TrajPattern: Mining Sequential Patterns from Imprecise Trajectories of Mobile Objects

Jiong Yang and Meng Hu

EECS, Case Western Reserve University
jiong.yang@case.edu, meng.hu@case.edu

Abstract. Mobile objects have become ubiquitous in our everyday lives, ranging from cellular phones to sensors, therefore, analyzing and mining mobile data becomes an interesting problem with great practical importance. For instance, by finding trajectory patterns of the mobile clients, the mobile communication network can allocate resources more efficiently. However, due to the limited power of the mobile devices, we are only able to obtain the imprecise location of a mobile object at a given time. Sequential patterns are a popular data mining model. By applying the sequential pattern model on the set of imprecise trajectories of the mobile objects, we may uncover important information or further our understanding of the inherent characteristics of the mobile objects, e.g., constructing a classifier based on the discovered patterns or using the patterns to improve the accuracy of location prediction. Since the input data is highly imprecise, it may not be possible to directly apply any existing sequential pattern discovery algorithm to the problem in this paper. Thus, we propose the model of the trajectory patterns and a novel measure to represent the expected occurrences of a pattern in a set of imprecise trajectories. The concept of pattern groups is introduced to present the trajectory patterns in a concise manner. Since the Apriori property no longer holds on the trajectory patterns, a new **min-max** property is identified and a novel TrajPattern algorithm is devised based on the newly discovered property. Last but not least, we apply the TrajPattern algorithm on a wide range of real and synthetic data sets to demonstrate the usefulness, efficiency, and scalability of this approach.

1 Introduction

Mobile devices have been widely used in our everyday life, from handheld devices, e.g., PDA, to embedded devices, e.g., sensors. The trend is expected to intensify in the coming years. It is projected that in the next few years, all Hertz rental cars will be equipped with global positioning systems (GPS). One may infer important information from the trajectories of mobile objects. Most of the recent research effort has been concentrated in modeling the trajectory of mobile objects [2, 10, 11, 12] and indexing mobile objects [7, 9]. However, mining mobile data has received little attention so far. In this paper, we investigate the problem of mining and analyzing trajectories of moving objects. The following is a list of applications of analyzing the mobile trajectory data.

– Due to the limited power on the mobile devices and the unreliable communication links, we may want to infer the location of a mobile object based on its previous

locations. If we can find some moving patterns that are common to a large set of mobile objects, then these moving patterns may be useful for predicting the locations of an object in the future.
– In location-based commerce advertisement, if customers are willing to receive advertisements, retail stores will distribute e-Flyers to potential customers' mobile devices based on their locations. In this setting, finding common moving patterns of mobile devices is valuable for inferring potential movement of mobile device users, and thus helps to efficiently distribute the advertisement.
– Using a remote sensing system, the animals in a large farming area can be tracked. The sensors are limited in power and may fail from time to time. By mining the imprecise trajectories of animals, it is possible to determine migration patterns of certain animal or groups of animals. These patterns could be useful to analyze the migration behavior of different species of animals.

The energy in a mobile device is very limited, so it is impossible for a mobile object to continuously send out its location information. To reduce the energy consumption, many methods [2, 11, 12] are developed for obtaining (predicting) the approximate location of a mobile object. At a high level, all of these methods share the same principle. These methods first use some predictive model, e.g., Kalman Filter, linear model, etc., to predict an expected location of a mobile object at a given time t. If the actual location of the mobile object differs too much from the predicted location, then the mobile object reports the new location. Otherwise, it does not report the new location.

In this paper, our aim is not to develop a data mining approach which depends on a particular prediction model, but rather develop a general data mining framework that can be applied to a large number of existing location prediction methods. In the data mining field, there exist a large number of different models, from association to classification. Among these models, frequent patterns are one of the most basic and widely employed models. In addition, most of the previous proposed location prediction models for mobile objects assume one type of movement, e.g., linear, quadratic, etc. However, the type of movement for a mobile object may not be known ahead. Moreover, a mobile object may change the type of movement at any time. Therefore, the accuracy of these models might not be high. The frequent patterns may help to improve the accuracy of the prediction module. If an object follows some moving patterns, e.g., an object always changes its velocity or directions after it moves in a certain manner, then this knowledge can be integrated into the location prediction module and the location prediction can be adjusted accordingly. Looking ahead, we will show the usefulness of the frequent patterns of the imprecise trajectories on real data sets via the location prediction.

The traditional frequent pattern models and approaches could not be directly applied to the trajectories due to their imprecise nature. In the traditional frequent sequential pattern setting, the sequences are synchronized and we know exactly the occurrences of the symbol or values at every synchronized point. In this paper, a series of synchronization points can be superimposed on the trajectories. The interpolated values (at synchronization points) can be taken as the input for the frequent pattern mining process. In many applications, it is more useful to find patterns on the velocities rather than the locations. In such a case, we can transform the location trajectories (sequences) into velocity trajectories (sequences). Thus, our frequent pattern model can be constructed on a set of either location or velocity trajectories.

A sequential pattern is an ordered list of symbols. It is intuitive and easy for a user to comprehend, thus we also use the sequential pattern model in this paper. A trajectory pattern is an ordered list of positions. For instance, a pattern $(p_1, p_2, \ldots, p_m)$ can be considered as the possible positions of an object at m consecutive snapshots. The support model is usually used to measure the importance of a pattern, i.e., if a pattern occurs a large number of times, then it is an important pattern. However, in the context of the imprecise trajectories, at any given moment, the location of an object in a trajectory is not precise, but rather a distribution of possible locations. Thus, we do not know for sure whether a pattern occurs or not. This could be a very challenging issue for formulating the frequent sequential pattern model. In this paper, we propose the normalized match (NM) measure to capture the importance of a trajectory pattern. We show the benefits of the NM measure over other measures in the experimental results section.

Most previous frequent sequential pattern algorithms utilize the Apriori property. However, the Apriori property does not hold for our NM measure. Fortunately, we are able to identify another property, called **min-max** property, which is weaker than the Apriori property. Thus it is necessary for us to devise a new algorithm for mining the NM patterns. Based on the min-max property, we develop a trajectory pattern mining algorithm called **TrajPattern**. The user will specify k, the number of trajectory patterns that he wants. Our goal is to mine the k patterns with the most NM. Due to the presence of noise in the trajectories, many similar patterns may be found in the mining process. The concept of **pattern groups** is introduced to compactly represent a large number of similar trajectory patterns via a small number of groups. The TrajPattern algorithm mines the patterns by a *growing* process. We first identify short patterns with high NM value, and then try to extend these short patterns to find longer patterns with high NM via the min-max property. With the min-max property, a novel pruning method is devised to reduce the number of candidate patterns, thus the efficiency of the mining algorithm can be greatly improved. In addition, the TrajPattern algorithm can be used for mining any type of patterns satisfying the min-max property.

The remainder of this paper is organized as follows. We briefly describe some related work at Section 2. The problem model is presented in Section 3. The TrajPattern algorithm is discussed in Section 4. Additional issues of the trajectory pattern model and algorithm are discussed in Section 5. The experimental results are shown in Section 6. Finally, we draw our conclusions in Section 7.

2 Related Work

There is a large amount of work in location modeling and prediction. In [2] the Kalman Filter is used to predict the location of a mobile object at a given time while in [11], the authors used not only a single previous location, but rather multiple locations to predict the current location. Authors in [12] assumed that the object moves in a piece-wise linear manner. Thus the location of an object can be predicted by its previous locations and velocities.

Data mining has been an active research area in the past decade. Many data mining models and approaches have been proposed. However, there is limited work on spatiotemporal data mining. In [5], the proposed algorithm treated spatiotemporal data as a generalization of pattern mining in time-series data to capture the frequent moving patterns of users from a set of log data in a mobile environment. Authors in [6] processed moving nearest-neighbor queries in R-trees by employing sampling. In [9] the TPR-tree is presented as an extension of the R-tree to answer prediction queries on dynamic objects. Very recently, researchers began to study the problem of mining the trajectories of mobile objects. In [3] the authors proposed a method on clustering the locations of mobile objects continuously. It groups nearby objects into small micro-clusters and each micro-cluster is treated as an entity so that the computation time can be saved. The authors of [4] proposed a method to find periodic patterns for trajectories of mobile objects. This work aims to find the periodic moving patterns in the history of one object. In addition, all above works also assume that the input data is a sequence of precise locations, which is quite different from our assumption that the locations of objects are imprecise. Therefore the support measure can be used to qualify the importance of patterns in [4], but it could not be applied to our problem. To the best of our knowledge, we are the first to tackle the problem of mining imprecise trajectories.

Sequential patterns has been an active research topic in recent years, and many sequential pattern mining models and approaches have been proposed. One category of sequential patterns is the periodic patterns [1, 4] which repeat themselves over the time. Another category is the frequent sequential patterns [8, 13, 14, 15], which is more related to the problem in this paper. A frequent sequential pattern is a pattern which occurs at a large number of sequences. Several models and approaches have been proposed for this problem. In [8] the authors used the prefix-tree to maintain the set of prefixes of frequent patterns and later grow the set of patterns. The author of [15] designed an efficient algorithm for mining frequent subsequences in a long sequence. Both above approaches assume that the symbol in each position is accurate and the Apriori Property is used for devising efficient algorithms.

In [14] the authors studied the problem that symbols in a set of sequences are not accurate and may mutate due to noise. There is a mutation matrix which shows the probability that a symbol a may mutate to b in the input data sequence. The match model is invented for representing the true (or expected) occurrences of a pattern. The match of a pattern within a sequence is the joint probability of the occurrence of the symbols in the pattern. The match value of a pattern is not normalized according to its length. As a result, the non-normalized match of a longer pattern is smaller. This property is not desirable in many applications where longer patterns are needed since longer patterns usually consist of more information. Since the Apriori property holds for the match measure, the authors in [14] devised an algorithm based on the Apriori property, which could not be directly applied to the problem in this paper.

3 Preliminaries and Problem Statement

In this paper, we study the problem of identifying sequential patterns of imprecise trajectories of mobile devices. We assume that there is a server and a set of mobile devices. The mobile devices have the capability to know their own locations (e.g. via GPS) and they asynchronously report their locations to the server via some wireless network.

3.1 Location Reporting Scheme

A mobile object may choose not to notify the server its current location for a long time when the location can be derived from its previous locations, speed, directions, etc. There are many methods (e.g., [2, 11, 12], etc.) for a server to predict the location of a mobile device. Our aim is to develop a general pattern mining framework that can be used with various different location inference models, so we only require that the location prediction method has the following property. At any given time t, each mobile object has a predicted location. The actual location of the mobile object follows a certain distribution around the predicted location.

Most of the proposed location inference techniques satisfy the above property. Without loss of generality, we choose the method proposed in [12] as an example to demonstrate our problem model and solution. For a given device o, let $last\_loc$ be the last known location of the object and v be the velocity vector of the object. The predicted position of o ($predict\_loc$) is defined as follows:

$$predict\_loc = last\_loc + v \times t \tag{1}$$

Here, t is the number of time units that have elapsed since the last known position of o. Since this is only a prediction, the actual position of o may vary from the predicted position. It is assumed that the actual position of o follows the k-dimensional multivariate normal distribution $N_k(\mu, \Sigma)$ where k is equal to the dimension of the space, the mean $\mu = predict\_loc$, and Σ is the variance-covariance matrix. The variance-covariance matrix is a symmetric $k \times k$ matrix with diagonal elements σ^2 equal to the variance of the marginal distributions. σ is defined as $\frac{1}{c}U$ where U is the tolerable uncertainty distance of the object and c is a constant. A mobile object may choose to report its actual location only if it is more than U away from the predicted position μ. There are several ways to assess the parameters U and c. U can be either a constant, a function of the elapse time t, or the expected traversed distance d. In this paper, we assume that U is a constant so that all objects in the database have the same uncertainty. This assumption has been practically used since it is difficult to find the uncertainty for each object. c is a constant which may depend on the network reliability, etc. With the greater c, the probability that the actual location close to the predicted location is higher. For instance, a mobile device is within U distance from μ with probability 0.68, 0.95 and 0.997 for $c = 1, 2, 3$, respectively. Since there may be an error during the communication between the mobile object and the server, the location information may be lost during the transmission. If there exists a 5% chance that the message will be lost during the location notification, then c should be set to 2 so that the probability that the actual location is more than U away from μ is equal to 5%.

3.2 Location and Velocity Trajectories

To provide a consistent view of all objects, a set of synchronous snapshots are generated on the server. A series of synchronization points can be superimposed on the asynchronous data. The interpolated values (at synchronization points) can be taken as the input to the data mining modules. Let's assume that we generate a snapshot at time point t. For every object we could calculate its locations at t via some prediction method.

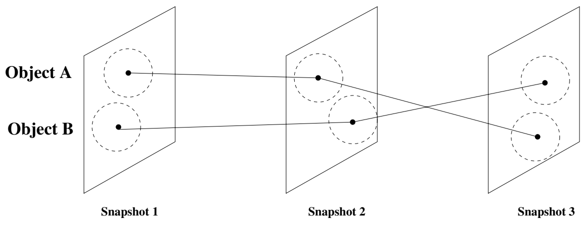

Fig. 1. Location Trajectories

For instance, we can apply Equation 1 to compute the expected location of an object based on the last reported location before t. Based on this model, at each snapshot, every object will have an expected location and a distribution of errors. The frequency of the snapshots may vary in different applications. We will discuss how to choose this parameter in a later section.

The locations of a mobile object o at each snapshot can form a sequence T. We call T the *location trajectory* of object o. Here $T = (l_1, \sigma_1), (l_2, \sigma_2), \ldots$ where l_i and σ_i are the mean and standard deviation of the distribution of the true location of o at ith snapshot, respectively. l_i and σ_i can be calculated via Equation 1. Figure 1 shows two location trajectories.

In many applications, two mobile objects may travel in different regions of space. Thus these two location trajectories could not be compared directly. On the other hand, the velocities may be more important. In these applications, we need to transform the location trajectories to a sequence of velocities, or *velocity trajectories*. This can be achieved by taking the difference between two consecutive snapshots. For example, let $T = (l_1, \sigma_1), (l_2, \sigma_2), (l_3, \sigma_3), \ldots$ be a location trajectory where l_i and σ_i are the expected location and the standard deviation of the mobile object at the ith snapshot. The velocity trajectory is generated as follows. We consider the location of a mobile object at ith and $i+1$th snapshot as two random variables of normal distribution with mean l_i and l_{i+1} and standard deviation σ_i and σ_{i+1}. The difference of these two random variables can be considered as the velocity of the mobile object at ith snapshot. The difference is also a normal distribution random variable where the mean is $l_{i+1} - l_i$ and the standard deviation is $\sqrt{\sigma_i^2 + \sigma_{i+1}^2}$. (A slightly more complicated formula can be used to compute the standard deviation if the two random variables are not independent.) Thus the new velocity trajectory T' is in the following form: $T' = (l'_1, \sigma'_1), (l'_2, \sigma'_2), (l'_3, \sigma'_3), \ldots$ where $l'_i = l_{i+1} - l_i$ and $\sigma'_i = \sqrt{\sigma_i^2 + \sigma_{i+1}^2}$. It is obvious that the transformed velocity trajectories are in the same form as the original location trajectories. Thus, we call both the velocity trajectories and location trajectories as *trajectories*. In this paper, we assume that the input data is a set of trajectories, each of which is in the form of T.

3.3 Model of Trajectory Pattern

A trajectory pattern P can be represented as $P = (p_1, p_2, \ldots, p_m)$ where p_i is a location. P can be interpreted as the following: the mobile object is located at p_1, p_2, $\ldots$, and p_m at m consecutive snapshots. The **length** of a pattern P is the number of positions in P, which is m in this example. We call a pattern of length 1 as a *singular* *pattern*. In theory, the space in which the objects travel is continuous, which means that there are infinite possible choices for a position in a pattern. To expedite the mining process, we discretize the space into small regions and only the centers of these regions may serve as the positions in a pattern. Let G_x, G_y be the grid size on a 2-dimensional space. As long as G_x, G_y are sufficiently small, our model will provide a very good approximation.

The support model has been used to measure the importance of a pattern in many applications [8, 15]. According to the traditional support model, we may define the support of a trajectory pattern as follows. A trajectory sequence T supports a pattern if there exists a consecutive segment $((l_k, \sigma_k), (l_{k+1}, \sigma_{k+1}), \ldots, (l_{k+m-1}, \sigma_{k+m-1}))$ such that l_{k+i-1} is equal to p_i for $1 \leq i \leq m$. However, in the context of this problem, the support model may not work well due to the presence of noises. The degradation of quality of the data may conceal the real frequent patterns. The spirit of the support model is to find frequently occurred patterns. Due to the uncertainty (which is described as a probabilistic function), we have to find expected frequently occurred patterns instead. In [14] the match model is proposed to measure the expected number of occurrences of a pattern. Intuitively, the match model computes the expectation on how likely a pattern occurs in a trajectory or the degree that a trajectory confirms (supports) a trajectory pattern.

Let δ be the indifferent parameter such that for any coordinate (x, y), if an object o is at most δ away from (x, y), then the location of o is considered indifferent from (x, y). With the indifferent parameter, we can define the match of a trajectory pattern as the following. If at a snapshot the expected location of an object is l with standard deviation σ, the probability that the true location of the object is within δ away from another location p is denoted as $Prob(l, \sigma, p, \delta)$, which represents how likely the object is truly very close to a position p. Let $T' = ((l_k, \sigma_k), (l_{k+1}, \sigma_{k+1}), \ldots, (l_{k+m-1}, \sigma_{k+m-1}))$ be a contiguous segment of a trajectory sequence and $P = (p_1, p_2, \ldots, p_m)$ be a trajectory pattern, the probability that for every $1 \leq i \leq m$, the true location of the mobile object is located within at most δ away from p_i is[1]

$$M(P, T') = Prob(P, T') = \Pi_{i=1}^{m} Prob(l_{k+i-1}, \sigma_{k+i-1}, p_i, \delta) \tag{2}$$

We call $M(P, T')$ the *match* between a pattern P and a trajectory T' of the same length. This is essentially the same measure as in [14].

Based on the definition of *match*, the value of *match* monotonously decreases with the growth of pattern length m. For example, if the probabilities of observing symbol a, b, and c at position 1, 2, and 3 are all 0.9, then the joint probability of (a, b) is 0.81 while the joint probability of (a, b, c) is 0.729. In this case, only short patterns can be found and the measurement can not be compared between patterns of different lengths. To normalize this effect, we choose the geometric mean to denote the match between T' and P, which is $(M(P, T'))^{\frac{1}{m}}$. To speed up the computation we use the logarithmic value to present the match between a trajectory and a pattern with the same length, i.e.,

$$NM(P, T') = \log M^{1/m}(P, T') = \frac{\log M(P, T')}{m}. \tag{3}$$

[1] We assume that the error in location prediction in T' is independent, but the locations of the mobile objects in T' are not assumed independent.

We call $NM(P, T')$ the *normalized match* (NM) of a pattern P with a trajectory T' of the same length. In reality, the length of a trajectory T is usually much longer than that of a pattern P (with m locations). Thus, we use the maximum NM between any continuous segment of m locations in T and P as the NM between T and P. Formally, the NM between T and P is defined as follows.

$$NM(P, T) = \max_{\forall T' \subseteq T, |T'| = |P|} NM(P, T') \tag{4}$$

In a data set $\mathcal{D}$, the NM of a pattern P is equal to $\sum_{T \in \mathcal{D}} NM(T, P)$, i.e., the sum of NM between P and each trajectory in $\mathcal{D}$. Here, the NM between P and a trajectory actually represents how likely the pattern occurs in the trajectory. The sum of NM measures the expected occurrence of the pattern in a trajectory set. This is essentially based on the same intuition that, in traditional frequent patterns, the support of a pattern is defined as the total number of exact occurrences in a data set. The match measure can be defined similarly. The Apriori property holds on the match measure, but not on the NM measure, because the NM is normalized according to the pattern length. Thus, the algorithm proposed in [14] can only be applied for mining patterns according to the match measure. We need to develop algorithms to mine the patterns according to the NM measure.

3.4 Definition of Pattern Group

Since the trajectories are imprecise, many mined trajectory patterns are very similar. At each snapshot of trajectory, the true location of the moving object follows a normal distribution where the mean is the expected location of the object. Therefore, due to the bell shape of the normal distribution, the probabilities that the true location of the object falls into two adjacent grids could be similar. As a result, the NM of two patterns consisting of nearby grids could be similar.

The pattern group is a concept which helps to compactly present the results of imprecise trajectory mining, in which many patterns are similar to each other. The similar patterns can be clustered into a small number of groups. Intuitively, similar patterns should be close to each other at any snapshot. The similar relation of patterns and the concept of pattern group are formally defined as below:

Definition 1. *Given two patterns of the same length, if at every snapshot of the patterns, the distance between the two patterns is no larger than a pre-defined value γ, we say that these two patterns are* **similar patterns**.

γ is called the maximum similar pattern distance. How to set this parameter is discussed in a later section.

Definition 2. *A* **pattern group** *is a set of patterns, which contains the maximum number of patterns that are similar to each other.*

Problem Statement

In this paper, we try to solve the following problem. For a given set of imprecise trajectories, we want to find k patterns with the most normalized match. These qualified patterns are represented via the concept of pattern groups.

3.5 Properties of Trajectory Patterns

As discussed above, the NM measure does not possess the Apriori property. As a result, many algorithms that utilize the Apriori property could not be applied here. However, the NM of trajectory patterns exhibit the following property which can be used to facilitate the mining process. Before stating the property, we first define some terms that will be used in the remainder of this paper.

Definition 3. *Let $P = (p_1, p_2, \ldots, p_m)$ and $P' = (p'_1, p'_2, \ldots, p'_n)$ be two trajectory patterns. P is a* **super-pattern** *of P' iff there exists an integer $i \geq 0$ such that for all $1 \leq j \leq n$, $p_{i+j} = p'_j$. In addition, P is called a* **proper super-pattern** *of P' if $m > n$.*

For example, let $P = (p_1, p_2, p_3)$ and $P' = (p_2, p_3)$. We call P a super-pattern or proper super-pattern of P'. On the other hand, we also call P' a sub-pattern or proper sub-pattern of P.

Definition 4. *A trajectory pattern P is called an i-trajectory pattern (or $i - pattern$ for short) if there are i positions specified in P, i.e., $P = (p_1, p_2, \ldots, p_i)$.*

Property 1. Given two trajectory patterns $P' = (p'_1, p'_2, \ldots, p'_i)$ and $P'' = (p''_1, p''_2, \ldots, p''_j)$. Let $P = (p'_1, p'_2, \ldots, p'_i, p''_1, \ldots, p''_j)$ be the trajectory pattern by appending P'' to the end of P'. Within a given set of trajectories $\mathcal{D}$, $NM(P) \leq \max(NM(P'), NM(P''))$. We call it the **min-max** property.

Proof. For each trajectory $T \in \mathcal{D}$, there exists a sub-trajectory T', where $|T'| = |P|$ and $NM(P,T) = NM(P,T')$. By definition, we have $(i + j) \times NM(P,T') \leq i \times NM(P',T') + j \times NM(P'',T') \leq i \times NM(P',T) + j \times NM(P'',T)$. Thus, $(i + j) \times NM(P) = \sum_{T \in \mathcal{D}} (i + j) \times NM(P,T) \leq \sum_{T \in \mathcal{D}} i \times NM(P',T) + \sum_{T \in \mathcal{D}} j \times NM(P'',T) = i \times NM(P') + j \times NM(P'')$. As a result, $NM(P) \leq \max(NM(P'), NM(P''))$.

Note that the above min-max property is very different from the Apriori property. The Apriori property states that the support of a pattern is less than or equal to any of its sub-patterns, while the min-max property is much looser. For each partition of a pattern P, we have two portions (sub-patterns) P_{left} and P_{right}. The min-max property requires that the NM of P is less than or equal to either P_{left} or P_{right}. The algorithms developed for mining the patterns satisfying the Apriori property may not be directly applied to the trajectory patterns with NM. As a result, it is necessary to develop a new algorithm for mining NM patterns.

4 TrajPattern Algorithm

In this section, we present the TrajPattern algorithm to mine the k trajectory patterns with the most normalized match (NM), and cluster these patterns into pattern groups. The following observations are used for the mining process.

1. The length of the discovered trajectory patterns is usually much shorter than the length of the trajectory. A trajectory could contain thousands of snapshots while a

qualified trajectory pattern often has much less positions, e.g., tens. Based on this observation, it is reasonable to start the search process from the short patterns, and grow to longer patterns.

2. Our goal is to find the k patterns with the most NM. If we know the NM threshold ω, then this threshold can be used for pruning the search space. Unfortunately, we do not know ω. However, if we find a set of patterns $\mathcal{Q}$, then the NM threshold ω should be greater than or equal to the kth maximum NM of the patterns in $\mathcal{Q}$. Based on this observation, we can dynamically maintain a set of patterns $\mathcal{Q}$, and the NM threshold ω should be the kth maximum NM of the patterns in $\mathcal{Q}$. With more patterns discovered, we can update the threshold ω, which could increase the pruning power.
3. Based on the min-max property, if a pattern P_1 is below a NM threshold ω, then in order to find a super-pattern $P = (P_1, P_2)$ such that $NM(P) \geq \omega$, the NM of the pattern P_2 has to be greater than or equal to ω. As a result, if the NM of a pattern P is below ω, then P will only be combined with patterns whose NM is at least ω to generate the candidate patterns. Thus, we may consider the set of patterns with NM at least ω as the seeds for generating the candidate patterns.

Based on the previous observations, we devise an algorithm called *TrajPattern* to mine the set of k trajectories with the most NM. We first partition the space into grids, and the grid centers serve as the singular patterns. Then we initialize the set $\mathcal{Q}$ to include all these singular patterns and set the NM threshold ω to be the kth maximum NM of patterns in $\mathcal{Q}$. The set of patterns in $\mathcal{Q}$ with NM lower than ω is marked as *low* patterns and denoted as $\mathcal{L}$ while the set of patterns in Q with NM greater than or equal to ω is labeled as *high* patterns and denoted as $\mathcal{H}$. We can generate the candidate patterns from the set of high patterns as follows. For each high pattern $P \in \mathcal{H}$, we extend P by adding each pattern $P' \in \mathcal{Q}$. Note that P' may be a high pattern or a low pattern. Let $P = (p_1, p_2, \ldots, p_m)$ and $P' = (p'_1, p'_2, \ldots, p'_l)$. Two candidate patterns $(p_1, \ldots, p_m, p'_1, \ldots, p'_l)$ and $(p'_1, \ldots, p'_l, p_1, \ldots, p_m)$ will be generated. The NM of these candidate patterns are computed and these newly generated patterns are inserted into $\mathcal{Q}$. Based on these patterns, we can update the threshold ω and mark all patterns as high or low according to the new threshold ω. Then, patterns in $\mathcal{Q}$ are pruned to reduce the cardinality and improve the efficiency (The detail is explain of the pruning step later.) The mining process terminates when the set of high patterns does not change during the last iteration. Lastly, pattern groups are discovered from the set of high patterns. The formal description of the TrajPattern algorithm can be found in [17].

4.1 Pruning

In the TrajPattern algorithm, the main problem is the size of $\mathcal{Q}$. If it is too large, then the algorithm would be very inefficient. During an iteration, the size of $\mathcal{Q}$ increases by $2k$ fold. Without any pruning, $\mathcal{Q}$ would grow to $2k^iG$ after ith iteration where G is the number of grids in the space. This could be too large. In order to provide an efficient algorithm, it is necessary to reduce the size of $\mathcal{Q}$. Fortunately, we can prune $\mathcal{Q}$ based on the following observation. We only need to keep the set of low patterns satisfying the following *1-extension* property.

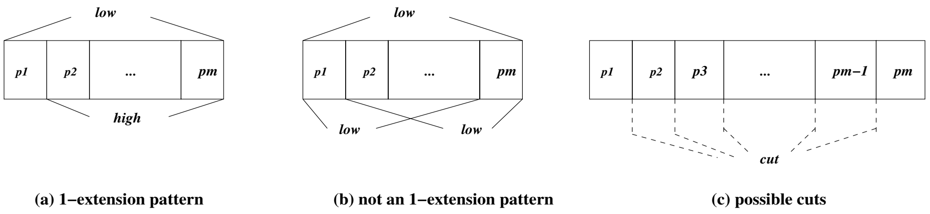

Fig. 2. 1-extension patterns

Definition 5. *(1) For a j-pattern($j > 1$) P, if there exists a $(j-1)$-pattern P' which is the proper sub-pattern of P and P' is a high pattern, then we say that P satisfies the 1-extension property. (2) Any 1-pattern satisfies the 1-extension property.*

For example, the pattern in Figure 2(a) can be viewed as a pattern satisfying the 1-extension property while the pattern in Figure 2(b) does not satisfy the 1-extension property. The reason that we only need to retain the set of low patterns satisfying the 1-extension property is due to the following lemma.

Lemma 1. *Any high pattern P can be obtained by extending a high pattern P' with either a high pattern or a low pattern P'' satisfying the 1-extension property.*

Proof. Let's consider the high pattern $P = (p_1, p_2, \ldots, p_m)$ shown in Figure 2(c). A "cut" partitions P into two non-overlapping complementary patterns P_{left} and P_{right} where P_{left} and P_{right} contains the sub-patterns left and right to the cut respectively. Assume the cut is made at the end of the first position of P, then $P_{left} = (p_1)$ and $P_{right} = (p_2, \ldots, p_m)$. There are three cases. (1) Both P_{left} and P_{right} are high patterns, then the lemma holds. (2) P_{left} is low while P_{right} is high. Since P_{left} is a 1-pattern (i.e., 1-extension pattern), then the lemma also holds. (3) P_{left} is high and P_{right} is low. In this case, we move the cut from the left to the right one position at a time. If P_{left} is always high with respect to all cuts, then this lemma also holds because when the cut is at the $(m-1)$th position, P_{right} is of length 1 (a 1-extension pattern). Let's assume that there exists a position $1 < i \leq m-1$, such that $(p_1, p_2, \ldots, p_i)$ is high and $(p_1, p_2, \ldots, p_i, p_{i+1})$ is low. This means that $(p_{i+2}, \ldots, p_m)$ is a high pattern by the min-max property. In addition, $(p_1, p_2, \ldots, p_{i+1})$ is a 1-extension pattern by the definition. Thus the lemma holds.

Armed with the above lemma, we can remove all low patterns that do not satisfy the 1-extension property. In the $Prune$ procedure, for each low pattern P, we examine whether P is an 1-extension pattern. This can be achieved by removing either the first or the last position in P and search whether the resulting pattern exists in H_{new}. If it exists, then P is an 1-extension pattern and it will remain in Q, otherwise, it is removed.

4.2 Pattern Groups Discovery

After obtaining the top k patterns, we first group these qualified patterns by their lengths, then cluster the patterns of the same length into pattern groups. The clustering process can be conducted in the following way. First, the patterns are clustered at each snapshot based on their distances. We refer to these clusters as snapshot groups. If any pattern is clustered into a single snapshot group at certain snapshot, according to the definition of pattern group, this pattern should be in a single pattern group. Then this pattern is removed from all remaining snapshot groups. Next, we start from the smallest snapshot group at all snapshots, denoted as G, to check whether G exists at other snapshots. If so, patterns in G are qualified as a pattern group at all snapshots, and should be removed from all remaining snapshot groups. If G does not exist at other snapshots, we find the snapshot group at other snapshots G', which makes $G \cap G'$ has the minimum number of patterns. We continue to check whether $G \cap G'$ exists at other snapshots, until we find a proper pattern group. This process continues until all patterns are grouped.

For example, assume we have six patterns of length two: $P_1 = (p_1, p'_1)$, $P_2 = (p_2, p'_2)$, $P_3 = (p_3, p'_3)$, $P_4 = (p_4, p'_4)$, $P_5 = (p_5, p'_5)$ and $P_6 = (p_6, p'_6)$. We cluster these six patterns according to their locations at the two snapshots. Assume that at the first snapshot we have snapshot groups (p_1, p_3, p_4, p_5) and (p_2, p_6); at the second snapshot we have snapshot groups (p'_1, p'_3, p'_6), (p'_2, p'_4) and (p'_5). We start with the snapshot group containing only one pattern, which is (p'_5) at the second snapshot. Then P_5 is assigned into a single pattern group and we remove P_5 from all remaining snapshot groups. After this step, (p_1, p_3, p_4) and (p_2, p_6) remain for the first snapshot, while (p'_1, p'_3, p'_6) and (p'_2, p'_4) remain for the second snapshot. Now the smallest snapshot group is (p_2, p_6). Since this snapshot group does not exist at the second snapshot, we find the smallest subset of (p_2, p_6) contained in any snapshot groups at the second snapshot, which is either P_2 or P_6. For the same reason as P_5, P_2 and P_6 are assigned into single pattern groups separately. After removing P_2 and P_6, P_4 is also assigned into a single pattern group. Now (p_1, p_3) and (p'_1, p'_3) remain for both snapshots, and (P_1, P_3) is qualified as a pattern group. Thus the final pattern groups are (P_2), (P_4), (P_5), (P_6), and (P_1, P_3).

4.3 Correctness Analysis

In this subsection we show the correctness of the TrajPattern algorithm.

Theorem 1. *Let $\mathcal{H}_{new}$ be the set of high patterns in $\mathcal{Q}$ when the TrajPattern algorithm terminates and $\mathcal{K}$ be the set of k patterns with the highest NM. Then $\mathcal{Q} = \mathcal{K}$.*

Proof. Since the cardinality of $\mathcal{H}_{new}$ and $\mathcal{K}$ is the same, ie., k, we only need to prove $\mathcal{K} \subseteq \mathcal{H}_{new}$. Let $P_i \in \mathcal{K}$ be the pattern of length i. We prove via induction that P_i is also in $\mathcal{H}_{new}$. First, when $i = 1$, P_i is a singular pattern. This pattern will be generated in $\mathcal{Q}$ at the beginning and thus $P_1 \in \mathcal{H}_{new}$. Assume that for each $i \leq m$, any pattern P_i is in $\mathcal{H}_{new}$ where m is a positive integer. For a pattern P_{m+1} there exists a proper subpattern P_m of P_{m+1} and $M(P_m) \geq M(P_{m+1})$ by the min-max property. P_{m+1} can be obtained via extending the high pattern P_m by adding an 1-extension pattern (a singular pattern). Thus at the latest P_{m+1} will be inserted into $\mathcal{H}_{new}$ after P_m is inserted into $\mathcal{H}_{new}$. Therefore P_{m+1} will be in $\mathcal{H}_{new}$ by the end of the TrajPattern algorithm.

4.4 Complexity Analysis

To analyze the complexity of the algorithm we need to determine the number of iterations executed by TrajPattern. For the same reason as in the previous proof, by the ith iteration, all high patterns with length less than or equal to i is inserted in $\mathcal{H}$. Therefore the max number of iterations is $O(M)$ where M is the maximum length of the pattern with top k NM.

Second, we need to analyze the number of patterns in $\mathcal{Q}$. $\mathcal{Q}$ consists of two types of patterns: high patterns and low patterns. The low patterns are the 1-extension patterns. Let G be the number of grids in the space. Each high pattern P can generate at most $2G$ low 1-extension patterns by extending one position before the first or after the last position. Therefore, we have at most $(2G|\mathcal{H}| + G)$ low patterns, which is $O(kG)$. During the candidate pattern generation phase there are a total of $O(k^2G)$ candidate patterns. The time complexity to compute the NM of a pattern is $O(MN)$ where M is the maximum length of a pattern and N is the size of the input trajectory data set, i.e., $|\mathcal{D}|$. Thus, during one iteration, the total time spent in computing the NM of all candidate patterns is $O(k^2MNG)$. All other operations, e.g., choosing top k patterns, extending high patterns, pruning, etc. have lower complexity than the computation of NM. As a result, the total time complexity of the TrajPattern algorithm is $O(k^2M^2NG)$.

The largest data structure to maintain is $\mathcal{Q}$, which has the space complexity $O(kMG)$. Although the input data set size N could be larger than that of $\mathcal{Q}$, it is not necessary to load the entire input data set at once since we only need a portion of the data set at a time for computing the NM. Thus the space complexity of our algorithm can be considered as $O(kMG)$.

5 Discussion

In this section we will further discuss some additional issues in the TrajPattern approach. First, in the context of the problem studied in this paper, it is desirable to find patterns with some wild card positions or gaps. A wild card position represented by the "*" symbol can be considered as a "don't care" position and any location can match this position. An additional parameter d can be used to limit the number of consecutive "don't care" symbols in a pattern. For each pattern P in $\mathcal{Q}$, we can add between 0 and d "*" symbols either in the left side or right side of P. A gap can be viewed as a variant number of consecutive "*"s. When computing the NM of a pattern, the dynamic programming technique can be used in this case.

In our current problem statement, the discovered pattern may contain any number of positions. In many applications, it may be desirable to find longer patterns, i.e., patterns longer than a certain threshold d, since longer patterns usually contain more information. This additional constraint poses a significant challenge due to the fact that we no longer know how large of a set of $\mathcal{Q}$ we need to track. To adapt the TrajPattern algorithm to this new problem, we only need to perform the following modification. The NM threshold ω is set to the minimum NM of the set of k patterns with the most NM of length at least d. In $\mathcal{Q}$, the set of patterns with NM more than ω are labeled as high patterns. The set of high patterns may be more than k. When more patterns of at least length d are inserted into $\mathcal{H}$, ω will be updated. This modification enables us to find patterns with the highest NM and at least length d.

In the TrajPattern algorithm, there are several parameters: the time interval between two consecutive snapshots t, the indifference threshold δ, the size of a grid g_x and g_y, and the maximum similar pattern distance γ. For the snapshot interval, we can use a small time unit, e.g., seconds or minutes. It can be specified by a domain expert. δ can be set to a small distance unit, which can be considered as ignorable by the domain experts. The unit length of a grid along the x and y directions g_x and g_y can be set to δ. The larger grid will reduce the computation complexity but provide inaccurate results, while the finer grid would increase the computation complexity but provide more accurate results. The sensitivity of our algorithm to δ and the computation cost of various grid size are analyzed in the experimental results section. For the maximum similar pattern distance γ, we can decide its value based on the probabilistic distribution of the location prediction model. Here due to the property of normal distribution, that is, the probability within the range between $-3 \times \sigma$ and $3 \times \sigma$ is approximately 0.97, we can set γ equal to $3 \times \sigma$.

6 Experimental Results

In this paper we implemented the TrajPattern algorithm in the C++ programming language. All experiments are running on a PC with a 3.2 GHz Pentium-4 processor and 1GB main memory. The PC is running Windows XP. It is also equipped with 160 GB disk of 7200 RPM rotation speed. We use both real and synthetical data to analyze the performance of the TrajPattern algorithm.

To illustrate the usefulness of the NM model, we compare it with the match model. The border collapsing algorithm in [14] is used to mine patterns according to match (since the Apriori property holds on the match). In addition, to show the scalability of the TrajPattern algorithm, we compare it with the PB approach [13] (used for mining the same set of NM patterns).

6.1 Effectiveness of the NM Model

We use two real data sets for demonstrating the usefulness of the trajectory patterns. One is a bus route data set, and the other is a human posture data set. Due to the space limitations, we only present the first one in this paper. The second has similar results.

In the bus data set, we have the locations of 50 buses belonging to 5 routes. Each bus is equipped with a small sensor and is able to obtain its locations via GPS. It transmits its location reading every minute. We obtain the traces of these 50 buses for 10 weekdays. Thus we have a total number of 500 traces. Each reading consists of the longitude and latitude of the bus' location. Although this data set does not use any predictive model, we can transform it to the predictive model $\mathcal{M}$ as follows. For a location reading at time t, if the location can be predicted with sufficient accuracy by the previous location(s) according to $\mathcal{M}$, then the location reading is omitted. As a result, we only retain these readings that can not be predicted by $\mathcal{M}$ accurately, which is the same as using the predictive model $\mathcal{M}$. Next we transform the location trajectories into velocity trajectories and align all 500 trajectories on a set of 100 snapshots.

For mining the trajectory patterns, we assume that the objects are traveling in a square, g_x, g_y, and δ are set to $\frac{1}{1000}$ of the side of the space. In the bus route data set, it takes TrajPattern a couple minutes to mine 1000 NM patterns. The average length of top-1000 match patterns with length at least 3 is about 3.18, while the average length of top-1000 NM patterns with length at least 3 is 4.2, which is much longer than that of match patterns.

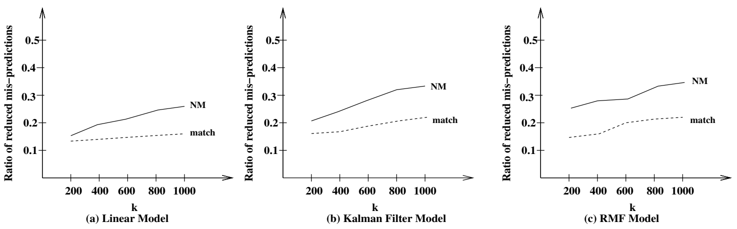

Fig. 3. Location Predictions for Bus Traces

To analyze the usefulness of the trajectory pattern model, we study the effects of employing trajectory patterns in the location prediction module. We assume that a particular module is used to predict the locations and integrate the trajectory patterns into the location prediction module. We first mine a set of k patterns of length at least 4 with the most match on the 450 velocity trajectories. Then we apply the discovered patterns to the location prediction module for the remaining 50 trajectories. When an object needs to decide whether to report a location, it first checks whether the previous portion of the trajectory confirms[2] with a discovered pattern. If so, we will use the pattern for the prediction. Otherwise, the location calculated according to the prediction module will be used. We chose three prediction modules, i.e., the linear model (LM) [12], linear Kalman Filter (LMF) [2], and the recursive motion function (RMF) [11] for the comparison. If the predicted location is too far away from the actual location such that a message has to be sent from the mobile object to the server, this is called a mis-prediction. Figure 3 shows the ratio of reduced mis-predictions by each approach. By employing the top-k NM patterns, the mis-predictions can be reduced by 20% to 40% for the three prediction methods, while with the top-k match patterns we only can reduce the mis-predictions by around 10% to 20%. This also demonstrates the effectiveness of the NM model and the trajectory patterns.

6.2 Scalability and Sensitivity

To further analyze the performance and sensitivity of our TrajPattern algorithm, we utilize a large set of synthetic data. A projection based (PB) approach [13] to mine the normalized match is presented as a baseline algorithm. We apply the TrajPattern algorithm and the PB algorithm on the synthetic data and analyze their scalability with respect to the growth of the number of patterns wanted, the number of grids in the space, the number of trajectories and the average length of the trajectories.

Synthetic data is generated according to the following parameters: the average length of a trajectory L, the number of trajectories S and the number of grids G. We generate the synthetic data in two different ways. The first data set is generated based on a similar data generation method as in [9]. The second data set is generated based on the ZebraNet data [16]. In the ZebraNet project, traces of wild zebras are recorded by deploying wireless devices on zebras in Kenya. We first extract the movement of zebras from the real traces, including the moving distance in a unit time and moving directions. There

[2] Here, we assume that a segment of trajectory confirms with a pattern if the probability that the trajectory segment is generated by the pattern (based on Equation 2) is above 90%.

are a certain number of zebra groups, within which zebras move together. For each time snapshot, each group is randomly assigned a moving distance and a moving direction that are extracted from the real traces. A randomness is added to every individual zebra to simulate noise in trajectories. Meanwhile, at each time snapshot, a certain small number of zebras will leave the group and move individually. In this paper, we only present the experimental results of the ZebraNet data set.

The projection based (PB) algorithm [13] suffers from the fact that a large set of prefixes need to be maintained. At each unspecified position, the maximum match of a position p is used as the up-bound of the possible match. However, this bound could be very loose. As a result, it could be true that every prefix up to length c could be extensible where c is a small positive integer. In this case, we need to keep G^c prefixes, which may be too large when c is larger than 3 or 4. This could render the projection based algorithm inefficient.

We compare the performance (efficiency) of the TrajPattern algorithm against the baseline projection based (PB) approach. The experimental results show that the TrajPattern algorithms outperforms the PB approach with a wide margin.

First we evaluate the performance with respect to the number of patterns needed, k. Figure 4(a) shows the average execution time of two algorithms with respect to k. Although the response time of the TrajPattern algorithm and the PB algorithm grow superlinearly with the increase of k, the response time of the TrajPattern algorithm grows at a much slower pace than that of the PB approach due to the following reason. In the previous section, we have shown that the time complexity of TrajPattern is quadratical to k, while in the PB approach the thresholds ω is lower and M is larger with larger k. As a result, the number of extensible prefixes in PB approach could increase at an exponential pace. Thus the TrajPattern is much more scalable than the PB algorithm as k increases.

The second aspect that we investigate is the scalability with the number of sequences S. The empirical results from Figure 4(b) have confirmed that the time complexity of the TrajPattern algorithm is linearly proportional to S. On the other hand, the response

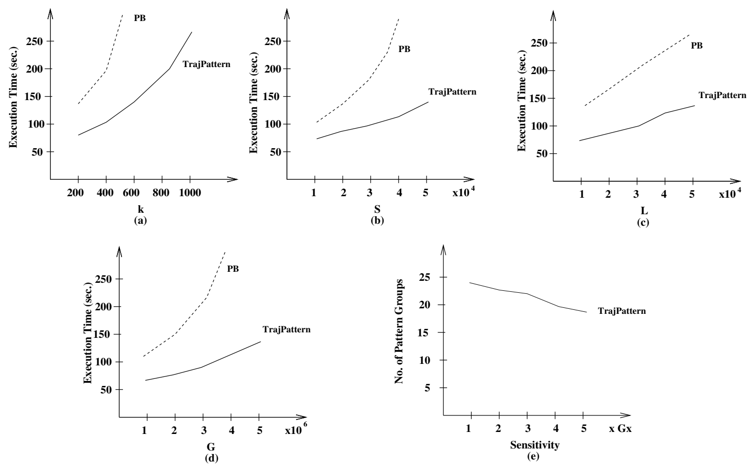

Fig. 4. Scalability and Sensitivity

time of the PB approach increases super-linearly with respect to S due to the following reason. When the number of trajectories increases, the NM of singular patterns increases and in turn the number of extendible prefixes increases exponentially. As a result, the response time of the PB algorithm increases at a much faster pace than those of the two TrajPattern algorithms.

Third, we study the effects of the average length of a sequence L. From Figure 4(c), L has similar effects on two algorithms since the time to scan a data set increases linearly with L.

Lastly, we examine the response time with various number of grids G. The TrajPattern algorithm is more scalable than the PB algorithm since the time complexity of the TrajPattern algorithm is linear with respect to G. On the other hand, in the PB approach, there are more candidate locations for each unspecified position, and in turn the number of extensible prefixes increases exponentially. Our empirical results in Figure 4(d) also confirm the theoretical analysis. The response time of the PB approach grows exponentially while the response time of the TrajPattern algorithm increases linearly.

The last experiment is performed to study the effect of the indifferent threshold δ on the mining results. Figure 4(e) shows that the number of discovered pattern groups decreases with the growth of the indifferent threshold δ. As analyzed in Section 3, the larger the indifferent threshold δ, the more grids will be considered indifferent from the expected location of the object, thus the more similar patterns will be found from the same set of trajectories. Because the number of patterns to mine is determined, the number of pattern groups becomes smaller when δ becomes larger, thus the discovered patterns represent a smaller amount of ”useful information”.

7 Conclusion

In this paper, we study a new problem, mining trajectory patterns from a set of imprecise trajectories. A novel measure is devised to represent the importance of a trajectory pattern. The min-max property is identified for the trajectory patterns. Based on this property, we develop the TrajPattern algorithm to mine the trajectory patterns, which first finds short patterns and then extends them in a systematic manner. The concept of pattern group is defined to present the trajectory patterns. Both real and synthetic data sets are used to demonstrate the usefulness of the trajectory patterns and the efficiency of the TrajPattern algorithm.

References

1. J. Han, G. Dong, and Y. Yin. Efficient mining partial periodic patterns in time series database. *Proc. of ICDE*, 1999.
2. A. Jain, E. Chang, Y. Wang. Adaptive stream resource management using Kalman filters. *Proc. of SIGMOD*, 2004.
3. Y. Li, J. Han, and J. Yang. Clustering mobile objects. *Proc. of KDD*, 2004.
4. N. Mamoulis, H. Cao. G. Kollios, M. Hadjieleftheriou, Y. Tao, D. Cheung. Mining, indexing, and querying historical spatiotemporal data. *Proc. of KDD*, 2004.
5. W.-C. Peng and M.-S. Chen. Developing data allocation schemes by incremental mining of user moving patterns in a mobile computing system. *IEEE TKDE*, 2003.

6. Song, Z., Roussopoulos, N. K-Nearest neighbor search for moving query point. *SSTD*, 2001.
7. J. Patel, Y. Chen, V. Chakka STRIPES: an efficient index for predicted trajectories. In *Proc. of SIGMOD*, 2004.
8. J. Pei, J. Han, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth. *Proc. of ICDE*, 2001.
9. S. Saltenis, C.S. Jensen, S.T. Leutenegger, and M.A. Lopez. Indexing the positions of continuously moving objects, In *Proc. of SIGMOD*, 2000.
10. Y. Tao, J. Sun, and D. Papadias. Selectivity estimation for predictive spatio-temporal queries. In *Proc. of ICDE*, pp. 417–428, Bangalore, India, Mar. 2003.
11. Y. Tao, C. Faloutsos, and D. Papadias. Prediction and indexing of moving objects with unknown motion patterns. In *Proc. of SIGMOD*, 2004.
12. O. Wolfson, P. Sistla, S. Chamberlain, and Y. Yesha. Updating and query databases that track mobile unites. *Distributed and Parallel Databases*, 7(3), 1999.
13. J. Yang, W. Wang, and P. Yu. Infominer: mining surprising periodic patterns. *Proc. of (KDD)*. pp. 395-400, 2001.
14. J. Yang, W. Wang, P. Yu, and J. Han. Mining long sequential patterns in a noisy environment. *Proc. of SIGMOD*, 2002.
15. M. Zaki. SPADE: an efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1/2):31-60, 2001.
16. Y. Wang, M. Martonosi, L.S. Peh, P. Zhang, C. Sadler, T. Liu, D. Rubenstein, S.A. Lyon. ZebraNet mobility data. unpublished, Princeton University 2004
17. J. Yang and M. Hu. TrajPattern: Mining sequential patterns from imprecise trajectories of mobile objects. *Technical Report*, EECS, Case Western Reserve University, 2005