Future Locations Prediction with Uncertain Data*

Disheng Qiu[1], Paolo Papotti[2], and Lorenzo Blanco[1]

[1] Università Roma Tre, Roma, Italy
{disheng,blanco}@dia.uniroma3.it
[2] Qatar Computing Research Institute (QCRI), Doha, Qatar
ppapotti@qf.org.qa

Abstract. The ability to predict future movements for moving objects enables better decisions in terms of time, cost, and impact on the environment. Unfortunately, future location prediction is a challenging task. Existing works exploit techniques to predict a trip destination, but they are effective only when location data are precise (e.g., GPS data) and movements are observed over long periods of time (e.g., weeks).
We introduce a data mining approach based on a Hidden Markov Model (HMM) that overcomes these limits and improves existing results in terms of precision of the prediction, for both the route (i.e., trajectory) and the final destination. The model is resistant to uncertain location data, as it works with data collected by using cell-towers to localize the users instead of GPS devices, and reaches good prediction results in shorter times (days instead of weeks in a representative real-world application). Finally, we introduce an enhanced version of the model that is orders of magnitude faster than the standard HMM implementation.

1 Introduction

The ability to predict future movements can enable novel applications in a wide range of scenarios. For example, in the context of Location-Based Services, it can be used to deliver advertising to customers approaching shops of interest.

Work has been done to predict future locations for moving objects, both in the short term [20] and in longer timeframes [11]. These works rely on two hypotheses: (i) moving objects follow some patterns in (most of) their movements, and (ii) such movements can be observed with a certain accuracy. The first argument relies on the intuition that most people drive among their points of interest along a usually small number of routes (e.g., from home to work, from work to the gym). The same observation holds for public transportation (such as buses or flights), and even animals in their migrations. The second argument is due to the increasing popularity of GPS devices in the last years. On one hand, such systems are becoming popular because of the large diffusion of smartphones equipped with GPS sensors. On the other hand, it is not reasonable to assume

* Supported in part by a Working Capital 2011 grant from Telecom Italia.

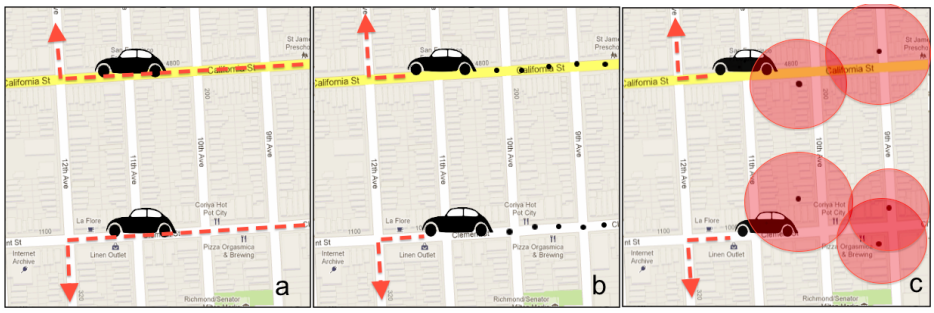


Fig. 1: Two cars running in different streets are going to turn in opposite directions. (a) shows their complete trajectories; (b) and (c) show how they are observed before they turn with GPS and cell-towers data, respectively.

the availability of these data on long periods. In fact, previous algorithms were designed with car navigation kits in mind, which do not have any constraints on power consumption. On the contrary, battery life is a primary issue for mobile users and GPS sensors are known for their high battery consumption. Existing technologies cannot be used for applications where movements of the users have to be continuously observed. For instance, consider the following application.

Carpooling is defined as a meeting of two or more people to share a car and travel together. Carpooling has a strong appeal for multiple reasons, but it experiences rather low levels of participation because of the complexity of social and work schedules, which make such arrangements hard to plan. Existing services (such as iCarpool and Avego) allow drivers and passengers to arrange occasional shared rides on short notice with smartphone applications. Such services facilitate one-time ride matches but users still have to define the routes and times of their trips, exactly as in traditional carpooling. This requirement is unacceptable in a context where users want more flexibility and short-term commitments.

This problem can be solved with methods to automatically predict the route and destination of the drivers, but existing solutions cannot be applied as they require the user to constantly keep the GPS sensor active. This is a tight requirement, as prediction techniques require to collect the routes of the users (to build the patterns repository) and to continuously verify their actual position.

Our approach tackles this problem by relying on location information inferred by the cell-towers the phone is connected to. Cell towers multilateration is the standard solution for location aware applications that need to run in background continuously: a standard network connectivity allows the smartphone OS to access a repository of GPS coordinates for the physical towers and to compute an estimated position for the user based on signal strength. Therefore network based localization comes at no additional cost and reduces battery consumption.

Consider Figure 1a, where two cars run on parallel streets before turning in opposite directions. Figure 1b shows how the cars are observed with GPS equipment, every dot is an *observation*. Given the current observations and an

history of previous trips of the user, a system can match the current trip with the repository to find the most probable destination for it. This is possible when there is a sufficient number of previous trips and one of them matches to some degree with the actual one. But what happens with noisy observations, such as in Figure 1c? Inferring location data with cell-towers is a difficult task: real-world data shows that the approximation in urban areas varies between 400 and 1600 meters. Predictor systems have problems in the matching step when locations are inferred with cell-towers, thus the prediction can fail.

To overcome these issues, we introduce a new prediction model, based on the Hidden Markov Model (HMM) [18], that naturally supports the uncertainty of the observations. HMMs have been used before in similar setting for *destination prediction* [19], and for handling *uncertain location data* [15], but our formulation tackles the two problems in a unified, principled solution. In particular, we designed a *future location prediction system* with the following contributions:

I We introduce a HMM for future locations prediction that naturally supports imprecise data and efficiently predicts both future locations (i.e., final destinations) and their corresponding trajectories (i.e., travel routes).
II A direct implementation of the HMM leads to unacceptable execution times in a real-world application. We therefore present a refined version with lossless optimization and a faster one with approximate solutions.
III We conduct an experimental study on the models and compare them with a state-of-the-art solution in simulated and real environments. For the synthetic scenarios, we introduce a tool to generate routes for moving objects.

In the following, Section 2 discusses related work and Section 3 formally introduces the problem. Sections 4 and 5 present our algorithm and its optimization, respectively. Finally, Section 6 validates the proposed approach with extensive experiments and Section 7 discusses future directions of research.

2 Related Work

Many studies have faced the problem of predicting future locations and routes for moving objects [16]. Considering the adopted techniques, they can be divided in two main trends. The first one is about the prediction of paths in an euclidean space [11, 20]: given the location and velocity of an object, the future location of the object is predicted with a function. The second trend is based on pattern matching: the algorithms compare the current location of the object with previously observed routes and return the best match as a prediction. Considering the domain of application, these can be further divided in three groups.

Prediction Based on GSM Network. Solutions for the prediction problem with wireless cellular communication networks differ from our approach as user history is not considered and they focus on optimizing network resources [7, 17, 21]. In [22], the authors build transaction rules based on neighbor cell towers with a support computed from the user history. The approach does not predict the final goal but only the most probable next cell tower.

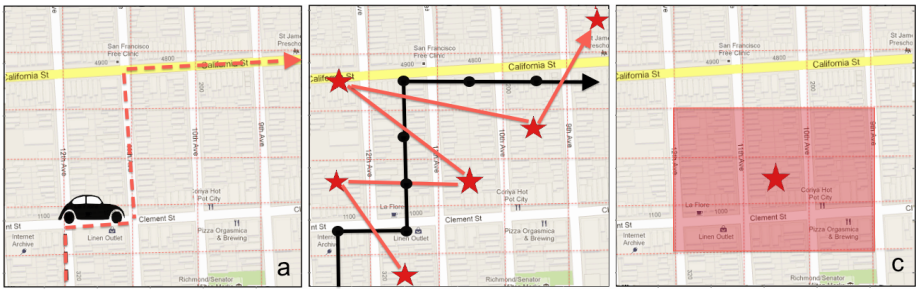


Fig. 2: A car running across an urban area: **a** shows its real trajectory; in **b** dots and stars show the trajectory with GPS and Network data, respectively; **c** shows one way to distribute the error rate for an observed block.

Prediction Based on Movement. Proposals that focus on predicting the destination and routes require precise locations [5, 9, 19], whereas our approach can handle uncertain observations. The notable exception is CRPM [6], as it can tolerate different kinds of disturbance in trajectory data. CRPM relies on a frequency analysis and adopts different heuristics to manage imprecise data.

Prediction Based on Location. Other methods focus on predicting the destination without considering the current partial route [1, 2, 12, 14]. They rely on different approaches, such as a prediction model leveraging the history of group of users [14], or a route recommendation system considering user interest and a database of geotagged photos [12].

HMMs have been successfully applied for many tasks related to sequence clustering [18], such as map matching with noisy observations [15], and to hidden intents recognition [1, 19]. We also differ from [4], where a traditional Markov model is used to classify different observations to an internal taxonomy. The work proposes a one-to-one mapping to couple state-symbol, but due to the absence of hidden states, Baum-Welch and Viterbi algorithms are not applied. Our proposal is the first attempt to addresses the problem of predicting the destination and the route to reach it with a HMM formulation over noisy observations.

3 Preliminaries

Road Networks. Our model does not require road networks given as input, as it infers the underlying roads from the observations. We introduce a simple urban trip to describe our setting. Consider the real trajectory of a car in Figure 2a. As in [23], we divide the area of interest in blocks and use the centers of such areas as observations points. Blocks can have variable size and their dimension depends on the granularity required by the actual application. Once blocks are fixed, we associate the reported location with the center of the block that contains the location. We model a road network as a directed graph $G = (V, E)$, where V is the set of vertices, E is the set of edges. A vertex $v \in V$ represents a block in the road network, an edge $(v, v') \in E$ represents blocks connected by a road.

Objects of Interest. We assume a set of moving objects $d \in D$ on the edges E of the road network G. At a given time t, each object d has its current spatial location $d.b$ (i.e., a block) and the predicted route $d.r^*$. A predicted route is a sequence of predicted locations $d.b_1^*, \ldots, d.b_n^*$, where $d.b_n^*$ is the final *goal* (or *destination*) of the trip.[3] The real route of an object d is denoted by $d.r$ and it is a sequence of locations $d.b_1, \ldots, d.b_n$. A route between two goals is called a *trip* or *trajectory*. The size (cardinality) of the set of objects D is denoted as $|D|$. The shortest path between locations $d.b$ and $d.b'$ in G is defined as $||d.b, d.b'||$.

Problem Statement. The problem faced in this work is the following: *given a time t, a moving object d with route $d.r$, and the subroute of d until t, set up a probabilistic prediction model s.t. the most probable predicted route corresponds to the route $d.r^*$ that minimizes the distance between $d.r^*$, $d.r$.*

The effective route is not available in general and the goal of our work is to minimize the distance between $d.r^*$ and $d.r$ with efficient algorithms. In other words, the goal is to predict the future locations of the moving objects and find the most similar route w.r.t. their real movements.

4 A Model for Destination and Route Prediction

The problem above can be modeled as a Markov chain reproducing the state of a system with a random variable for the final destination that changes through time. A state coincides with an observation and the distribution for this variable depends only on the distribution of the previous observations. Given a Markov chain and a sequence of observations, it is possible to predict the resulting state distribution [2]. Unfortunately, such approach fails in practice: (i) it predicts only the final destination and not the route; (ii) it is designed to work on precise GPS data, thus it suffers the problems we discussed above when used with noisy location information. The first problem can be faced with a model where l is a known location and every state represents a transition from location l to l' with a pair $(l'|l)$ [23]. By considering a n order Markov Model, this approach predicts all the possible routes for the user by considering, for every transition, all the n transitions that it has seen before. Unfortunately, Markov chains are not flexible w.r.t. the order n (which is fixed and has to be chosen at design time), so it is not easy to find a compromise between the right complexity and a good prediction.

Given the limits of Markov models, we build our solution on top of a Hidden Markov Model (HMM), for which the modeled system is assumed to be a Markov process with unobserved (hidden) states. In our solution, a state represents a pair (block, goal) $s_i = (b|g)$, and each state can be associated to multiple observations O_{s_i} with a given distribution of probability, i.e., O_{s_i} is the set of blocks observable from s_i. If we ignore hidden states observations, a moving object in a trip towards a given goal g covers the states $S = \{(b_i|g), (b_j|g), \ldots, (b_k|g)\}$. Given the sequence of states S, multiple observed sequences are possible. In fact, there is no one-to-one mapping from the state to the observation. Given a sequence

[3] A goal is a location visited by the observed object for long periods of time.

of states $S = \{s_1, \ldots, s_n\}$, the possible observation sequences are $b_1, b_2, \ldots, b_n$ with $b_i \in O_{s_i}$. In our setting $|O_{s_i}|$ grows if the measurement error of the block associated to the state is high and it decreases to 1 if measurement error is low (e.g., GPS observations). In Figure 2c, $|O_s| = 9$, because 9 blocks are covered by the measurement error of the observation.

Given an observation b, the possible related states are $s_i : b \in O_{s_i}$. Notice that this formulation models both noisy observations and user intents. In fact, given b_1 and b_2 close to observed b, possible states are $(b_1|g_1)$ or $(b_2|g_1)$, modeling uncertainty w.r.t. the current location, but also $(b_1|g_1)$ and $(b_1|g_2)$, modeling uncertainty w.r.t. the future location.

At prediction time, the HMM is used to calculate what is the most likely sequence of states associated to the sequence of blocks observed so far: the last state $(b_k|g)$ models that the location is b_k and the predicted goal g.

We build our Hidden Markov Model as follows:

- $A = \{a_{i,j}\}$ is the *state transition probability matrix*, where $a_{i,j}$ is the transition probability from s_i to s_j. If s_i=$(b_n|g)$ and s_j=$(b_m|g)$, $a_{i,j}$ is the probability to move from b_n to b_m with same goal g, when the object is in s_n.
- B=$\{s_j(o_k)\}$ is the *observation probability distribution matrix*. If s_j=$(b_n|g)$ and o_k=b_m, then $s_j(o_k)$ is the probability to see b_m, when in b_n with goal g.
- π is the *initial state distribution*.

HMM is a useful representation for our application. Consider again the example in Figure 2. Dots in Figure 2b show the trajectory with observed data as blocks (i.e., data observed with GPS), while stars show the trajectory with the blocks identified with the typical degraded data observed with systems based on cell-towers only. Data retrieved with cell-towers are approximate, but we can use a function to assign a certain probability to an observed block and spread the rest of the information over the 8 blocks around it as shown for the third observation in Figure 2c. As cell-towers localization services in smartphones give an estimation of the error of measurement, the number of blocks that have to be considered dynamically adapts to the quality of the observations.

Training. *Baum-Welch* algorithm [18] is used to train the model. Given a sequence of observations, it computes a maximum likelihood estimation, i.e., it produces A, B, and π such that the probability that the given sequence is observed is maximized. Unfortunately, it cannot be directly applied in our setting:

1. In our setting the system receives as input many sequences of observations (i.e., different trips), but the algorithm is designed to train a single sequence.
2. We want to train our model with a route to a destination. In the training, for each observation the corresponding state has two pieces of information: the current block b and the final destination g. The algorithm, given an observation for block b for which two states exist with distinct goals g and g', would train both states $(b|g)$ and $(b|g')$. This is motivated by the fact that no constraints are present in the HMM about the modeled goal.
3. The algorithm complexity is $O(\mathcal{N}^2\mathcal{T})$, where $\mathcal{N}$ is the number of states and $\mathcal{T}$ is the size of the observation sequence. Its performance is unacceptable in settings with a large number of states and long sequences of observations.

To overcome the above issues, we implemented an ad-hoc training. First, we split the states domain by considering a matrix for each destination. Instead of training a matrix A of size $\mathcal{N}=(\mathcal{GB})^2$, where $\mathcal{G}$ is the number of goals and $\mathcal{B}$ is the number of blocks, we separately train $\mathcal{G}$ matrices of size $\mathcal{B}^2$ (thus addressing issue 3). Second, we split the training set in groups of trips with the same destination; this allows us to train our model with multiple sequences (thus addressing issue 1) for which each goal is known and can be used to train the correct sequence of states (thus addressing issue 2).

From the algorithmic point of view, we initialize A with flat values, $a^0_{i,j} = avg = \frac{1}{\mathcal{N}}$. Given a sequence of trips $T = \{trip^1, \ldots, trip^{|T|}\}$, for each $trip^t$ we train a $\mathcal{B}^2$ matrix $S^t = \{s^t_{i,j}\}$ associated to the observed destination g^t and we interpolate S with A. In this way we constrain the training process to consider one trip at a time and with the interpolation process we train the original matrix A. In particular, each value $a^t_{i,j}$ is updated as follows:

$$a^t_{i,j} = \begin{cases} K \cdot s^t_{i,j} + (1-K) \cdot a^{t-1}_{i,j} & \text{, iff the goals of } s_i \text{ and } s_j \text{ are equal to } g^t \\ F \cdot avg + (1-F) \cdot a^{t-1}_{i,j} & \text{, otherwise} \end{cases}$$

Where K is a parameter called "learning factor", F is the "forgetting factor", $s^t_{i,j}$ is the trained value in S^t, and $0 \leq s^t_{i,j} \leq 1$, $0 \leq F \leq 1$, $0 \leq K \leq 1$. Blocks of states $s^t_{i,j}$ have to match the blocks of states $a^t_{i,j}$. By tuning K and F, we can find a trade-off between the importance given to old trips and new ones.[4] We train our model by repeating this task for every trip in T.

Predicting Destinations and Routes. Given a sequence of observations O, we use the *Viterbi algorithm* [18] to efficiently compute the most likely states sequence associated to the observations. The algorithm matches the current observations O (unaware of the final destination) with all the previous trips and finds the one that maximizes the probability of the given sequence of observations. By applying Viterbi we obtain the sequence of states $S = \{s_1, s_2, \ldots, s_n\}$ that maximizes the probability of observing O. S models the information about the final destination. In the actual implementation we consider only the last predicted state s_n and its goal.

Once we are able to recognize the current state of an object, we can find its destination and the route to reach it. We compute how the object reaches the predicted goal in s_n by using Dijkstra's algorithm in a connected graph where the nodes are the states of the model, and the costs on the edges are the probabilities from the matrix A. Our goal is to find the most likely path by maximizing the probability to get from the current state to the destination.

This approach can predict a route $d.r^*$ that is closer to $d.r$ than each of the previous observed trips. In fact, given a training set with a route observed multiple times, the noise in the observations is likely to be in different portions for each of the previous trips. But, when the shortest path is computed from A, the algorithm combines pieces matching the current route from multiple previous trips, thus automatically filtering noisy observations. Experiments show that this behavior improves the quality of the predicted path w.r.t. alternative approaches.

[4] We experimentally identified $F = 0.05$ and $K = 0.9$ as the optimal parameters.

Dijkstra algorithm is one of the several algorithms that can be used to compute the route to the predicted goal given a trained matrix A, while the prediction is done by Viterbi algorithm. The role of the HMM is to feed the weights to the graph used by the search algorithm to efficiently find a clean $d.r^*$.

5 Revisiting the Model to Improve Performance

State Pruning. For a fixed $\mathcal{T}$, all algorithms are $O(\mathcal{N}^2)$ with $\mathcal{N} = \mathcal{G} * \mathcal{B}$, where $\mathcal{B}$ is the number of blocks and $\mathcal{G}$ the number of goals. In particular, the cardinality of A is $(\mathcal{GB})^2$, thus the matrix grows by a factor $\mathcal{B}^2$ for each new goal. This easily leads to a very large number of states. A natural approach to the problem is to reduce the size of the matrices. In fact, reducing the matrix dimension has direct effects in response time in all stages. We reduce its size with two pruning techniques.

Lossless Pruning In the model, **all** possible pairs $(b|g)$ are considered as states, even if most of them are useless as many pairs are never observed in the user routes. For example, it is possible to remove all the blocks that are not crossed by an objects in its trips. In our experiments with a large urban area, this simple heuristic removes up to 85% of the blocks. Moreover, there are blocks crossed to reach multiple distinct goals, but there is a larger number of blocks crossed to always reach the same goal. In the original model, the set of all observed blocks L contains different noisy states such as follows: given f distinct goals and the user going to goal g_i through block b, state $(b|g_i)$ is generated together with $(b|g_1), \ldots, (b|g_{f-1})$. These $f-1$ states would be part of the model matrix, even if b has never been observed with the user moving to a goal different from g_i. Therefore $f-1$ states are pruned to reduce the size of the matrix.

Consider an incremental creation of the matrix A of dimension $\mathcal{N}$, by adding a new (i.e., never seen before) sequence $trip^{t+1}$ of size M to a trained model; the original model would generate a new matrix A of dimension:

$$(L+M)(\mathcal{G}+1) = \mathcal{N} + (M\mathcal{G} + M + L)$$

On the contrary, the optimized model generates a new matrix of dimension $\mathcal{N} + M$, because a new goal is added only for the blocks that lead to it and the already trained blocks are not involved.

This optimized model is already orders of magnitude faster than the original model in both the training and the prediction, and the results of the prediction are of same or better quality. In the following, we will refer to this model implementing the lossless pruning as the **standard HMM**.

Approximate Pruning With network observations the accuracy of the measures varies between 400 and 1600 meters. With such uncertain signals we spread the information over a number of blocks that depends on the observation quality. This approach leads to a big A matrix: from GPS to network data there is at least a five times increase in the number of observed blocks.

A possible solution is to manage the blocks added for error compensation in the B matrix, without reporting all blocks in A. In the standard model, when we

observe a block, **all** the blocks within its measurement error are "promoted" to states inside the HMM. Here instead, we do not create new states for each block reachable within the measurement error, but all the blocks added to distribute the error for an observation are stored in the emission matrix B.

We pushed this idea further, by doing the same pruning also for observed blocks. The basic intuition is the following: suppose that during a trip, observed through network data, we obtain in a sequence three blocks: b_1, b_2, b_3, with b_2 inside the error area of b_1. Given b_1, it is added to A, but b_2 is stored in the emission matrix B, while b_3 is promoted to A if it is out of the measurement error of b_1. In the approximate model only observed blocks out of the measurement error are promoted as states. In Section 6 we show how this aggressive pruning drastically reduces the execution times without reducing the prediction quality.

Notice that uncertainty is managed by the model and the pruning is applied only for efficiency purpose. The complexity of HMM algorithms is $O(\mathcal{N}^2\mathcal{T})$ and it is not affected by the dimension of matrix B. In fact, HMM is designed to support a large set of possible observations, and its growth does not increase the complexity of the system. Consider an observation sequence of 10 distinct blocks with an estimated error of 600 meters. If for each block we add the 8 blocks around it, adding this sequence adds $10 * 9$ new observations. By managing the 8 extra blocks as observations, we add only 10 new blocks and the remaining $10 * 8$ are managed without an overhead in the model. In the following, we will refer to this model implementing the approximate pruning as the **smart HMM**.

Split Training. Splitting if the training also brings a performance improvement. Instead of working with a matrix of size $\mathcal{N}^2 = (\mathcal{G} * \mathcal{B})^2$, we train $\mathcal{G}$ matrices of size $\mathcal{B}^2$ so that, given a trip to a goal, we train it only for the B matrix of its goal. Given $|T|$ sequences of observations, $\mathcal{G}$ goals, and $\mathcal{B}$ known symbols, the cost for a training of the original matrix is $|T| \cdot \mathcal{T}(\mathcal{G}\mathcal{B})^2$ (considering every observation of fixed length $\mathcal{T}$). The cost for the split training is instead $|T| \cdot \mathcal{T}(\mathcal{B})^2$ for the Baum-Welch execution and $|T| \cdot (\mathcal{G}\mathcal{B})^2$ for the interpolation step, with an important improvement w.r.t. the original training:

$$\frac{Original}{Split} = \frac{|T| \cdot \mathcal{T}\mathcal{G}^2\mathcal{B}^2}{|T| \cdot \mathcal{T}\mathcal{B}^2 + |T| \cdot \mathcal{G}^2\mathcal{B}^2} = \frac{\mathcal{T}\mathcal{G}^2}{\mathcal{T} + \mathcal{G}^2}$$

For 6 goals and $\mathcal{T} = 30$, the improvement is $\frac{1080}{66} = 16.36$.

Maximum Observation Length. Response time can be improved also by optimizing the observation sequence length $\mathcal{T}$. This applies in the Viterbi algorithm, as we do not need the complete sequence of states, but cannot be used with Baum-Welch, because in the model setup we need a training for the entire observation sequences. We experimentally observed that only significant changes to the size of the observations sequence affect the results. A short sequence is not sufficient to recognize a route, thus providing wrong predictions, while large sizes do not improve predictions quality while increasing execution times. In the experiments we set m=10 after verifying the prediction accuracy in real data.

6 Experiments

We implemented and tested alternative models to compare predictions quality and execution times. Algorithms have been implemented in Java and tested on an i5 2.4GHz CPU with 8GB of RAM.[5]

Metrics. We consider a training set of routes T, and an evaluation dataset E. Two kinds of dataset are used: *synthetic*, built with a route generator based on features obtained from real data, and *real-world*, made with a collection of real users movements in a large urban area. The real route $d.r$ is known for every object $d \in E$ and is used as ground truth for evaluation.

In the evaluation, we use as input a fragment i of the real route (e.g., the first 10% observations) to compute the predictions. For each object $d \in E$, we collect the predicted route $d.r^* = d.l_1^*, \ldots, d.l_n^*$, with the last location being the predicted goal. We test eight executions with an increasing percentage of input data from $d.r$ (from 10% to 80%).

We rely on two metrics for quality evaluation. In the first metric, we match the predicted goal $d.l_n^*$ with the effective goal in $d.r$. A match is valid if the predicted goal is in the area covered by the measurement error of the block containing the effective goal. We report the percentage of errors in the goal prediction in a set of executions as the Wrong Goal Percentage (**WG-P**). The second metric is based on our definition of distance and represents the quality of the predicted route. We define the distance between $d.r^*$ and $d.r$ by using the *Levenshtein* edit distance [13] with the following costs:
- Substitute: the cost to replace a wrong location $d.b_i^*$ with the correct one $d.b_i$ is computed as the distance between them: $||d.b_i^*, d.b_i||$.
- Add: the cost to add a location $d.b_i$ to $d.r^*$ is the distance between the location to be added and the last predicted location in the route: $||d.b_n^*, d.b_i||$.
- Delete: the cost to remove a location $d.b_j^*$ from $d.r^*$ is the distance between the location to be deleted and the closest location $d.b_j \in d.r$: $||d.b_j^*, d.b_j||$.

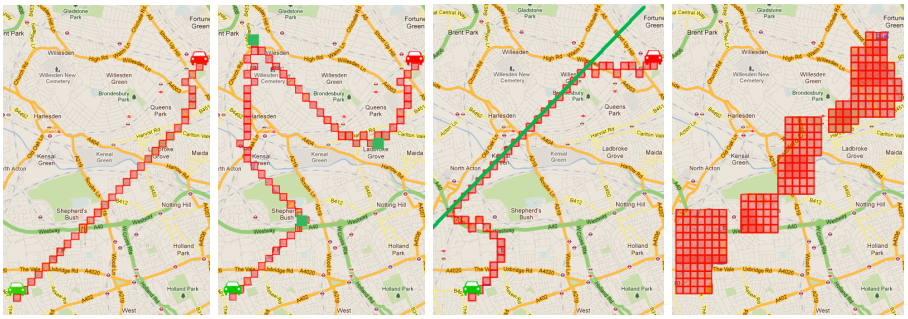
For settings with several goals, an incorrect goal prediction can be close to the correct one. We therefore distinguish two sets of routes: the ones with correct predicted goal are measured in the Correct Goal Route Distance (**CG-RD**); those with erroneous predictions are in the Wrong Goal Route Distance (**WG-RD**).

Synthetic Data. Despite there exist tools to generate routes [8, 10], we created our own generator to control parameters that are peculiar to our setting, such as the probability of having alternatives route between the same pair of goals. This cannot be defined with existing tools where the route is the shortest path between two points. The generator takes as input the following parameters.

Goals. Goals can be randomly distributed or manually set, to verify ad-hoc settings. Goals act as both starting and ending points for a route.

Roads. We initialize a weighted undirected graph, with a node for every block in the grid, and a weighted edge for every pair of adjacent blocks. The weight for

[5] The scenario generator, the implementation of the models, and a sample of the real-world datasets can be downloaded at http://www.placemancy.com/public/code.zip



(a) Route r **(b)** Alternative r' **(c)** r with roads **(d)** r with noise

Fig. 3: Synthetic routes generation.

an edge is in inverse proportion to the speed associated to the block transaction. Once goals are set, a route is the shortest path among them as in Figure 3a.

Alternative Routes. To model alternative paths to reach the same destination, different routes between a pair of goals are computed with two parameters: probability to have an alternative route and maximum number of alternatives.

Fast Roads. Users use fast roads when available, thus such roads are added to make the scenarios more realistic as in Figure 3c. This makes more challenging the prediction as users often drive the same fast roads to reach different goals. Weight over normal edges is three times the weight of edges for fast roads.

Noise. We add noise to the above routes to simulate real-world observations. We extracts two measurement error distributions (GPS and Network) from real data. An error probability distribution is the probability that a specified error rate is seen for a given block. Given a distribution, we associate to every block an error rate according to it, then we take a route as input and use the error rate associated to its blocks to "perturb" it. Given route r in Figure 3a, for each block $i \in r$ with error rate e_i, we remove from r the following $e_i - 1$ blocks. We then shift block i by a random number of blocks limited by e_i as in Figure 3d.

Results. We discuss results for the following models: standard HMM, smart HMM, and CRPM [23]. CRPM is one of the current state of the art solutions and it has been shown that it improves by 71% the goal prediction and by 30% the route distance w.r.t. a second order Markov model. We tested [19], but it failed in every test as it is not resistant to uncertain observations.

We consider 20 users and report the average results. We start with a *standard configuration* with a 80×80 grid, 5 goals distributed randomly, and 8 executions with increasing input. Once the parameters for the route generator are fixed, for the same scenario and for every user we create T (50 trips) and E (50 trips).

Table 1 shows a comparison of the standard configuration for Smart HMM and CRPM over three settings with different complexity.

- In *Base*, fast roads and alternatives are not enabled. This simple scenario can be used as a measure of comparison for the other settings. HMM has a lower

Table 1: Results on synthetic scenarios.

	Base		*Fast Roads*		*Alternatives*	
	Smart HMM	CRPM	Smart HMM	CRPM	Smart HMM	CRPM
WG-P	0.19	0.25	0.28	0.42	0.19	0.27
CG-RD	82	82	65	77	167	256
WG-RD	509	715	1131	1791	651	1064

error rate in the prediction of the goals (WG-P). This is reflected in the lower values for WG-RD, while there is no difference in the easier case (CG-RD).
- With *Fast Roads* enabled, the goal prediction becomes more challenging: errors in WG-P increases by 68% for CRPM, while for Smart HMM is limited to 47%. Consequentially, the gap between HMM and CRPM increases in all measures.
- In *Alternatives*, each route has 3 possible alternatives. WG-P values are close to the *Base* setting for both models, but the predicted routes differ more from $d.r$. The distances increase more for CRPM than Smart HMM.

We now discuss how results are affected by varying the parameters fixed in the standard configuration. Finally, we report execution times.

Grid Size The standard configuration represents a city in a 80x80 blocks grid, with a 500 meters side for each block. An increase in the number of blocks does not shows remarkable differences in the goal error rate. Figure 4(a) reports that the two models show a 10% increase of the WG-P measure with a 230x230 grid w.r.t. the 80x80 version, while for other measures the results are stable.

Number of Goals and Training Size Figure 4(b) shows results with increasing number of randomly located goals for models with different training sets. We denote with T_1 a set of 20 trips and with T_2 a set of 70 trips, and with CRPM_{T_x} (Smart HMM_{T_x}) the model trained on T_x. Smart HMM outperforms CRPM in all scenarios, and, as expected, increasing the number of goals makes the prediction more difficult: while CG-RD is of course stable, results degrade for WG-P and WG-RD. The number of goals is important, but also their location play a role. In fact, WG-P has a lower growth for both models starting from 7 goals because of their positions: when there are many of them, even a wrong goal can lead to a route which is relatively close to the target $d.r$. Interestingly, goals number can be reduced by using clustering techniques [2]. With a clustering radius of 1.5 miles they reduce the number of goals for all users to less then 10. Given the size of our blocks, we implicitly consider a clustering radius of 500m from raw GPS data. As expected, models improve with bigger training sets. For complex settings, longer training is necessary to obtain acceptable results. Similar trends are observed with other measures.

Increasing Input Figures 4(c-d) show how an increasing percentage of the effective route $d.r$ given as input linearly increases the performances of the algorithms. This is important: when the user starts a trip the system may predict a wrong goal, but it promptly changes prediction along the route. HMMs out-

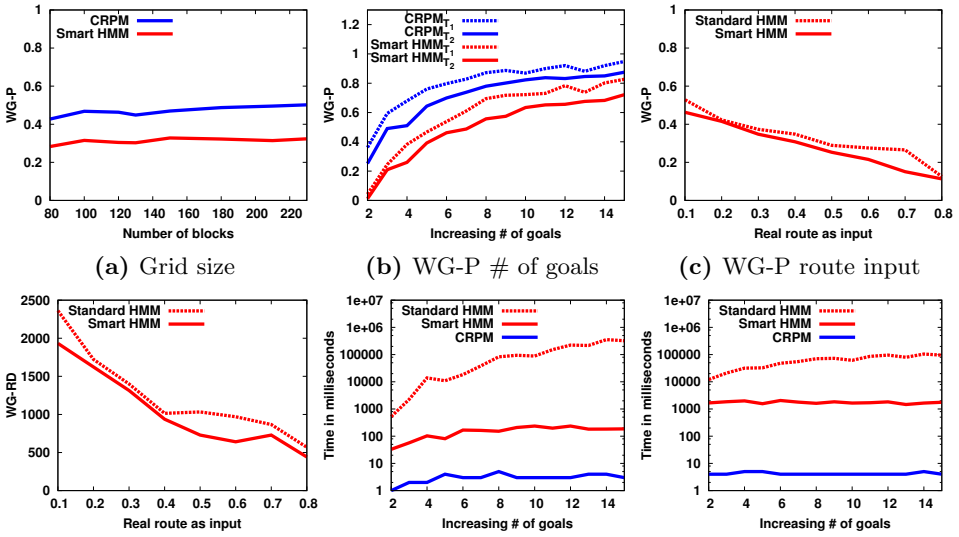
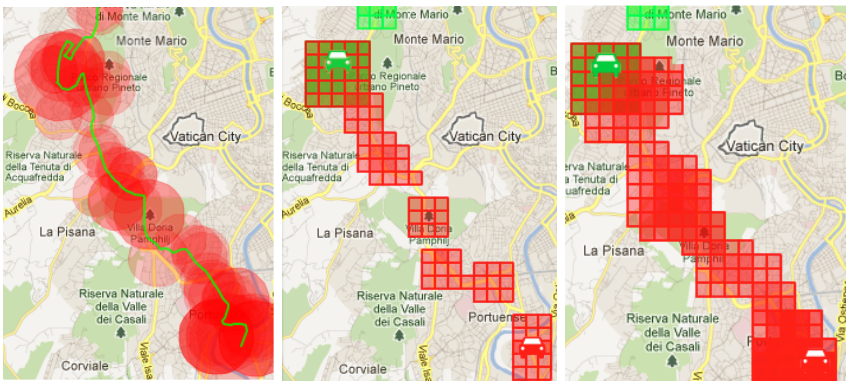


Fig. 4: Synthetic experiments: (a) grid size, (b) number of goals with different training sizes, (c-d) route input, (e) prediction time, (f) training time.

perform CRPM and similar trends are observed with CG-RD. The smart HMM obtains a slightly better error goal rate than the standard. Similarly, it has a lower average value in route distance. This indicates that, even in cases where it is not possible to predict the correct goal, the smart model can predict better routes and the pruning does not penalize the performances.

Execution Times The better prediction performance of the HMMs has a cost in term of computational time. Figures 4(e-f) show that CRPM is faster than HMMs in every test. However, the execution times for Smart HMM are acceptable and its execution times do not show a quadratic growth w.r.t. the number of blocks. Compared to the standard model in the worst setting (15 goals), the smart model reduces the prediction time from about 8 minutes to less than 0.2 second, and from about 1.5 minute to 2 seconds for the training.

Real-World Data. We collected real-world data from 10 volunteers with mobile applications for smartphones. Raw data trajectories were collected for 4 weeks by separately storing and cleaning [9] both GPS and network data. The datasets allow a comparison of the prediction models with precise and imprecise real data on the same routes; we report an example of the same trip observed by the two technologies in Figure 5a. Data has been pre-processed in order to identify goals and isolate the sequences of observations to be evaluated. For each learning system, we created one model with GPS data and one with network data. Once the four models were trained, we computed the routes to conduct a leave-one-out cross-validation. Examples of the predictions are in Figure 5(b-c).



(a) GPS and cell-tower **(b)** HMM cell-tower **(c)** CRPM cell-tower

Fig. 5: In (a) the green line is the trajectory with GPS data, while the red circles represent the cell-towers data with the measurement error for each observation. In (b-c) green blocks are the input observations, while red are predictions done with cell-towers data (fewer blocks indicate a more precise prediction).

Table 2: Experimental results with both GPS and cell-towers real data.

		HMM		CRPM	
		GPS	Network	GPS	Network
WG-P	Average all users	0.21	0.30	0.21	0.39
	Standard deviation	0.05	0.07	0.09	0.11
ALL-RD	Average all users	106.2	244.0	137.7	384.2
	Standard deviation	38	66	57	134

In Table 2 we report statistics about the observed distributions over the collected data. The standard deviation of the distributions is averaged over 10 users. GPS data are used as the ground truth. We denote with **ALL-RD** the route distance computed on all predictions. Training for Smart HMM required 259ms and a prediction required an average of 52ms. For goal prediction, Smart HMM outperforms CRPM on real network data by more than 23% (error goal rate of 0.30 compared to 0.39), while there is no significant difference on GPS data. For route distance on GPS data, Smart HMM returns routes closer to the correct one by 24% w.r.t to CRPM distance (distance of 106 compared to 137), while for the cell-tower data the difference in favor of HMM increases to 37% (244 compared to 384). In particular, Smart HMM outperforms CRPM for all our volunteers. The advantage in WG-P ranges from 15% to 27%, while for the ALL-RD it goes from 16% to 50%. HMM predictions are more stable than CRPM ones w.r.t. the standard deviation.

The differences in WG-P are due to a better modeling of the problem in the HMM. Even in cases when the WG-P values coincide, HMM obtains a better ALL-RD because the route is mined over all user trained paths (i.e., finding the shortest path inside A), while CRPM returns the single trained path that matches best. The execution times and the accuracy of predictions are in line with those obtained with the synthetic data generator. In particular, from the study of real data, an average user has less than 7 goals, and it is easy to verify that the numbers in Table 2 match those computed in Figure 4(b). Moreover, the number of goals for a user can be reduced by considering models with weekend days or weekdays only, or if we distinguish goals observed in the morning hours from the ones in the evening time. This leads to a solution with multiple HMMs for each user (e.g., one for the weekdays, one for the weekend, etc), such that the number of goals is always small.

Memory Consumption HMM is known to be space consuming. We use a trained HMM for every user, thus it can result in a large amount of parameters that have to be maintained. For a space complexity analysis, we trained the two HMMs over an increasing number of users with a month of cell-tower observations. For each user, a new prediction model is created. Then, each trained model is added to a global list, making sure that the models are not deleted. In this setting, a laptop (8GB RAM) can manage up to 50 trained models with the Standard HMM and up to 2000 models with the Smart HMM. This is a consequence of the different number of states in the models: the number of states in the smart implementation is almost an order of magnitude smaller than the standard model. In fact, the trained matrix has a $|S| \times |S|$ dimension, with $|S|$ the number of the states, and occupies about 95% of the allocated memory.

7 Conclusions

We presented novel prediction models for moving objects designed for imprecise data. We first modeled the problem with a HMM and then refined it to improve quality of the results and execution times. Experiments show that our model needs a shorter training than existing methods and predicts more precise routes.

Our work can be easily apply to a large audience. Existing location aware applications, such as Foursquare, already collect users' data and have APIs to access them. Using such systems brings advantages w.r.t. ad-hoc solutions: (1) a large user base; (2) optimized software to save battery power, also using network-based localization, thus highlighting the necessity of supporting imprecise data; (3) historical observations that enable our model to make predictions from the installation; (4) privacy issues are managed by the application provider. We plan to implemented our solution on top of such location aware applications.

Our model is effective when users travel over routes known to the system. This does not apply for new routes. A promising approach to overcome this limitation is to automatically extract appointments and events from social media (such as Facebook and shared calendars [3]) to enable the prediction of new locations.

References

1. Alvarez-Garcia, J.A., Ortega, J.A., Gonzalez-Abril, L., Velasco, F.: Trip destination prediction based on past gps log using a HHM. ESA 37(12), 8166–8171 (2010)
2. Ashbrook, D., Starner, T.: Using gps to learn significant locations and predict movement across multiple users. Personal Ubiquitous Comput. 7, 275–286 (2003)
3. Blanco, L., Crescenzi, V., Merialdo, P., Papotti, P.: Flint: Google-basing the web. In: EDBT. pp. 720–724 (2008)
4. Bonchi, F., Castillo, C., Donato, D., Gionis, A.: Taxonomy-driven lumping for sequence mining. Data Mining and Knowledge Discovery 19, 227–244 (2009)
5. Burbey, I., Martin, T.: When will you be at the office? predicting future locations and times. In: Mobile Computing, Applications, and Services, vol. 76 (2012)
6. Chen, L., Lv, M., Ye, Q., Chen, G., Woodward, J.: A personal route prediction system based on trajectory data mining. Inf. Sci. 181, 1264–1284 (April 2011)
7. Cheng, C., Jain, R., van den Berg, E.: Location prediction algorithms for mobile wireless systems. In: Wireless internet handbook, pp. 245–263 (2003)
8. Duntgen, C., Behr, T., Guting, R.H.: Berlinmod: a benchmark for moving object databases. The VLDB Journal 18(6), 1335–1368 (2009)
9. Froehlich, J., Krumm, J.: Route prediction from trip observations. In: Society of Automotive Engineers (SAE) 2008 World Congress (2008)
10. Giannotti, F., Mazzoni, A., Puntoni, S., Renso, C.: Synthetic generation of cellular network positioning data. In: GIS. pp. 12–20 (2005)
11. Jeung, H., Yiu, M.L., Zhou, X., Jensen, C.S.: Path prediction and predictive range querying in road network databases. The VLDB Journal 19, 585–602 (2010)
12. Kurashima, T., Iwata, T., Irie, G., Fujimura, K.: Travel route recommendation using geotags in photo sharing sites. In: CIKM. pp. 579–588 (2010)
13. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady 10(8), 707–710 (1966)
14. Monreale, A., Pinelli, F., Trasarti, R., Giannotti, F.: Wherenext: a location predictor on trajectory pattern mining. In: KDD. pp. 637–646 (2009)
15. Newson, P., Krumm, J.: Hidden markov map matching through noise and sparseness. In: GIS. pp. 336–343 (2009)
16. Nizetic, I., Fertalj, K.: Automation of the moving objects movement prediction process independent of the application area. ComSIS 7(4), 931–945 (2010)
17. Paramvir, T.L., Liu, T., Bahl, P., Chlamtac, I.: Mobility modeling, location tracking, and trajectory prediction in wireless atm networks. IEEE Journal on Selected Areas in Communications 16, 922–936 (1998)
18. Rabiner, L.R.: A tutorial on hidden Markov models and selected applications in speech recognition. Proceedings of the IEEE 77(2), 257–286 (1989)
19. Simmons, R., Browning, B., Zhang, Y., Sadekar, V.: Learning to predict driver route and destination intent. In: IEEE ITSC. pp. 127 –132 (2006)
20. Tao, Y., Faloutsos, C., Papadias, D., Liu, B.: Prediction and indexing of moving objects with unknown motion patterns. In: SIGMOD. pp. 611–622 (2004)
21. Xiao, Y.Y., Zhang, H., Wang, H.Y., Wang, F.: Location Prediction of Moving Objects with Uncertain Motion Patterns. DCDIS 14(S2), 503–507 (2007)
22. Yavas, G., Katsaros, D., Ulusoy, O., Manolopoulos, Y.: A data mining approach for location prediction in mobile environments. Data Knowl. Eng. 54(2), 121–146 (2005)
23. Ye, Q., Chen, L., Chen, G.: Predict personal continuous route. In: IEEE ITSC. pp. 587–592 (2008)