A Unifying Framework of Mining Trajectory Patterns of Various Temporal Tightness

Jae-Gil Lee, *Member, IEEE*, Jiawei Han, *Fellow, IEEE*, and Xiaolei Li

Abstract—Discovering trajectory patterns is shown to be very useful in learning interactions between moving objects. Many types of trajectory patterns have been proposed in the literature, but previous methods were developed for only a specific type of trajectory patterns. This limitation could make pattern discovery tedious and inefficient since users typically do not know which types of trajectory patterns are hidden in their data sets. Our main observation is that many trajectory patterns can be arranged according to the strength of temporal constraints. In this paper, we propose a *unifying framework* of mining trajectory patterns of various temporal tightness, which we call *unifying trajectory patterns* (*UT-patterns*). This framework consists of two phases: *initial pattern discovery* and *granularity adjustment*. A set of initial patterns are discovered in the first phase, and their granularities (i.e., levels of detail) are adjusted by split and merge to detect other types in the second phase. As a result, the structure called a *pattern forest* is constructed to show various patterns. Both phases are guided by an information-theoretic formula without user intervention. Experimental results demonstrate that our framework facilitates easy discovery of various patterns from real-world trajectory data.

Index Terms—Trajectory pattern mining, synchronous movement patterns, moving object trajectories, trajectory clustering

1 Introduction

Trajectory data are ubiquitous in the real world. Recent progress on satellite, sensor, RFID, video, and wireless technologies has made it possible to systematically track object movements and collect huge amounts of trajectory data, e.g., animal movement data, ship navigation data, and person tracking data. Accordingly, there is an ever-increasing interest in performing extensive data analysis over trajectory data [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15].

Moving objects may show *synchronous* movement patterns since they often communicate or interact with each other. There are several kinds of synchronous movement patterns: for example, a set of moving objects move together; a set of moving objects chase another set of moving objects with a small time delay. On the other hand, moving objects may show *asynchronous* movement patterns: for example, a set of moving objects follow the same path every year. Such trajectory patterns are called collectively as *unifying trajectory patterns* (*UT-patterns*) in this paper.

UT-patterns are informally defined as sets of moving objects that are closely related in terms of location, time, or both. An example of UT-patterns can be observed in deer migration since a herd of deer move together at the same time and thus are always close to each other. Another example is wolf predation on wild ungulates since the wolves chase prey over long distances in open habitat.

- *J.-G. Lee is with the Department of Knowledge Service Engineering, KAIST, Daejeon 305-701, Republic of Korea. E-mail: jaegil@kaist.ac.kr.*
- *J. Han is with the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801. E-mail: hanj@cs.uiuc.edu.*
- *X. Li is with Twitter, San Francisco, CA 94103. E-mail: xli@twitter.com.*

Manuscript received 13 Aug. 2014; revised 14 Nov. 2104; accepted 17 Nov. 2014. Date of publication 4 Dec. 2014; date of current version 27 Apr. 2015. Recommended for acceptance by J. Xu.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2014.2377742

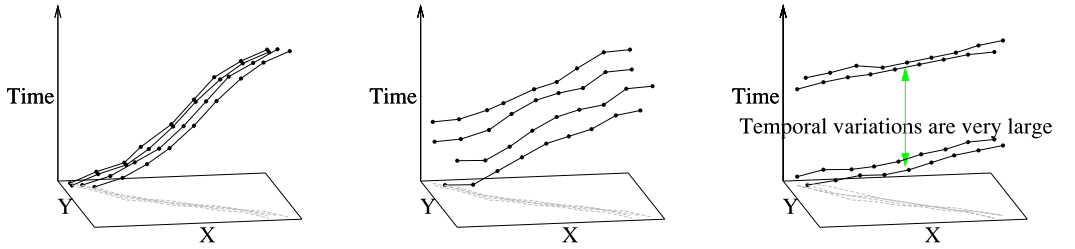
It is obvious that mining UT-patterns is very useful since we can learn interactions between moving objects and possibly group dynamics. Many applications can benefit from the UT-patterns. From animal trajectories, zoologists can learn migration patterns of animals such as deer and elk. From battleship trajectories, commanders can discover unknown tactics of the enemy. From soccer-player trajectories, coaches can guess attack strategies of the opponent team. From person trajectories, sociologists can discovery communities of the people who share common interests on physical location.

Considerable effort has been devoted to discovery of trajectory patterns in data mining and computational geometry. A list of well-known studies include flock patterns [4], [5], [6], [14], [15], convoy patterns [3], swarm patterns [10], moving clusters [16], [17], time-relaxed trajectory joins [1], hot motion paths [12], and sub-trajectory clusters [7]. Notice that each of them corresponds to just *one type* in a broad range of trajectory patterns, and previous methods have been developed to handle only one specific type.

This limitation could make pattern discovery tedious and inefficient since users typically do not know which types of trajectory patterns are hidden in their data sets. For example, a data set may contain sets of moving objects arriving at several locations within one-minute intervals, one-hour intervals, and so on. Thus, one nice way of classifying existing trajectory patterns is to consider the rigidity of temporal constraints on the patterns. This observation motivates our study which develops a framework having the potential capability of navigating the patterns at all different levels of temporal tightness.

In this paper, a *unifying framework* of mining UT-patterns is proposed, which can cover a broad range of temporal tightness classified into the three types of Fig. 1. To cover these three types, the framework goes through two phases: *initial pattern discovery* and *granularity adjustment*. A good set of initial UT-patterns are discovered in the first phase, and

1041-4347 © 2014 IEEE. Translations and content mining are permitted for academic research only. Personal use is also permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.



(a) A *time-constrained* pattern. (b) A *time-relaxed* pattern. (c) A *time-independent* pattern.

Fig. 1. Three types of UT-patterns supported by our framework.

their granularities (i.e., levels of detail) are adjusted by split and merge to detect other types in the second phase. Both phases are guided by an information-theoretic formula, based on the minimum description length (MDL) [18] principle, and do not require user intervention.

In summary, the contributions of this paper are as follows:

- We develop a unifying framework of mining trajectory patterns of various temporal tightness, which we call *UT-patterns*. More specifically, the UT-patterns are classified into three types depending on the strength of temporal constraints, and our framework covers all three types.
- We present an algorithm of discovering a good set of *initial* UT-patterns for the first phase. To ensure high efficiency, a stepwise approach is proposed, where spatial constraints are checked first to filter sub-trajectories, and temporal constraints are examined only for those satisfying spatial constraints. Moreover, to ensure high quality, the discovery process is guided by an information-theoretic formula.
- We present an algorithm of adjusting the granularity of UT-patterns for the second phase. The novel concept of a *pattern forest* is introduced to represent their granularity hierarchies, which is constructed by split and merge of the patterns. The construction process is controlled by the almost same information-theoretic formula as in the first phase.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 gives an overview of our framework. Sections 4 and 5 propose the algorithms for the first and second phases, respectively. Section 6 presents the results of experimental evaluation. Finally, Section 7 concludes the study.

2 Related Work

Trajectory patterns can be classified mainly into three types depending on the tightness of temporal constraints as in Fig. 2. Flock patterns, convoys, or moving clusters take one extreme of the spectrum, and sub-trajectory clusters take the other extreme. As going from right to left on the spectrum, temporal constraints become tighter. In addition, time-relaxed trajectory joins or hot motion paths are placed between the two extremes. These three types are summarized as follows.

1) *Flock patterns* [4], [5], [6], [14], [15], *convoys* [3], or *moving clusters* [16], [17]. A *flock* in a time interval I, where the duration of I is at least k, consists of at least m entities such that for every point in time within I, there is a disk of radius r that contains all the m entities. The *convoy* is an extension of the flock using the concept of density-based clustering. A *moving cluster* is a sequence of (snapshot) clusters $c_1, c_2, \ldots, c_k$ such that for each timestamp i $(1 \leq i < k)$, c_i and c_{i+1} share a sufficient number of common objects. In Type 1, objects should move together at the same time.
2) *Time-relaxed trajectory joins* [1] or *hot motion paths* [12]. Two trajectories are *time-relaxed joined* if there exist time intervals of the same length δ_t such that the distance between the locations of the two trajectories during these intervals is no more than a spatial threshold ε, and the relative matching between the two trajectories occurs within some time distance. A *hot motion path* is a route frequently followed by multiple objects within a tolerance margin ε in the last W timestamps. In Type 2, objects may follow other objects with some time delay.
3) *Sub-trajectory clusters* [7]. A trajectory is partitioned into a set of line segments, and these trajectory partitions are grouped into a cluster according to their spatial similarities only. That is, a *sub-trajectory cluster* is a set of trajectory partitions that are close to each other and are heading to a similar direction. In Type 3, temporal information is not considered at all.

Since the UT-pattern is defined differently from these existing patterns, it does not really subsume or incorporate any of the existing patterns. Its most prominent advantage is that it can fall into *any* of the three types in Fig. 2, whereas existing patterns into *only one* of the three types.

Contrary to the patterns in Fig. 2, several patterns do not force strong spatial constraints. For example, a *swarm* [10] is a group of moving objects containing at least min_o individuals who are in the same cluster for at least min_t timestamp snapshots. Since some objects are allowed to leave the cluster for some time period, the intermediate paths between timestamps are not very important differently from a UT-pattern. As another example, a *trajectory pattern* [2] is defined as a sequence of important locations with a transition time

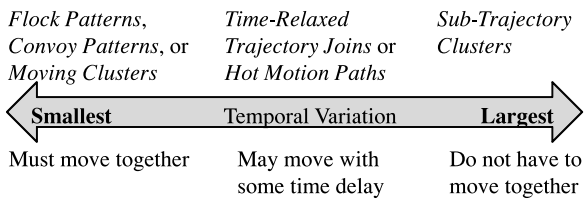


Fig. 2. A spectrum of existing trajectory patterns.

between two consecutive locations. Thus, these patterns are not included in the spectrum of Fig. 2.

The difference from our previous papers [7], [8], [9] is twofold. First, this unifying framework considers both spatial and temporal information, whereas our previous methods consider only spatial information. Second, the aims of these methods are diverse: to find UT-patterns in this framework, to find outlying trajectories in the partition-and-detect framework [8], and to predict the label of a trajectory in the *TraClas* method [9]. However, the partition-and-group framework [7] is used for preprocessing in this framework.

More studies on trajectory patterns are presented in Appendix A (supplemental material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2014.2377742).

3 UT-Pattern Mining

3.1 Problem Statement

The input of our framework is a set of trajectories $\mathcal{I} = \{TR_1, \ldots, TR_{num_{tra}}\}$. A *trajectory* is a sequence of the locations of an object at each timestamp and is denoted as $TR_i = p_1 p_2 \ldots p_j \ldots p_{len_i}$. Here, p_j $(1 \leq j \leq len_i)$ is a point (x_j, y_j, t_j) in a three-dimensional space, where (x_j, y_j) indicates the location of the object at the time t_j. The length len_i of a trajectory can be different from those of other trajectories. A trajectory $p_{c_1} p_{c_2} \ldots p_{c_k}$ $(1 \leq c_1 < c_2 < \cdots < c_k \leq len_i$, where $c_k = c_{k-1} + 1)$ is called a *sub-trajectory* of TR_i. A *trajectory partition* is a line segment $p_i p_j$ $(i < j)$, where p_i and p_j are the points chosen from the same trajectory.

The output of our framework is a set of UT-patterns $\mathcal{O} = \{UT_1, \ldots, UT_{num_{pat}}\}$. A *UT-pattern* is a set of trajectory partitions with a reference movement. Trajectory partitions that belong to the same UT-pattern are close in terms of location and time according to a similarity function. A *reference movement* is a line segment, where the first point is the starting location and time of the overall movement, and the second point the ending location and time. In summary, a UT-pattern is $UT_i = \langle \overrightarrow{R_i}, \mathbb{L}_i \rangle$, where $\overrightarrow{R_i}$ is the reference movement, and $\mathbb{L}_i$ the set of trajectory partitions.

Example 1. Fig. 3 illustrates an example of UT-patterns on a set of trajectories whose movements are similar. The upper three objects and the lower three ones moved at different times, so the two groups belonged to different UT-patterns. Besides, the lower three objects stayed at a particular point for a long time, so the former part and the latter part belonged to different UT-patterns. The reference movements are displayed as the background arrows. Overall, $UT_1 \sim UT_3$ are discovered from the six sub-trajectories.

3.2 Overall Framework

Algorithm 1 shows the skeleton of our pattern mining algorithm. It executes two sub-algorithms (Lines 6 and 12): the first one generates a good set of initial UT-patterns for each sub-trajectory cluster; and the second one constructs a pattern forest by splitting and merging them. The two sub-algorithms will be explained in Sections 4.5 and 5.3, respectively. Optionally, the UT-patterns in the forest are marked explicitly as one of the three types.

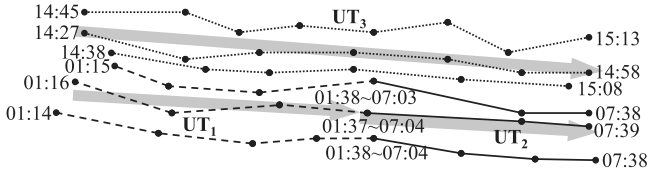


Fig. 3. An example of UT-patterns.

Algorithm 1. UT-Pattern Mine

Input: A set of trajectories $\mathcal{I} = \{TR_1, \ldots, TR_{num_{tra}}\}$
Output: A set of UT-patterns $\mathcal{O} = \{UT_1, \ldots, UT_{num_{pat}}\}$
1: /* Phase I: Initial Pattern Discovery */
2: Perform sub-trajectory clustering over $\mathcal{I}$ based on the partition-and-group framework [7];
3: Get a set $\mathcal{C}_{all}$ of sub-trajectory clusters;
4: **for** each $C \in \mathcal{C}_{all}$ **do**
5: /* Algorithm 2 */
6: Execute *Initial Pattern Generation* over C;
7: Get a set $\mathcal{P}$ of UT-patterns as the result;
8: Accumulate $\mathcal{P}$ into a set $\mathcal{P}_{all}$;
9: **end for**
10: /* Phase II: Granularity Adjustment */
11: /* Algorithm 3 */
12: Execute *Pattern Forest Construction* over $\mathcal{P}_{all}$;
13: Return the set of UT-patterns in the forest;
14: /* Optional */
15: Classify the UT-patterns into the three types;

Sub-trajectory clustering is performed *as preprocessing*. Notice that only spatial constraints are verified at this stage, and trajectory partitions that do not satisfy spatial constraints are thrown away since they cannot be part of the result. While executing the sub-algorithms, temporal constraints are checked among only the trajectory partitions that remain after preprocessing. The general approach of first considering spatial constraints and then temporal constraints is also found in time-focused clustering [11] since handling both constraints at the same time is inherently difficult.

4 Initial Pattern Discovery

4.1 Stepwise Approach

The phase of initial pattern discovery is processed in two steps as shown in Fig. 4. First, sub-trajectory clusters in the (X, Y)-space are discovered based on the partition-and-group framework [7] *without considering temporal constraints*. A sub-trajectory cluster is a set of trajectory partitions that moved close together in a similar direction. Then, for each sub-trajectory cluster, initial UT-patterns in the $(location, time)$-space are retrieved *by considering both spatial and temporal constraints*. Notice that the locations of trajectory partitions within a particular sub-trajectory cluster can be represented using one dimension since their directions are very similar. The first step is explained in Section 4.2, and the second step in Sections 4.3 $\sim$ 4.5.

The advantage of this stepwise approach is to *improve efficiency* for the following reasons. First, the dimensionality is reduced from three to two. Second, the search space is confined to a particular sub-trajectory cluster, not the entire data set.

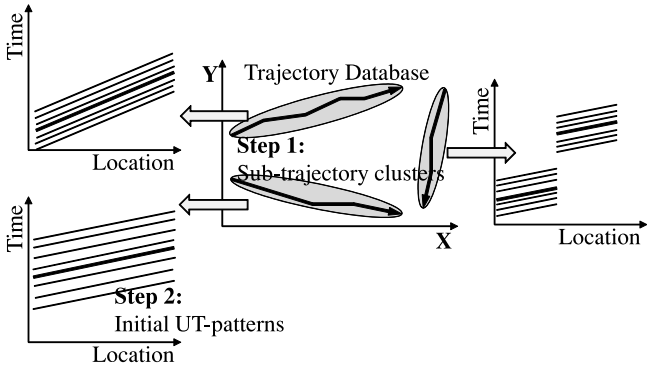


Fig. 4. The stepwise approach of discovering initial UT-patterns.

4.2 Sub-Trajectory Clustering

First of all, *time-independent* patterns are found by *sub*-trajectory clustering. On the other hand, clustering whole trajectories is not suitable for our problem since similar *portions* of the trajectories are not detected. Sub-trajectory clustering is based on the partition-and-group framework that consists of two phases as in Fig. 5.

1) The *partitioning* phase: Each trajectory is partitioned into a set of line segments (i.e., trajectory partitions) whenever its moving direction changes rapidly. The problem of finding the optimal set of such partitioning points is formulated by the MDL principle. An $O(n)$ approximate algorithm is proposed to efficiently find the near-optimal partitioning.
2) The *grouping* phase: Similar line segments are grouped into a cluster using a density-based clustering method analogous to DBSCAN [19]. The basic notions of density-based clustering for points are changed to those for line segments as in Appendix B (available in the online supplemental material). There are two parameters that need to be configured: ε is determined such that the size of an ε-neighborhood ($|N_\varepsilon(L_i)|$) is skewed among line segments as much as possible, and $MinLns$ is determined using the average of $|N_\varepsilon(L_i)|$ at the optimal value of ε.

At the final stage of the grouping phase, a model called a *representative trajectory*, which is a sequence of points just like an ordinary trajectory, is generated for each cluster. It is an imaginary trajectory that indicates the major movement pattern of the trajectory partitions belonging to the cluster and is obtained by calculating the average coordinates of those trajectory partitions.

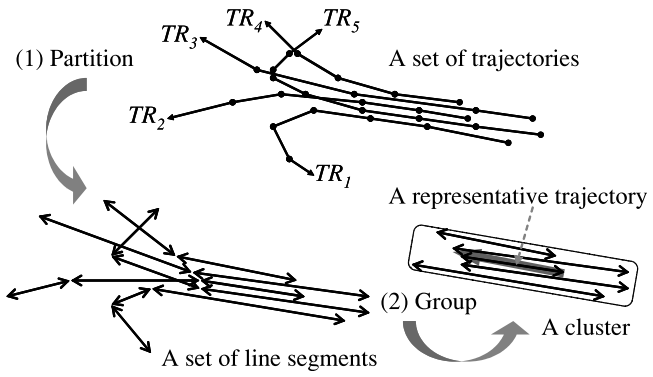


Fig. 5. An overall procedure of sub-trajectory clustering [7].

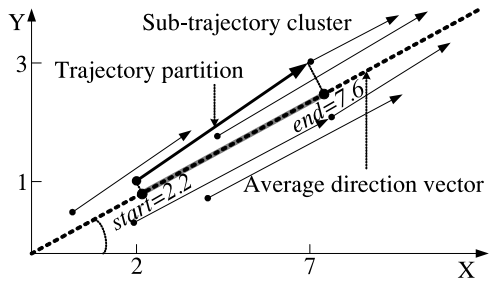


Fig. 6. Dimensionality reduction for locations within a particular cluster.

4.3 Representation of Trajectory Partitions

Trajectory partitions in the $(X, Y, time)$-space are converted into those in the $(location, time)$-space. The real interest here is making the analysis simpler and easier almost without losing accuracy. This is possible because the directions of trajectory partitions within a cluster are very similar. This dimensionality reduction is illustrated in Fig. 6. Each endpoint of a trajectory partition is projected onto the average direction vector [7] of a cluster. Its projection point can be easily calculated using the rotation matrix. Also note that the original two-dimensional location can be restored from this one-dimensional location using the *inverse* of the rotation matrix. The (inverse) rotation matrix is constructed by just keeping the angle of the average direction vector for each cluster. Variations perpendicular to the average direction vector are usually small, so they can be safely ignored.

Example 2. Suppose the thick trajectory partition in Fig. 6 is $(2, 1, 3)(7, 3, 8)$ in the $(X, Y, time)$-space, and $\phi = \pi/6$. The projection points are obtained by multiplying the rotation matrix $R(-\pi/6)$ to $(2, 1)$ and $(7, 3)$, which are 2.2 and 7.6, respectively. Thus, the trajectory partition becomes $(2.2, 3)(7.6, 8)$ in the $(location, time)$-space. When we need to restore the locations in the (X, Y)-space, we multiply the inverse of $R(-\pi/6)$ to $(2.2, 0)$ and $(7.6, 0)$. The restored locations are $(1.9, 1.1)$ and $(6.6, 3.8)$, which are very close to the original locations.

4.4 Information-Theoretic Formulation

Once all trajectory partitions in a sub-trajectory cluster are represented in the $(location, time)$-space, the next step is to find a set of *reference movements* that best capture the underlying patterns of those trajectory partitions. Then, every trajectory partition is allocated to the closest reference movement according to a similarity measure. More specifically, we try to maximize the compression of the trajectory partitions using the set of reference movements.

This approach provides two major advantages. First, it has no parameter to be specified by users. Second, it returns a natural partitioning of a sub-trajectory cluster owing to the intuitive information-theoretic principle of maximizing the data compression [20]. These advantages are essential to our problem since users do not know the number of UT-patterns and their sizes either.

4.4.1 Similarity Measure

A similarity measure between line segments is required for quantifying the degree of data compression. Here, we use the similarity function developed in our earlier work [7],

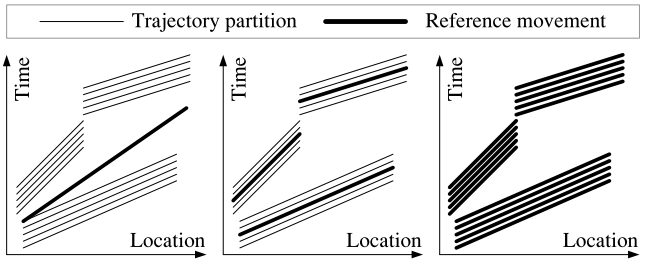


Fig. 7. Preciseness and conciseness between two extreme cases.

[8], [9], which was proven to be reliable. The similarity function takes two line segments and measures (i) the *perpendicular distance*, (ii) the *parallel distance*, and (iii) the *angle distance* between them. Similar techniques have been widely used for measuring the difference between line segments in the area of pattern recognition [21]. Note that, since the similarity function is applied on the $(location, time)$-space, its value represents both spatial and temporal similarity.

4.4.2 Goodness Criteria

To maximize data compression, a set of UT-patterns should possess two desirable properties: *preciseness* and *conciseness*. Preciseness means that the difference between a reference movement and a set of its trajectory partitions should be as small as possible. Conciseness means that the number of reference movements should be as small as possible.

Preciseness and conciseness are rivalry measures. If all trajectory partitions are represented by only one pattern as in Fig. 7a, preciseness may become lowest, but conciseness becomes highest. In contrast, if every trajectory partition is represented by its own pattern as in Fig. 7c, preciseness becomes highest, but conciseness becomes lowest. Thus, we need to find a good tradeoff between the two properties as in Fig. 7b.

Example 3. The trajectory partitions in Fig. 3 are represented on the $(location, time)$-space as in Fig. 8. They span the same space but do *not* span the same time period. Thus, the three shaded groups take different positions of the $time$ axis. To optimize data compression, it is obvious that each group should become a separate UT-pattern.

TABLE 1
The Notation for the MDL Cost

Symbol	Definition
num_{pat}	the number of UT-patterns
num_{loc}	the number of unique starting and ending locations of trajectory partitions
num_{time}	the number of unique starting and ending times of trajectory partitions
$\overrightarrow{R_i}$	the reference movement for the ith UT-pattern $(1 \le i \le num_{pat})$
$\mathbb{L}_i$	the set of trajectory partitions belonging to the ith UT-pattern
$dist(\overrightarrow{R_i}, L_j)$	the similarity of a trajectory partition L_j to the reference movement $(1 \le j \le \lvert\mathbb{L}_i\rvert)$ (defined in Section 4.4.1)
$C(\overrightarrow{R_i}, \mathbb{L}_i)$	the code length of trajectory partitions in the ith UT-pattern
$C(\theta_i)$	the code length of a probability parameter for the ith UT-pattern

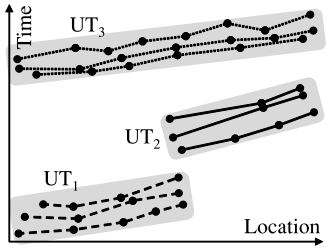


Fig. 8. The UT-patterns derived from Fig. 3.

4.4.3 MDL Cost Function

The MDL cost consists of two components [18]: $L(H)$ and $L(D|H)$. Here, H means the hypothesis, and D the data. The two components are informally stated as follows [18]: "$L(H)$ is the length, in bits, of the description of the hypothesis; and $L(D|H)$ is the length, in bits, of the description of the data when encoded with the help of the hypothesis." The best hypothesis H to explain D is the one which minimizes the sum of $L(H)$ and $L(D|H)$.

The MDL principle fits very well our problem. H corresponds to a set of UT-patterns, and D to the set of trajectory partitions. As a result, finding a good set of UT-patterns translates to finding the best hypothesis using the MDL principle. The notation for this formulation is summarized in Table 1.

Description Complexity. $L(H)$ is formulated by Eq. (1). The first term is required to describe the number of UT-patterns. Here, $\log^*$ is the universal code length for integers.[1] The second term is required to describe the set of reference movements. If there are num_{pat} reference movements, $(2 \cdot num_{pat})$ endpoints need to be encoded. The code length for an endpoint is $\log_2(num_{loc} \cdot num_{time})$ because it is selected from $(num_{loc} \cdot num_{time})$ points

$$L(H) = \log^*(num_{pat}) + 2 \cdot num_{pat} \cdot \log_2(num_{loc} \cdot num_{time}). \quad (1)$$

Code Length. $L(D|H)$ is formulated by Eq. (2). The first term encodes the number of trajectory partitions within the ith UT-pattern. The second term $C(\overrightarrow{R_i}, \mathbb{L}_i)$ encodes the similarities of the trajectory partitions with respect to the reference movement in the ith UT-pattern. By Shannon's classical information theory, the data can be encoded in $-\log_2 p(x)$ bits when $p(x)$ is the probability of occurrence of x [22]. It is usually assumed that the distance between a reference object and an object conforms to a certain probability distribution such as a Gaussian distribution $N(0, \sigma^2)$ [23] and an exponential distribution $Exponential(1) = \lambda e^{-\lambda x}$ $(\lambda = 1)$ [24]. The third term encodes a parameter for the probability distribution. Finally, $L(D|H)$ is obtained by summing up the three terms of every UT-pattern $(1 \le i \le num_{pat})$

$$L(D|H) = \sum_{i=1}^{num_{pat}} \left(\log^* |\mathbb{L}_i| + C(\overrightarrow{R_i}, \mathbb{L}_i) + C(\theta_i)\right). \quad (2)$$

1. It can be shown that $\log^*(n) = \log_2 c + \log_2 n + \log_2 \log_2 n + \cdots$, where the sum only includes positive terms and $c \approx 2.865$ [22]. If n is large, $\log^*(n) \approx \log_2 n$.

$C(\overrightarrow{R_i}, \mathbb{L}_i)$ for $N(0, \sigma^2)$ is formulated by Eq. (3), and that for $Exponetial(1)$ by Eq. (4). In addition, $C(\theta_i)$ for the former is $\frac{1}{2}\log_2|\mathbb{L}_i|$, and that for the latter is zero. For $N(0, \sigma^2)$, one parameter σ needs to be encoded.[2] Actually, a proper probability distribution depends on the application and data set. Our experience indicates that when $Exponetial(1)$ is used, $C(\overrightarrow{R_i}, \mathbb{L}_i)$ increases more rapidly as $dist()$ increases. Thus, it tends to generate a larger number of UT-patterns than $N(0, \sigma^2)$

$$C_g(\overrightarrow{R_i}, \mathbb{L}_i) = \sum_{L_j \in \mathbb{L}_i} -\log_2 \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{dist(\overrightarrow{R_i}, L_j)^2}{2\sigma^2}\right) \quad (3)$$

$$C_e(\overrightarrow{R_i}, \mathbb{L}_i) = \sum_{L_j \in \mathbb{L}_i} -\log_2 \exp(-dist(\overrightarrow{R_i}, L_j)). \quad (4)$$

As mentioned before, our objective is to find the set of UT-patterns that minimizes $L(H) + L(D|H)$. This is exactly the tradeoff between conciseness and preciseness in the sense that $L(H)$ measures the degree of conciseness, and $L(D|H)$ that of preciseness. Although theoretically pleasing, our problem is computationally expensive: since even the optimal num_{pat} is unknown, it is required to find the best set for every possible cardinality. Thus, an approximate algorithm is developed in the next section.

4.5 An Approximate Algorithm

Algorithm 2 shows an approximate algorithm for generating an initial set of UT-patterns. It receives a set of trajectory partitions that belong to the same sub-trajectory cluster and returns a set of UT-patterns. Notice that the patterns retrieved here are either *time-constrained* or *time-relaxed*.

At the beginning, one UT-pattern containing all trajectory partitions is assumed to exist (Lines 1 $\sim$ 2). The algorithm progressively refines the set of UT-patterns by splitting one of them as long as the MDL cost decreases. Among the current set of UT-patterns, the algorithm selects the one that has the maximum code length (Line 4) and splits it into two splits (Lines 5 $\sim$ 15). When splitting a UT-pattern, the algorithm selects the pair (L_p, L_q) of trajectory partitions that are the most dissimilar (Line 6) and assigns the remaining ones to the more similar one from L_p and L_q (Lines 8 $\sim$ 15). By Lemma 1, this split always decreases $\sum C(\overrightarrow{R_i}, \mathbb{L}_i)$ in $L(D|H)$. However, the description complexity $L(H)$ evidently increases. If this split decreases the MDL cost ($= L(H) + L(D|H)$), the same procedures are repeated.

Lemma 1. $C(\overrightarrow{R_i^1}, \mathbb{L}_i^1) + C(\overrightarrow{R_i^2}, \mathbb{L}_i^2) \leq C(\overrightarrow{R_i}, \mathbb{L}_i)$, *where* $\mathbb{L}_i = \mathbb{L}_i^1 \cup \mathbb{L}_i^2$.

Proof. The code length $C(\overrightarrow{R_i}, \mathbb{L}_i)$ monotonically decreases as the distance between a trajectory partition and a reference movement does. If a reference movement is added, some trajectory partitions must be closer to the added one than to the rest and thus can be encoded using a smaller number of bits. The equality holds only when either $\mathbb{L}_i^1$ or $\mathbb{L}_i^2$ is empty. □

2. If a parameter θ is unknown, one could estimate it from the data and then apply Shannon's theory with the unknown θ replaced by its estimate $\hat{\theta}$. A maximum likelihood estimate computed from n data points can be effectively encoded by $\frac{1}{2}\log_2 n$ bits [22].

Algorithm 2. Initial Pattern Generation

INPUT: A set $\mathcal{L}$ of trajectory partitions in a cluster C
OUTPUT: A set $\mathcal{P}$ of initial UT-patterns
1: $\mathbb{L}_1 \Leftarrow \mathcal{L}$, $\overrightarrow{R_1} \Leftarrow$ *DeriveRefMovement*($\mathbb{L}_1$);
2: $\mathcal{P} \Leftarrow \{\langle \overrightarrow{R_1}, \mathbb{L}_1 \rangle\}$;
3: **repeat**
4: Choose the mth UT-pattern from $\mathcal{P}$, where
$$m = \underset{\langle \overrightarrow{R_m}, \mathbb{L}_m \rangle \in \mathcal{P}}{\operatorname{argmax}} C(\overrightarrow{R_m}, \mathbb{L}_m);$$
5: /* Split the mth UT-pattern into two splits */
6: Choose the pair of trajectory partitions, where
$$(L_p, L_q) = \underset{L_p, L_q \in \mathbb{L}_m}{\operatorname{argmax}} dist(L_p, L_q);$$
7: /* Distribute t-partitions of $\mathbb{L}_m$ into two */
8: $\mathbb{L}_m^p \Leftarrow \emptyset$, $\mathbb{L}_m^q \Leftarrow \emptyset$;
9: **for** each $L_i \in \mathbb{L}_m$ **do**
10: **if** $dist(L_i, L_p) < dist(L_i, L_q)$ **then**
11: $\mathbb{L}_m^p \Leftarrow \mathbb{L}_m^p \cup \{L_i\}$;
12: **else**
13: $\mathbb{L}_m^q \Leftarrow \mathbb{L}_m^q \cup \{L_i\}$;
14: **end if**
15: **end for**
16: /* Derive a reference movement for each split */
17: $\overrightarrow{R_m^p} \Leftarrow$ *DeriveRefMovement*($\mathbb{L}_m^p$);
18: $\overrightarrow{R_m^q} \Leftarrow$ *DeriveRefMovement*($\mathbb{L}_m^q$);
19: /* Replace the mth pattern by the new ones */
20: $\mathcal{P}' \Leftarrow \mathcal{P} - \{\langle \overrightarrow{R_m}, \mathbb{L}_m \rangle\} \cup \{\langle \overrightarrow{R_m^p}, \mathbb{L}_m^p \rangle, \langle \overrightarrow{R_m^q}, \mathbb{L}_m^q \rangle\}$;
21: /* Check if $L(H) + L(D|H)$ decreases */
22: **if** $MDL(\mathcal{P}') < MDL(\mathcal{P})$ **then**
23: $\mathcal{P} \Leftarrow \mathcal{P}'$;
24: **end if**
25: **until** $MDL(\mathcal{P}') \geq MDL(\mathcal{P})$
26: Return the set $\mathcal{P}$ of initial UT-patterns;

27: **function** *DeriveRefMovement*($\mathbb{L}_k$):
28: /* Consider each t-partition as a candidate */
29: $\mathbb{R}_k \Leftarrow \{L \mid \forall L \in \mathbb{L}_k\}$;
30: /* Find the one that minimizes the code length */
31: Return the sth candidate $\overrightarrow{R_k^s}$, where
$$s = \underset{\overrightarrow{R_k^s} \in \mathbb{R}_k}{\operatorname{argmin}} C(\overrightarrow{R_k^s}, \mathbb{L}_k);$$
32: **end function**

The function *DeriveRefMovement* (Lines 27 $\sim$ 32) is called to derive a reference movement for a UT-pattern. Simply, each trajectory partition becomes a candidate for the reference movement. Among these candidates, the function picks the one that minimizes the code length $C(\overrightarrow{R_k}, \mathbb{L}_k)$.

Lemma 2. *The time complexity of Algorithm 2 is* $O(num_{pat} \cdot n^2)$, *where* n *is the number of trajectory partitions in a particular sub-trajectory cluster.*

Proof. The algorithm exits the loop after executing Lines 4 $\sim$ 24 num_{pat} times. At each iteration, the algorithm checks every pair of trajectory partitions of a UT-pattern in Lines 6, 17, and 18. □

TABLE 2
Three Types of UT-Patterns

Correlation	Temporal cohesion	Type
None	$\|r\| < 0.1$	*Time-independent*
Small	$0.1 \leq \|r\| < 0.3$	*Time-relaxed*
Medium	$0.3 \leq \|r\| < 0.5$	*Time-relaxed*
Large	$\|r\| \geq 0.5$	*Time-constrained*

4.6 Interpretation of UT-Patterns

To quantify the temporal proximity in a UT-pattern, we introduce the notion of the *temporal cohesion* and formally define it in Definition 1. Basically, it is the Pearson product-moment correlation coefficient between $location$ and $time$. The correlation coefficient is being widely used as a measure of the strength of linear dependence between two variables. Following the notation of the correlation coefficient, the temporal cohesion is denoted by r. By definition, r gives a value between -1 and $+1$ inclusive. The closer to $|1|$, the stronger the temporal cohesion is.

The correlation coefficient aims at measuring how close the timestamps at the same location are with each other. If these timestamps are very close with each other, linear dependence between $location$ and $time$ should be very strong. If each moving object visited the same location at different timestamps, points in the ($location$, $time$)-space would spread widely over the $time$ axis, resulting in weak linear dependence. Thus, the correlation coefficient is found to be suitable for our problem.

Definition 1. *Consider the set $\mathbb{L}_i$ of trajectory partitions in UT_i. For a given trajectory partition L_j, $L_j.start\_loc$ and $L_j.start\_time$ denote its starting location and time, respectively. Consider a set of (x, y) points constructed as $\{(L_j.start\_loc, L_j.start\_time) \,|\, \forall L_j \in \mathbb{L}_i\}$. Then, the* temporal cohesion *of UT_i is the Pearson product-moment correlation coefficient [25] defined as Eq. (5). Here, n means $|\mathbb{L}_i|$ which is the number of trajectory partitions in UT_i*

$$r = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \overline{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \overline{Y})^2}}. \quad (5)$$

The type of a UT-pattern is determined by its temporal cohesion according to Table 2. Since the temporal cohesion is based on the correlation coefficient, we adopt a guideline for the interpretation of the correlation coefficient [26]. However, this guideline should not be observed too strictly because the interpretation depends on the context and purposes.

Example 4. Suppose UT_i consists of five trajectory partitions $\mathbb{L}_i = \{(1,2)(5,5),\ (2,2)(6,5),\ (3,2)(7,5),\ (4,3)(6,6),\ (5,6)\ (7,8)\}$. Then, the temporal cohesion is 0.82 by Definition 1. This pattern is categorized as *time-constrained* per our guideline in Table 2 and means that the five objects have moved very closely together.

5 Granularity Adjustment

5.1 Pattern Forest

Although the initial set best captures the underlying patterns of trajectory partitions according to the MDL cost, some users may still want to find smaller (finer) or larger (coarser) patterns. This requirement motivates the phase of granularity adjustment. Example 5 discusses one application example for animal monitoring.

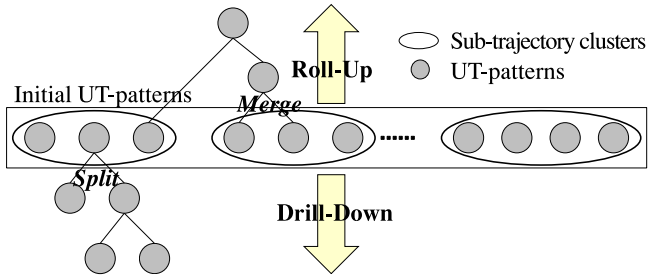


Fig. 9. Construction of the pattern forest from initial UT-patterns.

Example 5. Animal scientists want to focus on some regions when investigating animal behaviors. If they find an interesting pattern from a specific region (e.g., in a valley), they may like to zoom in the region to see multiple finer-granularity patterns. In this way, they study the impact of regional environments on animal behaviors in the animal's habitat. On the other hand, animal scientists investigate animal behaviors for different granularities of time. They may like to see *seasonal* patterns as well as *daily* patterns. Please refer to [27] as an example of this kind of study.

To this end, a *pattern forest* is introduced in Definition 2, which is a set of trees of UT-patterns. As shown by Fig. 9, it is constructed by splitting and merging UT-patterns: smaller patterns are obtained by splits, and larger ones by mergers.

Definition 2. *A* pattern forest *$\mathcal{FR}$ is an undirected graph $(\mathcal{V}, \mathcal{E})$. Here, $v \in \mathcal{V}$ is a UT-pattern; $e \in \mathcal{E}$ represents either a split or merge relationship.*

1) *$\mathcal{FR}$ is the set of trees $\mathcal{T}_i$ $(1 \leq i \leq num_{tree})$, each consisting of the vertexes $\mathcal{V}_i \subseteq \mathcal{V}$ and the edges $\mathcal{E}_i \subseteq \mathcal{E}$. num_{tree} denotes the number of such trees in the pattern forest. Each $\mathcal{V}_i$ is a component of $\mathcal{V}$ (i.e., $\forall i \neq j, \mathcal{V}_i \cap \mathcal{V}_j = \emptyset$ and $\bigcup \mathcal{V}_i = \mathcal{V}$); each $\mathcal{E}_i$ is a component of $\mathcal{E}$ (i.e., $\forall i \neq j, \mathcal{E}_i \cap \mathcal{E}_j = \emptyset$ and $\bigcup \mathcal{E}_i = \mathcal{E}$).*
2) *The vertexes for the initial UT-patterns are called the* base vertexes. *The set of base vertexes is denoted by $\mathcal{V}_{base} \subseteq \mathcal{V}$.*

Definitions 3 and 4 describe the split and merge operations in the context of the pattern forest.

Definition 3. *A* split *of a UT-pattern (vertex) $v \in \mathcal{V}$ is adding two vertexes u, w and two edges $(v, u), (v, w)$. That is, a split makes a new pattern forest $\mathcal{FR}_{new}$ such that $\mathcal{V}_{new} = \{u, w\} \cup \mathcal{V}$ and $\mathcal{E}_{new} = \{(v, u), (v, w)\} \cup \mathcal{E}$.*

Definition 4. *A* merger *of two UT-patterns (vertexes) $u, w \in \mathcal{V}$ is adding one vertex v and two edges $(v, u), (v, w)$. That is, a merger makes a new pattern forest $\mathcal{FR}_{new}$ such that $\mathcal{V}_{new} = \{v\} \cup \mathcal{V}$ and $\mathcal{E}_{new} = \{(v, u), (v, w)\} \cup \mathcal{E}$.*

Split and merger of UT-patterns are semantically analogous to drill-down and roll-up [25] widely used in OLAP. However, the operations in our problem are more

complicated since the drill-down and roll-up dimensions are not specified by users but are automatically determined to be either *location* or *time*. This choice is cleanly formulated using the almost same information-theoretic formula as in the previous phase. Moreover, as far as we know, there has been no work for *automatically* building concept hierarchies for spatio-temporal patterns.

5.2 OLAP-Like Operations

5.2.1 Drill-Down Operation

Drill-down is done only if a split decreases the MDL cost just like Algorithm 2. Its main purpose is to derive multiple time-constrained (or *smaller* time-relaxed) patterns from a time-relaxed pattern. The drill-down dimension is selected automatically so as to minimize the MDL cost. The condition for a split is presented in Definition 5.

Definition 5. *Consider a UT-pattern* $UT_i = \langle \overrightarrow{R_i}, \mathbb{L}_i \rangle$. *Let* $UT_i^1 = \langle \overrightarrow{R_i^1}, \mathbb{L}_i^1 \rangle$ *and* $UT_i^2 = \langle \overrightarrow{R_i^2}, \mathbb{L}_i^2 \rangle$ *be two UT-patterns satisfying* $\mathbb{L}_i^1 \cup \mathbb{L}_i^2 = \mathbb{L}_i$ *and* $\mathbb{L}_i^1 \cap \mathbb{L}_i^2 = \emptyset$. *If* $MDL(\{UT_i\}) > MDL(\{UT_i^1, UT_i^2\})$, UT_i can be split *into* UT_i^1 *and* UT_i^2. *Here,* $MDL()$ *denotes the sum of Eqs. (1) and (2).*

There can be many choices to split UT_i into UT_i^1 and UT_i^2 with satisfying the requirements in Definition 5. Basically, it is preferable to obtain the split that minimizes $MDL(\{UT_i^1, UT_i^2\})$. Thus, we use the heuristic for determining a set of initial UT-patterns. More specifically, UT_i^1 and UT_i^2 are generated by performing Lines 5 $\sim$ 18 of Algorithm 2. In addition, the reference movement for a new UT-pattern is obtained by the same method *DeriveRefMovement* in Algorithm 2 throughout the paper.

5.2.2 Roll-Up Operation

Roll-up is the reverse operation of drill-down. The condition for a merger is presented in Definition 6. A *relaxation factor* γ is introduced since the merger does not decrease the MDL cost. γ designates the amount of sacrificing preciseness for conciseness. Empirically, $\gamma \simeq 0.95$ works well in most cases.

Definition 6. *Consider two UT-patterns* $UT_i = \langle \overrightarrow{R_i}, \mathbb{L}_i \rangle$ *and* $UT_j = \langle \overrightarrow{R_j}, \mathbb{L}_j \rangle$. *Let* $UT_{ij} = \langle \overrightarrow{R_{ij}}, \mathbb{L}_i \cup \mathbb{L}_j \rangle$ *be a merger of* UT_i *and* UT_j. *If* $\gamma \cdot MDL(\{UT_{ij}\}) < MDL(\{UT_i, UT_j\})$ $(0 < \gamma < 1)$, UT_i *and* UT_j can be merged *into* UT_{ij}.

5.3 A Construction Algorithm

Algorithm 3 summarizes the procedure of constructing a pattern forest. This algorithm is self-explanatory. The first half is the drill-down phase, and the second half the roll-up phase. The drill-down phase precedes the roll-up phase just for following the order of presentation.

Lemma 3. *The time complexity of Algorithm 3 is* $O(m^2)$, *where* m *is the number of initial UT-patterns.*

Proof. Let d be the maximum depth of a given pattern forest $\mathcal{FR}$. During the drill-down phase, the algorithm repeats at most $m + 2 \cdot m + \cdots + 2^{d-1} \cdot m$ times. During the roll-up phase, the algorithm repeats at most $\binom{m}{2} + \binom{\frac{m}{2}}{2} + \cdots + \binom{\frac{m}{2^{d-1}}}{2}$ times. Thus, the time complexity is dominated by the roll-up phase (Lines 12 $\sim$ 23), and it becomes $O(m^2)$. □

One might wonder why the UT-patterns that were not split in Algorithm 2 could be split in Algorithm 3. Please note that the MDL formulation for Algorithm 3 is slightly different from that for Algorithm 2. Only a single pattern is considered each time in Algorithm 3 (Lines 4 $\sim$ 5), whereas the set $\mathcal{P}$ of all patterns is considered in Algorithm 2. As a result, splitting a specific pattern here is not affected by the costs of other patterns.

Algorithm 3. Pattern Forest Construction

```
INPUT: A set P_all of initial UT-patterns
OUTPUT: A pattern forest FR
1: FR ⇐ P_all, Q ⇐ P_all; /* Q is a queue */
2: /* 1. DRILL-DOWN */
3: while Q ≠ ∅ do
4:    Pop a UT-pattern UT_i from Q;
5:    if UT_i can be split into UT_i^1 and UT_i^2 by Definition 5 then
6:       Push UT_i^1 and UT_i^2 into Q;
7:       /* Update the pattern forest */
8:       Add two vertexes for UT_i^1 and UT_i^2 into FR;
9:       Add two edges for (UT_i, UT_i^1) and (UT_i, UT_i^2) into FR;
10:   end if
11: end while
12: /* 2. ROLL-UP */
13: /* P_c is a set of UT-patterns in the cth cluster */
14: for each P_c ⊆ P_all do
15:    for each pair of UT_i ∈ P_c and UT_j ∈ P_c do
16:       if UT_i and UT_j can be merged into UT_ij by Definition 6
          then
17:          Add UT_ij into P_c;
18:          /* Update the pattern forest */
19:          Add one vertex for UT_ij into FR;
20:          Add two edges for (UT_ij, UT_i) and (UT_ij, UT_j) into
             FR;
21:       end if
22:    end for
23: end for
24: Return the pattern forest FR;
```

6 Experimental Evaluation

6.1 Experimental Setting

Real-World Data Sets. Two real data sets of animal movements[3] are used for our experiments. By the Starkey Project, radio-telemetry locations of animals were collected in northeastern Oregon. Deer's trajectories from April to August in 1995 and elk's trajectories from May to August in 1993 are selected for pattern discovery. The deer data set has 32 trajectories and 20,065 points, and the elk data set 33 trajectories and 47,204 points. Each point consists of a real coordinate (location) and a timestamp: a coordinate is represented using the UTM coordinate system; a timestamp is the time when the coordinate was recorded, which is the cumulative number of minutes that have elapsed since the Greenwich Mean Time of 00 : 00 : 00 on 12/31/87.

Synthetic Data Set 1. For accuracy test in Section 6.3, a given number of UT-patterns are injected into a sub-trajectory cluster. In each UT-pattern, a set of moving objects

3. http://www.fs.fed.us/pnw/starkey/data/tables/

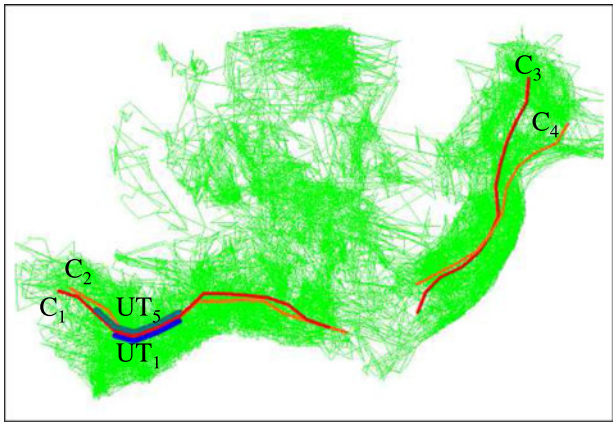


Fig. 10. Sample UT-patterns for the deer data.

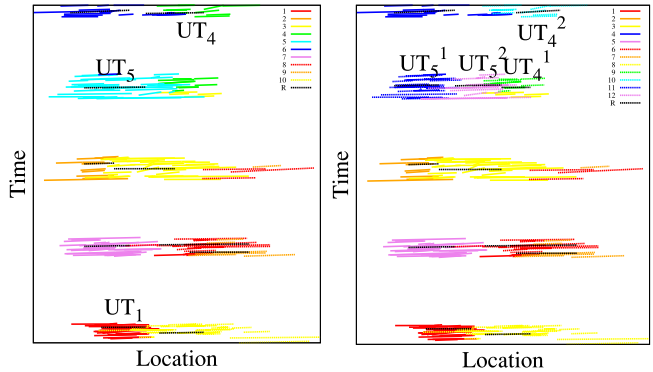


Fig. 11. Drill-down on C_1 in Fig. 10.

traverse much the same path with the interarrival times following a Gaussian distribution $N(\mu, \sigma^2)$. The time periods of UT-patterns are disjoint, and a time gap exists between consecutive UT-patterns. Overall, to simulate different movement characteristics, we vary the number of UT-patterns, the number of trajectories, the variance of interarrival times, and the time gap between UT-patterns.

Synthetic Data Set 2. For scalability test in Section 6.4, large-scale synthetic data sets are generated using the network-based generator of moving objects [28]. The map of San Francisco is used for data generation. The *base* data set has 20,000 trajectories, and each trajectory contains 231 points on average. Thus, the data set has about 4.6 million points in total, which is sufficiently large to verify scalability. This base (1×) data set is replicated by 2∼5 times to generate larger data sets having 9.2 (2×), 13.8 (3×), 18.4 (4×), and 23.0 (5×) million points, respectively. More than one thousand initial UT-patterns are detected from the base data set.

$Exponential(1)$ is used for modeling the code length. The best value of the rate parameter of the exponential distribution was found empirically to be unity [24]. The relaxation factor γ, which is related to only roll-up operations, is set to be 0.95. The lower the value of γ is, the more the UT-patterns are merged by roll-up. A value between in $[0.90, 0.95]$ works well in most cases. In addition, two parameters for sub-trajectory clustering are set using the heuristic proposed by Lee *et al.* [7], e.g., $\varepsilon = 39, MinLns = 8$ for the deer data set and $\varepsilon = 34, MinLns = 9$ for the elk data set. If other values are used for ε or $MinLns$, some useful UT-patterns could be missing, due to inappropriate sub-trajectory clustering, as discussed in Appendix C (available in the online supplemental material).

All experiments are conducted on a Linux server with a quad-core processor and 16 GBytes of main memory. The clock speed of the processor is 2.13 GHz. Ubuntu 12.04 runs on this server. Everything is implemented in C++ using gcc version 4.6.3.

6.2 UT-Patterns from Real Data Sets[4]

6.2.1 A Deer Data Set

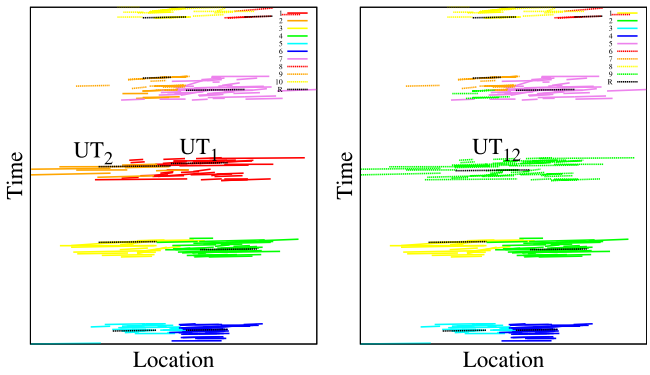
Four sub-trajectory clusters are discovered as in Fig. 10, and each two have similar traces in the opposite directions indicated by red and orange colors. The percentage of non-noise objects after clustering is 59.6 percent. Unless the percentage of non-noise objects is too low, the usefulness of the resulting UT-patterns indeed depends on the data set and application. Twenty nine initial UT-patterns are discovered in total from the four clusters. Two UT-patterns UT_1 and UT_5 from C_1 are displayed as well. They overlap in space, but not in time. UT_1 in blue color is time-constrained, and UT_5 in cyan color time-relaxed.

To avoid redundant explanation, the results for only C_1 and C_2 are shown in Figs. 11 and 12. Two sub-figures in the left column are obtained after the first phase (initial pattern discovery), and those in the right column after the second phase (granularity adjustment). The UT-patterns (i.e., sets of trajectory partitions) are displayed in the $(location, time)$-space. Here, a color and line type (either solid or dashed) represents a UT-pattern. A black line segment indicates its reference movement. These figures are *not* the output of the framework, but are presented to give the reader a better understanding of drill-down and roll-up operations. The final output is a set of UT-patterns of the form $UT_i = \langle \overrightarrow{R_i}, \mathbb{L}_i \rangle$, each of which is optionally classified into one of the three types, or a set of movement paths on a map with their values of the temporal cohesion.

Fig. 11 shows a set of UT-patterns from C_1 before and after drill-down. Drill-down continues as long as there exists a UT-pattern satisfying Definition 5. Fig. 11b is the snapshot obtained when drill-down is no longer applicable. Two UT-patterns UT_4 and UT_5 in Fig. 11a are split into (UT_4^1, UT_4^2) and (UT_5^1, UT_5^2) in Fig. 11b, respectively. The drill-down dimension is automatically chosen so that the MDL cost is minimized: UT_4 is split along the time dimension, and UT_5 along the location dimension. Due to this drill-down, two smaller time-relaxed patterns UT_5^1 and UT_5^2 are derived from a time-relaxed pattern UT_5. It is observed that $r(UT_5^1) = 0.36$ and $r(UT_5^2) = 0.49$, showing that the temporal cohesion increases by the drill-down.

Fig. 12 shows a set of UT-patterns from C_2 before and after roll-up. Roll-up continues as long as there exists a pair of UT-patterns satisfying Definition 6. Fig. 12b is the snapshot obtained when roll-up is no longer applicable. Two UT-patterns UT_1 and UT_2 in Fig. 12a are merged into one UT-pattern UT_{12} in Fig. 12b. Due to this roll-up, one larger

4. We kindly recommend the reader to see color images in the electronic file.



(a) Before roll-up. (b) After roll-up.

Fig. 12. Roll-up on C_2 in Fig. 10.

time-relaxed pattern is derived from one time-independent pattern and one time-constrained pattern. It is observed that $r(UT_1) = 0.02, r(UT_2) = 0.56$, and $r(UT_{12}) = 0.13$. When we decrease the value of γ from 0.95 to $0.90, UT_3$ and UT_8, which are the two topmost UT-patterns in Fig. 12, are merged. This merger looks reasonable since the two UT-patterns overlap in time and space. Afterward no more merger happened until $\gamma = 0.80$. Thus, we confirm that reasonable mergers take place with a value of γ between 0.90 and 0.95.

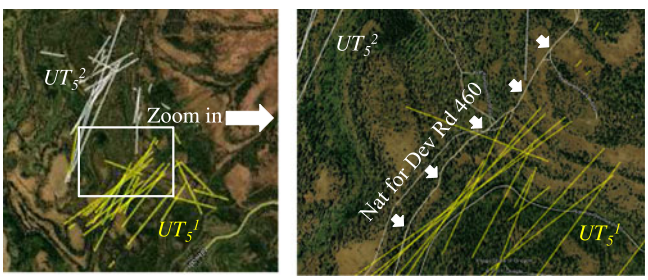
6.2.2 An Elk Data Set

Eighteen sub-trajectory clusters are discovered, and they are used as the basis of pattern discovery. The percentage of non-noise objects after clustering is 39.0 percent. Forty-nine initial UT-patterns are discovered in total from the 18 clusters. The detailed results for the elk data set was moved to Appendix D (available in the online supplemental material) because of space limitations.

6.2.3 Usefulness of the Pattern Forest

An example is given to show usefulness of the pattern forest in the real world. Let's look at the drill-down result of Fig. 11. Fig. 13a shows the UT-pattern UT_5 overlaid on a satellite image, which is generated by Google Earth. The deer in this UT-pattern move from southwest to northeast, and UT_5 is split into UT_5^1 and UT_5^2. Fig. 13b zooms in the rectangle of Fig. 13a to show a detailed look of the boundary between UT_5^1 and UT_5^2.

It is worth noting that "Nat for Dev Rd 460" divides UT_5 into UT_5^1 and UT_5^2. The trajectory partitions of UT_5^1 are located on the right of the road, and those of UT_5^2 on the left



(a) UT_5. (b) A detailed look.

Fig. 13. Detailed investigation of a drill-down result in the deer data.

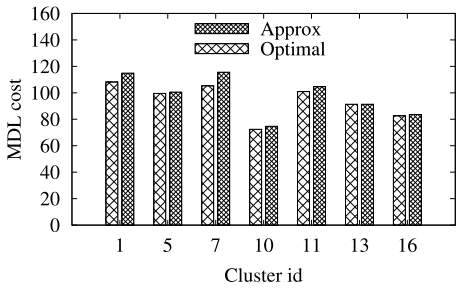


Fig. 14. Approximation Error of Algorithm 2.

of the road. A user would *not* know that a herd of deer moved at both sides of the road if the user did not look into *every* pattern including UT_5. Due to a possibly large number of patterns, checking everything could be very tedious. In contrast, after performing drill-down, the user can easily notice that UT_5 is divided into UT_5^1 and UT_5^2 by checking the pattern forest. Thus, the user conjectures that there must be something in UT_5 and will examine it in more detail. Then, the user finds out that Rd 460 impacted greatly on the movements of the deer. This kind of information is known to be valuable in studying animal behaviors.

6.3 Accuracy Results

6.3.1 Approximation Error

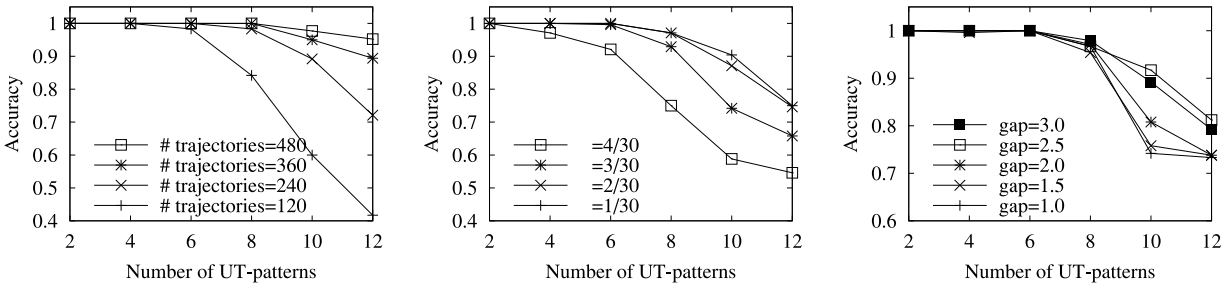
Since Algorithm 2 is an approximate algorithm, we would like to empirically show that the approximation error is negligible. In Fig. 14, for each cluster from the elk data set, the first bar indicates the MDL cost of the approximate solution, and the second bar indicates that of the optimal solution. The approximation error, which is the difference between the two bars, is shown to be very small—3.58 percent on the average. Here, only small clusters whose size is less than 25 are selected, because it is not feasible to find the optimal solution for a large cluster owing to an enormous number[5] of possible partitions of a cluster.

6.3.2 Accuracy of UT-pattern Discovery

To show that UT-Pattern Mine works well for diverse data sets with different movement characteristics, we use synthetic data sets created by varying four control parameters, as described in Section 6.1. We generate a sub-trajectory cluster seven times per the values of the control parameters. We measure the accuracy of UT-pattern discovery and report the average of the five numbers excluding the minimum and maximum numbers. Here, the *accuracy* is defined as the ratio of the number of the trajectory partitions allocated to the correct UT-pattern to the total number of trajectory partitions in a cluster. This set of experiments was done using Synthetic Data Set 1.

In the three plots of Fig. 15, the accuracy increases as the number of UT-patterns decreases. The probability that a trajectory partition is allocated to the correct UT-pattern is higher when there are fewer UT-patterns. The accuracy was mostly 100 percent when the number of UT-patterns per

5. For example, the number of the partitions of a set whose cardinality is 26 reaches 49631246523618756274, which is practically impossible to enumerate all of them. Please refer to the *Bell numbers* for details.



(a) Effect of the number of trajectories. (b) Effect of the variation of timestamps. (c) Effect of the gap between UT-patterns.

Fig. 15. Accuracy of finding UT-patterns.

cluster was less than or equal to 6. Overall, the accuracy was very high (80 $\sim$ 100 percent) for various settings except a few unusual cases.

Fig. 15a shows the effect of the number of trajectories per cluster on accuracy. Intuitively, a larger number of trajectories per cluster implies stronger support for the existence of patterns since the cluster is a group of similar sub-trajectories. When the number of trajectories was about 500, the accuracy was kept close to 100 percent until the number of UT-patterns became 10. This property that handles larger clusters well is indeed desirable in the big data era.

Fig. 15b shows the effect of the variation of timestamps on accuracy. Interarrival times were generated to follow a Gaussian distribution $N(\mu, \sigma^2)$; $[\mu - 3\sigma, \mu + 3\sigma]$ accounts for about 99.7 percent of the interarrival times by definition. If σ is very small, the ith timestamp is roughly $i \cdot \mu$. On the other hand, as σ gets larger, moving objects reach similar locations at more diverse times. Thus, pattern discovery is more demanding when the variation is large, and the accuracy drops more rapidly with the growing number of UT-patterns when σ becomes larger.

Fig. 15c shows the effect of the temporal gap between successive UT-patterns on accuracy. The length of a gap was varied between 1.0 $\sim$ 3.0 times of that of a UT-pattern. The accuracy drops more rapidly with the growing number of UT-patterns when the gap becomes smaller. This is because the UT-patterns become less discriminable when the gap is small.

6.4 Performance Issues and Scalability Results

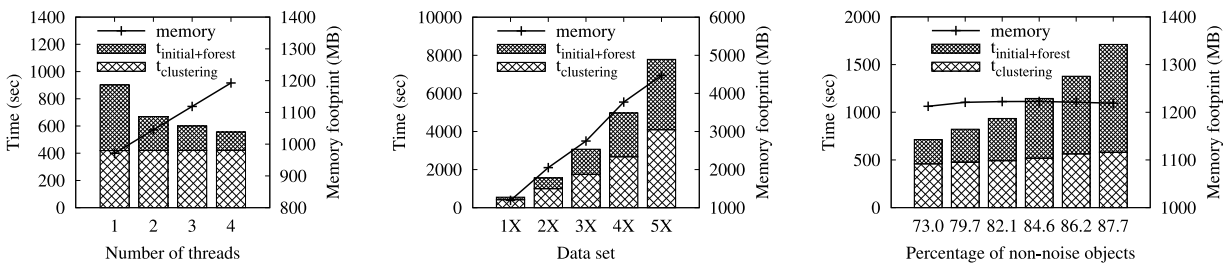
By taking advantage of recent hardware trends, our prototype has improved to run in parallel on multiple cores (i.e., threads) so that it efficiently supports large-scale data sets. The unifying framework has a very nice property for parallelization. After constructing sub-trajectory clusters, subsequent operations are executed separately for each cluster. Thus, each cluster naturally becomes a unit of a task, and each task is assigned to a thread.

Please recall that there are three components of UT-Pattern Mine in Algorithm 1: sub-trajectory clustering (Lines 2 $\sim$ 3), *Initial Pattern Generation* (Lines 4 $\sim$ 9), and *Pattern Forest Construction* (Lines 12 $\sim$ 13). The time complexity of the first component is $O(n \log n)$ since a spatial index is used for finding a neighborhood in sub-trajectory clustering; however, that of the second component is $O(n^2)$ by Lemma 2.

This parallelization accelerates the two components executed after sub-trajectory clustering, thus solving the scalability issue of the most problematic component, *Initial Pattern Generation*, of the time complexity $O(n^2)$. Sub-trajectory clustering is not improved by parallelization, but its time complexity $O(n \log n)$ is already reasonable. Four threads are created for experiments since our server is equipped with a quad-core CPU. Each thread alternatively takes a cluster and works for the cluster.

Processing time is measured separately for sub-trajectory clustering and *Initial Pattern Generation + Pattern Forest Construction*. It is hard to divide the latter into the two components since they run in parallel. The former is denoted by $t_{clustering}$, and the latter by $t_{initial+forest}$. This set of experiments was done using Synthetic Data Set 2.

Fig. 16a shows the effect of the number of threads on processing time and memory footprint in the 1$\times$ data set. The parallelization is shown to improve performance significantly: $t_{initial+forest}$ improves by 3.59 times as the number of threads increases from 1 to 4. $t_{clustering}$, however, remains constant since sub-trajectory clustering is not parallelized. The speed-up ratio is expected to increase further as the number of CPU cores increases. Memory footprint also increases linearly as the number of threads increases, because more threads consume memory concurrently.



(a) Varying the number of threads. (b) Varying the size of the data set. (c) Varying the ratio of non-noise objects.

Fig. 16. Scalability of the unifying framework.

TABLE 3
Number of Flock Patterns for the Deer Data ($r = 39$)

k	2	3	4	5	6	7	8	9	10
$m = 2$	69	50	46	45	44	41	39	38	32
$m = 3$	24	21	16	14	12	10	10	9	9
$m = 4$	5	3	3	3	1	0	0	0	0
$m = 5$	1	0	0	0	0	0	0	0	0
$m = 6$	0	0	0	0	0	0	0	0	0

TABLE 4
Number of Flock Patterns for the Elk Data ($r = 34$)

k	2	3	4	5	6	7	8	9	10
$m = 2$	165	128	93	72	60	53	42	39	33
$m = 3$	74	60	47	34	25	20	15	13	8
$m = 4$	42	28	20	12	11	10	6	5	1
$m = 5$	18	11	10	8	7	6	4	2	0
$m = 6$	8	7	7	5	3	3	1	0	0
$m = 7$	8	6	5	3	1	1	0	0	0
$m = 8$	4	4	3	2	0	0	0	0	0
$m = 9$	3	2	2	0	0	0	0	0	0

Fig. 16b shows the effect of the data size on processing time and memory footprint. $t_{clustering}$ increases almost linearly, whereas $t_{initial+forest}$ exhibits quadratic growth as the data size increases. Please note that this quadratic growth will *lessen* as the degree of parallelism increases *by using more CPU cores*. Since *Initial Pattern Generation* receives only the trajectory partitions included in clusters, its processing time is affected by the proportion of trajectory partitions belonging to clusters, i.e., the percentage of *non*-noise objects. This percentage is 61.1 percent in Fig. 16b. In addition, memory footprint increases proportionally to the data size.

Fig. 16c shows the effect of the percentage of *non*-noise objects on processing time and memory footprint in the $1\times$ data set. The value of the parameter ε is set differently to control the percentage of non-noise objects. It can be regarded that spatial constraints become more strict as this percentage decreases. $t_{initial+forest}$ gets shorter as the percentage decreases since more trajectory partitions are filtered out early on. This verifies the effectiveness of the stepwise approach in Section 4.1. Moreover, when the percentage is small (e.g., below 50 percent), the quadratic growth of $t_{initial+forest}$ may not be a big issue. In addition, memory footprint is not affected by the percentage because memory consumption during sub-trajectory clustering is dominant in the entire process.

6.5 Comparison with Flock Patterns

We finally compare our UT-patterns with the flock patterns (classified as time-constrained). There are several variations of flock patterns, and we adopt the definition proposed by Benkert et al. [14] and Gudmundsson and Kreveld [15].[6] Since it is impossible to do an apples-to-apples comparison between the two different methods, we just would like to claim that the flock patterns are sometimes too restrictive to find useful patterns. This restrictiveness indeed shows the necessity of a unifying framework.

Table 3 shows the number of flock patterns found in the deer data set. As the minimum number of objects (m) increases or the minimum duration (k) increases, a smaller number of flock patterns are found. When m becomes greater than 5, no flock pattern is found from the data set. That is, no information is available on large patterns involving more than five objects. Please note that $MinLns = 8$ is used for our unifying framework, thereby enabling us to find UT-patterns involving eight or more objects—i.e., with harder constraints on the number of trajectories. From this quick observation, we believe that detecting only flock patterns can result in loss of some useful patterns. The visualization of the flock patterns is available in Appendix E (available in the online supplemental material).

Table 4 shows that a similar result is obtained for the elk data set. $MinLns = 9$ is used for our unifying framework, and 49 UT-patterns are found.

6. Please note that these two papers proposed almost the same definition.

7 Conclusions

A unifying framework for mining UT-patterns has been proposed in this paper. Based on this framework, a pattern mining algorithm UT-Pattern Mine has been developed. The main advantage of the algorithm is the detection of the trajectory patterns of *various temporal tightness*: time-constrained, time-relaxed, and time-independent. The algorithm first discovers initial UT-patterns using the intuitive information-theoretic principle of maximizing data compression and then constructs a pattern forest by drill-down and roll-up to discover more patterns. Experiments using real-world data sets show that UT-Pattern Mine easily discovers various types of trajectory patterns. Its accuracy is shown to be very high ($80 \sim 100$ percent) for various movement characteristics. Its performance has improved by parallelization on multi-core CPUs to support large-scale data sets. Overall, we believe that our framework has good usability since users do not have to pay attention to the types of trajectory patterns hidden in their data sets.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (2012R1A1A1012954).

References

[1] P. Bakalov, M. Hadjieleftheriou, and V. J. Tsotras, "Time relaxed spatiotemporal trajectory joins," in *Proc. 13th ACM Int. Symp. Geograph. Inf. Syst.*, Bremen, Germany, Nov. 2005, pp. 182–191.

[2] F. Giannotti, M. Nanni, F. Pinelli, and D. Pedreschi, " Trajectory pattern mining,, " in *Proc. 13th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, San Jose, CA, USA, Aug. 2007, pp. 330–339.

[3] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen, "Discovery of convoys in trajectory databases," *Proc. VLDB Endowment*, vol. 1, no. 1, pp. 1068–1080, Sep. 2008.

[4] P. Laube and S. Imfeld, "Analyzing relative motion within groups of trackable moving point objects," in *Proc. 2nd Int. Conf. Geograph. Inf. Sci.*, Boulder, CO, USA, Sep. 2002, pp. 132–144.

[5] P. Laube, M. J. van Kreveld, and S. Imfeld, "Finding REMO—Detecting relative motion patterns in geospatial lifelines," in *Proc. 11th Int. Symp. Spatial Data Handling*, Leicester, U.K., Aug. 2004, pp. 201–214.

[6] P. Laube, S. Imfeld, and R. Weibel, "Discovering relative motion patterns in groups of moving point objects," *Int. J. Geograph. Inf. Sci.*, vol. 19, no. 6, pp. 639–668, Jul. 2005.
[7] J.-G. Lee, J. Han, and K.-Y. Whang, "Trajectory clustering: A partition-and-group framework," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, Beijing, China, Jun. 2007, pp. 593–604.
[8] J.-G. Lee, J. Han, and X. Li, "Trajectory outlier detection: A partition-and-detect framework," in *Proc. 24th Int. Conf. Data Eng.*, Cancun, Mexico, Apr. 2008, pp. 140–149.
[9] J.-G. Lee, J. Han, X. Li, and H. Gonzalez, "TraClass: Trajectory classification using hierarchical region-based and trajectory-based clustering," *Proc. VLDB Endowment*, vol. 1, no. 1, pp. 1081–1094, Sep. 2008.
[10] Z. Li, B. Ding, J. Han, and R. Kays, "Swarm: Mining relaxed temporal moving object clusters," *Proc. VLDB Endowment*, vol. 3, no. 1, pp. 723–734, Sep. 2010.
[11] M. Nanni and D. Pedreschi, "Time-focused clustering of trajectories of moving objects," *J. Intell. Inf. Syst.*, vol. 27, no. 3, pp. 267–289, Nov. 2006.
[12] D. Sacharidis, K. Patroumpas, M. Terrovitis, V. Kantere, M. Potamias, K. Mouratidis, and T. K. Sellis, "On-line discovery of hot motion paths," in *Proc. 11th Int. Conf. Extending Database Technol.*, Nantes, France, Mar. 2008, pp. 392–403.
[13] H.-P. Tsai, D.-N. Yang, and M.-S. Chen, "Mining group movement patterns for tracking moving objects efficiently," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 2, pp. 266–281, Feb. 2011.
[14] M. Benkert, J. Gudmundsson, F. Hubner, and T. Wolle, "Reporting flock patterns," in *Proc. 14th Eur. Symp. Algorithms*, Zurich, Switzerland, Mar. 2006, pp. 660–671.
[15] J. Gudmundsson and M. J. van Kreveld, "Computing longest duration flocks in trajectory data," in *Proc. 14th ACM Int. Symp. Geograph. Inf.. Syst.*, Arlington, VA, USA, Nov. 2006, pp. 35–42.
[16] P. Kalnis, N. Mamoulis, and S. Bakiras, "On discovering moving clusters in spatio-temporal data," in *Proc. 9th Int. Symp. Spatial Temporal Databases*, Angra dos Reis, Brazil, Aug. 2005, pp. 364–381.
[17] Y. Li, J. Han, and J. Yang, "Clustering moving objects," in *Proc. 10th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Seattle, WA, USA, Aug. 2004, pp. 617–622.
[18] P. D. Grünwald, I. J. Myung, and M. A. Pitt, *Advances in Minimum Description Length: Theory and Applications*. Cambridge, MA, USA: MIT Press, 2005.
[19] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. 2nd Int. Conf. Knowl. Discovery Data Mining*, Portland, OR, USA, Aug. 1996, pp. 226–231.
[20] C. Böhm, C. Faloutsos, J.-Y. Pan, and C. Plant, "Robust information-theoretic clustering," in *Proc. 12th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Philadelphia, PA, USA, Aug. 2006, pp. 65–75.
[21] X. Yu and M. K. H. Leung, "Shape recognition using curve segment hausdorff distance," in *Proc. 18th Int. Conf. Pattern Recognit.*, Hong Kong, China, Aug. 2006, pp. 441–444.
[22] T. C. M. Lee, "An introduction to coding theory and the two-part minimum description length principle," *Int. Stat. Rev.*, vol. 69, no. 2, pp. 169–183, Aug. 2001.
[23] H. Bischof, A. Leonardis, and A. Selb, "MDL principle for robust vector quantization," *Pattern Anal. Appl.*, vol. 2, no. 1, pp. 59–72, Apr. 1999.
[24] A. R. Kelly and E. R. Hancock, "Grouping line-segments using eigenclustering," in *Proc. 11th British Mach. Vis. Conf.*, Bristol, U.K. , Sep. 2000, pp. 59–68.
[25] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd ed. San Mateo, CA, USA: Morgan Kaufmann, 2011.
[26] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*, 2nd ed. Mahwah NJ, USA: Lawrence Erlbaum Associates, Publishers, 1988.
[27] A. A. Ager, B. K. Johnson, J. W. Kern, and J. G. Kie, "Daily and seasonal movements and habitat use by female rocky mountain elk and mute deer," *J. Mammal.*, vol. 84, no. 3, pp. 1076–1088, Aug. 2003.
[28] T. Brinkhoff, "A framework for generating network-based moving objects," *GeoInformatica*, vol. 6, no. 2, pp. 153–180, Jun. 2002.

Jae-Gil Lee received the PhD degree from KAIST in 2005. He is an associate professor in the Department of Knowledge Service Engineering, KAIST. He worked at IBM Almaden Research Center before joining KAIST. He is a member of the IEEE.

Jiawei Han is an Abel Bliss professor in the Department of Computer Science, University of Illinois at Urbana-Champaign. He has been researching into data mining, information network analysis, database systems, and data warehousing, with more than 600 journal and conference publications. He is a fellow of the ACM and IEEE. His book *Data Mining: Concepts and Techniques* has been used popularly as a textbook worldwide.

Xiaolei Li received the PhD degree in computer science from the University of Illinois at Urbana-Champaign. He is an engineering manager at Twitter, working on consumer products.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.