Handling Uncertainty in Trajectories of Moving Objects in Unconstrained Outdoor Spaces

Eleazar Leal, Le Gruenwald
School of Computer Science
University of Oklahoma
Norman, OK, USA
{eleal, ggruenwald}@ou.edu

Jianting Zhang
Department of Computer Science
City College of New York
New York City, NY, USA
jzhang@cs.ccny.cuny.edu

Abstract— **A trajectory is a polygonal line consisting of the positions that a moving object occupies as time passes, and as such, it can be derived by periodically sampling the positions of the object. In this manner, and due to the proliferation of location-sensing devices, it has been possible to create large datasets of trajectories. Using these datasets it is possible to derive much information about the movement patterns of the objects. However, trajectory data is uncertain, and this can negatively impact the accuracy of data mining algorithms used to obtain the movement patterns of objects. One of the sources of trajectory uncertainty is the error inherent to GPS measurements, and another source relates to the fact that in practice many trajectories have sampling points that are on average too far in time, so it is difficult to determine its movement between points. In this paper, we propose a technique called TrajEstU that estimates the trajectory of a moving object in an unconstrained space. The algorithm is applied when there is uncertainty in trajectories due to measurement errors and/or low sampling rates. Experiments show that TrajEstU achieves up to 98% accuracy on real life and synthetic trajectory datasets.**

Keywords- Trajectory Processing; Trajectory Mining; Uncertain Trajectories

I. INTRODUCTION

A trajectory is a polygonal line connecting the sequence of points that a moving object occupies in time. If the object is moving outdoors, then these points that make up the trajectory can be obtained by periodically sampling the positions of the object with location-based sensors like GPS. As a consequence of the widespread availability of these sensors, and of the fact that trajectories contain much information about the past, present and future characteristics of the movements of objects, large trajectory datasets have been collected [14] with the intention of using them to issue queries that help understand the moving objects themselves.

Applications: Among the queries that can be issued against trajectory datasets to understand the behavior of moving objects are top-K trajectory similarity queries, which find the K trajectories most similar to a given query trajectory. One example application of these queries is predicting the landfall of a developing hurricane. Because hurricanes are known to follow similar paths, meteorologists can search their records and use a trajectory similarity function to retrieve past hurricanes that have the most similar initial tracks to that of the developing hurricane, and use the trajectories of these past hurricanes to help predict its future landfall [1][9]. Another example is about understanding the way animals relate to their environment [2][8]. Animal ecologists want to understand how diseases are transmitted among birds, and how birds make use of space. By using trajectory similarity queries, ecologists can mine the trajectories of these animals to discover unknown movement patterns that can help them better understand how diseases are transmitted.

In those two applications, and just like in many other phenomena, uncertainty is a key obstacle for accurately knowing the past, present and future states of the objects under study. It is important to filter out the noise associated with trajectories when processing queries because, otherwise, the query results could end up being incorrect. The reason is that trajectory similarity measures rely on trajectory points in order to determine how similar two trajectories are, and if the points are noisy, or if the points are too distant from each other in time (due to low-sampling rate), then this noise carries over to the similarity measure, affecting its accuracy. So, when performing trajectory similarity analysis, it is important to reduce trajectory noise.

Trajectory Uncertainty: Trajectory uncertainty can come from different sources that can be classified into two major classes: the sources related to the measurement / instrumentation process, and the ones related to the model dynamics. One of the noise sources related to the measurement process is the noise inherent to GPS device measurements [2][12]. This noise arises because no measurement is perfectly accurate, but also arises from the environmental conditions surrounding the sensor at the moment when the measurement is made. For example, in our second application, the GPS measurement errors can be greater if there are overcast skies in the place where the animals are, or if the animals have tampered with their GPS collars, etc. This type of noise is illustrated in Figure 1, where the trajectories of three objects, Q, R and S are captured. Here we see that around each position (trajectory point) sampled, Q_i, R_i, and S_i, for each of the three corresponding trajectories Q, R and S, there is an "area of uncertainty." If we ignore the measurement uncertainty, the most similar trajectory to Q is trajectory R. However, if we consider the measurement to be noisy, then there is a high probability that trajectory S is the most similar trajectory to Q because the points of S have low uncertainty and are almost as close to Q as the points in R. On the other hand, the

points on R have large uncertainty, so there is a non-zero probability that the points R_0 and R_2 are farther away from Q than S_0 and S_2, respectively.

One of the sources of uncertainty related to the model dynamics is the interaction between the linear interpolation model for trajectories and the inconsistencies in the sampling rate. If the time interval between two consecutive sampled points in a trajectory is very long, i.e. the trajectory has a low sampling-rate, and the object moves at a high speed with a non-constant velocity and/or acceleration during that time interval, then the linear interpolation model underlying trajectories may not be a good approximation to the object's movement. Figure 2 shows an example illustrating this situation. The dotted lines correspond to the actual path taken by the moving objects, while the straight lines connect consecutive sampled points. If we ignore model uncertainty, then trajectory R is the most similar to trajectory Q because its points are all closer to Q than the points in S. However, if we consider the actual true paths, we see that S is the object with the most similar path to Q. The uncertainty arises because the sampling rate was too low. This source of uncertainty is present in our animal ecology example because scientists want to maximize the lifetime of the expensive telemetry devices that they attach to animals, which makes these devices subject to energy utilization constraints. Therefore, to save energy, geolocators cannot work continuously, so animal trajectories cannot have high sampling rates [2]. As a consequence of this, there is great uncertainty in the trajectory that an animal takes between the sampled points. Another source of uncertainty in the animal ecology application relates to the fact that there frequently are missing data in the trajectories because of failed attempts at geolocation, in which the GPS device cannot successfully determine the animal's position [2]. Hence, we see that there is a concern about the impact of trajectory model uncertainty in their studies.

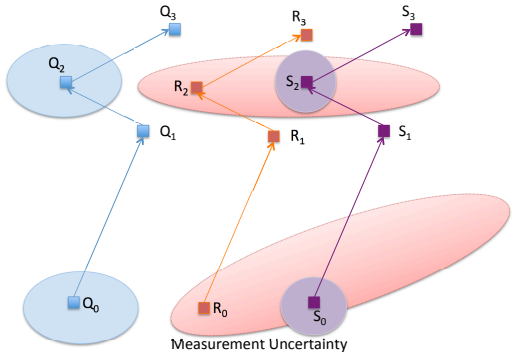

Figure 1. Measurement/Instrumentation Uncertainty

Problem: The problem tackled in this paper consists in designing an algorithm, called TrajEstU, that estimates the trajectory of a moving object in unconstrained spaces (i.e. spaces where the objects' movements are not constrained by roads), considering both the measurement and model uncertainties. This is a computationally challenging problem. One of the reasons is the volume of the data involved: using a trajectory database to improve upon the quality of the estimates can potentially involve scanning through a large set of nearby trajectories, each of which may consist of hundreds of thousands of sampled points. Another reason relates to the veracity of the data involved: as it has been mentioned before, trajectories are inherently uncertain. These reasons make it possible to argue that this paper's problem falls in the realm of Big Data.

To tackle the problem of filtering out the noise in trajectories in unconstrained spaces, we propose to model a trajectory with a sequence of generalized linear regression models that accounts for an object that changes its velocity and/or acceleration between sampled points.

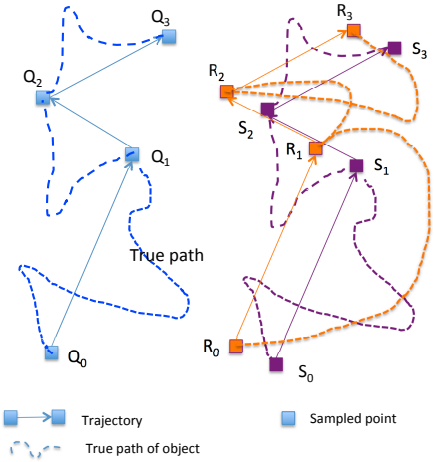

Figure 2. Model Uncertainty

This strategy, however, faces another challenge when dealing with low sampling rate trajectories because of the lack of data that is necessary to fit a generalized regression model. To address this challenge, we use the knowledge provided by other trajectories that are close to the trajectory to improve its position estimates. One advantage of modeling trajectories this way is that this type of model can then be used to support other queries on trajectories, such as trajectory similarity queries, whose accuracy can be negatively impacted by uncertainty. Another advantage of this type of model is that, since these are continuous-time models (which means that the model can predict the position of the moving object at any time instant during its lifespan, and not just estimate the position of the object at discrete time instants), they make it possible to resample the original noisy trajectory. By resampling the trajectory, we can obtain more points in those portions of the trajectory's lifetime where the trajectory has changing acceleration and velocity (hence, it deviates from a straight-line movement), and eliminate sampled points in other portions where the object's velocity is constant. Doing so allows us to use existing and well-known trajectory similarity measures that do not address trajectory uncertainty [3] [13] to compute trajectory similarities on noisy trajectories.

Contributions: The major contributions of this paper are the following:

• We propose an algorithm to estimate the trajectory of an object moving in an unconstrained space that takes into account the noise originated from location-based sensor measurements (measurement uncertainty), and from low-sampling rates (model uncertainty).

• We conduct experiments to measure the quality of the trajectory estimates.

The remainder of this paper is organized as follows. Section II discusses the related work. Section III introduces the background concepts. Section IV presents the proposed algorithm. Section V contains the performance analysis of the algorithm. Finally, Section VI presents conclusions and future work.

II. Related Work

In this section, we discuss the work related to TrajEstU in the areas of uncertain trajectory models, techniques to infer routes for low-sampling rate trajectories, and trajectory smoothing.

Uncertain Trajectory Models: Ma et al. [12] propose a probabilistic model for trajectories based on a polygonal line model that associates a probability density/mass function to every sampled point, and then connects the consecutive sampled points using linear interpolation. In this manner they can estimate the positions of a moving object during its lifetime, and then use these estimates to perform trajectory similarity search. There are two differences between this model and ours. The first difference is that our work is designed to address the issues of model and measurement uncertainty in trajectories, while the model in [12] addresses only the measurement uncertainty issue and not the model uncertainty because it performs linear interpolation between points; therefore, if two consecutive points are very far apart in time, the object's path might significantly diverge from a straight line. The second difference is that our work makes use of surrounding trajectories as a means to improve the quality of the estimations for the query trajectory, which is not done in [12]. Another technique for uncertain trajectories was introduced in [6] by Emrich et al., which models trajectories using Markov chains. The difference between this technique and ours is that the former assumes that objects move in a discrete space while our technique works in a continuous space. A potential issue when using Emrich et al.'s technique in an unconstrained continuous space is that it requires discretizing the space, and in doing so, we can potentially end up with a discrete space-time with a prohibitively large number *S* of states. Additionally, there is the potential that different moving objects can have very different movement dynamics, so this would mean that for each object we would have to compute an even larger *SxS* transition matrix.

Route inference: Zheng et al. [15] address the problem of reducing the uncertainty of low-sampling rate trajectories, i.e. trajectories such that the average interval between consecutive points is over 2 min, in road networks (constrained space). To accomplish this, they propose an algorithm called HRIS that fills the low-sampling rate sections of a trajectory by searching for other nearby trajectories (called reference trajectories) in the database that satisfy certain network constraints, like velocity constraints. After performing this search, their algorithm finds a set of associated road network paths corresponding to those reference trajectories that maximize an objective function based on popularity and uniformity of the traffic through that road network path. This work, unlike our work, assumes that objects move in a constrained space, so it exploits the knowledge provided by a road network to reduce the uncertainty in the low-sampling rate trajectories. Therefore, it is not applicable in the scenario of objects moving in unconstrained space (like hurricanes and animal trajectories), where objects do not move on road networks, or where there is no road network available.

Trajectory smoothing: The work of Cao and Krumm [17] addresses the problem of mining a road network using a database of trajectories moving in this network. In doing so, they perform trajectory smoothing to try to discover the underlying path described by the moving objects. However, this work is specifically designed to deal with trajectories of objects in constrained spaces, unlike our work which deals with unconstrained ones.

III. Background

A. Preliminaries

1) Notation

Given a trajectory *traj*, the list of points that make up *traj* is denoted by *traj.pts*. If $I = [t_0, t_1]$ is a time interval, then the notation *traj.pts[I]* denotes the sub-list of points of trajectory *traj* such that their corresponding time instants fall within the time interval *I*. On the other hand, if *i* is an integer, then *traj.pts[i]* denotes the ith point of *traj*. Given a point *p*, then *p.x*, *p.y*, and *p.t* denote the x-component, the y-component, and the time component of *p*, respectively. The notation *traj.numPts* refers to the number of points of trajectory *traj*. If *M* is a real matrix, then *M[i,j]* denotes the element of *M* located in the ith row and jth column, and *M[i,:]* denotes the ith row of *M*. Finally, if *c* is a cluster of trajectories, then *c.repr* denotes its representative trajectory.

2) Definition of a Trajectory

Given a set $\{(x_i,y_i,t_i) \mid t_i \leq t_{i+1}, 1 \leq i < n\}$ of points in R^3 sampled from the movement of an object with a location sensor, a *trajectory* over *S* is a continuous function $\tau : [1,n] \rightarrow R^3$ where $\tau(i) = (x_i,y_i,t_i)$ for all integers $i \in [1,...,n]$ and such that $\tau(x)$, with $x \in [t_i,t_{i+1})$, is the interpolated value between $\tau(i)$ and $\tau(i+1)$ [5].

3) Definition of a Sub-trajectory

Given a trajectory *traj* with sample point set $S = \{(x_i,y_i,t_i) \mid t_i \leq t_{i+1}, 1 \leq i < n\}$, we define the *sub-trajectory of traj* during the interval *[a,b]*, denoted by *traj[a,b]*, with $a < b$ as the set of all points of the trajectory *traj* with timestamps between a and b. More formally, it is the subset $\{(x_i,y_i,t_i) \in S \mid a \leq t_i \leq t_{i+1} \leq b, 1 \leq i < n\}$.

4) Extended Minimum Bounding Rectangle

Given a set of points *S* and $\varepsilon > 0$, the epsilon Extended Minimum Bounding Rectangle (EMBR) of *S* is a minimum bounding rectangular (MBR) of *S* each side of which has

been extended by ε in every direction, so that if an MBR has the lower left corner *(lx, ly)* and the upper right corner *(ux, uy)*, then the corresponding EMBR(ε) has the lower left corner *(lx - ε , ly - ε)*, and the upper right corner *(ux + ε , uy + ε)*.

5) Low-sampling rate trajectory

Given a trajectory, it is said to be a *low-sampling rate trajectory* if the average time span between any two of its consecutive points is greater than a predefined threshold. A trajectory that is not a low-sampling rate trajectory is said to be a high-sampling rate trajectory.

IV. THE PROPOSED TECHNIQUE: TRAJESTU

A. Overview

TrajEstU receives as input a trajectory database *db* and an uncertain query trajectory *q*. To estimate the true path of the trajectory *q* of a moving object in an unconstrained space when there is uncertainty due to measurement errors and/or low sampling rates, TrajEstU goes through three stages: (i) a pre-processing stage, (ii) a model fitting stage, and (iii) a trajectory generation stage.

In the first stage, the pre-processing stage, TrajEstU performs local segment clustering of the trajectories in *db* [11] with the intention of finding the spatial patterns that the trajectories in the database exhibit. The output of this local segment clustering algorithm is a set of segment clusters, each of which has an associated representative trajectory describing the behavior of its cluster. Once the database trajectories have been locally clustered, TrajEstU builds an R-tree *clusterTree* containing the representative trajectories of each of the segment clusters found.

In the second stage, the model fitting stage, TrajEstU identifies the time intervals where *q* has a near-constant acceleration. Then, for each one of these time intervals *I*, the algorithm finds the extended Minimum Bounding Rectangle (EMBR) containing the points of *q* with timestamps contained in *I*, called *q.pts[I]*, and uses this EMBR to perform a range search over *clusterTree*. The output of this range search is a set of representative trajectories called *clustersI*. Then, for each representative trajectory *r* in *clustersI*, TrajEstU builds a separate linear constant acceleration model for $q.pts[I] \cup r.pts$. The result of this operation is one candidate constant acceleration model for *q.pts[I]* per representative trajectory *r*. Out of all these models for *q.pts[I]*, TrajEstU chooses the one with the highest goodness of fit. The collection of constant acceleration models for all intervals *I* makes up the kinematic model for *q*, and allows us to predict the true path of the object at any given time.

In the final stage, given the kinematic model found in the second stage, TrajEstU generates a new trajectory with uniform sampling rate, called the *estimated trajectory*, whose points are the ones predicted by the kinematic model.

B. Pre-processing stage

In this section we explain the first phase of TrajEstU, which consists of the preprocessing operations that need to be performed only once. Then, *as more query trajectories arrive, these operations need not be performed again*.

In this stage, the idea is to identify the local patterns displayed by the set of moving objects in *db*. To this end, we perform local trajectory clustering [11] of the trajectories in the database, and find the representative trajectory associated with each cluster. It is in these local cluster trajectory representatives that the movement patterns are condensed.

Our proposed algorithm uses local trajectory clustering to find local trajectory patterns. Lee et al. [9][11] introduced the idea of first partitioning a set of trajectories into segments and then clustering the resulting segments, instead of clustering the trajectories as a whole. This serves our objectives because by clustering trajectories into segments, we can obtain the movement patterns in a given small area, instead of globally clustering the trajectories, which would not be able to discover patterns at a local scale.

C. Model Fitting Stage

In this section we explain TrajEstU's second phase, the model fitting stage. We describe the kinematic trajectory model used and how to estimate its parameters. The model is based on kinematics. First, we identify the time intervals of the object's trajectory where it has constant acceleration, and build a constant acceleration model for each of these intervals. The collection of these constant acceleration models makes up the kinematic trajectory model. Then, the constant acceleration models for the two consecutive time intervals $[t_0, t_1]$ and $[t_1, t_2]$ need to be smoothly connected in order for them to be consistent around t_1.

1) Constant Acceleration Model

It is known that if an object *o* with initial position x_0 and initial velocity v_0 moves *with constant acceleration a* during a time interval $[t_0, t_0 + deltaT]$, $deltaT > 0$, we can then accurately determine the position of *o* at any time $t_0 + t$, with $0 < t < deltaT$ using $x(t) = x0 + v0 \cdot t + a \cdot t^2/2$. This is the time-linear dynamic trajectory model that we will use for our trajectories, and its parameters are x_0, v_0 and a.

In the case of an uncertain trajectory, the problem is that it consists of a sequence of only *uncertain* positions (called sampled points or observations), so the velocity and acceleration, if computed straightforwardly from the observed positions, are also uncertain quantities. To address this problem, we find the best linear constant acceleration trajectory model that fits the observed positions by using a standard linear regression model [10]. The form of the linear model is $Z = HX$, where Z is the observation vector, H is the model matrix, and X is the parameter vector. As is the case in standard linear regression, we seek to find the parameter vector X that minimizes the sum of the squares of

the errors $(Z - HX)^T(Z - HX)$. We now explain with an example how Z, H and X are found.

Suppose for example that we are given the sequence of $n=3$ points $\{(x_0,y_0,t_0), (x_1,y_1,t_1), (x_2,y_2,t_2)=(x_{n-1},y_{n-1},t_{n-1})\}$ belonging to an uncertain trajectory. To find the best linear constant acceleration trajectory model fitting these data, we compute $\Delta t_i = t_i - t_{i-1}$ for $1 \le i < n = 3$, and then build the model matrix H of size $2n \times 6 = 6 \times 6$ as follows:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & \Delta t_1 & \Delta t_1^2 & 0 & 0 & 0 \\ 1 & \Delta t_2 & \Delta t_2^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta t_1 & \Delta t_1^2 \\ 0 & 0 & 0 & 1 & \Delta t_2 & \Delta t_2^2 \end{bmatrix}$$

and the observation vector Z of size $2n \times 1 = 6 \times 1$ as follows:

$$Z = [x_0 \quad x_1 \quad x_2 \quad y_0 \quad y_1 \quad y_2]^T$$

Then, we use the standard formula $X = (H^T H)^{-1} H^T Z$ to compute the parameter vector corresponding to the linear constant acceleration trajectory model that minimizes the sum of the squares of the residuals $(Z - HX)^T(Z - HX)$:

$$X = (H^T H)^{-1} H^T Z = \left[x_0 \quad v_0^{(x)} \quad \frac{1}{2}a^{(x)} \quad y_0 \quad v_0^{(y)} \quad \frac{1}{2}a^{(y)}\right]^T,$$

where the vector (x_0, y_0) is the initial position of the object during the interval $[t_0, t_1]$, $v_0 = (v_0^{(x)}, v_0^{(y)})$ is the initial velocity, and $a = (a^{(x)}, a^{(y)})$ is the constant acceleration during that interval. This means that the fitted model during the interval $[t_0, t_1]$ has x-component $m^{(x)}(t) = x_0 + v_0^{(x)}t + 0.5a^{(x)}t^2$, and y-component $m^{(y)}(t) = y_0 + v_0^{(y)}t + 0.5a^{(y)}t^2$.

One key assumption made when building the above linear model, besides that the acceleration is constant, is that the trajectory has a high-sampling rate during the time interval corresponding to the observed data, which means that there are enough data from which to discover the parameters of the model. However, when building a linear model for low-sampling rate trajectories, there may not be enough data to build an *accurate* model. To solve this problem, we can exploit the knowledge provided by the database of trajectories by mining the movement patterns described by the trajectories in the database around the spatial area where our given trajectory query has a low-sampling rate. To discover these movement patterns we use a trajectory clustering algorithm. However, algorithms for clustering trajectories may not be appropriate for discovering the movement patterns around a particular area of interest (where the given trajectory has a low-sampling rate) because regular trajectory clustering algorithms cluster trajectories globally. Instead, we propose using an algorithm that performs local clustering of trajectory segments [9] because it first splits trajectories into smaller segments and then clusters the segments. To each cluster, the clustering algorithm assigns a representative trajectory that captures the behavior of the segments in the corresponding cluster. By incorporating the knowledge of these representative trajectories into the constant acceleration model, we can overcome the difficulty of building a linear model for low-sampling rate sections of trajectories.

2) Incorporation of Trajectory Patterns

Once the database trajectories have been locally clustered, we have the knowledge of the spatial patterns that the trajectories in *db* exhibit in space. To exploit this knowledge, during the generation of a constant acceleration model in the time interval $[t_0, t_1]$ with $t_0 < t_1$, we do the following. We first construct an extended minimum bounding rectangular EMBR(ε) with ε >0 surrounding the sampled points of the trajectory with times in the range $[t_0, t_1]$. Then, we locate the set of clusters *Clusters* that intersect with this EMBR. This is illustrated in Figure 3, where we have $t_0 = P_2.t$ and $t_1 = P_3.t$. There we see that the intersection of the EMBR(ε) of $Traj.pts[P_2.t, P_3.t)]$ with *Clusters* is *{Cluster1, Cluster2, Cluster3}*. We will consider each of these clusters individually, and for each one, we will build a constant acceleration model using the points in the set $Traj.pts[t_0, t_1] \cup Clusters[i]$, where $Clusters[i]$, denotes the ith cluster in the list *Clusters*. However, one obstacle here is that the points in *Clusters[i]*, unlike the points in $Traj.pts[t_0, t_1]$, do not have timestamps; therefore, we cannot directly build the matrix H, which depends on the timestamps of the points.

To solve this problem, we assign times to these points using the closest trajectory points. To this end, we associate an empty list L_k with every point P_k in the trajectory. Then for every point C_j in *Clusters[i]* we find the closest consecutive pair of trajectory points P_m, P_{m+1} and insert C_j into L_m. This is illustrated in Figure 4 where we see that the points C_0, C_1 and C_2 in *Cluster3* have the consecutive pair of trajectory points P_2 and P_3 as the closest one; so $L_2=[C_0, C_1, C_2]$. Then we consider the list of points $[P_2, C_0, C_1, C_2, P_3]$ and use this list to linearly interpolate the timestamps for C_0, C_1 *and* C_2, assuming a uniform sampling rate. Therefore, the timestamp of C_1 will be $\frac{2}{5}(P_3.t - P_2.t)$, and the time stamp for C_2 will be $\frac{3}{5}(P_3.t - P_2.t)$. Once every point C_j in *Clusters[i]* has

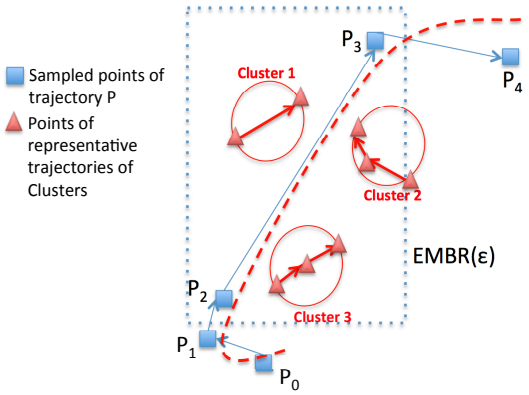

Figure 3. Finding representative trajectories

been assigned a timestamp, we can proceed with the fitting of a candidate constant acceleration model for the set of points $Traj[t_0, t_1] \cup Clusters[i]$.

3) Selecting the Best Constant Acceleration Model

In Subsection IV.C.2 we examined how we can build a constant acceleration model for a trajectory *Traj* during the time interval $[t_0, t_1]$ with $t_0 < t_1$, using a single segment cluster in the database found by intersecting the EMBR(ε) surrounding the points of $Traj.pts[t_0, t_1]$ with the set of segment clusters *Clusters*. However, the result of this intersection could yield more than one intersecting segment cluster, which would imply that we have 1 candidate constant acceleration model per intersecting cluster. Out of all these possible constant acceleration models for $Traj.pts[t_0, t_1]$, we pick the model that maximizes the goodness of fit, i.e. the one that best explains the variance of the data, using the standard formula for the coefficient of determination.

4) Variable Acceleration Model

A moving object could, however, change its acceleration throughout the extent of its movement, rendering the above constant acceleration model incapable of accurately predicting the movement of the object. Hence, we identify the time instants at which the moving object changes its acceleration. To accomplish this, we use a real positive number $tol > 0$ as a constant parameter, and keep an integer *startInterval* with initial value 0, and scan the query trajectory *q* from the beginning to the end, and computing the average acceleration at each point with an index *i*, starting from *0*. If the absolute value of the difference between the acceleration at *q[startInterval]* and at *q[i]* exceeds the fixed parameter *tol*, then we consider that *[q[startInterval].t, q[i].t]* is a constant acceleration interval. Then we assign *i* to *startInterval* and then keep scanning the trajectory in search for the next interval.

The average acceleration is, by virtue of being computed from the observed positions of an uncertain trajectory, an uncertain quantity. By using the *tol* parameter as a threshold to identify the time intervals where the trajectory has

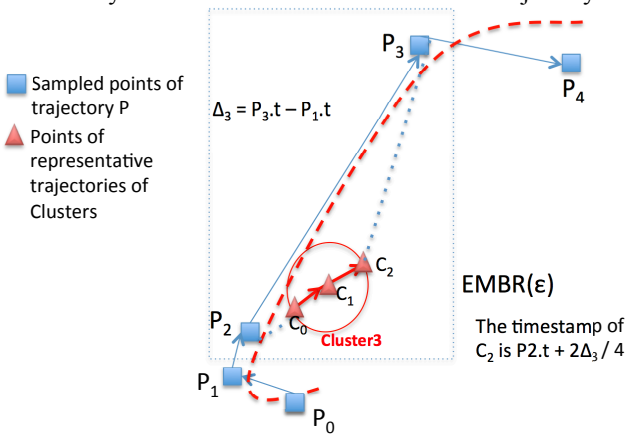

Figure 4. Timestamp calculation for one set of cluster points

constant acceleration, we are able to address this uncertainty problem.

5) Model Coupling Function

As mentioned in the sections IV.C.1 to IV.C.4, to model the movement of an object, we identify the time intervals where the object's acceleration does not vary beyond a pre-specified tolerance $tol > 0$, and then fit a constant acceleration model for each of these intervals. However, at the time instant that lies at the boundary between two consecutive time intervals (for example, if two intervals of near-constant acceleration are $[t_0,t_1]$ and $[t_1,t_2]$, with $t_0 < t_1 < t_2$, then at time t_1 we have two estimates for the position of the object: one estimate arises from the constant acceleration model during $[t_0,t_1]$, and the other from the model during $[t_1,t_3]$, and these two estimates could potentially differ. To overcome these difficulties we smooth the trajectory model by smoothly connecting, or "coupling" [7], the constant acceleration model during $[t_0,t_1]$ with the constant acceleration model during $[t_1,t_2]$. This is illustrated in Figure 5 where we see two constant acceleration models: Model $P_0.t$ to P_2.t, and Model $P_2.t$ to $P_4.t$, showed as the pointed and dashed lines, respectively. These models disagree in their estimations around P_2, but when we incorporate the model coupling function, then both models are smoothly connected.

Assume we have a constant acceleration model m_1 during $[t_0,t_1]$ with $m_1^{(x)}(t)$ and $m_1^{(y)}(t)$ as the x and y-components, respectively, and another constant acceleration model during $[t_1,t_2]$ with $m_2^{(x)}(t)$ and $m_2^{(x)}(t)$ as its x and y-components, respectively. To smoothly connect the two models we can use the hyperbolic tangent, which is a function that is known to be suitable for this purpose [7]. The model coupling function $c(m_1, m_2)^{(x)}(t)$ smoothly connects both models and has an x-component given by

$$c(m_1, m_2)^{(x)}(t) = m_1^{(x)}(t) + \frac{\tanh(t-t_1)+1}{2}(m_2^{(x)}(t) - m_1^{(x)}(t)),$$

and the y-component is identical, but replacing *x* by *y*. This function smoothly connects both models because *tanh* converges to 1 as *t* goes to infinity, and to -1 as *t* goes to minus infinity. Therefore, for large *t*, $c(m_1, m_2)^{(x)}(t)$ converges to $m_2^{(x)}(t)$, and for *t* values less than t_1, it converges to $m_1^{(x)}(t)$.

D. Trajectory Estimation Stage

In this section we explain the final stage of TrajEstU. The purpose of this stage is to generate a trajectory, called the

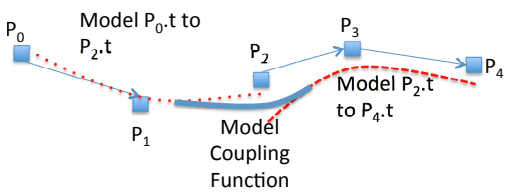

Figure 5. Model coupling function

estimated trajectory, using the kinematic model found during the model fitting stage. This kinematic model is a collection of constant acceleration models $\{(model_I, I) \mid I \text{ is a constant acceleration interval of } q\}$. Each $model_I$ can be used to estimate the true trajectory of the moving object during the time interval I. Given a number of points *numPoints*, and the lifetime *Inter = [startTime, endTime]* of q, we can generate a uniform sampling rate trajectory by querying the kinematic model at time instants $\{startTime, startTime + \Delta_t, startTime + 2\Delta_t, ...\}$, thereby obtaining the estimated trajectory.

E. Details of TrajEstU

In this section we explain the details of TrajEstU, the pseudocode of which is presented in Figure 6. TrajEstU consists of a preprocessing stage where the spatial trajectory patterns of the database are mined, a model fitting stage where we build a trajectory model, and then a trajectory estimation stage where we use such model to produce an estimated trajectory.

1) Pre-processing Stage

First, the algorithm invokes the function *cluster_segments* implementing the local clustering algorithm [9] to get the set of segment clusters for all trajectories in the database *db* (Line 44). Once this is done, the algorithm constructs an R-tree containing the representative trajectories of each local segment cluster (Line 45). If there are multiple query trajectories, then *this stage is to be executed only once off-line, so it does not to be run with each query*.

2) Model Fitting Stage

Once the pre-processing stage has finished, TrajEstU proceeds to fit a model for every query trajectory by calling function *FILLDATA* (Line 46). The function *FILLDATA* receives as input arguments a query trajectory q, a database of trajectories *db*, an R-tree *clusterTree* with the representative trajectories of the clusters, and a set of clusters *clusters*, and is in charge of fitting a kinematic trajectory model for q. It first computes the set *intervals* of time intervals where q has near-constant acceleration (Line 3). It then builds for each constant acceleration interval I of q a separate model describing the behavior of q during I (Lines 4-15). It achieves this by constructing an Extended Minimum Bounding Rectangle EMBR(ε) enclosing the portion of q during I to account for the uncertainty of q during I (Line 5). Then, it performs a range search over the R-tree *clusterTree* using the EMBR as input to determine the set $clusters_I$ of all trajectory segment clusters located near q during I (Line 6). It then finds a candidate constant acceleration model for each trajectory segment cluster in $clusters_I$ by calling the function CONSTACCMODEL and computes the goodness of fit for each model (Lines 7-10). Finally, it selects the model with the greatest goodness of fit (Lines 11-12).

The function *CONSTACCMODEL* in Line 16 takes a set of trajectory points *points* as its input argument, and finds a linear regression model that fits the trajectory points in *points* following the approach explained in Section IV.C. By using this linear regression approach to fit a constant acceleration trajectory model, we are able to address the issue of measurement uncertainty because linear regression

Algorithm TrajEstU(q, db)
Input: A query trajectory q. A trajectory database *db*.
Output: A trajectory $traj$ that estimates q.
1: **function** FILLDATA($q, clusterTree, clusters$)
// Obtains a list of const. acceleration models for q
2: $models \leftarrow \emptyset$
3: $intervals \leftarrow const\_acc\_intvls(q)$
4: **for** $I \in intervals$ **do**
5: $mbr_I \leftarrow EMBR(\epsilon)$ of q during I
6: $clusters_I \leftarrow rangeSearch(clusterTree, mbr_I)$
7: **for** $c \in clusters_I$ **do**
// c.repr is the repr. trajectory of c
8: $model_I^c \leftarrow$ CONSTACCMODEL($q[I].pts) \cup c.repr.pts$)
9: $r_c^2 \leftarrow goodness\_of\_fit(model_I^c)$
10: **end for**
11: $model_I \leftarrow argmax_{c \in clusters_I} r_c^2$
12: $models \leftarrow models \cup \{(I, model_I)\}$
13: **end for**
14: **return** $models$
15: **end function**
16: **function** CONSTACCMODEL($points$)
// Fits a constant acceleration model to $points$
17: Sort $points$ by increasing timestamp values
18: $H \leftarrow$ new matrix of size $(2 \cdot |points| \times 6)$
19: $Z \leftarrow$ new vector of size $(2 \cdot |points| \times 1)$
20: $H[1,:] \leftarrow [1,0,0,0,0,0]$ and $H[1+|points|,:] \leftarrow [0,0,0,1,0,0]$
21: **for** $i \in [2, \ldots, |points|]$ **do**
22: $\Delta_i \leftarrow points[i].t - points[i-1].t$
23: $H[i,:] \leftarrow [1, \Delta_i, \Delta_i^2, 0, 0, 0]$
24: $H[i+|points|,:] \leftarrow [0,0,0,1,\Delta_i,\Delta_i^2]$
25: $Z[i] \leftarrow points[i].x$
26: $Z[i+|points|] \leftarrow points[i].y$
27: **end for**
28: **return** $(H^T H)^{-1} H^T Z$
29: **end function**
30: **function** ESTIMATETRAJECTORY($models, numPoints, Inter$)
// Obtains an estimated trajectory from the models
31: $traj \leftarrow$ empty trajectory
32: $\Delta_t \leftarrow (Inter.endTime - Inter.startTime)/numPoints$
33: **for** $i \in [1, \ldots, numPoints]$ **do**
34: $\tau_i = startTime + i \cdot \Delta_t$
35: Find $(model_I, I) \in models$ such that $\tau_i \in I$
36: $t_i \leftarrow \tau_i - I.start$
37: // $model_I$ is a vector $[x_0, v_0^{(x)}, (1/2)a_x, y_0, v_0^{(y)}, (1/2)a_y]^T$
38: $p_i.x \leftarrow model_I \cdot [1, t_i, t_i^2, 0, 0, 0]$
39: $p_i.y \leftarrow model_I \cdot [0,0,0,1,t_i,t_i^2]$
40: Add point $(p_i.x, p_i.y, \tau_i)$ to $traj$
41: **end for**
42: **return** $traj$
43: **end function**
44: $clusters \leftarrow cluster\_segments(db)$ // Pre-processing
45: Create an r-tree $clusterTree$ with $\{c.repr \mid c \in clusters\}$
46: $model \leftarrow$ FILLDATA($q, db, clusterTree, clusters$)
47: $Inter \leftarrow q$'s lifetime
48: $traj \leftarrow$ ESTIMATETRAJECTORY($model, q.numPts, Inter$)
49: **return** traj

Figure 6. TrajEstU pseudocode

does not force the resulting model to agree with every single sampled point. By considering the representative trajectories of the segment clusters, we are able to address the problem that may arise when trying to fit a model with very few data points available.

3) Trajectory Estimation Stage

After the model fitting stage is completed, TrajEstU proceeds to compute the lifetime *Inter* of the query trajectory *q* (Line 47) and then invokes the function *ESTIMATETRAJECTORY* to estimate the true trajectory of *q* (Line 48). The function *ESTIMATETRAJECTORY*, shown in Line 30 of Figure 6, receives as input parameter a kinematic model *models* that is a set of pairs of the form ($model_I$, *I*), where every $model_I$ is a constant acceleration model valid during the time interval *I*. This function also receives as an input parameter the number of points (*numPoints)* of the final trajectory estimate, and an interval *Inter=[startTime, endTime]* that is the lifetime of the estimated trajectory. This function first creates an empty trajectory (Line 31) that will eventually be the estimated trajectory, and then computes the time span between consecutive points, assuming uniform sampling (Line 32). The function then computes the points of the estimated trajectory one by one (Lines 33-41). To do so, it finds the model ($model_I$, *I*) in *models* that is valid at time τ_i (Line 34-37), and computes the estimated position at time τ_i using scalar products (Lines 38-39). Finally, it adds the estimated position to the estimated trajectory (Line 40).

V. PERFORMANCE ANALYSIS

A. Experimental Setting

1) Hardware and Software Environment

The algorithms used in these experiments were implemented in Java 8, and were run on a workstation equipped with an Intel Xeon E5 and 64GB of RAM.

2) Datasets

For our experiments, we use two real datasets and one synthetic dataset. The first real dataset used is the deer dataset collected by the Starkey project [9] consisting of the trajectories of deer from 1993 to 1996, and obtained through radio-telemetry. This dataset has 32 trajectories and 20,000+ points. The second real dataset is the hurricane dataset [9] consisting of the trajectories of Atlantic Hurricanes occurring during the period from 1950 to 2004. This dataset contains 608 trajectories and 19,000+ points. Since the real life datasets are small, we generated a larger synthetic dataset to test the scalability of our technique. This synthetic dataset consists of 100,000 trajectories whose points sum up to 10,000,000, and was generated using moveHMM [16], which simulates animal trajectories. These trajectories correspond to movements in unconstrained spaces.

3) Dataset Preparation

Obtaining Ground Truth Data: We assume that all the trajectories in the database are the ground truth data, i.e. the trajectories in the database are considered correct with no uncertainty.

Generating Query Trajectories: Query trajectories are trajectories with uncertainty for which we want to compute estimates for its positions. To generate our query trajectories, we randomly remove high sampling rate trajectories from the database and then add Gaussian noise with distribution $N(0,\sigma^2)$ to every one of its sample points, in order to simulate measurement uncertainty. Then, to simulate the model uncertainty, a subset of the sample points is removed from these trajectories to ensure a low-sampling rate. We artificially add timestamps to all the points of all trajectories, and that timestamp added to each point is its index in its corresponding trajectory. Therefore, this scheme assumes that the trajectories in the datasets have a uniform sampling rate. For example, to generate a query trajectory with half the sampling rate of a given ground truth trajectory, we remove every other point from the latter. The resulting query trajectory is then said to have a *sampling rate multiplier* of 0.5 because its sampling rate is twice as long as that of the ground truth.

B. Evaluation Approach

Evaluation Metrics: To measure the estimation accuracy we use the EDR trajectory similarity measure [4] to determine the similarity between the trajectory suggested by our algorithm and the ground truth. The EDR trajectory similarity between two trajectories q_1 and q_2 is similar to the edit distance on strings, so that it computes the minimum number of points that have to be modified in order to transform one of the input trajectories into the other. The key difference between EDR and the edit distance lies in the matching function: in EDR two points match if they are within a distance of δ, where δ is a fixed positive real number. For calculating the EDR distance, we use δ = 20m. We see then that the larger the distance, the more dissimilar q_1 and q_2 are. Therefore, the maximum possible distance is $max(q_1.numPts, q_2.numPts)$ where $q_i.numPts$ is the number of points in the trajectory q_i. Using this, we define the *EDR match percentage* between q_1 and q_2 as $(1 - EDR(q_1, q_2)) / max(q_1.numPts, q_2.numPts)$. Hence, the EDR match percentage is a number from 0 to 1, and the higher the EDR match percentage the more similar q_1 and q_2 are.

Our second evaluation metric is the average query execution time, where we measure the time from the moment when the query starts executing until it finishes. The average execution time is taken over 30 runs of the same query.

Evaluation Experiments: We conduct extensive experiments to study the impacts of the application parameters (sampling rate, query length, noise standard deviation and dataset size) and the algorithm's parameters (acceleration tolerance and epsilon EMBR) on the performance of our algorithm. The details and the results of these experiments are reported in the next section.

C. Experimental Results

1) Impact of the Sampling Rate

We study the impact of the model uncertainty by studying the impact of the sampling rate of the points in the query trajectory as the lower the sampling rate, the higher the model uncertainty is. Figures 7a, 7e and 7i show the impacts of the sampling rate on the accuracy and execution time for the deer, hurricane and synthetic datasets, respectively. The x-axis denotes the sampling rate multiplier. To simulate different sampling rates we choose a set of query trajectories, called the *original set of query trajectories*, and then remove points from them to obtain other query trajectories with lower sampling rates. The value 1 of the sampling rate multiplier refers to the case where we use the original set of query trajectories, the value 1/2 refers to the case where we use the query trajectory set resulting from removing every other point from the original query trajectories, and the value 1/3 refers to the case where we use the query trajectory set resulting from removing every other three points from the original query trajectories, and so on. Therefore, the higher the value of the sampling rate multiplier, the higher the sampling rate. The figures show that the lower the sample rate, the lower the accuracy is. This is because a lower sampling rate implies higher model uncertainty, which in turn leads to more difficulty in accurately predicting the true path of the moving object. We observe that the accuracy seems to decrease exponentially from 98%, which can be explained because the number of points of the query trajectories also decreases exponentially. Also, the execution time decreases exponentially because the bulk of the work is proportional to the number of sampled points considered when building the linear regression models.

2) Impact of the Query Length

Figures 7b, 7f and 7j show that the length of the query trajectory does not impact the accuracy very much. This is expected because TrajEstU works by splitting the trajectory into intervals where the acceleration does not change significantly, and the number and duration of these constant-acceleration intervals is not a function of the length of the query trajectory. The execution time does, however, increase with the length of the query trajectory because, under a constant acceleration tolerance, the longer the trajectory, the greater the number of points and the greater the number of near-constant acceleration models that need to be fitted.

3) Impact of the Noise Standard Deviation

In this experiment we study the impact of the magnitude of the variance of the Gaussian noise that was artificially added to every point in the query trajectories in order to simulate measurement uncertainty. Figures 7c, 7g and 7k show that the greater the standard deviation of the measurement noise, the lower the accuracy is. This is expected because it becomes harder to accurately predict the true path of the moving object when the measurement/instrumentation

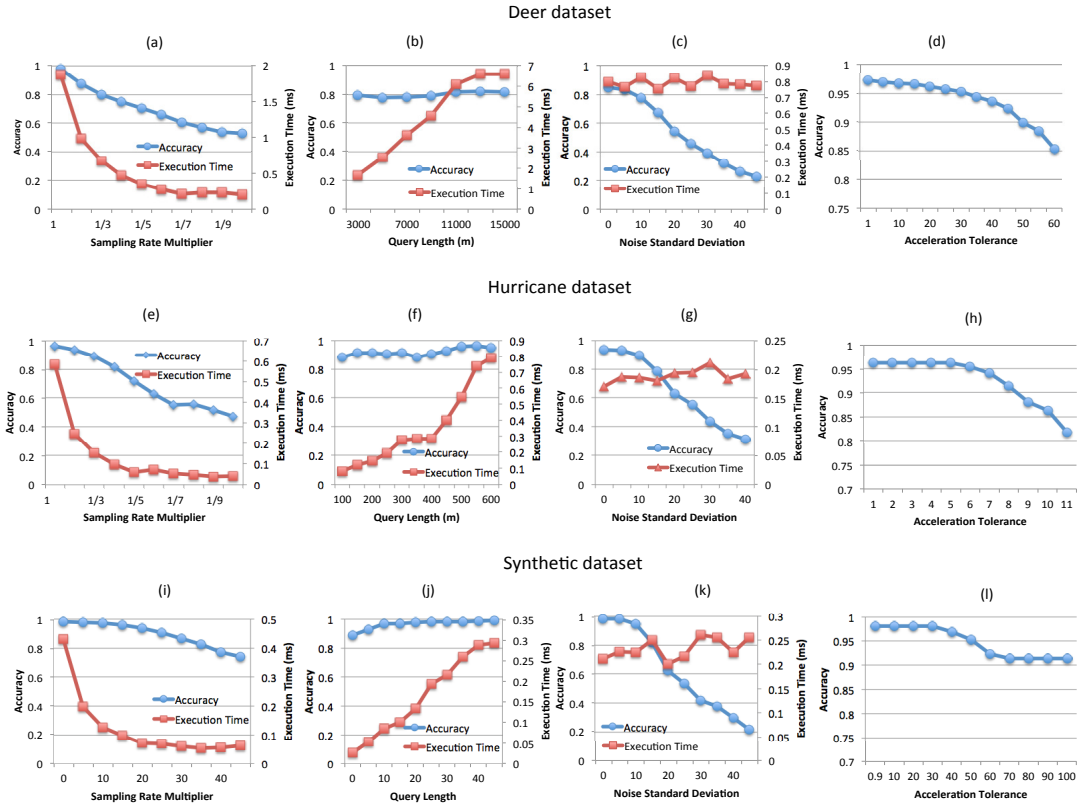

Figure 7. Experimental Results

uncertainty is higher. The figures also show that the execution time is not impacted by the noise standard deviation because when building the linear regression models, the amount of work performed by the technique is the same independently of the noise standard deviation.

4) Impact of the Acceleration Tolerance

The acceleration tolerance is used to split a trajectory into near-constant acceleration intervals. Figures 7d, 7h and 7l show that as the acceleration tolerance increases, the accuracy decreases. This is because as the acceleration tolerance is larger, then within each near-constant acceleration interval, the acceleration varies more markedly so that our linear regression model, which assumes constant acceleration, cannot adequately fit the data.

5) Impact of the Epsilon EMBR

In this experiment we study the impact of the ε parameter representing the magnitude of the extension in EMBR(ε). The experiments show that ε does not impact the accuracy. If ε is large, this means that the area explored by TrajEstU in search for trajectory clusters is large, and therefore, the technique will try to build the linear regression models using the sampled points for the clusters that are far away. However, such resulting models will likely have smaller goodness of fit, and thus will not be chosen. On the other hand, if ε is too small, then it is possible that the region explored by TrajEstU does not contain any cluster that can improve the quality of the model.

6) Impact of the Dataset Size

Despite the fact that the synthetic dataset is significantly larger than the real life ones, we observe that the average query execution time for any given experiment did not change significantly between the datasets. This is because the dataset size only impacts the pre-processing stage where we cluster the trajectories, and this stage is performed off-line. The impact of the dataset size could then only influence the average query execution time through the number of resulting trajectory clusters that need to be considered in Line 7 of Figure 6. In our three datasets, we observe that with the clustering parameters recommended by [11], the cluster density is the same so that our algorithm considers a similar amount of clusters. From Figure 7 we also observe that the accuracy did not vary much among datasets. This is expected in the deer and synthetic datasets because the latter dataset was generated using an animal movement simulator, and because the acceleration tolerance chosen led to constant acceleration intervals of about the same size; therefore, the linear regression models fitted sets of points with similar movement patterns and with about the same number of points.

VI. Conclusions and Future Work

In this paper we proposed TrajEstU, a technique for estimating the true path of moving objects in unconstrained spaces that takes into account both measurement and model uncertainty. TrajEstU works by splitting the lifetime of an object's trajectory into time intervals where the object's acceleration is nearly constant. Then TrajEstU uses the local trajectory clusters to obtain the movement patterns that are prevalent in the areas where trajectories have low-sampling rates, and linear regression to fit a kinematic constant acceleration model to the observed positions of the moving object. By using a linear regression model, TrajEstU reduces the uncertainty arising from the GPS measurements and the low-sampling rate of trajectories. The experiments on real and synthetic datasets show that TrajEstU obtained an accuracy of up to 98%, while running in the order of ms. For future work, we plan to design a top-K similarity query processing algorithm for uncertain trajectories that leverages the dynamic model built by TrajEstU, in order to determine the similarity between two uncertain trajectories.

Acknowledgment

This work is supported in part by the National Science Foundation under Grant No. 1302439 and 1302423.

References

[1] M. Buchin, S. Dodge, B. Speckmann. Similarity of trajectories taking into account geographic context. J. of Spatial Inf. Science, 2014.

[2] F. Cagnacci, L. Boitani, R. A. Powell, M. S. Boyce. Animal ecology meets GPS-based radiotelemetry: a perfect storm of opportunities and challenges. Phil. Trans. R. Soc. B, 2010.

[3] L. Chen, R. Ng. On the marriage of lp-norms and edit distance. VLDB 2004.

[4] L. Chen, M.T. Ozsu, V.Oria. Robust and fast similarity search for moving object trajectories. SIGMOD 2005.

[5] H. Cao, O. Wolfson, G. Trajcevski. Spatio-temporal data reduction with deterministic error bounds, *The VLDB Journal*, 15(3), 2006.

[6] T. Emrich, H. Kriegel, N. Mamoulis, M. Renz, A. Zuefle. Querying Uncertain Spatio-Temporal Data. ICDE 2012.

[7] S. Gupta, R. Manchanda. TANH Spline Interpolation for Analytical Modelling of BK Ion Channels in Smooth Muscle. Asia Modelling Symposium, 2014.

[8] J. S. Horne, E. O. Garton, S. M. Krone, J. S. Lewis. Analyzing animal movements using brownian bridges. *Ecology*, 88(9), 2007.

[9] J. Lee, J. Han, K. Whang. Trajectory Clustering: A Partition-and-Group Framework. SIGMOD 2007.

[10] M. Lewis, S. Lakshmivarahan, S.K. Dhall. Dynamic Data Assimilation: A least squares approach. Cambrid. Univ. Press, 2006.

[11] Z. Li, J. Lee, X. Li, J. Han. Incremental Clustering for Trajectories. DASFAA 2010.

[12] C. Ma, and H. Lu. KSQ: Top-K Similarity Query on Uncertain Trajectories. TKDE 2013.

[13] S. Ranu, P. Deepak, A. Telang, P. Deshpande, S. Raghavan. Indexing and matching trajectories under inconsistent sampling rates. ICDE 2015.

[14] Y. Zheng, X. Xie, W.Y. Ma. Geolife: A collaborative social networking service among user, location and trajectory. *IEEE Database Engineering Bulletin*, 2010.

[15] K. Zheng, Y. Zheng, X. Xie, X. Zhou. Reducing Uncertainty of Low-Sampling Rate Trajectories. ICDE 2012.

[16] T. Michelot, R. Langrock, T. A. Patterson. moveHMM: an R package for the statistical modelling of animal movement data using hidden markov models. Methods in Ecology and Evolution, 2016.

[17] L. Cao, J. Krumm. From GPS Traces to a Routable Road Map. ACM GIS 2009.