Predicting Future Locations for Moving Objects

Lorenzo Blanco, Paolo Papotti, Disheng Qiu

RT-DIA-186-2011 October 2011

Dipartimento di Informatica e Automazione
Università di Roma Tre
00146 Roma – Italy

http://www.dia.uniroma3.it
In the context of transportation optimization, the possibility to predict future movements for moving objects is a crucial innovation in order to make the best 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 such proposals are effective only when location data are precise (e.g., collected with GPS tracking systems), and movements are observed for long periods of time (e.g., weeks). In this work we introduce an approach based on a Hidden Markov Model which overcomes these limitations and improves existing results in terms of precision of the prediction, for both the final destination and the route (i.e., trajectory) to get there. This new 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 much shorter times (days instead of weeks in a representative application). Finally, in order to enable the use of the technology in an application for the Social Web, we introduce an enhanced version of the model with a speed-up in the execution times measurable in order of magnitudes w.r.t. the standard HMM implementation.

To show the applicability of the approach we show an application of the technology in the context of a carpooling service. Carpooling is an effective way to reduce travel costs, but the increasing complexity of work and social schedules have made it less desirable and have kept rather low the level of participation. In particular, drivers want flexibility in their schedule and the possibility to commit to others only when they are certain about their time and route. We show how our model enables a novel “real-time” carpooling, where drivers do not have to commit to potential passengers and do not have to manually define the start and end points for their trips.
## Contents

1 Introduction .................................................. 5  
  1.1 Problem Overview ......................................... 6  

2 Prediction Model ............................................. 8  
  2.1 Markov Chains for Prediction ............................... 8  
  2.2 Hidden Markov Model for Routes and Destination Prediction ................................. 9  
    2.2.1 Training ................................................ 10  
    2.2.2 Predicting the Destinations ......................... 10  
    2.2.3 Predicting Routes ..................................... 10  

3 Revisiting the Model to Improve Performances .......... 11  
  3.1 Reducing the Matrix Dimension ......................... 11  
  3.2 Algorithm Implementation ............................... 13  
  3.3 Split Training ............................................ 14  
  3.4 Maximum Observation Length ......................... 14  

4 Experiments .................................................. 15  
  4.1 Metrics .................................................. 15  
  4.2 Synthetic Data .......................................... 16  
    4.2.1 Route Generation .................................... 16  
    4.2.2 Cell-towers simulation .............................. 18  
    4.2.3 Model execution .................................... 19  
    4.2.4 Results .............................................. 20  
  4.3 Real World Data ......................................... 25  
  4.4 An example of application ............................... 26  

5 Related Work ................................................ 28  
  5.1 Indexing ................................................. 28  
  5.2 Goals and routes prediction ............................. 28  

## List of Figures

1 Two cars running in different street 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 ............. 6  
2 A car running across an urban area. (a) shows its real trajectory; (b) dots show the trajectory with precise data, while stars show the trajectory observed by using cell-towers data; (c) shows how we distribute probability for an observed block ................................................. 7  
3 Markov chain with three states, each state can be directly observed ......................... 8  
4 Hidden Markov Model with three states, each state can be associated to different observations with a given probability ......................... 9  
5 A large city with a 80 × 80 grid .................................................. 12  
6 Parameters used in the experiments generation and in the evaluation. The default values are presented in in bold. The first section represents general parameters, the second section parameters for the synthetic datasets generation, and the third section parameters of the HMM model ......................... 17
Transformation from a goal to goal route to more complex representations.  

Functions used to specify fast roads in the grid: two circles and a set of five lines. 

From the original route to one observed with cell-towers. 

Evaluation of different values for the HMM model parameters: (a-b) Middle Probability, (c-d) Interpolation factor, (e-f) Forgetting factors, (g-h) size of the Observation sequence. 

Models comparison considering different values for the size of the grid we use to divide the area of interest. 

Goal error rate of CRPM and HMM changing the error bias parameter; the different execution are made considering three different training sets with 225, 90 and 50 trajectories, respectively. 

Models comparison considering the presence or absence of roads. 

Models comparison considering the presence or absence of alternative routes. 

Models comparison considering the error goal rate and the route distance; the different setting are generated using 2 to 5 fixed goals. 

Models comparison considering the error goal rate and the route distance changing the number of fixed goals. 

Experimental results with the standard model and the smart one on a network simulated setting. 

Models comparison considering the error goal rate and the predicted route distance changing the input size. 

Execution of a set of random settings and model comparison on goal error rate and route distance; different lines for every model reflect the performance after 1 week/ 1 month/ 6 months of observations given as training. 

Models comparison on execution time. 

HMM comparison on execution time, with and without “smart target”. 

Prediction with real-world data by our HMM model. Route are represented by using cells, both with GPS tracker data (left hand side) and cell towers data (right side). Blocks in green represent the input observations, the ones in red the predicted routes and destinations. 

Prediction with real-world data by the CRPM model. Route are represented by using cells, both with GPS tracker data (left hand side) and cell towers data (right side). Blocks in green represent the input observations, the ones in red the predicted routes and destinations. 

Experimental results with both GPS and cell-towers data with our model and CRPM. 

A screenshot of the tool showing the predicted routes for two moving objects.
1 Introduction

The main contribution we present in this work is a novel future location prediction algorithm for moving objects that relies only on standard information provided by smartphones to localize the users, thus not requiring GPS sensor data. In particular, the prediction algorithm allows the drivers to not necessarily describe their routes in order to being able to share their car with potential passengers. The gain in power consumption is given by the exclusive use of GSM cell-towers localization information instead of the precise GPS coordinates for the users: every smartphone is connected to a cell-tower with a known location and our solution relies on this information in order to locate the user with an acceptable approximation.

There are many other applications for the technology we present in this work. In fact, in the wide context of transport optimization, the possibility to predict future movements is a crucial innovation in order to make the best decision in terms of time, cost, and impact on the environment. For instance, being able to predict the route taken by a car gives information about future changes in elevation and speed, such that hybrid control systems can optimize vehicles charge/discharge schedule [7].

In particular, we discuss and demonstrate algorithms that enable a new generation of real-time carpooling services. Carpooling (also called ridesharing) can be defined as a meeting of two or more people to share a car and travel together. It allows individuals to benefit from the convenience of the car, share travel costs, and alleviate the associated problems of congestion and negative environmental impacts. Carpooling has a strong appeal for multiple reasons, in the interest of both the society (from better use of transport infrastructure to improved energy efficiency and reduced carbon footprint) and the traveling passengers (private cars are often faster than public transport and have more capacity). Nevertheless, carpooling still continues to experience rather low levels of participation. In fact, the increasing complexity of social and work schedules, and the related increase in vehicle trip complexity, such as trip chaining, have made carpooling arrangements less desirable.

Traditionally, carpooling arrangements between two or more unrelated individuals for commuting purposes are long-term agreements limited in flexibility. In recent years, an answer for this problem arrived from innovative carpooling services relying on mobile terminals. The main feature of such services is the added flexibility to allow drivers and passengers to arrange occasional shared rides on short notice. Such feature is known as “real-time” carpooling, or “dynamic” ridesharing. A real-time carpooling system facilitates the ability of drivers and potential passengers to make one-time ride matches close to the time when the travel is desired, with enough convenience and flexibility to be used on a daily basis. Examples of real-time carpooling services can be found both in commercial companies (such as iCarpool, MyRideBuddy, Carticipate, Avego1), and in research initiatives (e.g., [5, 1]).

The advent of these novel carpooling services seemed the right innovation to quickly enlarge the number of participants, but their diffusion is still quite low for many reasons. One of them is the necessity for the participants of the service to manually define the start and end points for their trips, exactly as in traditional carpool. In fact, existing services require to know when the driver wants to carpool and the route that she will take. This requirement is reasonable and it is a natural effort for the users in traditional carpooling, but it is now unacceptable in a context where the users want more flexibility and prefer short-term commitments (i.e., they do not want to schedule precisely their movements in advance). Another requirement of current services is to keep the smartphones GPS application active in order to get the positions of the users. This is usually not acceptable for users, as GPS services are well known for consuming quite a lot of power, thus reducing the autonomy of the battery for the smartphone.

In this work we show how our model for the prediction of routes for moving objects can overcome the above issue. In particular:

- we show how our novel model for future location prediction based on hidden Markov model [10] efficiently predicts both future locations (i.e., final destinations) and the corresponding trajectories (i.e., travel routes) to get there; when multiple goal (or trajectories) are possible, the model returns them listed with their respective probabilities;
- we compare our prediction model with existing ones and show how is able to predict with both GPS data and cell-towers information; with respect to previous models, which are restricted to precise data, we show that the techniques in our tool enable the management of more realistic and complex scenario even with imprecise location data;
- we show that our tool can efficiently support a real-time carpooling application even for large scenarios by showing both a simulated environment (with configurable parameters such as the number of drivers and their trajectories) and a real-world use case with data collected from volunteers in a large urban area; we let the audience impersonate both the drivers and the passengers and actively interact with the scenarios.

In the following section we discuss the main challenges of the problem and how we faced them by describing our system and the technology behind it. We then explain a concrete application for our technology. Finally, we conclude the paper by discussing related works.

1.1 Problem Overview

Future location prediction is a challenging task. There are many works that exploit techniques to predict a trip route and destination [11, 4], but such proposals have been designed for applications where the trajectories of the moving objects were collected with precise measurements (i.e., GPS systems). On one hand, such systems are becoming more and more popular thanks to the large and increasing diffusion of smartphones equipped with GPS sensors. On the other hand, it is not reasonable to assume the availability of these data on long periods. In fact, previous algorithms were designed with GPS car navigation kits in mind, which do not have any constraints on power consumption. On the contrary, battery life has become a primary issue for mobile users and GPS sensor are well known for their high battery consumption. Our approach to this problem is to rely on location information inferred by the cell-towers the phone is connected to. In fact, a standard network connectivity allows the smartphone operating system (such as iOS and Android) to access a repository of GPS coordinates for the towers and compute an estimated position for the user.

Figure 1: Two cars running in different street 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
To see an example of the different data quality for the two technologies, consider the simple scenario in Figure 1.a, where two cars are observed while running on two parallel streets and then turning in opposite directions, the first North and the second South. Figure 1.b shows how the cars are observed with GPS equipment every few seconds. Consider every dot as an observation. In literature there are many ways to handle them: they can be associated to a street, to part of it (e.g., the portion between two cross-roads), or to something defined ad-hoc for the prediction system, such as a small portion of the city. The main intuition to predict the driver destination is quite simple: compute the current trajectory and match it with a repository of previous movements to find the most probable goal of the trip. This is always possible when there are a sufficient number of previous observations and one of them matches to some degree with the actual one, such as what is represented in Figure 1.b: Car 1 is driving westbound along Ward St and this is usually associated to turning right at 2nd Ave.

But what happen with noisy observations, such as those in Figure 1.c? Inferring location data with cell-towers is an uneasy task: the best a smartphone OS can do is combining different signals and their strength in order to estimate the location of the moving object. Real-world data shows that the approximation in urban areas vary between 100 and 1500 meters. This implies that the prediction problem becomes a more challenging task. Moreover, in 1.c the last observation for Car 2 is in a different street (3rd Ave) from its real location, but possibly 3rd Ave has not been seen in previous trips. It is easy to see that the predictor system has problems in the matching step when the locations are inferred by using cell-towers. In this case the prediction can easily fail, as the current and previous observations are likely to be different.

To overcome these issues, we introduce a new model that naturally supports the uncertainty of the observations. In fact, we designed prediction system with the following properties: (i) native support for data retrieved with localization systems based on cell-towers; (ii) better quality of the future location predictions than existing solutions; (iii) scalable solutions in order to enable services that are useful in practical setting (e.g., for social applications on the Web with large number of users).

Figure 2: A car running across an urban area. (a) shows its real trajectory; (b) dots show the trajectory with precise data, while stars show the trajectory observed by using cell-towers data; (c) shows how we distribute probability for an observed block.

Our model does not require road networks given as input, in fact it works directly on GPS coordinates. However, for the sake of presentability, we introduce another small example on top of a simple urban trip to describe of our approach. Consider the real trajectory of a car in Figure 2.a. In order to be independent from the road network, we divide the area of interest (e.g., a city) in blocks and use the centers of such areas as observations in the system. Notice that blocks can have fixed or variable size and their dimension depends on the granularity required by the actual application. Once blocks are fixed, we associate the GPS location estimated with the center of the block whose contains the location. Dots in Figure 2.b show the trajectory with the blocks containing the locations observed with precise data (e.g., data observed with a GPS tool), while stars show the trajectory with the block identified with the typical degraded data.
observed with systems based on cell-towers only.

2 Prediction Model

A simple way to model the prediction problem above is a Markov chain, which reproduces the state of a system with a random variable for the final destination that changes through time (e.g., [2]). A state coincides with an observation, 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.

For the sake of clarity, we first introduce Markov chains and their limits and then our model.

2.1 Markov Chains for Prediction

A Markov chain can be modeled by using a matrix $A = \{a_{i,j}\}$ that represents a state transition probability. If the model has $N$ states, $A$ will be a $N \times N$ matrix, and for every $a_{i,x}$ with $i, x \in \{1, 2, ..., N\}$ we have the probability to have a transition from $i$ to $x$ with $\sum_{x=1}^{N} a_{i,x} = 1$.

Figure 3: Markov chain with three states, each state can be directly observed

Models based on Markov chains (e.g., [2]) associates a state to a location that is frequently visited for by the observed user for long periods of time. It is common to refer to such locations as destinations. The probability distribution is then populated considering the frequency for the different locations in the history of user’s trips. For every destination $i$, $\text{tot}_i$ is the sum of all the trips occurrences from $i$, and for every reachable destination $j$ we compute the occurrence $n_{i,j}$ of the trips from $i$ to $j$, thus the probability of a trip from $i$ to $j$ is $\frac{n_{i,j}}{\text{tot}_i}$. With this naive approach it is possible to predict what is the next probable destination given the current one. This simple model can be extended by considering a $n$ order Markov chain, where for every destination the prediction of the destination depends on the $n$ states that come before. With this extension the estimation depends not only on the current location for the user but also on her previous $n-1$ locations.

Such approaches have two important limits: (i) they predict only the final destination and not the route, thus they not useful for applications such as real-time carpooling systems; (ii) they are design to work on precise GPS data, thus they suffer from the problems we discussed above when used with noisy location information. The first problem can be tackled with a simple extension to the model, as proposed in [9]. In this model every state represents a pair $(r'|r)$, where $r$ is a know location (ROI) and $(r'|r)$ represents a transition between two ROIs (from $r'$ to $r$). By considering an $n$ order Markov Model, this approach predicts all the possible route paths for the user. In fact, this is possible if for every transition it can consider all the $n$ transitions that it has seen before, but, as observed by [9], 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.
2.2 Hidden Markov Model for Routes and Destination Prediction

Given the limits of Markov models, we built our solution on top of a Hidden Markov Model (HMM) [10], which overcomes their limits and achieve a good prediction rate. A HMM is a Markov model in which the system being modeled is assumed to be a Markov process with unobserved (hidden) states. In our solution, a state is associated to a pair (block, destination) \((b,d)\) and each state can be associated to multiple observations with a given distribution of probability.

An Hidden Markov Model is modeled by three matrices \(A, B, \pi\), where \(A\) is the state transition probability (as seen for Markov Chains), \(B\) is the observation probability distribution, and \(\pi\) is the initial state distribution.

\[
\begin{align*}
A & \text{ is the state transition probability matrix,} \\
B & \text{ is the observation probability distribution,} \\
\pi & \text{ is the initial state distribution.}
\end{align*}
\]

The initial state distribution can be modeled as a vector \(\pi = \{\pi_i\}\) where \(\pi_i = p[q_1 = S_i]\) with \(1 <= i <= N\). So for every state \(i\) it shows the probability that the first state is \(S_i\).

In the matrix \(B\), for every state \(j\) there is a value \(b_j(k)\), where \(b_j(k) = P(v_k \text{ at time } t|q_t = S_j)\) for \(1 <= i <= N\) and \(1 <= j <= m\). The value \(b_j(k)\) denotes the probability that an observation \(v_k\) is associated to the state \(S_j\).

In our model a state is represented by a couple \((b,d)\), so if \(B\) is the number of blocks, and \(G\) is the number of destinations, in a naive implementation the cardinality of \(A\) is \((GB)^2\) (we will discuss later how we optimize such cardinality to boost performance). It is easy to see that, given an observation, we can not understand the state of our model because every block can be associated to multiple destinations. For example, even if we consider a correct observation (i.e., such one obtained with a precise GPS tracker), we can observe that an user is in a block \(b_1\), but this observation can be associated to multiple states \((b_1|d_1), (b_1|d_2), ..., (b_1|d_G)\).

This is a perfect representation for our settings. Consider again the simple example in Figure 2.c: we know that the data retrieved with cell-towers for the third observation are approximate, but we can use a function to assign a certain probability to the observed block (say, 0.5) and spread the rest of the information over the 8 blocks around it.\(^2\)

Once the HMM has been defined, we train it with the set of observed trips by using the \textit{Baum-Welch algorithm} [10]. Then, given a sequence of observations, we use the \textit{Viterbi algorithm} [10] to obtain the current state (i.e., current location and possible final destinations with their probabilities). Finally, we rely on Dijkstra’s algorithm to compute the routes to the possible destinations.

\(^2\)Notice that cell-towers localization services in smartphones give also an estimation of the error, so the number of blocks that have to be considered dynamically adapt to the quality of the observations.
2.2.1 Training

Baum-Welch is a training algorithm that, given a sequence of observations, estimates the maximum likelihood estimation, that is, it maximizes the probability that the sequence is seen, modeling $A$, $B$, and $\pi$. In order to use such algorithm in our model, we had to adapt it to our settings for the following reasons:

1. The algorithm is designed to train a sequence of observations and not multiple sequences, but in our setting the system does not receive as input a complete sequence of user movement but only fragments of it (i.e., the different trips).

2. We want to train our model by using the paths that the user made to reach a destination. This implies that in the training, for each observation the corresponding state has two information: the current location $b$ and the final destination $g$. The original algorithm, given two observations for the same location $b$, but with two different destinations $g$ and $g'$, does not consider the latter information, and it would train both states $(b|g)$ and $(b|g')$ as a step for the both destinations.

3. The complexity for the Baum-Welch algorithm is $O(N^2T)$ where $N$ is the number of states and $T$ is the size of the observation sequence to train. Its performance can be unacceptable in our settings when we deal with long observations.

To overcome the above issues, we implemented an ad-hoc training for our setting. In particular, we split the states domain by considering a matrix for each destination. In this way, from our initial matrix of $(GB)^2$ we obtain $G$ split matrices of $B^2$ dimension (thus addressing issue 3). Similarly, we can also split the training set of our observations according to every single trip in relation to its known destination (thus addressing issues 1 and 2).

From the algorithmic point of view, we start the training with a flat $(GB)^2$ matrix, then, for every trip we define a $B^2$ matrix associated to the observed destination, and we interpolate this new matrix trained for this goal to our original matrix. By repeating this approach for every trip we train our model.

2.2.2 Predicting the Destinations

The Viterbi algorithm is used to resolve efficiently the following problem: given a sequence of observations, we want to find out the most likely states sequence associated to the observations. In our setting, the algorithm matches the current observations (unaware of the final destination) with all previous trips and finds the one that matches and maximizes the probability of that sequence of observations. From this result, we are able to recognize where the user is going, where she comes from, and where she is at this moment. From this information we can finally find the path to the destination by using the Dijkstra algorithm.

2.2.3 Predicting Routes

The last part of our approach will find out how the user will reach her goal, by using Dijkstra in a connected graph where the nodes are the states for our model, and the costs on the edges are the probabilities from the matrix $A$. Our goal is to find the most likely path, so we want to maximize the probability to get from the current state to the destination one value. Notice that we do not have problems with cycles as there are no negative value for the costs.

Finally, in order to be able to work with cell towers, where the precision can differ from every observation, we include the observation error in our $B$ matrix, that can be varied for every training and every prediction. With respect to previous solutions, our model is therefore
resistant to errors in the observations and lead to better predictions with cell-towers location data. Moreover, we optimized the model in order to achieve execution times comparable to those of previous solutions. In fact, the complexity for the two main tasks of the model (training and prediction) is quadratic w.r.t. the number of states. Therefore one way to improve the performance is to aggressively reduce the number of states of the model, without compromising the quality of the prediction. To achieve this result we reduce the size of the matrix for every user as much as possible, for instance by putting in the matrix only the blocks he has visited in previous trips. Another approach we tackled to improve efficiency is to adapt the Baum-Welch and Viterbi algorithms to our context.

3 Revisiting the Model to Improve Performances

Our prediction model is based on a HMM where every state is a couple $l|g$ ($\text{link} \rightarrow \text{goal}$), all the three prediction processes use known algorithms, and their complexities are $N^2 \times T$ for the Baum-Welch and Viterbi algorithms, and $N^2$ for the Dijkstra search. In order to make these algorithm scalable up to large amount of data, we optimized the algorithmic solutions, also to have a HMM prediction model comparable in execution times with existing models in literature. In fact, the challenge here is to modify algorithms that are known to be time consuming in order to enable real-time applications in real-world settings.

In order to optimize the performance of our model, we focused on improvements in all the three stages of the prediction process. As Baum-Welch algorithm is used to train the model, an improvement in this stage is needed to get a faster setup of the user model. Viterbi and Dijkstra algorithms are used to predict users’ future routes, thus improvements of the response time in these two stages is needed to allow acceptable user experience with high workloads.

Our optimization techniques can be clustered into four general approaches as follows.

3.1 Reducing the Matrix Dimension

A natural approach to the problem is to reduce the size of the matrices involved in the process. In fact, the optimization of the matrix dimension has direct effects in response time in all the three stages. The most efficient way to reduce the response time is therefore to have a transition matrix as small as possible. If we focus on the $N$ parameter, we can have a quadratic improvement (in fact, fixed $T$ all algorithms are $N^2$). We remind that in the first version of the model we introduced $N = L \times G$, where $L$ is the number of cells and $G$ the number of goals; thus the matrix grows of a factor $N$ for every new goal. This easily leads to a very large number of states. To overcome this issue, we reduce the user matrix with four different approaches:

1. We consider in $L$ only cells that have been seen in the user’s routes. This first optimization reduces drastically the value of $L$. In fact, we divided our area of interest in a grid $80 \times 80$ for a total of 6400 cells, as shown in Figure 5. In our real-world experiments an average user covers 800 cells in a month. Given this number, we have a $(6400/800)^2$ improvement (i.e., $(\#\text{TotalCells}/\#\text{ObservedCells})^2$), a 98.5% reduction for the number of cells.

2. We can use information about the time of observation to further reduce the number of goals. In fact, we observed in our experiments that an average user has a number of goals in order of tens after a month of observation. The quite large set of goals can be found by looking to the entire user history, but it is easy to see that a user has a much smaller number of goals if we consider the weekend days or the week days only, or if we distinguish goals observed in the morning hours from the ones in the evening time. This consideration reduces the goals dimension (thus improving execution times) and improves the prediction,
as a user going outside from his house during the weekend probably will not go to the same destination that he would go in a normal week day.

3. In the naive implementation, we considered as states of our model all the possible pairs \( l|g \). We observe most states in this matrix are useless, since many pairs \( l|g_i \) are never observed in the user routes. In particular, there can be cells that are crossed by users when going to multiple distinct goals, but there is also a larger number of cells that are crossed to reach a unique destination. If the user cross the cell \( l_x \) to reach \( g_y \) only, why should we keep in the model also the states modeling the same cell with all the other possible goals reached by the user? In other words, in the first model we were considering in the set \( L \) all the cells generated as follows: given \( f \) distinct goals and observed the user going exclusively to goal \( g_1 \) with a cell \( l \), states \( l|g_1, \ldots, l|g_f \) would be generated. These \( f - 1 \) states would be part of the model matrix, even if \( l \) has never been observed with the user moving to a goal different from \( g_1 \). Therefore these \( f - 1 \) states are not useful, because they have never been observed in the user’s routes, and they only make the matrix bigger.

If we consider an incremental creation of the matrix, we can see that by adding a new (i.e., never seen before) sequence of size \( M \) to a trained model, the model would generate a new matrix of dimension:

\[
(L + M) \times (G + 1) = L \times G + M \times G + M + L = N + (M \times G + M + L)
\]

On the contrary, the optimized model would generate a new matrix of dimension \( N + M \), because a new goal is added only for the cells that lead to it and the already trained cells are not touched.

In our experiments the response time for the optimized model is about ten times smaller then the original implementation in both training and prediction, and the results of the prediction is same or better.

4. We discussed how an average user covers a small subset of the cells for the city of reference (e.g., 800 for one month of movement). This consideration has been made observing GPS traces, where the accuracy is below 100 meters during most of the observations.
we switch to network observations, the accuracy reported by the smartphone operating system varies between 100 and 1500 meters. It is clear that with such uncertain signals it is not possible to identify only one cell in our representation. As discussed in Section 2, we need to spread the information given by such observation over a number of cells depending on the quality of the signal. This approach easily leads to bigger matrix with more states involved. For example, the same user covering 800 cells with GPS data ends up covering about 20% of the entire city when observed with cell towers data with the compensation of the error estimation. This behavior can make the matrix too big and thus unusable for our applications.

A solution that we implemented is to manage the cells coming from the error compensation as observations for a state (instead that state themselves) in the $B$ matrix. The intuition here is that in the prediction phase it is enough to find a cell in the observation matrix in order to recognize it, so we do not need a state for each cell from the error compensation in order to use it in the prediction. We remind here that the complexity of HMM algorithm is $N^2T$ and does not concern the dimension of the observation set $M$. In fact, HMM is designed to support a big set of possible observations, and the grown on the observation set does not increase the complexity of the system. Consider an observation sequence of 10 distinct cells with an estimated error of 600 meters, adding this sequence adds $10 \times 9$ new observations: for each cell we add the 8 cells around it because of the uncertainty. If we manage the 8 extra cells as observations, we will add only 10 new cells and the remaining $10 \times 8$ are managed without a overhead in the model.

**3.2 Algorithm Implementation**

We are working with known algorithms and is not possible in general to overcome the issue of the algorithm complexity ($N^2T$). But we can propose some optimizations that can be applied for our setting and for our implementation.

**Memoization and choice of the data structures.** As described by the Baum-Welch and Viterbi original formulation, in most cases we need to reuse estimation computed in a previous step. Consider the forward variable induction step:

$$\alpha_{t+1}(j) = \sum_{i=1}^{N} \alpha_t(i)a_{i,j}b_j(O_{t+1})$$

When we have to compute $\alpha_{t+1}(j)$ we have to recursively compute for every $i$, $\alpha_t(i)$. If we have to compute $\alpha_{t+1}(z)$ is still necessary the results of all $\alpha_t(i)$ computed previously. From this consideration is easy to find that for a good implementation is necessary a pre-computation and a caching of this values. For the caching of values for the computation of $\alpha$ or $\beta$ (same discussion as $\alpha$) a linear ($N \times T$) space is needed, which has not been a problem in our setting.

Consider now the $\xi_t(i,j)$ variable used during the training with the Baum-Welch algorithm:

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}B_j(O_{t+1})\beta_{t+1}(j)}{P(O|\lambda)}$$

The $\xi$ variable is used to compute the re-estimated transition matrix. For example, if we compute the state transition probability $\alpha_{1,2}$, we need to compute all $\xi_t(1,j)$ for every $t$ and every state $j$; so, when we compute $\alpha_{1,3}$, we can reuse the information previously computed if we cache $\xi$ as we cached $\alpha$. In this case, variable $\xi$ needs $N \times N \times T$ space, and
the quadratic complexity can become a problem if we work with a big matrix.\footnote{We remark that this problem become serious working with multiple matrices in a concurrent implementation of the system in order to take advantage of modern multicore processors.} There is no general solution for this problem. In our implementation we used data structures having less space overheads as possible. In particular, we use multi-dimension arrays instead of maps.

**Pruning of the possible transitions.** The Viterbi and Baum-Welch algorithm have been designed considering generic settings, where every \( a_{i,j} \) and \( b_j(k) \) is greater than 0. On the contrary, in our settings we get very often \( b_j(k) = 0 \). Consider for instance the map of a city and all the possible 80 × 80 cells; if we consider a state a couple \( cell|goal \), the possible observations from this state are a small subset of all possible cells.

Consider in fact the Baum-Welch algorithm and the expected number of transitions between two states. The expected number of transitions is computed given a sequence of observations. In our setting, given as input in the training a single path, only few states will be considered, and all the other states transitions that are not seen in the input path will be set to 0. Given an observation and the observation matrix, we reduce \( N \) by keeping in the matrix only states with \( b_j(k) \neq 0 \), where \( k \) is a observed state.

For example, consider a single goal matrix trained with 4 different paths with fixed size \( T \) and independent one from the other. If we suppose that the path sequences contain no duplicates, the single goal matrix has \( 4 \times T \) size. During the Baum-Welch training of each of these 4 paths, the algorithm we will work with a matrix of size \( T \) instead of the original one.

### 3.3 Split Training

As we already discussed in the previous section, we use a split approach for the training phase. The splitting of the training brings also a performance improvement. In fact, instead of working with a matrix \( N = G \times B \), we work with \( G \) matrices of \( B \) dimension, and given a path to a destination we train this path only for the \( B \) matrix of the related \( goal \). Given \( L \) sequences of observations, \( G \) number of goals, and \( B \) known symbols, the cost for a training of the original matrix is \( L \times T (GB)^2 \) (considering every observation with a fixed length of \( T \)). The cost for the split training is instead \( L \times T \times (B)^2 \) for the Baum-Welch execution and \( L \times (GB)^2 \) for the interpolation step, with an important improvement w.r.t. the original training:

\[
\frac{Original}{Split} = \frac{L \times T \times G^2 \times B^2}{L \times T \times B^2 + L \times G^2 \times B^2} = \frac{T \times G^2}{T + G^2}
\]

Given a user with 6 goals and an average observation length of 30, the obtained improvement is \( 16 \times \left( \frac{66}{16} \right) = 16.36 \).

Baum-Welch is an iterative algorithm, and to find the best solution (the local best) it often needs several iterations. In our setting we do not need to find the best solution as a good one satisfies our requirements. For this reason we set at 1 the number of iterations.

### 3.4 Maximum Observation Length

As it has been described, Baum-Welch and Viterbi complexity is \( N^2 \times T \). We already showed how we optimized the matrix dimension \( N \), we now discuss how to reduce the response time by optimizing the observation sequence length \( T \).

Our approach is to fix a parameter named *maximum observation length m*: an observation sequence cannot be longer than this parameter. When this is the case, the observation is cut by...
considering only the last $m$ observations. Notice that this optimization can only be applied in the Viterbi algorithm. In fact, when we use it to predict the user future position and her goal, we do not need the exact sequence of her states. On the other hand, it cannot be used with Baum-Welch, because during the model setup we want a complete training made on the entire observation sequences.

The right size of the observation length is important, because a short one would not be sufficient to recognize a route, thus providing a wrong prediction, while one that is too long would not give additional information and would being too expensive in terms of performances. In our setting the domain is a map of $80 \times 80$ blocks and a user can move from one side of the map to the opposite, i.e., observation sequences can be longer than 80 blocks. We experimentally observed that $m = 10$ is a good compromise, allowing a good path recognition and an efficient response time.

Choosing the right $m$ is not trivial, suppose that a drives always crosses certain cells: when she crosses them from her home she normally goes to work, when she crosses them coming from the supermarket near her home she always goes to her friend’s house. If $m$ is too small in the prediction phase, there is the possibility that these cells fill the observation sequence and the system is not be able to predict that the driver is going to her friend’s house when she comes from the supermarket. To overcome this issue, it is possible to find the right $m$ during a precomputation on the user history, in order to find the maximum length of observations in common with different paths for different goals.

4 Experiments

We now show the benefits of using our model w.r.t. previous works. We developed alternative models in order to experimentally evaluate the differences in the quality of the predicted routes and in their execution times.

4.1 Metrics

The experiments are conducted with a training set of routes $T$, in order to set-up the models, and an evaluation dataset $D$ to verify their performances. In particular, we evaluate the models in terms of quality of the prediction and execution times. For every trip to be evaluated in $D$, we collect the predicted goal $g_p$ and the corresponding route to the goal $r_p$.

We use two kinds of dataset to validate the models: we built the first one with a route generator based on features obtained from real-world data, while the second one is a collection of real users movements in a large urban area.

For both datasets, we test every route in $r \in D$ and evaluate the quality of the prediction. In particular, we give to the model a fragment $i$ of the actual route (say, the first 10% observations) and compare the computed prediction $r_p$ versus the real route $r$. In other terms, to evaluate a prediction we consider as “golden data” the original route as this is available both in the synthetic dataset and in the real world setting.

We rely on two metrics for the evaluation. In the first metric, we match the predicted goal for $r_p$ with the actual goal in the golden data $r$. We report this boolean value as the **Error Goal**. To match the goals we compare the last symbol of the predicted route and the last symbol of the golden route. This match has to be flexible as we must consider the error rate for both goals. In fact, because of the localization error, we often do not find exactly the same symbol for the same destination. For this reason we accept as valid matches predicted goals that are in the area covered by the error rate of the cell for the golden goal. In the second metric, we start by refining the routes to compare them more easily. For every pair of consecutive predicted points $r_p(t)$ and $r_p(t+1)$ that are not neighbor, we add the shortest path from $r_p(t)$ to $r_p(t+1)$ to
we do the same for \( r_p \). Then, we evaluate the distance between \( r_p \) and \( r \) by computing the Levenshtein edit distance [8] between them with the following costs:

- Substitute: the cost to replace a symbol with another one is computed as the distance between them.
- Add: the cost to add a new symbol is the (shortest) distance between the symbol that must be added and the last predicted symbol in the route.
- Delete: the cost to delete a symbol is the (shortest) distance between the symbol that must be deleted and the original path.

The obtained distance between the predicted route and the real route shows how the model is working. As we want to evaluate the behavior of the models both with a small input (i.e., only a small fragment of the route is observed) and with a larger input, we always test eight different executions with an increasing percentage of input data from \( r \). We collect the distance for each execution and report the average as the Route distance (the smaller, the better).

In the results section we discuss two cases: the one when we compute the route distance by considering only routes when the predicted goal was correct, and the case when we consider all the routes (with correct or wrong predicted goal). This corresponds to evaluating how the models perform when they are in the best condition (the predicted goal is the right one) and how they perform in more difficult situations (such as the cases when predicting the right goal is not possible).

One important remark here is that we evaluate predictions made on noisy network data by comparing them with precise data (such as the perfect route in the synthetic scenarios and the GPS data in the real-world ones). For this reason, for the synthetic setting we also measure how much of the error in the output is due to the error function to simulate the network data, and how much of it is from errors in the prediction.

4.2 Synthetic Data

There are many tools to generate routes for moving objects [3, 6], but we decided to create our own generator in order to being able to control parameters that are peculiar for our setting such as the amount of noise introduced by cell-towers, the probability of having an alternative route between the same pair of goals, and so on.

4.2.1 Route Generation

The synthetic scenario generator takes as input the parameters listed in the second section of Figure 6 and returns a set of routes. The number of goals can be chosen randomly considering some bounds on the length, or can be manually set by defining their coordinates. Manually defined goals are used to verify ad-hoc settings, while random generated goals permit to explore a large variety of settings automatically. Once defined, goals act both as starting and as ending points for a route, which is defined by as a sequence of cells crossed by the moving object.

To define routes, we initialize a weighted undirected graph, with a node for every cell in the grid, and a weighted edge for every pair of adjacent cells. The weight for an edge is in inverse proportion to the speed we associate to the cell transaction. In our setting, we fixed such speed at 20km/h. Once goals are specified, a route is generated by computing the shortest path among them as shown in Figure 7a.

An user often has different alternatives to go to the same destination; in order to model this behavior a set of different alternative routes can be defined between a pair of goals as follows. From the original route, two points are selected randomly: the one where the user is going to
Figure 6: Parameters used in the experiments generation and in the evaluation. The default values are presented in bold. The first section represents general parameters, the second section parameters for the synthetic datasets generation, and the third section parameters of the HMM model.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Percentage of the route given as input</td>
<td>10%, \ldots, 80%, <strong>avg</strong></td>
</tr>
<tr>
<td>Number of Trips in training</td>
<td>45, 90, <strong>225</strong> (except for weeks/months test)</td>
</tr>
<tr>
<td>Number of Trips in evaluation</td>
<td><strong>25</strong></td>
</tr>
<tr>
<td>Number of executions/users</td>
<td>10, 20</td>
</tr>
<tr>
<td>Number of goals</td>
<td>2, 3, 4, 5, 6, \ldots, 16</td>
</tr>
<tr>
<td>Alternative path probability</td>
<td>0%, 80%</td>
</tr>
<tr>
<td>Max number alternative paths</td>
<td>0, 3</td>
</tr>
<tr>
<td>Roads</td>
<td><strong>true</strong>, false</td>
</tr>
<tr>
<td>Grid dimension</td>
<td>60, 70, <strong>80</strong>, 90, \ldots, 130</td>
</tr>
<tr>
<td>Bias</td>
<td>0.25, 0.5, <strong>0.75</strong>, 1</td>
</tr>
<tr>
<td>Error probability distribution (from real data)</td>
<td>GPS, <strong>Network</strong></td>
</tr>
<tr>
<td>Middle prob</td>
<td>0.1, 0.2, \ldots, 0.9</td>
</tr>
<tr>
<td>Interpolation Factor</td>
<td>0, 0.1, \ldots, <strong>0.9</strong>, 1</td>
</tr>
<tr>
<td>Forgetting Factor</td>
<td>0, 0.1, \ldots, <strong>0.9</strong>, 1</td>
</tr>
<tr>
<td>Size of Observations Length</td>
<td>1, 2, \ldots, <strong>10</strong>, 11, \ldots, 22</td>
</tr>
</tbody>
</table>

Figure 7: Transformation from a goal to goal route to more complex representations.
transition between two cells covered by a fast road is fixed at 60km/h. In other words, when we want to find the shortest path over such graph, the weight over normal edges is 3 and the weight over special (fast) edges is 1. Fast roads are defined with a set of functions as shown in Figure 8. In this way, given two goals, the shortest path between them will use the fast roads, making the prediction more challenging as shown in Figure 7c.

4.2.2 Cell-towers simulation

Routes generated in the previous section are realistic in a urban context, but their data correspond to very precise GPS observations. In order to evaluate our techniques in more critical settings, we carefully add noise to the data in order to simulate the data we collect with real-world observations. We start by extracting error distributions from real data collected both with GPS and cell-towers technologies. By using data collected in a large city with several users we computed different error probability distributions that we associated to two precision profiles (i.e., GPS and Network). An error probability distribution is represented as the probability that a specified error rate is seen for given a cell. Once fixed a profile, we associate to every cell in grid an error rate according to the actual distribution. In this way, given a route from one goal to another, we are able to get for every observed cell its error rate.

When real routes are observed by a network profile, it is easy to find “holes” between observations. In particular, the size of the holes depends on the error rate. Moreover, as network observations are uncertain, we must consider the error estimation returned by the device: we associate to an observation an area depending on the reported error rate, instead of a single cell.
To simulate these characteristics of real data, we take our clean generated route as input and use the error rate associated to the cell to “perturb” the route. Given the original route $r$ in Figure 7a with an associated precision profile, we iteratively build the new route as follows. For every symbol $i \in r$ (starting from the first) we use its error rate $e_i$ (expressed in number of cells) to enlarge the possible area of observations for it. We then remove the following $e_i - 1$ observations (i.e., those covered by the uncertain area of the previous observation) as shown in Figure 9b, and shift the observation by a random number of blocks limited by $e_i$ as shown in Figure 9c. With the above operations we make sure that is hard to have two sequences that are exactly the same even when they are generated from the same original path.

4.2.3 Model execution

In order to evaluate our model, and compare it with alternative ones, we fixed a scenario w.r.t. the parameters in the previous sections and collected experimental data with multiple executions, that is, for every test we generate at least 10 different users and report the average results in the graphs. Once all the parameters for the route generator module described above are defined, we can simulate routes for different users to create first the training $T$ and then the evaluation set $D$. In order to fine-tune the parameters for the model, we experimentally identified the optimal values for the four of them as follows.

![Graphs](image)

Figure 10: Evaluation of different values for the HMM model parameters: (a-b) Middle Probability, (c-d) Interpolation factor, (e-f) Forgetting factors, (g-h) size of the Observation sequence.

**Middle Probability** As discussed in previous sections, in the network setting we spread the probability of the observation over the cells around the one identified with uncertainty. We leave the 10% of the probability in the one reported by the original observation and distribute the uniformly over the set of cells included by considering the error. In order to identify this value, we conducted the experiments reported in Figure ??.

**Interpolation and Forgetting Factors** These parameters are important in the configuration of the model as they regulate the behavior of the matrix over different computations. In order to identify their best values for our setting, we conducted the experiments reported in Figure 10.

---

4 We shift the observation only among the right-upper diagonal since observations tend to repeat among routes and thus allowing a completely random shift would make the data more noisy than the real one. The effect of limiting this shift is controlled by the bias parameter.
**Size of Observations Length** We also tested the right size for the sequence of observations to be passed as input to the model during the prediction step. The results are reported in Figure ??.

4.2.4 Results

We first report the results for the parameters we fixed in our model. We then discuss qualitative results for the considered models: standard HMM, smart HMM, and CRPM [9], the state of the art solution. We tried also another model [11], but it failed in every test as it is not resistant to uncertain observations.

![Figure 11: Models comparison considering different values for the size of the grid we use to divide the area of interest.](image)

**GRID size** In our experiments we consider a large city divided in a 80x80 cells grid, with a 500 meters side for each cell. This size has been chosen after studying different alternatives because is a good compromise between cells dimension and number of states in our domain. In fact, a 160x160 matrix would give a measure of the user position slightly more precise but at the cost of a large increased number of potential observations (4 times more than those of the original grid). Moreover, Figure 11 shows how a larger grid strongly penalizes the alternative model in the goal prediction, as the data are much noisier. It is interesting to observe that the goal prediction results of our model do not depend on the size of the grid. Both models shows increasing route distance, but this is not surprising as the distance is defined in terms of number of blocks between the predicted route and the correct one: a finer grain in the grid leads to an higher distance.

**BIAS, Training set size** We generate graphs of the prediction quality (distance and goal error rate) considering the variation of the error bias. We remind that the bias factor affects the way synthetic data are generated. In particular, it defines how much the observation is concentrated on the the right-upper diagonal, in order to have some repeating blocks among different generations of the same route (i.e., to avoid having synthetic data more noisy than the real one). Higher values for bias leads to the generation of routes that share more observations, in this way is much more likely to find the same path during different movements, while a low error bias value decreases that possibility. We compared results for the synthetic scenario with the results from a real-world scenario with similar setting and fixed 0.75 as our default setting as it is the value where models were giving similar quality results. From Figure 12 is easy to see how the HMM model has a really good noise tolerance (almost invariant w.r.t. the error bias),
while CRPM has a really high Goal error rate with a low error bias value, that drops when this parameter goes near to 1. Notice also that the HMM model is able to predict with low errors when given only a few observations, while the CRPM is able to do so only with a high number of input data, but without matching our results. In most cases we have a better route distance even when we only consider the cases with correct goal predicted. This is motivated because HMM computed a route by considering all previous user movements, while the CRPM operation is based on finding the route with the best match. When we have a big training set (that has to be even bigger if the error bias is small) the two models can be compared. Finally, also in the case when we predict a wrong goal, the model returns a plausible route: this information can be seen by looking at the low distance in the case with both good and wrong goal predictions.

Roads, Alternatives The alternative routes and the roads network were motivated to make the prediction more challenging. In fact, in Figure 13 and Figure 14 it is possible to observe how the error rate and the Levenshtein distance increases by considering these parameters.
Figure 14: Models comparison considering the presence or absence of alternative routes

**Number and position of Goals** To understand how the number and the position of the goals influences the model, we choose a set of fixed goal distributions varying from a 2 goals to a 5 goals setting.

![Graph](image)

(a) Model goals error rate

(b) Model route distance

Figure 15: Models comparison considering the error goal rate and the route distance; the different setting are generated using 2 to 5 fixed goals

Figure 15 and Figure 16 show how an increasing number of goals makes the prediction more challenging. Notice that the number of goals is important, but the main difference is made by the locations of the goals. In fact, in the 5 goals setting, a goal is just positioned in the middle of the other 4 goals making the prediction more challenging because many routes across the outer goals go through the one in the center.

**HMM standard vs smart** We proposed a lot of different optimizations for the HMM model and we report their effects on the predictions in Figure 17. On GPS data, the two models have the same results, but differences can be found by looking at the network setting. The smart model obtains a better Goal error and, more importantly, has a much lower distance in the route comparison when all the input data. This is a clear indication that, even in cases when it is not possible to predict the correct final goal, the smart model can better predict the route.

**Increasing % input data** Figure 18 show the results with growing input observations. Given
Figure 16: Models comparison considering the error goal rate and the route distance changing the number of fixed goals.

<table>
<thead>
<tr>
<th></th>
<th>HMM with smart target</th>
<th>HMM without smart target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal error</td>
<td>GPS</td>
<td>Cell-towers</td>
</tr>
<tr>
<td>Route distance good</td>
<td>0.04</td>
<td>0.09</td>
</tr>
<tr>
<td>Route distance tot</td>
<td>44.98</td>
<td>88.39</td>
</tr>
</tbody>
</table>

Figure 17: Experimental results with the standard model and the smart one on a network simulated setting.

Figure 18: Models comparison considering the error goal rate and the predicted route distance changing the input size.

a longer observation sequence, the prediction models are able to improve their prediction. HMM model seems to take better advantage of this information as it moves from an high goal error with the first 10% of the input to very good results. This is an important expected result: it is acceptable that when the user starts her trip the system may predict a wrong destination, but is not acceptable that the system is not able to promptly adapt and change prediction along the
Random Goals, Different training size Figure 19 represents the results considering three different training sizes for a setting with weak days: a week, a month, and half a year of. All the routes are generated by the random settings generator with an increasing number of goals. The error bias has been set to 0.75 and the minimum distance between two goals has been set to 10 cells. The graph is plotted considering the average value of a set of 10 users for every setting and an increasing number of generated goals. Models get better results with bigger training sets and for more complex settings, with a lot of possible goals, is necessary a longer training to get acceptable results. The HMM model gets good prediction rate (0.2 error rate) for small settings (3, 4 goals), even when is provided only a very small training set (as a week of observations). CRPM is able to reach a 0.3 error rate only after one month of observation. After a month of observations, the error rate of the HMM model is under 0.7 for complex settings with up to 15 goals, thus outperforming the CRPM with six months of training.

Figure 20: Models comparison on execution time

Execution times, Training size The better prediction performance of the HMM model has a cost in term of computational time. Figure 20 shows that CRPM is faster than our approach in every test. Anyway, the execution times for the HMM model are acceptable: for a
complex scenario with 15 goals with 6 months of history, the training takes 20 seconds and the prediction about 1 second. From the Figure can be observed that we the HMM is not facing a quadratic grown in execution time. This behavior can be explained because even when we cover a really wide area with the error rate, the smart model considers only a small set of the possible states.

![Figure 21: HMM comparison on execution time, with and without “smart target”](image)

Figure 21 shows the improvement in speed gained with the smart HMM. In fact, in the standard implementation the grown in computational time is more related to the number of goals, while for the smart model the number of trained symbols is more important. Considering a complex setting with 15 goals and a month of observation, with the smart model we pass from a prediction made in a few minutes minutes to a prediction made in 0.1 second, and from a training time of about 2 minutes to a training time of 2 seconds. From the prediction quality in Figure 19 it is clear that after a month of training our model is ready to manage even complex settings, while for simple settings one week is enough.

### 4.3 Real World Data

We collect real-world data with a group of volunteers in Rome, Italy. Data have been collected with a Java application we developed for the Android system, which is available for readers that are willing to test the quality of their trajectories data collected with GPS sensors against the same data collected by using cell-towers only.\(^5\)

We used half of the data collected by each volunteer to train our HMM model and the CRPM model. We did this operation for every user. We trained distinct models with GPS data and cell-towers data.

Once the models are trained, we used the remaining data for each user to verify the quality of the predicted goals and routes. As for the synthetic data, we tested different configurations with an increasing amount of input data. For every trip of the user, we did a first prediction with the first 10% of the observations for the actual trip and collected the results. We did the same operation for 20%, ..., 80% of the data. As expected, the larger is the sequence of input observations, the better is the quality of the prediction. Examples of the predictions can be observed in Figures 22 and 23.

To evaluate the predictions we used the two metrics introduced above. We considered entire sequence of the observed data as the “golden data”, that is, we relied on the real information

\(^5\)Logger can be downloaded at [http://www.placemancy.com](http://www.placemancy.com)
to verify the predictions of the models. The results are in Figure 24. It can be seen from the experimental results that our model always outperforms existing solutions when we deal with real data.

4.4 An example of application

A carpooling application has been developed in Java. The system has two modules: the client is a mobile application for Android, while the algorithms for the HMM are implemented in a server module. Algorithms work in an interactive graphical representation of the portion of a city and in a web application relying on Google maps. In particular, one of the scenarios supported is a simulation of an urban area as depicted in Figure 25. Users of the system can act both as driver and as passenger by using a PC or an Android device, thus experiencing how easy would be to have real-time carpool services with our solution.

In a first phase, routes for moving objects can be defined as described in the previous sections, or real data from our users can be used. In this phase, users can define the routes in an interactive editor and we show them with animations over the map. In the second phase, the predictions model is applied on the defined scenarios. On the left hand side, a map shows the real current routes for the moving objects (e.g., a weekday morning), while the map in the right hand side shows the uncertain data given as input to the prediction models and the predicted future movements they produce.

As can be seen in Figure 25, for each object the top two predicted routes are reported, the thickness of the line represents the probability computed by the algorithm for the actual movement. As explained above, the granularity of the prediction depends on the precision of the
Figure 23: Prediction with real-world data by the CRPM model. Route are represented by using cells, both with GPS tracker data (left hand side) and cell towers data (right side). Blocks in green represent the input observations, the ones in red the predicted routes and destinations.

<table>
<thead>
<tr>
<th>HMM Model</th>
<th>CRPM Model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Goal error</strong></td>
<td><strong>Goal error</strong></td>
</tr>
<tr>
<td>GPS Data</td>
<td>0.21</td>
</tr>
<tr>
<td>Cell-towers Data</td>
<td>0.30</td>
</tr>
<tr>
<td><strong>Route distance</strong></td>
<td><strong>Route distance</strong></td>
</tr>
<tr>
<td>GPS Data</td>
<td>0.28</td>
</tr>
<tr>
<td>Cell-towers Data</td>
<td>2.67</td>
</tr>
</tbody>
</table>

Figure 24: Experimental results with both GPS and cell-towers data with our model and CRPM.

localization returned by the smartphone. In the system is possible to compare different input data and verify how such precision affects the quality of the prediction by showing the use of GPS data compared with the use of data relying on cell-towers only. Moreover, it is possible to compare the output produced by our approach with those of previous algorithms [11, 4].

In a third phase, the system supports a real-time carpooling application. Users can place a potential passenger on the map and click on her final destination. The system checks for moving objects that are first going to pass nearby the passenger and then nearby her destination. The matching drivers are then ranked with a scoring function depending on the space and time dimensions: (i) the time the passenger has to wait in order to meet the driver, (ii) the distance from the current position of the passenger to the pick-up point, and (iii) the distance from the drop point to the passenger final destination. If the passenger finds a suitable match, the driver is notified and the ride arranged. In the application it is possible to see how the same ride is likely to find a good match depending on the scenario by switching between the weekday and the weekend examples with the same ride request.
5 Related Work

Many works have studied the management of data for moving objects (e.g., [12, 13]).

5.1 Indexing

The goal of moving object indexing is to give efficient support to workloads with both queries and (frequent) updates. This latter requirement makes earlier works on spatial indexing unsuitable for the task, as they were based on the assumption of relatively static data and the focus was on query performance. Several alternative indexing approaches have been proposed lately and benchmark shows that their relative performances depend on the actual workload. However, these indexes usually focus on spatial proximity and offer no support for future long-term location predictions.

5.2 Goals and routes prediction

They offer many useful techniques to store, index, and query data, but no much attention has been posed on future locations prediction. On the other hand, in the transport community there are quite a few works on this topic (e.g., [11, 4, 2]), but they only focus on precise data, i.e., location data retrieved with GPS systems. Moreover, in cases where the HMM is used [11], in order to predict routes and destinations the authors rely on a frequency analysis, while we use the native algorithms for training and prediction. Another work [9], not based on HMM, is still focused on a frequency analysis but enriched with a route structure for pattern mining for recognizing a trip. Their work is still focused on precise data.

Foundational studies on the importance of real-time carpooling have also been recently done [1]. Interestingly, such study highlight the importance of reducing the transaction costs (i.e., the time needed to establish a ride arrangement), which is exactly one of the benefits of a real-time carpooling system based on our technology.
Many companies are beginning to offer a so called real-time ridesharing service with the support of smartphones for localization and of matching software to organize rides. However, such services do not have any support for predicting future locations, thus they only rely on manually entered trips.

References


