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

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

$$\phi(\tau|g^{1:T}) = \left[\prod_{t=1}^{T-1} \omega(g^t|x^t)\underline{\delta}(x^t, p^t)\eta(p^t)\bar{\delta}(p^t, x^{t+1})\right] \cdot \omega(g^T|x^T). \tag{3}$$

Nonnegative function $\phi$ is called the *potential function.* Trajectory $\tau$ is said to be a *compatible trajectory with observation sequence* $g^{1:T}$ if $\phi(\tau|g^{1:T}) > 0$. When properly scaled, potential $\phi$ defines a probability distribution over all possible trajectories, given a sequence of observations, i.e.,

$$\pi(\tau|g^{1:T}) = \frac{\phi(\tau|g^{1:T})}{Z}. \tag{4}$$

Variable $Z$, which is called the *partition function*, is the sum of the potentials over all the compatible trajectories, i.e.,

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

We have combined observation model $\omega$ and transition model $\eta$ into a single potential function $\phi$, which defines an unnormalized distribution over all trajectories. Such a probabilistic framework is called a CRF [24]. A CRF is an undirected graphical model, which is defined as the unnormalized product of factors over cliques of factors (see Fig. 8). There can be an exponentially large number of paths; therefore, the partition function cannot be computed by simply summing the value of $\phi$ over every possible trajectory. As will be shown in Section IV, the value of $Z$ needs to be computed only during the training phase. Furthermore, it can be efficiently computed using dynamic programming.

*The case against the HMM approach.* The classical approach to filtering in the context of trajectories is based on HMMs or their generalization, i.e., DBNs [28]; a sequence of states and trajectories form a trajectory, and the coupling of trajectories

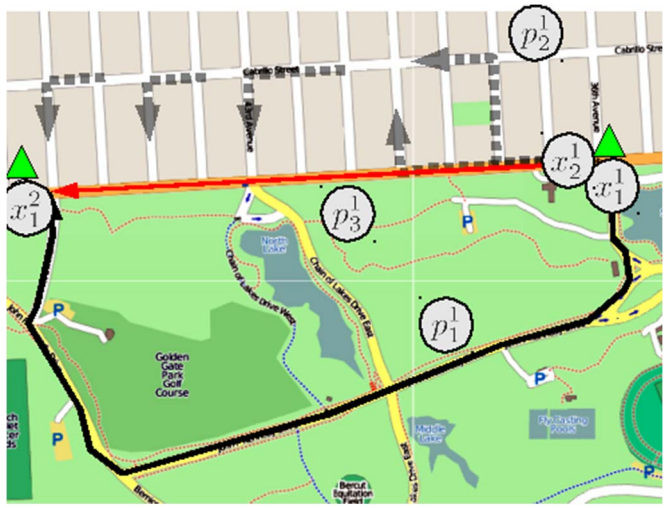


Fig. 10. Example of a failure case when using an HMM. The solid black path will be favored over all the other paths. Paths $p_i^1$ connect the states $x_j^1$ to $x_k^2$. See Section II for more details on the notations.

and states is done using transition models $\hat{\pi}(x|p)$ and $\hat{\pi}(p|x)$. (See Fig. 9 for a representation.)

This results in a chain-structured directed probabilistic graphical model in which 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 [30], [31], the forward algorithm or the Viterbi algorithm [4], [5], or particle filters [17].

HMM representations, however, suffer from the *selection bias problem*, which was first noted in the labeling of words sequences [24], making them not the best fit for solving path inference problems. Consider the example trajectory $\tau = x^1p^1x^2$ observed in our data, as represented in Fig. 10. For clarity, we consider only states $x_1^1$ and $x_2^1$ associated with the GPS reading $g^1$ and single state $x_1^2$ associated with $g^2$. The paths $(p_j^1)_j$ 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 start point. Since there is a single path from $x_2^1$ to $x^2$, the probability of taking this path from state $x_1^1$ will be $\hat{\pi}(p_1^1|x_1^1) = 1$; therefore, the overall probability of this path is $\hat{\pi}(p_1^1|g^1) = \hat{\pi}(x_1^1|g^1)$. Consider now the paths from $x_2^1$ to $x_1^1$. Several of these paths will have a similar weight since they correspond to different turns and across the lattice of roads. For each path $p$ among these $N$ paths of similar weight, the

Bayes assumption implies that $\hat{\pi}(p|x_2^1) \approx (1/N)$; therefore, the overall probability of this path is $\hat{\pi}(p|g^1) \approx (1/N)\hat{\pi}(x_2^1|g^1)$. In this case, $N$ can be large enough that $\hat{\pi}(p_1^1|g^1) \geq \hat{\pi}(p|g^1)$, and the remote path will be selected as the most likely path.

Due to their structures, all HMM models will be biased toward 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 a 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 CRFs, 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 $\tau$, i.e.,

$$\tau^* = \arg\max_{\tau} \pi(\tau|g^{1:T}) \tag{5}$$

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

As will be shown, 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* [28] and can be computed in time complexity linear in the time horizon by using a dynamic programming formulation. Computing the most likely trajectory is a particular instantiation of a standard dynamic programing algorithm called the *Viterbi algorithm* [14]. Using a classic machine learning algorithm for chain-structured junction trees (the *forward–backward algorithm* [6], [36]), all the conditional marginals can be computed in two passes over the variables. In the succeeding section, we detail the justification for the Viterbi algorithm, and in Section III-C, we describe an efficient implementation of the forward–backward algorithm in the context of this application.

### B. Finding the Most Likely Path

For the remainder 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 $t \in [1, \ldots, T-1]$, 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 following set $\varsigma$ of valid trajectories in the Cartesian space defined by these projections and these paths:

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

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 the most likely trajectory $\tau^*$ can be computed in $O(TI^*J^*)$ computations, with $I^* = \max_t I^t$ and $J^* = \max_t J^t$.

Partition function $Z$ does not depend on the current trajectory $\tau$ and need not be computed when solving (5), i.e.,

$$\begin{aligned} \tau^* &= \arg\max_{\tau \in \varsigma} \quad \pi(\tau|g^{1:T}) \\ &= \arg\max_{\tau \in \varsigma} \quad \phi(\tau|g^{1:T}). \end{aligned}$$

Define $\phi^*(g^{1:T})$ as the maximum value over all the potentials of the trajectories compatible with the observations $g^{1:T}$, i.e.,

$$\phi^*(g^{1:T}) = \max_{\tau \in \varsigma} \phi(\tau|g^{1:T}). \tag{6}$$

The trajectory $\tau$ 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 [1]. This is why we will only consider the computation of this maximum. Function $\phi$ depends on probability distributions $\omega$ and $\eta$, which are left undefined so far. These distributions will be presented in depth in Section III-D and E.

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

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

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

$$\begin{aligned} \phi(\tau^{1:t}|g^{1:t}) = \omega(g^1|x^1) \prod_{t'=1}^{t-1} \underline{\delta}\left(x^{t'}, p^{t'}\right) \eta\left(p^{t'}\right) \\ \cdot \bar{\delta}\left(p^{t'}, x^{t'+1}\right) \omega\left(g^{t'+1}|x^{t'+1}\right). \end{aligned} \tag{7}$$

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

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

One sees

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

The partial potentials defined in (7) follow an inductive identity, i.e.,

$$\begin{aligned} \phi_i^1 &= \omega\left(g^1|x_i^t\right) \\ \forall t, \phi_i^{t+1} &= \max_{\substack{i' \in [1, I^t] \\ j \in [1, J^t]}} \left[\phi_{i'}^t \underline{\delta}\left(x_{i'}^t, p_j^t\right) \eta\left(p_j^t\right) \right. \\ &\qquad \left. \cdot \bar{\delta}\left(p_j^t, x_i^{t+1}\right) \omega\left(g^{t+1}|x_i^{t+1}\right)\right]. \end{aligned} \tag{8}$$

By using this identity, the maximum potential $\phi^*$ can be computed efficiently from the partial maximum potentials $\phi_i^t$.

The computation of the trajectory that realizes this maximum potential ensues by tracing back the computation to find which partial trajectory realized $\phi_i^t$ for each step $t$.

### C. 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 $\pi(x^t|g^{1:T})$ and $\pi(p^t|g^{1:T})$ 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 state $x_i^t \in \mathbf{x}^t$ at time $t$, given all the observations $g^{1:T}$ of the trajectory, is defined as

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

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 state time $t$ (and not to the actual unscaled value of $\pi(x_i^t|g^{1:T})$). This is why we consider $(\bar{q}_i^t)_i$ as a choice between a (finite) set of discrete variables, i.e., one choice per possible state $x_i^t$. A natural choice is to scale distribution $\bar{q}_i^t$ so that the probabilistic weight of all possibilities is equal to 1, i.e.,

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

From a practical perspective, $\bar{q}^t$ can be computed without the knowledge of partition function $Z$. Indeed, the only required elements are the unscaled values of $\pi(x_i^t|g^{1:T})$ for each $i$. The distribution $\bar{q}^t = (\bar{q}_i^t)_i$ 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 among the set $(x_{i'}^t)_{1 \le i \le I^t}$, given all the observations $g^{1:T}$.

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

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

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

This problem of smoothing in CRFs is a classic application of the junction tree algorithm to chain-structured graphs [28]. For the sake of completeness, we derive an efficient smoothing algorithm using our notations.

The definition of $\pi(x_i^t|g^{1:T})$ requires summing the potentials of all the trajectories that pass through 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 allows us to 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 $t$:

$$\pi(x^t|g^{1:T}) \propto \pi(x^t|g^{1:t})\pi(x^t|g^{t+1:T}). \tag{9}$$

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 $\pi(x^t|g^{1:t})$ are related to each other by an equation that propagates *forward* in time, whereas the terms $\pi(x^t|g^{t+1:T})$ are related through an equation that goes *backward* in time. This is why we call $\pi(x^t|g^{1:t})$ the *forward distribution for the states*, and we denote it[6] by $(\overrightarrow{q}_i^t)_{1 \le i \le I^t}$, i.e.,

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

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

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

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*

$$\overleftarrow{q}_i^t \propto \pi\left(x_i^t|g^{t+1:T}\right)$$
$$\overleftarrow{r}_j^t \propto \pi\left(p_j^t|g^{t+1:T}\right).$$

Using this set of notations, (9) can be rewritten

$$\overline{q}_i^t \propto \overrightarrow{q}_i^t \cdot \overleftarrow{q}_i^t$$
$$\overline{r}_j^t \propto \overrightarrow{r}_j^t \cdot \overleftarrow{r}_j^t.$$

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

$$\overrightarrow{q}_i^1 \propto \pi\left(x_i^1|g^1\right)$$
$$\overrightarrow{r}_j^t \propto \eta\left(p_j^t\right) \left( \sum_{j:\underline{\delta}\left(x_i^t, p_j^t\right)=1} \overrightarrow{q}_i^t \right) \tag{10}$$
$$\overrightarrow{q}_i^t \propto \omega\left(x_i^t|g^t\right) \left( \sum_{j:\bar{\delta}\left(p_j^{t-1}, x_i^t\right)=1} \overrightarrow{r}_j^{t-1} \right). \tag{11}$$

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

$$\overleftarrow{q}_i^T \propto 1$$
$$\overleftarrow{r}_j^t \propto \eta\left(p_j^t\right) \left( \sum_{j:\bar{\delta}\left(p_j^t, x_i^{t+1}\right)=1} \overleftarrow{q}_i^{t+1} \right) \tag{12}$$
$$\overleftarrow{q}_i^t \propto \omega\left(x_i^t|g^t\right) \left( \sum_{j:\underline{\delta}\left(x_i^t, p_j^t\right)=1} \overleftarrow{r}_j^t \right). \tag{13}$$

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

Details of the forward algorithm and the backward algorithm are provided in Algorithms 1 and 2. The complete algorithm for smoothing is detailed in Algorithm 3.

**Algorithm 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:
$\forall i = 1 \cdots I^1$: $\overrightarrow{q}_i^1 \leftarrow \omega(x_i^1|g^1)$
Normalize $\overrightarrow{q}^1$
For every time step $t$ from 1 to $T-1$:
Compute the forward probability over the paths

$$\overrightarrow{r}_j^t \leftarrow \eta\left(p_j^t\right)\left(\sum_{j:\underline{\delta}\left(x_i^t,p_j^t\right)=1} \overrightarrow{q}_i^t\right) \qquad \forall j = 1,\ldots,J^t$$

Normalize $\overrightarrow{r}^t$
Compute the forward probability over the states

$$\overrightarrow{q}_i^{t+1} \leftarrow \omega\left(x_i^{t+1}|g^{t+1}\right)\left(\sum_{j:\bar{\delta}\left(p_j^t,x_i^{t+1}\right)=1} \overrightarrow{r}_j^t\right) \qquad \forall i = 1,\ldots,I^{t+1}$$

Normalize $\overrightarrow{q}^{t+1}$
Return the set of vectors $(\overrightarrow{q}^t)_t$ and $(\overrightarrow{r}^t)_t$

**Algorithm 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
$\overleftarrow{q}_i^T \leftarrow 1 \; \forall i = 1,\ldots I^T$
For every time step $t$ from $T-1$ to 1:
Compute the forward probability over the paths

$$\overleftarrow{r}_j^t \leftarrow \eta\left(p_j^t\right)\left(\sum_{j:\bar{\delta}\left(p_j^t,x_i^{t+1}\right)=1} \overleftarrow{q}_i^{t+1}\right) \qquad \forall j = 1,\ldots,J^t$$

Normalize $\overleftarrow{r}^t$
Compute the forward probability over the states

$$\overleftarrow{q}_i^t \leftarrow \omega\left(x_i^{t+1}|g^{t+1}\right)\left(\sum_{j:\underline{\delta}\left(x_i^t,p_j^t\right)=1} \overleftarrow{r}_j^t\right) \qquad i = 1,\ldots,I^t$$

Normalize $\overleftarrow{q}^t$
Return the set of vectors $(\overleftarrow{q}^t)_t$ and $(\overleftarrow{r}^t)_t$

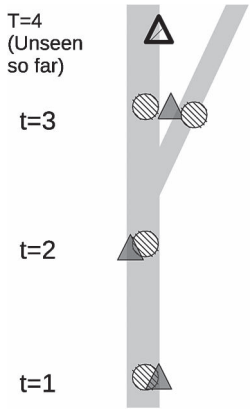


Fig. 11. Example of a case handled by lagged smoothing, which disambiguates the results provided by tracking. An observation is available close to an exit ramp of a highway, for which the algorithm has to decide if it corresponds to the vehicle exiting the highway. Lagged smoothing analyzes subsequent points in the trajectory and can disambiguate the situation.

Algorithm 3 (smoothing) requires all the observations of a trajectory 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 are available up to $t$ only. However, this algorithm can be adapted for a variety of scenarios:

- *Smoothing (offline filtering).* This corresponds to obtaining the best estimate given all observations, i.e., to computing $\pi(x^t|g^{1:T})$. The Algorithm 3 describes this procedure.
- *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 $\pi(x^t|g^{1:t})$. This is exactly the case the forward algorithm (Algorithm 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 (lagged filtering).* A few points of data are stored and processed before returning a result. Algorithm 4 details this procedure, which involves computing $\pi(x^t|g^{1:t+k})$ for some $k > 0$. A tradeoff 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 state $x^t$. As shown in Section VI, 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 Fig. 11.

**Algorithm 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 $(\overrightarrow{q}^t)_t$ and $(\overrightarrow{r}^t)_t$ using the forward algorithm.

Compute $(\overleftarrow{q}^t)_t$ and $(\overleftarrow{r}^t)_t$ using the backward algorithm.
For every time step $t$:
  $\overline{r}_j^t \leftarrow \overrightarrow{r}_j^t \cdot \overleftarrow{r}_j^t \; \forall\, j = 1, \ldots, J^t$
  Normalize $\overline{r}^t$
  $\overline{q}_i^t \leftarrow \overrightarrow{q}_i^t \cdot \overleftarrow{q}_i^t \; \forall\, i = 1, \ldots, I^t$
  Normalize $\overline{q}^t$
Return the set of vectors $(\overline{q}^t)_t$ and $(\overline{r}^t)_t$

**Algorithm 4** Lagged smoothing algorithm

Given integer $k > 0$, and a last-in–first-out 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 $\overline{r}^{t-k}$

### D. Observation Model

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

$$\omega(g|x) = p\left(d(g,\, x)\right) = \frac{1}{\sqrt{2\pi}\sigma}\left(-\frac{1}{2\sigma^2}d(g,\, x)^2\right) \qquad (14)$$

in which the function $d$ is the distance function between geocoordinates. The standard deviation $\sigma$ is assumed to be constant over all the network. This is not true in practice because of well-documented urban canyoning effects [10], [39], [40] 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 later in Sections IV-C and V-D, a value of $\sigma$ between 10 and 15 m could be estimated for the data of interest in this paper.

### E. 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 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, i.e.,

$$\eta(p) \propto \exp\left(\mu^T \varphi(p)\right). \qquad (15)$$

Function $\varphi$ is called the *feature function*, and vector $\mu$ is called the behavioral *parameter vector* and simply encodes a weighted combination of the features.

In a simple model, vector $\varphi(p)$ may be reduced to a single scalar, such as the length of the path. Then, the inverse of $\mu$, which is a length, can be interpreted as a characteristic length. This model simply states that the driver has a preference for shorter paths, and $\mu^{-1}$ indicates how aggressively this driver wants to follow the shortest path. Such a model is explored in Section VI. 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 PIF is the input to a model designed to learn travel times; therefore, the feature function does not include dynamic features such as the current travel time. Assuming this information is available, it would be easy to be added as a feature.

## IV. Training Procedure

The procedure detailed so far requires the calibration of the observation model and the driver behavior model by setting some values for weight vector $\mu$ and standard deviation $\sigma$. 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 VI. Computing likelihood will require the computation of the partition function (which depends on $\mu$ and $\sigma$). We first present a procedure that is valid for any path or point distributions that belong to the *exponential family*, and then we show how the models we presented in Section III fit into this framework.

### A. Learning Within the Exponential Family and Sparse Trajectories

There is a striking similarity between state variables $x^{1:T}$ and path variables $p^{1:T}$, particularly between the forward and backward distributions introduced in (10). This suggests to generalize our procedure to a context larger than the 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 $\overline{\delta}$). 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 succeeding section, we will describe the relation between the new variables that we are going to introduce and the parameters of our model.

Consider a joint sequence of multinomial random variables $\mathbf{z}^{1:L} = \mathbf{z}^1, \ldots, \mathbf{z}^L$ drawn from some space $\prod_{l=1}^{L}\{1, \ldots, K^l\}$, 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 nonnegative potential function $\psi(z^{1:L})$ over the sequence of variables. This potential function is controlled by parameter vector $\theta \in \mathbb{R}^M$:

$\psi(z^{1:L}) = \psi(z^{1:L};\theta)$.[7] Furthermore, we assume that this potential function is also defined and nonnegative over any subsequence $\psi(z^{1:l})$. Finally, we assume that there exists at least one sequence $z^{1:L}$ that has a positive potential. As in earlier section, the potential function $\psi$, when properly normalized, defines a probability distribution of density $\pi$ over variables $\mathbf{z}$, and this distribution is parametrized by vector $\theta$, i.e.,

$$\pi(z;\theta) = \frac{\psi(z;\theta)}{Z(\theta)} \tag{16}$$

with $Z = \sum_z \psi(z;\theta)$ being the *partition function*. We will show the partition function defined here is the partition function introduced in Section III-C.

We assume that $\psi$ is an unscaled member of the *exponential family*; it is of the form

$$\psi(z;\theta) = h(z)\prod_{l=1}^{L} e^{\theta\cdot T^l(z^l)}. \tag{17}$$

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

$$T^l_i = T^l(z^l = i) \qquad \forall\, i \in \{1,\ldots,K^l\}.$$

The sequence of variables $\mathbf{z}$ represents the choices associated with a single trajectory, i.e., the concatenation of the $x$'s and $p$'s. 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 $(\mathbf{z}^{(u)})_u$, which follows the given form and each can define a potential $\psi(z^{(u)};\theta)$ and a partition function $Z^{(u)}(\theta)$. 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 $\theta^*$ that maximizes the likelihood of all the realizations of the trajectories

$$\begin{aligned}\theta^* &= \arg\max_\theta \sum_u \log \pi^{(u)}\left(z^{(u)};\theta\right)\\ &= \arg\max_\theta \sum_u \log \psi\left(z^{(u)};\theta\right) - \log Z^{(u)}(\theta)\\ &= \arg\max_\theta \sum_u \sum_{l=1}^{L^{(u)}} \theta\cdot T^{l^{(u)}}\left(z^{l^{(u)}}\right) - \log Z^{(u)}(\theta)\end{aligned} \tag{18}$$

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 (18), 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,

$$\log\psi(z;\theta) - \log Z(\theta) = \sum_{l=1}^{L} \theta\cdot T^l(z^l) - \log Z(\theta). \tag{19}$$

The first part of (19) is linear with respect to $\theta$, and $\log Z(\theta)$ is concave in $\theta$. (It is the logarithm of a sum of exponentiated linear combinations of $\theta$ [7].) As such, maximizing (19) yields a unique solution (assuming no singular parametrization), and some superlinear algorithms exist to solve this equation [7]. 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.

*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 that the definition of the partition function is

$$Z(\theta) = \sum_z h(z)\prod_{l=1}^{L} e^{\theta\cdot T^l(z^l)}.$$

So far, function $h$ was defined in a generic way (it is nonnegative and does not depend on $\theta$). We consider a particular shape that generalizes the functions $\underline{\delta}$ and $\bar{\delta}$ introduced in the earlier section. In particular, function $h$ is assumed to be a binary function, from the Cartesian space $\prod_{l=1}^{L}\{1,\ldots,K^l\}$ to $\{0,1\}$, that decomposes to the product of the binary functions over consecutive pairs of variables, i.e.,

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

in which every function $h^l$ is a binary indicator $h^l : \{1,\ldots,K^l\}\times\{1,\ldots,K^{l-1}\} \to \{0,1\}$. These functions $h^l$ generalize functions $\underline{\delta}$ and $\bar{\delta}$ for arguments $z$ equal to either $x$'s or $p$'s. It indicates the compatibility of the values of instantiations $z^l$ and $z^{l-1}$.

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

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

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

[8]This is again a specific application of the junction tree algorithm. See [28] for an explanation of the general framework.

This partial summation can be also recursively defined as

$$Z_i^l(\theta) = e^{\theta \cdot T_i^l} \sum_{j \in [1,\ldots,K^{l-1}]: h^l(z^i, z^j)=1} Z_j^{l-1}(\theta). \tag{20}$$

The start of the recursion is for all $i \in \{1, \ldots, K^1\}$, i.e.,

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

and the complete partition function is the summation of the auxiliary values, i.e.,

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

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) 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 be also computed using (20), i.e.,

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

The Hessian matrix can be evaluated in similar fashion, i.e.,

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

### B. Exponential Family Models

We now express our formulation of CRFs to a form compatible with (17).

Consider $\epsilon = \sigma^{-2}$ and $\theta$ the stacked vector of the desired parameters, i.e.,

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

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

$$\begin{aligned} z^{2t} &= r^t \\ z^{2t-1} &= q^t \end{aligned}$$

and the feature vectors are simply the alternating values of $\varphi$ and $d$, which are completed by some zero values, i.e.,

$$\begin{aligned} T_i^{2t} &= \begin{pmatrix} 0 \\ \varphi\left(p_i^t\right) \end{pmatrix} \\ T_j^{2t-1} &= \begin{pmatrix} -\frac{1}{2} d\left(g, x_j^t\right)^2 \\ \mathbf{0} \end{pmatrix}. \end{aligned}$$

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.

### C. Supervised Learning With Known Trajectories

The most straightforward way to learn $\mu$ and $\sigma$ or, equivalently, to learn the joint vector $\theta$ 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 $p_{\text{observed}}^t$ was taken and which state $x_{\text{observed}}^t$ produced the GPS observation $g^t$. We make the assumption that the observed path $p_{\text{observed}}^t$ is one of the possible path among the set of candidate paths $(p_j^t)_j$, i.e.,

$$\exists j \in [1, \cdots, J^t]: \quad p_{\text{observed}}^t = p_j^t$$

and similarly, that the observed state $x_{\text{observed}}^t$ is one of the possible states

$$\exists i \in [1 \cdots I^t]: \quad x_{\text{observed}}^t = x_i^t.$$

In this case, the values of $r^t$ and $q^t$ are known (they are the matching indexes), and the optimization problem of (18) can be solved using methods outlined in Section IV-A.

### D. Unsupervised Learning With Incomplete Observations: EM

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 $\mathbf{z}^{1:L}$, i.e.,

$$\mathcal{L}(\theta) = \mathbb{E}_{z \sim \pi(\cdot|\theta)} \left[\log\left(\pi(z;\theta)\right)\right] \tag{21}$$

$$= \sum_z \pi(z;\theta) \log\left(\pi(z;\theta)\right). \tag{22}$$

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 $\pi(z;\theta)$. The challenge lies in the dependence in $\theta$ 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 (18).

[9] In particular, care should be taken to implement all the relevant computations in the log domain due to the limited precision of floating-point arithmetic on computers. The reference implementation [1] shows one way to do it.

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 [27]. This algorithm performs an iterative projection ascent by assigning some *distributions* (rather than singular values) to every $z^l$, and it uses these distributions to update parameters $\mu$ and $\sigma$ using the procedures shown in Section IV-C. This iterative procedure performs two steps.

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

$$\max_{\theta} \mathbb{E}_{z\sim\tilde{\pi}(\cdot)} \left[\log\left(\pi(z;\theta)\right)\right]. \tag{23}$$

This problem is significantly simpler than the optimization problem in (21) since the expectation itself does not depend on $\theta$ and is thus not part of the optimization problem.

An algorithmic description of this algorithm is given in Algorithm 5. Under this procedure, the expected likelihood is shown to converge to a local maximum [28]. It can be shown that good values for the plug-in distribution $\tilde{\pi}$ are simply the values of posterior distributions $\pi(p^t|g^{1:T})$ and $\pi(x^t|g^{1:T})$, i.e., the values $\bar{q}^t$ and $\bar{r}^t$. Furthermore, owing to the particular shape of the distribution $\pi(z)$, taking the expectation is a simple task; we simply replace the value of the feature vector by its *expected value* under distribution $\tilde{\pi}(z)$. More practically, we simply have to consider

$$\begin{aligned} T^{2t}(z^{2t}) &= \mathbb{E}_{p\sim\pi(\cdot|\theta,g^{1:T})}\left[\begin{pmatrix} 0 \\ \varphi\left(p_r^t\right)\end{pmatrix}\right] \\ &= \begin{pmatrix} 0 \\ \mathbb{E}_{p\sim\pi(\theta,g^{1:T})}\left[\varphi\left(p_r^t\right)\right]\end{pmatrix} \end{aligned} \tag{24}$$

in which

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

$$T^{2t-1}(z^{2t-1}) = \begin{pmatrix} \frac{1}{2}\mathbb{E}_{x\sim\pi(\cdot|\theta,g^{1:T})}\left[d\left(g, x_{q^t}^t\right)^2\right] \\ \mathbf{0}\end{pmatrix} \tag{25}$$

so that

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

These values of feature vectors plug directly into the supervised learning problem in (18) and produce updated parameters $\mu$ and $\sigma$, which are then used in turn for updating the values of $\bar{q}$ and $\bar{r}$ and so on.

**Algorithm 5** EM algorithm for learning parameters without complete observations.

Given a set of sequences of observations, an initial value of $\theta$
Repeat until convergence:
For each sequence, compute $\overline{r}^t$ and $\overline{q}^t$ using Algorithm 3.
For each sequence, update expected values of $T^t$ using (24) and (25).
Compute a solution of problem (18) using these new values of $T^t$.

## V. Results From Field Operational Test

The PIF and its learning procedures were tested using field data through the *Mobile Millennium* system. Ten San Francisco taxis were fit with a high-frequency GPS device (1-s sampling rate) in October 2010 during a two-day experiment. Together, they collected about 700 000 measurement points that provided high-accuracy ground truth. Additionally, the unsupervised learning filtering was tested on a significantly larger data set: one-day 1-min samples of 600 taxis from the same fleet, which represents 600 000 points. For technical reasons, the two data sets could not be collected the same day but were collected the same day of the week (a Wednesday), i.e., three weeks prior to the high-frequency collection campaign. Even if the GPS equipment was different, both data sets presented the same distribution of GPS dispersion. Thus, we evaluate two data sets collected from the same source with the same spatial features: a smaller set at high frequency, which is called Data Set 1, and a larger data set sampled at 1 min for which we do not know ground truth, which is called Data Set 2 (see Fig. 12). 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.

### A. Experiment Design

The testing procedure is described in Algorithm 6. The filter was first run in trajectory reconstruction mode (Viterbi algorithm) with settings that were tuned for a high-frequency application, using all the samples, 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:

**Algorithm 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

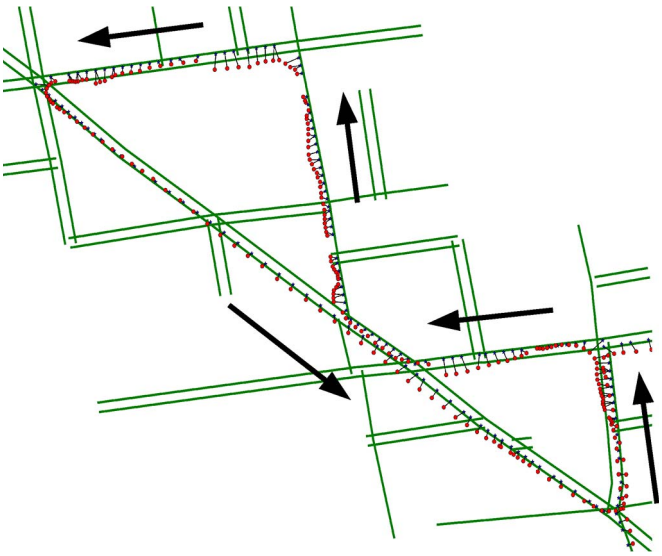

Fig. 12. Example of points collected in "Data Set 1," in the Russian Hill neighborhood in San Francisco, CA, USA. The red dots are the GPS observations (collected every second), and the green lines are road links that contain a state projection. The black lines show the most likely projections of the GPS points on the road network, using the Viterbi algorithm on a gridded state space with a 1-m grid for the offsets.

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

---

- *The sampling rate*. The following values were tested: 1 s, 10 s, 30 s, 1 min, 1.5 min, and 2 min.
- *The computing strategy.* Pure filtering ("online" or forward filtering), fixed-lagged smoothing with a one- or two-point buffer ("one-lag" and "two-lag" strategies), Viterbi and smoothing ("offline", or forward-backward procedure).
- Different models:
  - The "Hard Closest Point" model is 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 nonprobabilistic model is the baseline against which we make a comparison of our study [16]. This greedy model may lead to nonfeasible trajectories, e.g., by assigning an observation to a dead-end link from which it cannot recover.
  - The "Closest Point" model is a nongreedy version of the hard closest point model. 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 a dead end and force the trajectory to break.
  - The "Shortest Path" model is 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 point projections are then recovered from the paths. This is similar to the model in [15] and [44].
  - The "Simple" model considers two features that could be tuned by hand:
    1) $\xi_1$, which is the length of the path;
    2) $\xi_2$, which is 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."
  - The "Complex" model is a 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 Data Set 1, which was obtained from high-frequency samples. Since this data set is quite small, the EM procedure was able to find an unbounded maximum and unbounded parameters. Unsupervised learning was run again with Data Set 2, using the simple model as a start point, and it converged properly this time. This set of parameters is presented under the label "EM-Complex."

All the given models 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 $|\xi_1| \gg 1$ and $|\xi_2| \ll 1$. We used $\xi_1 = -1000$ and $\xi_2 = -0.001$
- "Closest Point" is a specific case of "Simple" with $|\xi_1| \ll 1$ and $|\xi_2| \gg 1$. 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.

Due to this observation, we implemented all the models using the same code, and we simply changed the set of features and the parameters [1].

These models were evaluated under a number of metrics.

- *The proportion of path misses*. For each trajectory, it is the number of times that 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, it is the number of times that 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 is a statistical measure that indicates the confidence assigned by the filter to its result. Small entropy (close to 0) indicates that one path is strongly favored by the filter against all the other paths, whereas large entropy indicates that all paths are equal.
- *The miscoverage of the route*. Given two paths $p$ and $p'$ the coverage of $p$ by $p'$, which is denoted as $\mathrm{cov}(p, p')$ is the amount of length of $p$ that is shared with $p'$ (it is a semidistance since it is not symmetric). It is thus lower than the total length $|p|$ of path $p$. We measure the dissimilarity of two paths by the *relative miscoverage* $\mathrm{mc}(p) = 1 - (\mathrm{cov}(p^*, p)/|p^*|)$. If a path is perfectly covered, its relative miscoverage will be zero.

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:

| Sampling rate (s) | 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 |

## B. 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 Figs. 13 and 14. 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 for the greedy model ("Hard Closest Point"). For sampling rates between 10 and 90 s, trained models

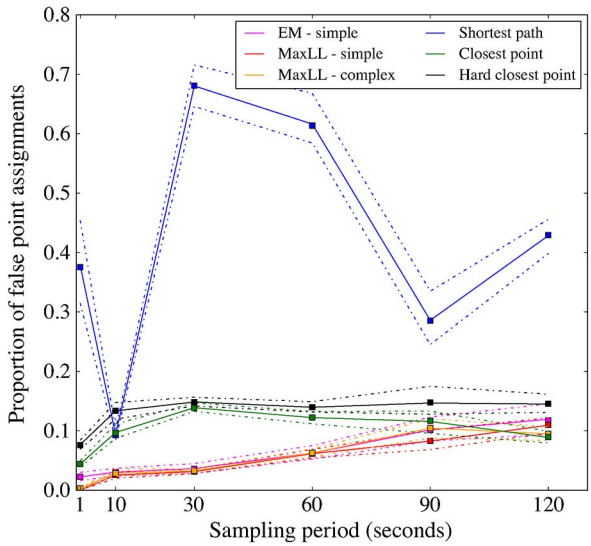


Fig. 13. Point misses using trajectory reconstruction (Viterbi algorithm) for different sampling rates as a percentage of incorrect point reconstruction for each trajectory (positive, smaller is better). The solid line denotes the median, the squares denote the mean, and the dashed lines denote the 94% 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 for the greedy model. For sampling rates between 10 and 90 s, tuned models show much higher performance compared with greedy models ("Hard Closest Point," "Closest Point," and "Shortest Path"). However, we will see that the errors made by the tuned models are more benign than errors made by the simple greedy models.

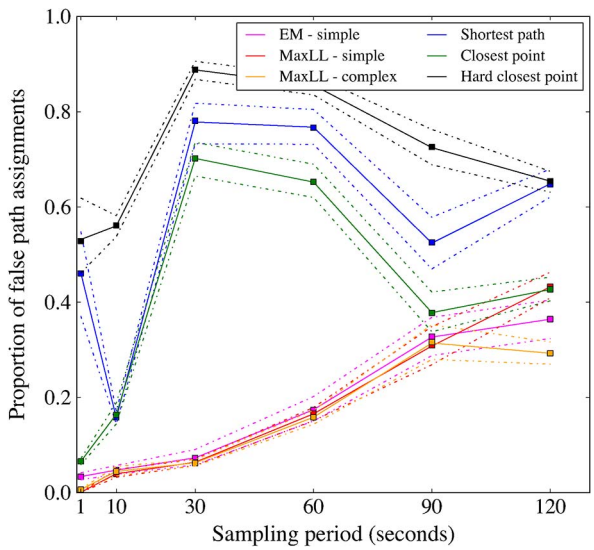


Fig. 14. Path misses using the Viterbi reconstruction for different models and different sampling rates, as a percentage on each trajectory (lower is better). The solid line denotes the median, the squares denote the mean, and the dashed lines denote the 98% percentiles. The error rate is close to 0 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-min 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 rapidly fails as the sampling increases. In addition, note how the shortest path heuristic performs poorly.

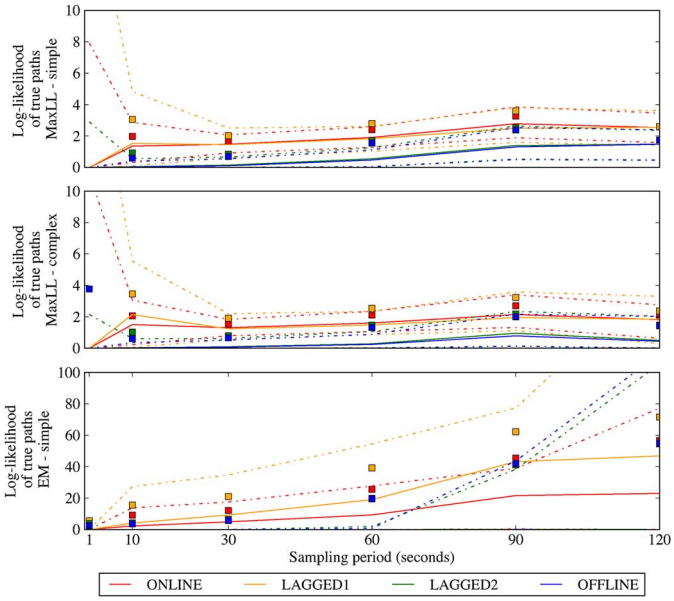


Fig. 15. (Negative of) Log likelihood of true paths for different strategies and different sampling rates (positive, lower is better). The error bars denote the first and last quartiles (the 25th and 75th percentiles). The solid line denotes the median, the squares denote the mean, and the dashed lines denote the 98% confidence interval. The likelihood decreases as the sampling interval increases, as expected. Note the relatively high mean likelihood compared with 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 (two-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-s 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 two-lagged filter. This difference disappears rapidly (it becomes insignificant at 10-s 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 two-lagged filtering (green) do not appear because they are too close to the one-lagged and offline strategies, respectively. Again, in the EM setting, the offline and two-lagged filters perform considerably better than the cruder strategies.

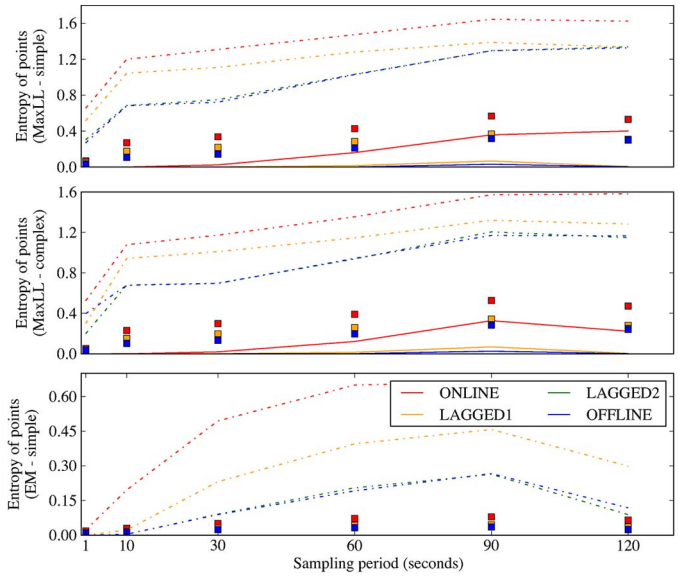


Fig. 16. Distributions of point entropy with respect to sampling and for different models. The colors show the performance of different filtering strategies (pure online, one lag, two lags, and offline). The entropy is a measure of the confidence of the filter on its output and quantifies the spread 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, whereas 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 entropy and the relatively wide confidence intervals. Note that using a more complex model improves the mean entropy by about 15%. In addition, 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 is not a good one.

("Simple" and "Complex") show much higher performance compared with 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 (see Fig. 15) and the entropy of the distribution of points or paths (see Figs. 16 and 17). 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 it stays relatively even across different time intervals.

We conclude the performance analysis by a discussion of the miscoverage (see Fig. 18). The miscoverage gives a good indication of how far the path chosen by the filter differs from the true path. Even if the paths are not exactly the same, some very similar path might be selected, which may differ by a turn around a block. Note that the metric is based on the length covered. However, at high frequency, the vehicle may be stopped and cover a length of 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/one-lag group (orange and red curves) and the two-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, particularly in the lower frequencies. The paths inferred by the filter are never dramatically different; at 2-min time intervals (for which the paths are 1.7 km 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 and all quartile metrics by more than 15%.

In the case of the complex model, the weights can provide some insight into the features involved in the decision-making process of the driver. In particular, for extended sampling rates

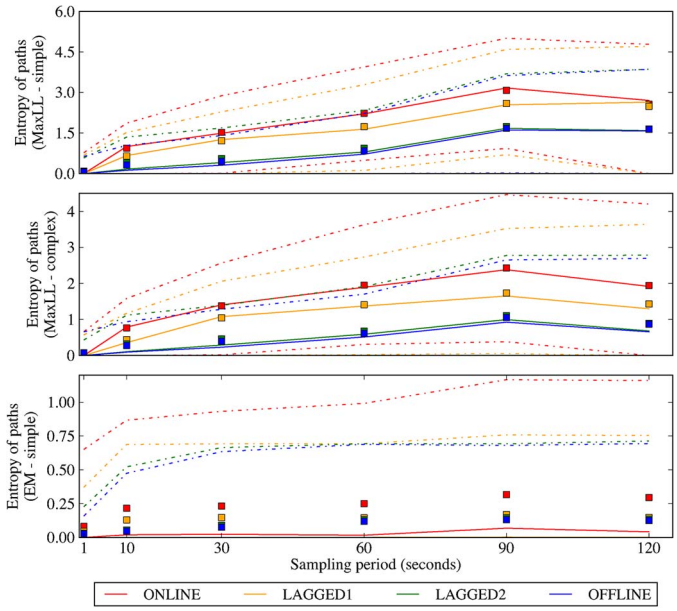


Fig. 17. Distributions of path entropy with respect to the sampling period and for different models (positive, lower is better). The colors show the performance of different filtering strategies (purely online, one lag, two lags, and offline). The entropy is a measure of the confidence of the filter 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 with the points, the path distributions have 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 two-lagged smoothing is as good as pure offline smoothing, for the same computing cost and a fraction of the data. Online and one-lagged smoothing perform about as well and definitely worse than two-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.

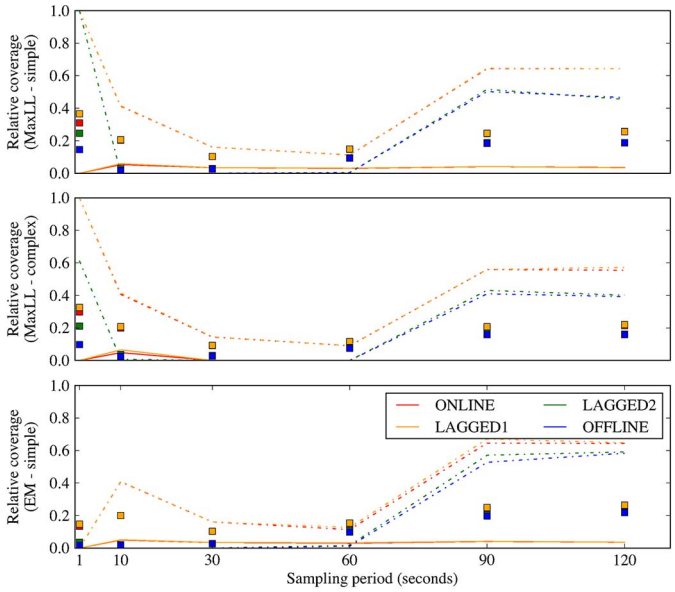


Fig. 18. Distribution of the relative miscoverage of the paths (between 0 and 1, lower is better). The solid line denotes the median, the squares denote the mean, and the dashed lines denote the 98% confidence interval. This metric evaluates how much of the true path the most likely path covers, with 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/one-lag group (orange and red curves) and the two-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, particularly in the lower frequencies. Note that the paths offered by the filter are never dramatically different; at 2-min time intervals (for which the paths are 1.7 km 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 and 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 zero length. This metric is thus less indicative at high frequency.

($t = 120$ s), 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, the drivers show a clear preference to turn on the right as opposed to the left, as shown in Fig. 19. 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, whereas two-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 farther apart. The errors steadily increase with the sampling rate until the 30-s 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 warming up. In particular, the tracking devices

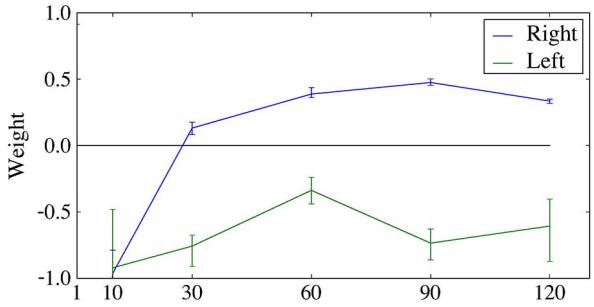


Fig. 19. Learned weights for left- or right-turn preferences. The error bars indicate the complete span of values computed for each time (0th and 100th percentile). For small time intervals, any turning is penalized, but rapidly, the model learns how to favor paths with right turns against paths with left turns. A positive weight even means that, with all other factors being equal, the driver would prefer turning on the right than going straight.

of fleet vehicle are usually designed to emit every minute as the road-level accuracy is not a concern in most cases.

### C. Unsupervised Learning Results

The filter was also trained for the simple and complex models using Data Set 2 (see Figs. 20 and 21). This data set does not include true observations but is two orders of magnitude larger

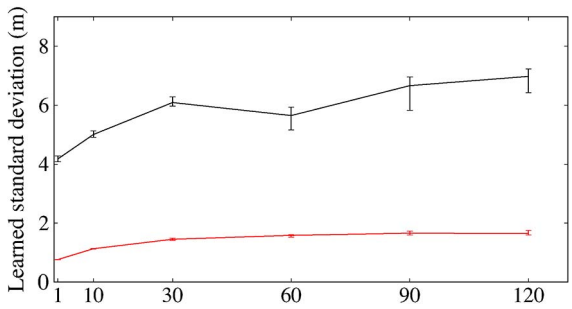


Fig. 20. Standard deviation learned by the simple models, in the supervised (maximum likelihood) setting and the EM setting. The error bars indicate the complete span of values computed for each time. Note that the maximum likelihood estimator rapidly converges toward a fixed value of about 6 m across any sampling time. The EM procedure also rapidly converges, but it is overconfident and assigns a lower standard deviation overall.

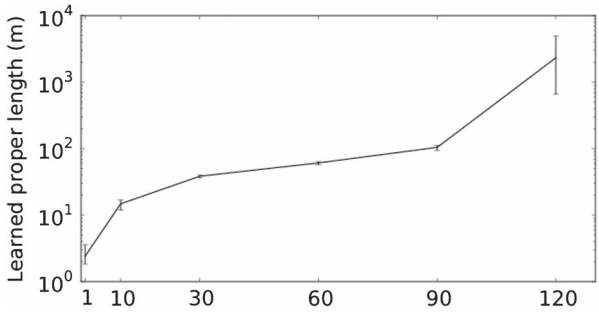


Fig. 21. Characteristic length learned by the simple models, in the supervised (maximum likelihood) setting and the EM setting. As hoped, it roughly corresponds to the expected path length. The error bars indicate the complete span of values computed for each time (0th and 100th percentile). Note how 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.

than Data Set 1 for the matching sampling period (1 min). We report some comparisons between the models previously trained with Data Set 1 ("MaxLL-Simple," "EM-Simple," and "MaxLL-Complex") and the same simple and complex models trained on Data Set 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 of all the features set to zero). The EM algorithm was then run for three 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 h and 400 000 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 shown in Figs. 22–24, a large training data set puts unsupervised methods on par with supervised methods as far as performance metrics are concerned. In addition, the inspection

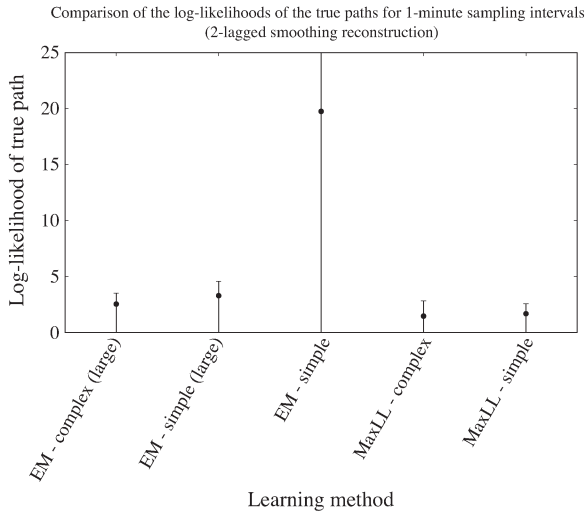


Fig. 22. Expected likelihood of the true path. The central point is the mean log likelihood, the error bars indicate the 70% confidence interval. Note that the simple trained unsupervised model with the small data set has a much larger error, i.e., it assigns low probabilities to the true path. Both unsupervised models tend to express the same behavior but are much more robust.

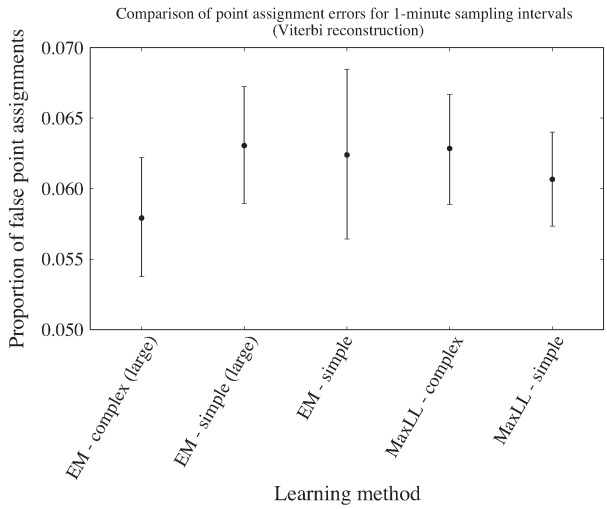


Fig. 23. Proportion of true points incorrectly identified, for different models evaluated with 1-min sampling (lower is better). The central point is the mean proportion, and the error bars indicate the 70% confidence interval. Unsupervised models are very competitive against supervised models, and the complex unsupervised model slightly outperforms all supervised models.

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

### D. 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 s.

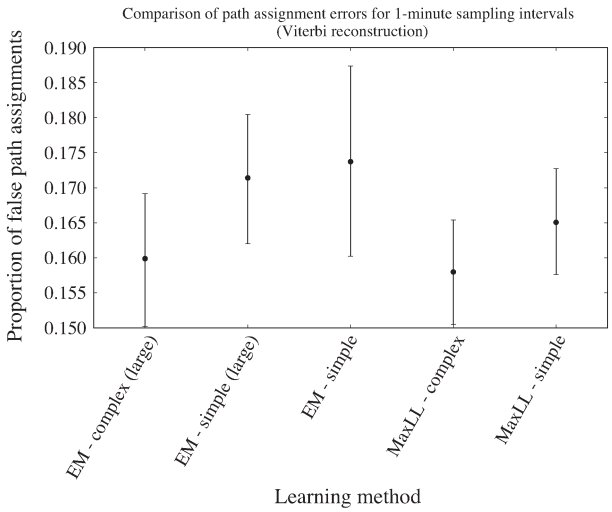


Fig. 24. Proportion of true paths incorrectly identified, for different models evaluated with 1-min sampling (lower is better). The central point is the mean proportion, and the error bars indicate the 70% confidence interval. The complex unsupervised model is as good as the best supervised model.

- A simple probabilistic heuristic ("Closest Point") gives good results for either very low frequencies (2 min) or very high frequencies (a few seconds) with more than 75% of paths and 94% of 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 (10–90 s), trained models (either supervised or unsupervised) have greatly improved accuracy compared with 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 with 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 one-lagged smoothing. The slower strategies (two-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-s sampling) for which smoothing clearly provides better results.
- Using a trained algorithm in a purely unsupervised fashion provides accuracy as good as when training in a supervised setting, i.e., within some limits and assuming that enough data are 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.

## VI. Conclusion and Future Work

We have presented a novel class of algorithms to track moving vehicles on a road network, i.e., the PIF. 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 CRF 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 s to 2 min) 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 is no need to collect expensive high-frequency data to train a model.

This algorithm supports a range of tradeoffs between accuracy, timeliness, and computing needs. In its most accurate settings, it extends the current state of the art [46], [48]. This result is supported by the theoretical foundations of CRFs. Because no standardized benchmark exists, the authors have released an open-source implementation of the filter to foster comparison with other methodologies using other data sets [1].

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, which is 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 a multicore architecture to achieve an average throughput of several thousand points per second [22]. 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 and 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 2 min). Although the filter already attempts to consider as few trajectories as possible, more aggressive pruning may be necessary 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 data sets that what we have presented here.

## Acknowledgment

The authors would like to thank the staff at the California Center for Innovative Transportation, particularly R. Herring and S. Apte, for their dedicated support; S. Samarayanake for the discussions that have been instrumental in writing this paper; and W. Hoburg and K. Kroetz for their thorough and insightful comments on the draft.

## References

[1] *Supporting Code for the Path Inference Filter*. [Online]. Available: https://github.com/tjhunter/Path-Inference-Filter/

[2] A. Bayen, J. Butler, and A. Patire, "Mobile Millennium final report," Univ. California, Berkeley, CA, USA, Tech. Rep. CCIT Research Report UCB-ITS-CWP-2011-6, 2011.
[3] J. L. Bentley and H. A. Maurer, "Efficient worst-case data structures for range searching," *Acta Informatica*, vol. 13, no. 2, pp. 155–168, Feb. 1980, 10.1007/BF00263991.
[4] M. Bierlaire, J. Chen, and J. Newman, "A probabilistic map matching method for smartphone {GPS} data," *Transp. Res. Part C, Emerging Technol.*, vol. 26, pp. 78–98, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0968090X12001064
[5] M. Bierlaire and E. Frejinger, "Route choice modeling with network-free data," *Transp. Res. Part C, Emerging Technol.*, vol. 16, no. 2, pp. 187–198, Apr. 2008.
[6] J. A. Bilmes, "A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models," *Int. Comput. Sci. Inst.*, vol. 4, pp. 1–13, 1998.
[7] S. P. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ Press, 2004.
[8] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk, "On map-matching vehicle tracking data," in *Proc. 31st Int. Conf. Very large Data Bases*, 2005, pp. 853–864, VLDB Endowment.
[9] *The Cabspotting Program*. [Online]. Available: http://cabspotting.org/
[10] Y. Cui and S. S. Ge, "Autonomous vehicle positioning with gps in urban canyon environments," *IEEE Trans. Robot. Autom.*, vol. 19, no. 1, pp. 15–25, Feb. 2003.
[11] A. Downs, *Still Stuck in Traffic: Coping With Peak-Hour Traffic Congestion*. Washington, DC, USA: Brookings Inst Pr, 2004.
[12] J. Du, J. Masters, and M. Barth, "Lane-level positioning for in-vehicle navigation and automated vehicle location (avl) systems," in *Proc. IEEE 7th Int. Conf. Intell. Transp. Syst.*, 2004, pp. 35–40.
[13] M. E. El Najjar and P. Bonnifait, "A road-matching method for precise vehicle localization using belief theory and kalman filtering," *Auton. Robots*, vol. 19, no. 2, pp. 173–191, Sep. 2005.
[14] G. D. Forney, Jr., "The Viterbi algorithm," *Proc. IEEE*, vol. 61, no. 3, pp. 268–278, Mar. 1973.
[15] L. Giovannini, "A novel map-matching procedure for low-sampling GPS data with applications to traffic flow analysis," M.S. thesis, Dept. Phys., Univ. Bologna, Bologna, Italy, 2011.
[16] J. S. Greenfeld, "Matching GPS observations to locations on a digital map," presented at the 81st Annual Meeting Transportation Research Board, Washington, D.C., USA, 2002.
[17] F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, and P. J. Nordlund, "Particle filters for positioning, navigation, and tracking," *IEEE Trans. Signal Process.*, vol. 50, no. 2, pp. 425–437, Feb. 2002.
[18] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. SSC-4, no. 2, pp. 100–107, Jul. 1968.
[19] A. Hofleitner, E. Come, L. Oukhellou, J. Lebacque, and A. Bayen, "Automatic inference of map attributes from mobile data," in *Proc. 15th Int. IEEE ITSC*, 2012, pp. 1687–1692.
[20] A. Hofleitner, R. Herring, P. Abbeel, and A. Bayen, "Learning the dynamics of arterial traffic from probe data using a dynamic bayesian network," *IEEE Trans. Intell. Transp. Syst.*, vol. 13, no. 4, pp. 1679–1693, Dec. 2012.
[21] B. Hoh, M. Gruteser, R. Herring, J. Ban, D. Work, J. Herrera, and A. Bayen, "Virtual trip lines for distributed privacy-preserving traffic monitoring," in *Proc. 6th Annu. Int. Conf. MobiSys, Appl. Serv.*, Breckenridge, CO, USA, Jun. 2008, pp. 15–28.
[22] T. Hunter, T. Moldovan, M. Zaharia, J. Ma, S. Merzgui, M. Franklin, and A. Bayen, "Scaling the mobile millennium system in the cloud," in *Proc. ACM Symp. Cloud Comput.*, 2011, pp. 28–35.
[23] Inrix Inc. [Online]. Available: http://www.inrix.com
[24] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Proc. 18th ICML*, San Francisco, CA, USA, 2001, pp. 282–289.
[25] K. Liu, T. Yamamoto, and T. Morikawa, "Study on the cost-effectiveness of a probe vehicle system at lower polling frequencies," *Int. J. ITS Res.*, vol. 6, no. 1, pp. 29–36, Jul. 2008.
[26] T. Miwa, T. Sakai, and T. Morikawa, "Route identification and travel time prediction using probe-car data," *Int. J.*, vol. 2, no. 1, pp. 21–28, 2004.
[27] T. K. Moon, "The expectation-maximization algorithm," *IEEE Signal Process. Mag.*, vol. 13, no. 6, pp. 47–60, Nov. 1996.
[28] K. P. Murphy, "Dynamic Bayesian Networks: Representation, Inference and Learning," M.S. thesis, Dept. Comput. Sci., Univ. California Berkeley, Berkeley, CA, USA, 1994.
[29] NAVTEQ Inc. [Online]. Available: http://www.navteq.com
[30] W. Y. Ochieng, M. Quddus, and R. B. Noland, "Map-matching in complex urban road networks," *Revista Brasileira de Cartografia*, vol. 55, no. 2, pp. 1–14, 2003.
[31] J. S. Pyo, D. H. Shin, and T. K. Sung, "Development of a map matching method using the multiple hypothesis technique," in *Proc. IEEE Intell. Transp. Syst.*, 2001, pp. 23–27.
[32] M. A. Quddus, W. Y. Ochieng, and R. B. Noland, "Current map-matching algorithms for transport applications: State-of-the art and future research directions," *Transp. Res. Part C, Emerging Technol.*, vol. 15, no. 5, pp. 312–328, Oct. 2007.
[33] M. A. Quddus, W. Y. Ochieng, L. Zhao, and R. B. Noland, "A general map matching algorithm for transport telematics applications," *GPS Solutions*, vol. 7, no. 3, pp. 157–167, Dec. 2003.
[34] S. Rogers, "Creating and evaluating highly accurate maps with probe vehicles," in *Proc. IEEE Intell. Transp. Syst.*, 2000, pp. 125–130.
[35] D. L. Schrank and T. J. Lomax, "2009 Urban mobility report," Texas A&M University, College Station, TX, USA, 2009.
[36] K. Seymore, A. McCallum, and R. Rosenfeld, "Learning hidden Markov model structure for information extraction," in *Proc. AAAI Workshop Mach. Learning Inf. Extraction*, 1999, pp. 37–42.
[37] S. Syed and M. E. Cannon, "Fuzzy logic-based map matching algorithm for vehicle navigation system in urban canyons," in *Proc. ION Nat. Tech. Meeting*, San Diego, CA, USA, 2004, vol. 1, pp. 982–993.
[38] Telenav Inc. [Online]. Available: http://www.telenav.com
[39] A. Thiagarajan, J. Biagioni, T. Gerlich, and J. Eriksson, "Cooperative transit tracking using smart-phones," in *Proc. 8th ACM Conf. Embedded Netw. Sensor Syst.*, 2010, pp. 85–98.
[40] A. Thiagarajan, L. S. Ravindranath, K. LaCurts, S. Toledo, J. Eriksson, S. Madden, and H. Balakrishnan, "Vtrack: Accurate, energy-aware traffic delay estimation using mobile phones," in *Proc. 7th ACM Conf. Embedded Netw.SenSys*, Berkeley, CA, USA, Nov. 2009, pp. 85–98.
[41] S. Thrun, "Probabilistic robotics," *Commun. ACM*, vol. 45, no. 3, pp. 52–57, 2002.
[42] N. R. Velaga, M. A. Quddus, A. L. Bristow, and Y. Zheng, "Map-aided integrity monitoring of a land vehicle navigation system," *IEEE Trans. Intell. Transp. Syst.*, vol. 13, no. 2, pp. 848–858, 2012.
[43] C. Wenk, R. Salas, and D. Pfoser, "Addressing the need for map-matching speed: Localizing global curve-matching algorithms," in *Proc. 18th Int. Conf. Sci. Statistical Database Manag.*, 2006, pp. 379–388, IEEE.
[44] C. E. White, D. Bernstein, and A. L. Kornhauser, "Some map matching algorithms for personal navigation assistants," *Transp. Res. Part C, Emerging Technol.*, vol. 8, no. 1-6, pp. 91–108, 2000.
[45] D. Work, O. Tossavainen, S. Blandin, A. Bayen, T. Iwuchukwu, and K. Tracton, "An ensemble Kalman filtering approach to highway traffic estimation using GPS enabled mobile devices," in *Proc. 47th IEEE Conf. Decision Control*, Cancun, Mexico, Dec. 2008, pp. 5062–5068.
[46] J. Yuan, Y. Zheng, C. Zhang, X. Xie, and G. Z. Sun, "An interactive-voting based map matching algorithm," in *Proc. 11th Int. Conf. MDM*, 2010, pp. 43–52, IEEE.
[47] T. Zhang, D. Yang, T. Li, K. Li, and X. Lian, "An improved virtual intersection model for vehicle navigation at intersections," *Transp. Res. Part C, Emerging Technol.*, vol. 19, no. 3, pp. 413–423, Jun. 2011.
[48] Y. Zheng and M. A. Quddus, "Weight-based shortest-path aided map-matching algorithm for low-frequency positioning data," presented at the 90th Annual MeetingTransportation Research Board, Washington, D.C., USA, 2011.

**Timothy Hunter** received the Engineering degree in applied mathematics from Ecole Polytechnique, Palaiseau, France, in 2007 and the M.S. degree in electrical engineering from Stanford University, Stanford, CA, USA, in 2009. He is currently working toward the Ph.D. degree with the Amplab and the Department of Electrical Engineering and Computer Science, College of Engineering, University of California, Berkeley, CA.

His research interests include new programming models for machine learning with big data and some applications to estimation and transportation.

**Pieter Abbeel** received a joint B.S. and M.S. degrees in electrical engineering from the University of Leuven (KU Leuven), Leuven, Belgium, and the Ph.D. degree in computer science from Stanford University, Stanford, CA, USA, in 2008.

Since 2008, he has been with the Department of Electrical Engineering and Computer Science, College of Engineering, University of California, Berkeley, CA. He has developed apprenticeship learning algorithms that have enabled advanced helicopter aerobatics, including maneuvers such as tictocs, chaos, and autorotation, which only exceptional human pilots can perform. His group has also enabled the first end-to-end completion of reliably picking up a crumpled laundry article and folding it. His work has been featured in many popular press outlets, including BBC, *MIT Technology Review*, *Discovery Channel*, *SmartPlanet*, and *Wired*.

Dr. Abbeel has received various awards, including Best Paper awards at the International Conference on Machine Learning and the IEEE International Conference on Robotics and Automation, the Sloan Fellowship, and the Okawa Foundation Award. He was listed in the 2011 TR35 of *MIT Technology Review*.

**Alexandre Bayen** received the Engineering Degree in applied mathematics from Ecole Polytechnique, Palaiseau, France, in 1998 and the M.S. and Ph.D. degrees in aeronautics and astronautics from Stanford University, Stanford, CA, USA, in 1999 and 2003, respectively.

From 2000 to 2003, he was a Visiting Researcher with NASA's Ames Research Center, Mountain View, CA, USA. He was the Research Director of the Autonomous Navigation Laboratory with the Ballistics and Aerodynamic Research Laboratory, French Ministry of Defence, Vernon, France, where he holds the rank of Major. He is currently an Associate Professor with the Department of Electrical Engineering and Computer Sciences, College of Engineering, University of California, Berkeley, CA, USA. He is the author of one book and over 100 articles in peer-reviewed journals and conferences.

Dr. Bayen received the Ballhaus Award from Stanford University in 2004, the CAREER Award from the National Science Foundation in 2009, and the Presidential Early Career Award for Scientists and Engineers from the White House in 2010. He is listed in NASA's Top 10 Innovators on Water Sustainability in 2010. His projects Mobile Century and Mobile Millennium received the Best of Intelligent Transportation Systems (ITS) Award for Best Innovative Practice at the ITS World Congress in 2008 and a TRANNY Award from the California Transportation Foundation in 2009. Mobile Millennium has been featured more than 100 times in the media, including TV channels and radio stations (e.g., CBS, NBC, ABC, CNET, NPR, KGO, the BBC), and in the popular press (e.g., *Wall Street Journal*, *Washington Post*, and *Los Angeles Times*).