Community Trend Outlier Detection using Soft Temporal Pattern Mining

Manish Gupta[1], Jing Gao[2], Yizhou Sun[1], and Jiawei Han[1]

[1] UIUC, IL ({gupta58, sun22, hanj}@illinois.edu)
[2] SUNY, Buffalo, NY (jing@buffalo.edu)

Abstract. Numerous applications, such as bank transactions, road traffic, and news feeds, generate temporal datasets, in which data evolves continuously. To understand the temporal behavior and characteristics of the dataset and its elements, we need effective tools that can capture evolution of the objects. In this paper, we propose a novel and important problem in evolution behavior discovery. Given a series of snapshots of a temporal dataset, each of which consists of evolving communities, our goal is to find objects which evolve in a dramatically different way compared with the other community members. We define such objects as *community trend outliers*. It is a challenging problem as evolutionary patterns are hidden deeply in noisy evolving datasets and thus it is difficult to distinguish anomalous objects from normal ones. We propose an effective two-step procedure to detect community trend outliers. We first model the normal evolutionary behavior of communities across time using soft patterns discovered from the dataset. In the second step, we propose effective measures to evaluate chances of an object deviating from the normal evolutionary patterns. Experimental results on both synthetic and real datasets show that the proposed approach is highly effective in discovering interesting community trend outliers.

1 Introduction

A large number of applications generate temporal datasets. For example, in our everyday life, various kinds of records like credit, personnel, financial, judicial, medical, etc. are all temporal. Given a series of snapshots of a temporal dataset, analysts often perform community detection for every snapshot with the goal of determining the intrinsic grouping of objects in an unsupervised manner. By analyzing a series of snapshots, we can observe that these communities evolve in a variety of ways – communities contract, expand, merge, split, appear, vanish, or re-appear after a time period. Most of the objects within a community follow similar evolution trends which define the evolution trends of the community. However, evolution behavior of certain objects is quite different from that of their respective communities. Our goal is to detect such anomalous objects as *Community Trend Outliers* (or *CTOutliers*) given community distributions of each object for a series of snapshots. In the following, we present *CTOutlier* examples and discuss importance of identifying such outliers in real applications.

***CTOutlier* Examples**

Consider the co-authorship network for the four areas in CS: data mining (DM), information retrieval (IR), databases (DB) and machine learning (ML). Every author can be associated with a soft distribution of their belongingness to each of these areas. One such sequence of distributions could be $\langle$*1:(DB:1*, *DM:0)*, *2:(DB:0.8*, *DM:0.2)*, *3:(DB:0.5*, *DM:0.5)*, *4:(DB:0.1*, *DM:0.9)*$\rangle$. Such a pattern represents the trend of a part of DB researchers gradually moving into the DM community. While most of the authors follow one of such popular patterns of evolution with respect to their belongingness distributions across different snapshots, evolution of the distributions associated with some of the other authors is very different. Such authors can be considered as *CTOutliers*.

As another example, consider all the employees working for a company. For each employee, one can record the amount of time spent in Office work, Household work, Watching TV, Recreation and Eating, for a month. Across different days, one can observe a trend where a person spends most of his time in office work on weekdays and in household work on weekends. Similarly, there could be different patterns for night workers. However, there could be a very few employees who follow different schedule for a few days (e.g., if an employee is sick, he might spend a lot of his time at home even on weekdays). In that case, that employee can be considered as a *CTOutlier*.

Besides these examples, interesting examples of *CTOutliers* can be commonly observed in real-life scenarios. A city with a very different sequence of land use proportion (agriculture, residential, commercial, open space) changes across time, compared to change patterns for other cities can be a *CTOutlier*. E.g., most of the cities show an increase in residential and commercial areas and reduction in agriculture areas over time. However, some cities may get devastated by natural calamities disturbing the land use drastically. Applications where *CTOutliers* could be useful depends on the specific domain. Outlier detection may be useful to explain future behavior of outlier sequences. E.g., one may analyze the diet proportion of carbohydrates, proteins and fats for a city across time. A city showing trends of increasing fats proportion in diet, may have a population with larger risk of heart attacks. *CTOutlier* detection may be used to trigger action in monitoring systems. E.g., in a chemical process, one may expect to observe a certain series of distribution of elements across time. Unexpected deviations from such a series, may be used to trigger a corrective action.

Brief Overview of *CTOutlier* Detection

We study the problem of detecting *CTOutliers* given community distributions of each object for a series of snapshots of a temporal dataset. Input for our problem thus consists of a soft sequence (i.e., a sequence of community distributions across different timestamps) associated with each object. For example, in DBLP, an author has a sequence of research-area distributions across years. The number of communities could change over time, so a soft sequence can consist of distributions of different sizes at different timestamps.

This problem is quite different from trajectory outlier detection [8,14,19] because: (1) In this problem, soft sequences consist of distributions obtained

by community detection rather than locations in trajectory outlier detection. (2) Soft patterns could be gapped and multi-level while trajectories are usually continuous. (3) Unlike trajectory based systems, we cannot rely on additional features such as speed or direction of motion of objects. Moreover, existing efforts on detecting outliers in evolving datasets [4,15] cannot detect temporal community trend based outliers because they do not involve any notion of communities. In the first step of discovering normal trends, probabilistic sequential pattern mining methods [9,21] can be used to extract temporal patterns, however the patterns detected by these methods are "hard patterns" which are incapable of capturing subtle trends, as discussed in Sec. 2.

We propose to tackle this problem using a two-step approach: pattern extraction and outlier detection. In the pattern extraction phase, we first perform clustering of soft sequences for individual snapshots. The cluster centroids obtained for each timestamp represent the length-1 frequent soft patterns for that timestamp. Support of a length-1 pattern (cluster centroid) is defined as the sum of a function of the distance between a point (sequence) and cluster centroid, over all points. The Apriori property [5] is then exploited to obtain frequent soft patterns of length ≥ 2. After obtaining the frequent soft patterns, outlier detection is performed. A sequence is considered a *CTOutlier* if it deviates a lot from its best matching pattern for many combinations of timestamps.

In summary, we make the following contributions in this paper.

- We introduce the notion of Community Trend Outliers *CTOutliers*. Such a definition tightly integrates the notion of deviations with respect to both the temporal and community dimensions.
- The proposed integrated framework consists of two novel stages: efficient discovery of a novel kind of patterns called *soft patterns*, and analysis of such patterns using a new outlier detection algorithm.
- We show interesting and meaningful outliers detected from multiple real and synthetic datasets.

Our paper is organized as follows. *CTOutliers* are defined as objects that deviate significantly from a novel kind of patterns. Thus, pattern discovery is the basis of the outlier detection step. Hence, first we introduce the notion of *soft patterns* and develop our method to extract temporal community trends in the form of frequent soft patterns in Sec. 2. Then, in Sec. 3, we discuss the algorithm for *CTOutlier* detection which exploits the extracted patterns to compute outlier scores. We discuss the datasets and results with detailed insights in Sec. 4. Finally, related work and conclusions are presented in Sec. 5 and 6 respectively.

2 Temporal Trends Extraction

In this section, we discuss how to extract soft patterns as temporal trends, which serve as the basis of community trend outlier detection in Sec. 3. We first introduce important definitions in Sec. 2.1. Next, we will carefully define support

for such soft patterns and discuss how to extract them from the soft sequence data in Sec. 2.2 and Sec. 2.3.

2.1 Problem Formulation

Let us begin with a few definitions. We will use the toy dataset shown in Fig. 1 as a running example. The toy dataset has 15 objects which consist of 4 patterns (▲, ◀, ▼, ▶) and two outliers (■, ★) across 3 timestamps. There are 3 (A, B, C), 3 (D, E, F) and 4 (G, H, I, J) clusters for the 3 timestamps respectively.

Soft Sequence: A soft sequence for object o is denoted by $S_o = \langle S_{1_o}, S_{2_o}, \ldots, S_{T_o} \rangle$ where S_{t_o} denotes the community belongingness probability distribution for object o at time t. In Fig. 1, for the point marked with a → across all 3 timestamps, the soft sequence is $\langle$*1: (A:0.1, B:0.8, C:0.1), 2: (D:0.07, E:0.08, F:0.85), 3: (G:0.08, H:0.8, I:0.08, J:0.04)*$\rangle$. Soft sequences for all objects are defined on the same set of T timestamps; all sequences are synchronized in time. For a particular timestamp, the data can be represented as $S_t = [S_{t_1}, S_{t_2}, \ldots, S_{t_N}]^T$.

Soft Pattern: A *soft pattern* $p = \langle P_{1_p}, P_{2_p}, \ldots, P_{T_p} \rangle$ is a sequence of probability distributions across L_p (possibly gapped) out of T timestamps, with support $\geq min\_sup$. Here, P_{t_p} denotes the community (probability) distribution at timestamp t. A soft pattern p defined over a set τ_p of timestamps is a representative of a set of sequences (similar to each other for timestamps $\in \tau_p$) grouped together by clustering over individual snapshots. Support of p is naturally defined proportional to the number of sequences it represents (Sec. 2.2 and 2.3). E.g., the pattern $p = \langle$*1:(DB:1, DM:0), 2:(DB:0.5, XML:0.3, DM:0.2), 4:(DB:0.1, DM:0.9)*$\rangle$ is defined over 3 timestamps 1, 2 and 4 and so L_p=3. In Fig. 1, $\langle$*1:(A:0.2, B:0.2, C:0.6), 2:(D:0.9, E:0.05, F:0.05), 3:(G:0.9, H:0.03, I:0.03, J:0.04)*$\rangle$ is a soft pattern covering the ▶ points.

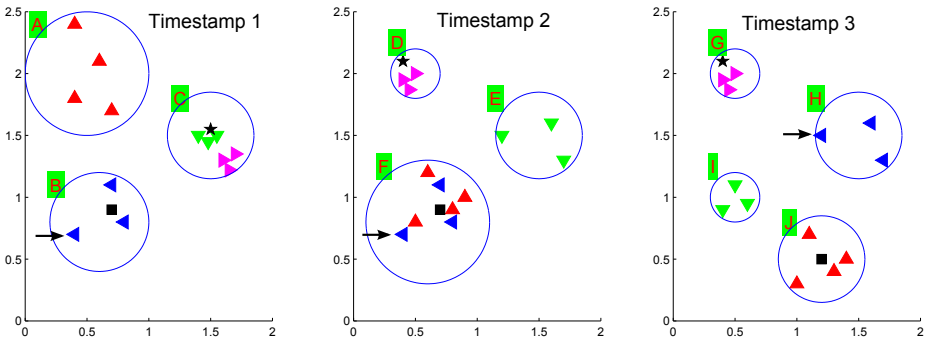

Fig. 1. Three Snapshots of a Toy Dataset

CTOutlier: An object o is a *CTOutlier* if its outlier score ranks within the top-k. The outlier score of an object o captures the degree to which it deviates from the best matching soft pattern across different combinations of timestamps (details in Sec. 3). E.g., outliers ★ and ■ deviate from the ▶ and ◀ patterns for the first and the third timestamps respectively.

The *CTOutlier* detection problem can then be specified as follows.

Input: Soft sequences (each of length T) for N objects, denoted by matrix S.

Output: Set of *CTOutlier* objects.

Before presenting the soft pattern discovery problem and the methodology for their efficient discovery, we discuss the reasons why we use soft rather than hard patterns to represent temporal trends.

Why use Soft Patterns?

Given soft sequence data, one can use probabilistic sequential pattern mining [9,21] to discover hard sequential patterns. In the DBLP example, a hard pattern can be $\langle$*1:DB*, *2:DB*, *3:DM*, *4:DM*$\rangle$, which expresses major transitions for an author changing his research area from DB to DM. However, most trends in temporal datasets are subtle and thus cannot be expressed using hard patterns. E.g., in Fig. 1, evolution of ▶ objects can be characterized by hard pattern $\langle$*1:C*, *2:D*, *3:G*$\rangle$. However, at the first timestamp, ▶ objects lie on the boundary of cluster C and are much different from the core cluster-C ▼ objects. Such subtle difference cannot be encoded in hard patterns.

Different from hard patterns, soft patterns contain a richer set of information. Consider two soft patterns which are not related to such drastic changes: p_1 = $\langle$*1:(DM:0.8*, *IR:0.2)*, *2:(DM:0.6*, *IR:0.4)*$\rangle$ and p_2 = $\langle$*1:(DM:0.48*, *DB:0.42*, *IR:0.1)*, *2:(DM:1)*$\rangle$ both of which map to the same hard pattern $\langle$*1:DM*, *2:DM*$\rangle$. This is not reasonable because clearly the two patterns represent two different semantic trends. On the other hand, let $\langle$*1:DB*, *2:DM*$\rangle$ denote a hard pattern, from which we cannot identify how the communities indeed evolve. The temporal trend could be that $\langle$*1:(DB:0.5*, *Sys:0.3*, *Arch:0.2)*, *2:(DM:0.5*, *DB:0.3*, *Sys:0.2)*$\rangle$ is frequent but $\langle$*1:(DB:0.9*, *Sys:0.1)*, *2:(DM:0.9*, *DB:0.1)*$\rangle$ is not frequently observed. Therefore, soft patterns are more desirable because they can express that core-DB authors did not move to DM, although some non-core-DB researchers started showing interest in DM. In Fig. 1, hard pattern based detection will miss ★ outlier, while soft pattern based detection will correctly identify it. To prevent such loss of information when using hard patterns, we propose to model temporal trends as soft patterns.

Soft Pattern Discovery Problem

The soft pattern discovery problem can be summarized as follows.

Input: Soft sequences (each of length T) for N objects, denoted by matrix S.

Output: Set P of frequent soft patterns with support $\geq$ *min_sup*.

Next, we will carefully define support for such soft patterns and discuss how to extract them from the soft sequence data. We will first discuss how to find length-1 patterns, and then discuss how to find patterns with length ≥ 2.

2.2 Extraction of Length-1 Soft Patterns

The task of discovering length-1 soft patterns for a particular timestamp is to find representative distributions from a space of N probability distributions where N = #objects. We solve this problem by performing *clustering* (we use *XMeans* [22]) on distributions. *The cluster centroids for such clusters are a representative of all the points within the cluster. Thus, a cluster centroid can be used to uniquely represent a length-1 soft pattern.* In the example shown in Fig. 1, for the first timestamp, *XMeans* discovers 4 clusters with centroids $(A:0.85, B:0.05, C:0.1)$, $(A:0.03, B:0.9, C:0.07)$, $(A:0.03, B:0.02, C:0.95)$ and $(A:0.2, B:0.2, C:0.6)$. Each of these cluster centroids represents a length-1 soft pattern (▲, ◀, ▼, ▶ resp).

Defining Support for Length-1 Soft Patterns

Traditionally, support for a sequential pattern is defined as the number of sequences which contain that pattern. Similarly, we can define support for a soft pattern (cluster centroid) P_{t_p} in terms of the degree to which the sequences (points) belong to the corresponding cluster (Eq. 1). Let $Dist(P_{t_p}, S_{t_o})$ be some distance measure (we use Euclidean distance) between the sequence distribution for object o at time t and the cluster centroid for pattern p at time t. Let $maxDist(P_{t_p})$ be the maximum distance of any point in the dataset from the centroid P_{t_p}. Then the support for the length-1 pattern p can be expressed as follows.

$$sup(P_{t_p}) = \sum_{o=1}^{N} \left[1 - \frac{Dist(P_{t_p}, S_{t_o})}{maxDist(P_{t_p})} \right] \tag{1}$$

From Eq. 1, one can see that an object which is closer to the cluster centroid contributes more to the support of a pattern (corresponding to that cluster centroid) compared to objects far away from the cluster centroid. E.g., at the first timestamp, cluster centroid $(A:0.85, B:0.05, C:0.1)$ gets good support from all the ▲ points because they are very close to it, but gets small amount of support from other points, based on their distance from it. Patterns with support $\geq min\_sup$ are included in the set of frequent patterns P.

A clustering algorithm may break a semantic cluster into multiple sub-clusters. Hence, some of the resulting clusters may be very small and so if we define support for a cluster centroid based on just the points within the cluster, we might lose some important patterns for lack of support. Hence, we define support for a cluster centroid using contributions from all the points in the dataset. To prevent the impact of distance based outliers (when computing $maxDist$), it might be beneficial to remove such outliers from each snapshot, as a preprocessing step.

2.3 Extraction of Longer Soft Patterns

Here we will discuss how to define support for longer patterns and compute them efficiently.

Defining Support for Longer Soft Patterns

The support by an object o for a pattern p is defined in terms of its support with respect to each of the timestamps in τ_p as shown below.

$$sup(p) = \sum_{o=1}^{N} \prod_{t \in \tau_p} \left[1 - \frac{Dist(P_{t_p}, S_{t_o})}{maxDist(P_{t_p})} \right] \quad (2)$$

Intuitively, an object for which the community distribution lies close to the cluster centroids of the pattern across a lot of timestamps will have higher support contribution for the pattern compared to objects which lie far away from the pattern's cluster centroids. As an example, consider the pattern $\langle$*1:(A:0.85, B:0.05, C:0.1), 2:(D:0.1, E:0.2, F:0.7), 3:(G:0.01, H:0.02, I:0.03, J:0.94)*$\rangle$. The ▲ points contribute maximum support to this pattern because they lie very close to this pattern across all timestamps. Patterns with support $\geq min\_sup$ are included in the set P of frequent patterns.

Apriori Property

From Eqs. 1 and 2, it is easy to see that a soft pattern cannot be frequent unless all its sub-patterns are also frequent. Thus, the Apriori property [5] holds. This means that longer frequent soft patterns can be discovered by considering only those candidate patterns which are obtained from shorter frequent patterns. This makes the exploration of the sequence pattern space much more efficient.

Candidate Generation

According to Apriori property, candidate patterns of length ≥ 2 can be obtained by concatenating shorter frequent patterns. For each ordered pair (p_1, p_2) where p_1 and p_2 are two length-l frequent patterns, we create a length-$(l+1)$ candidate pattern if (1) p_1 excluding the first timestamp, matches exactly with p_2 excluding the last timestamp; and (2) the first timestamp in p_1 is earlier than the last timestamp in p_2. A candidate length-$(l+1)$ pattern is generated by concatenating p_1 with the last element of p_2.

3 Community Trend Outlier Detection

In this section, we will discuss how to exploit set P of frequent soft patterns obtained after pattern extraction (Sec. 2) to assign an outlier score to each sequence in the dataset. When capturing evolutionary trends, length-1 patterns are not meaningful, as they are defined on single snapshots. So, we remove them from set P. Although the input sequences are all of length T, each pattern could be of any length $\leq T$ and could be gapped. While a sequence represents a point distribution at each timestamp, a pattern represents a cluster centroid for each timestamp. Each cluster is associated with a support, and there might be other statistics to describe the cluster, such as maximum distance of any point from the centroid and average distance of all points within cluster from the centroid. Intuitively, patterns consisting of compact clusters (clusters with low average distances) with high support are the most important for outlier detection.

Outlier Detection Problem

Input: Set P of frequent soft patterns with support $\geq min\_sup$.

Output: Set of *CTOutlier* objects.

3.1 Pattern Configurations and Best Matching Pattern

A non-outlier object may follow only one frequent pattern while deviating from all other patterns. Hence, it is incorrect to compute outlier score for an object by adding up its outlierness with respect to each pattern, weighted by the pattern support. Also, it is not meaningful to compute outlier score just based on the best matching pattern. The reason is that often times, even outlier sequences will follow some length-2 pattern; but such a short pattern does not cover the entire length of the sequence. Therefore, we propose to analyze the outlierness of a sequence with respect to its different projections by dividing the pattern space into different configurations.

Configuration: A *configuration* c is simply a set of timestamps with size ≥ 2. Let P_c denote the set of patterns corresponding to the configuration c.

Finding Best Matching Pattern

A normal sequence generally follows a particular trend (frequent pattern) within every configuration. A sequence may have a very low match with most patterns but if it matches completely with even one frequent pattern, intuitively it is not an outlier. (Here we do not consider the situation of group outliers, where all sequences following a very different pattern could be called outlier.) Hence, we aim at finding the pattern which matches the sequence the most for that configuration. Note that patterns corresponding to big loose clusters match a large number of sequences, and thus we should somehow penalize such patterns over those containing compact clusters.

Based on the principles discussed above, we design the following matching rules. Let a pattern p be divided into two parts ϕ_{po} and θ_{po}. ϕ_{po} (θ_{po}) consists of the set of timestamps where sequence for object o and pattern p match (do not match) each other. E.g., considering pattern $p =$ ▶ and sequence $o =$ ★, $\theta_{po} = \{1\}$ and $\phi_{po} = \{2, 3\}$.

Match Score: We define the match score between an object o and a pattern p as follows.

$$match(p, o) = \sum_{t \in \phi_{po}} \frac{sup(P_{t_p}) \times sup(P_{t_p}, S_{t_o})}{avgDist(P_{t_p})} \tag{3}$$

where $avgDist(P_{t_p})$ is the average distance between the objects within the cluster and the cluster centroid P_{t_p}, and $sup(P_{t_p}, S_{t_o}) = 1 - \frac{Dist(P_{t_p}, S_{t_o})}{maxDist(P_{t_p})}$. This definition is reasonable because the score is higher if (1) the object and the pattern match for more timestamps; (2) pattern has higher support; (3) pattern contains compact clusters; and (4) object lies closer to the cluster centroid across various timestamps.

Best Matching Pattern: The best matching pattern bmp_{co} is the pattern $p \in P_c$ with the maximum match score $match(p, o)$. In the toy example, the best matching pattern for sequence ★ with respect to configuration $\{1, 2, 3\}$ is ▶.

3.2 Outlier Score Definition

Given a sequence, we first find the best matching pattern for every configuration and then define the outlier score as the sum of the scores of the sequence with respect to each configuration. The outlier score of object o is thus expressed as:

$$outlierScore(o) = \sum_{c=1}^{|C|} outlierScore(c, o) = \sum_{c=1}^{|C|} outlierScore(bmp_{co}, o) \quad (4)$$

where bmp_{co} is the best matching pattern for configuration c and object o, and C is the set of all configurations.

Let $\tilde{p}$ denote the best matching pattern bmp_{co} in short. Then we can express the mismatch between $\tilde{p}$ and o by $\sum_{t \in \theta_{\tilde{p}o}} sup(P_{t_{\tilde{p}}}) \times \frac{Dist(P_{t_{\tilde{p}}}, S_{t_o})}{maxDist(P_{t_{\tilde{p}}})}$. Thus, mismatch between the pattern and the soft sequence for o is simply the timestamp-wise mismatch weighted by the support of pattern at that timestamp. Finally, the importance of the pattern is captured by multiplying this mismatch score by the overall support of the pattern. As can be seen, $outlierScore(\tilde{p}, o)$ as expressed in Eq. 5 takes into account the support of the pattern, number of mismatching timestamps, and the degree of mismatch.

$$outlierScore(bmp_{co}, o) = outlierScore(\tilde{p}, o) = sup(\tilde{p}) \times \sum_{t \in \theta_{\tilde{p}o}} sup(P_{t_{\tilde{p}}}) \times \frac{Dist(P_{t_{\tilde{p}}}, S_{t_o})}{maxDist(P_{t_{\tilde{p}}})} \quad (5)$$

Time Complexity Analysis

Finding best matching patterns for all sequences takes $O(N|P|TK)$ time where $|P|$ is the number of patterns. Number of configurations $|C| = 2^T - T - 1$. So, outlier score computation using the best matching patterns takes $O(N(2^T - T - 1)KT)$ time where K is the maximum number of clusters at any timestamp. Thus, our outlier detection method is $O(NTK(2^T - T - 1 + |P|))$ in time complexity, i.e., linear in the number of objects. Generally, for real datasets, T is not very large and so the complexity is acceptable. For larger T, one may use sampling from the set of all possible configurations, rather than using all configurations. Our results (Sec. 4) show that considering only length-2 ungapped configurations can also provide reasonable accuracy.

Note that we did not include the pattern generation time in the time complexity analysis. This is because it is difficult to analyze time complexity of algorithms using Apriori pruning. However it has been shown earlier that Apriori techniques are quite efficient for pattern generation [5].

3.3 Summing Up: *CTOutlier* Detection Algorithm (CTODA)

The proposed *CTOutlier* Detection Algorithm (Algo. 1) can be summarized as follows. Given a dataset with N soft sequences each defined over T timestamps, soft patterns are first discovered from Steps 1 to 12 and then outlier scores are computed using Steps 13 to 20.

Next, we discuss two important practical issues in implementing the proposed community trend outlier detection algorithm.

Effect of Varying *min_sup*

Algorithm 1 *CTOutlier* Detection Algorithm (CTODA)

Input: (1) Soft sequences for N objects and T timestamps (represented using matrix S). (2) Minimum Support: min_sup.
Output: Outlier Score for each object.

Set of frequent patterns $P \leftarrow \phi$ ▷ Pattern Extraction
Let L_l be the set of length-l frequent patterns. $\{L_l\}_{l=1}^{T} \leftarrow \phi$.
Let C_l be the set of length-l candidate patterns. $\{C_l\}_{l=1}^{T} \leftarrow \phi$.
for each timestamp t **do**
    $C_1 \leftarrow$ Cluster S_t (i.e., part of S for timestamp t).
    $L_1 \leftarrow L_1 \cup \{f | f \in C_1$ and $sup(f) \geq min\_sup\}$
end for
for l=2 to T **do**
    $C_l \leftarrow getCandidates(L_{l-1})$.
    $L_l \leftarrow \{f | f \in C_l$ and $sup(f) \geq min\_sup\}$.
    $P \leftarrow P \cup L_l$.
end for

$C \leftarrow$ Set of configurations for T timestamps. ▷ Outlier Detection
for each object o **do**
    for each configuration $c \in C$ **do**
        Compute the best matching pattern $\tilde{p} \in P$ for object o and configuration c using Eq. 3.
        Compute $outlierScore(\tilde{p}, o)$ using Eq. 5.
    end for
    Compute $outlierScore(o)$ using Eq. 4.
end for

min_sup decides the number of patterns discovered, given a temporal dataset. Higher min_sup implies that some patterns may not get detected and hence even non-outlier sequences may get marked as outliers. However, their outlier scores will still be lower than the scores of extreme outliers because they are closer to the cluster centroids for each individual timestamp. Also, the number of configurations for which normal sequences deviate from patterns, will be smaller than the number of configurations for outlier sequences. However, a very high min_sup might mean that no patterns get discovered for a lot of configurations. In that case, many sequences might be assigned same high outlier scores, which is undesirable.

If min_sup is too low, then the discovered patterns may represent an overfitted model of the data. Thus, even outliers may get modeled as normal patterns, and then we may not be able to discover many outliers. Also a lower value of min_sup will generate a bigger set of patterns so that the overall pattern generation may become very inefficient with respect to time and memory.

Therefore, there is a tradeoff between lower and higher min_sup. The best way to select min_sup is to determine what percentage of sequences can be considered to represent a significant pattern. This could be quite domain dependent. In some domains, it might be completely fine even if a very few objects demonstrate a pattern while in other domains, one might need to use a larger min_sup value.

Hierarchical Clustering

In this paper, we performed single-level clustering of the distributions corresponding to the community detection results. However, one can also perform hierarchical clustering. Multi-level soft patterns discovered using such a hierarchical clustering per snapshot, could be more expressive. Using DBLP example

again, one may be able to express that a sub-area in timestamp 1 (lower level cluster) evolved into a mature research area in timestamp 2 (higher level cluster). We plan to explore the benefits of using such hierarchical clustering methods as part of future work.

4 Experiments

Evaluation of outlier detection algorithms is quite difficult due to lack of ground truth. We generate multiple synthetic datasets by injecting outliers into normal datasets, and evaluate outlier detection accuracy of the proposed algorithms on the generated data. We also conduct case studies by applying the method to real data sets. We perform comprehensive analysis to justify that the top few outliers returned by the proposed algorithm are meaningful. The code and the data sets are available at: http://blitzprecision.cs.uiuc.edu/CTOutlier

4.1 Synthetic Datasets

Dataset Generation

We generate a large number of synthetic datasets to simulate real evolution scenarios, each of which consists of 6 timestamps. We first create a dataset with normal points and then inject outliers. The accuracy of the algorithms is then measured in terms of their effectiveness in discovering these outliers. For each dataset, we first select the number of objects (N), the number of full-length (i.e., length=6) patterns ($|F|$), the percentage of outliers (Ψ) and the outlier degree (γ). Next, we randomly select the number of clusters per timestamp. Each cluster is represented by a Gaussian distribution with a fixed mean and variance. Figure 2 shows the clusters with their 2σ boundaries. For each full pattern, we first choose a cluster at each timestamp and then select a Gaussian distribution with small variance within the cluster. Once we have fixed the Gaussian distribution to be used, we generate $N/|F|$ points per timestamp for the pattern. Each full-length pattern results into $2^6 - 6 - 1 = 57$ patterns. We ensure that each cluster is part of at least one pattern. Once patterns are generated, we generate outliers as follows. For every outlier, first we select the base pattern for the outlier. An outlier follows the base pattern for $\lceil T \times \gamma \rceil$ timestamps and deviates from the pattern for the remaining timestamps. We fix a set of $\lceil T \times \gamma \rceil$ timestamps and randomly select a cluster different from the one in the pattern for the remaining timestamps. Figure 2 shows the first 4 timestamps (out of 6 – for lack of space) of a dataset created with N=1000, $|P|$=570 ($|F|$=10), Ψ=0.5 and γ=0.6. Colored points are normal points following patterns while larger black shapes (■, ▼, ◀, ▲, ♦) are the injected outliers. For example, the outlier (▼) usually belongs to the ★ pattern, except for the third timestamp where it switches to the yellow ▶ pattern.

Results on Synthetic Datasets

We experiment using a variety of settings. We fix the minimum support to $(100/|F|-2)\%$. For each setting, we perform 20 experiments and report the mean

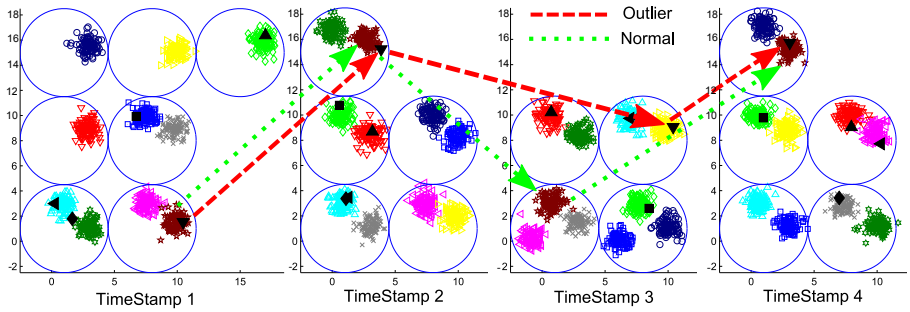

Fig. 2. First Four Snapshots of our Synthetic Dataset

N	Ψ	$\|\gamma\|$ = 0.5									$\|\gamma\|$ = 0.8								
		$\|F\|$ = 5			$\|F\|$ = 10			$\|F\|$ = 15			$\|F\|$ = 5			$\|F\|$ = 10			$\|F\|$ = 15		
	(%)	CTO	BL1	BL2	CTO	BL1	BL2	CTO	BL1	BL2	CTO	BL1	BL2	CTO	BL1	BL2	CTO	BL1	BL2
1000	1	**95.0**	92	89	**92.5**	86	90	**93.5**	83	92	**95.5**	85.5	92	83	76.5	**84.0**	**92.0**	77	86.0
	2	**98.0**	94.2	95.5	**94.0**	88.2	92	**95.5**	87.2	93.2	**98.2**	94.5	96.5	**91.2**	86.5	90	**95.5**	76	94.0
	5	**99.5**	96.8	97.4	**96.5**	95.3	96.2	**97.9**	93.1	97.1	**99.0**	95.7	97.3	**96.3**	91	95.9	**97.4**	79.3	96.7
5000	1	**97.0**	90.6	91.5	**91.9**	86	89.4	**91.8**	84.3	89.9	**95.8**	83.5	89.8	**84.4**	76.6	84.4	**88.4**	73.1	86.1
	2	**97.2**	92	92.8	**94.0**	91.2	93	**96.4**	89	94	**97.9**	89.6	94	**89.4**	85.6	88.4	**95.4**	79.8	93.1
	5	**99.4**	96.9	97.3	**96.3**	94.7	96.3	**97.6**	91	96.3	**98.8**	95.4	97.6	**95.0**	90.5	94.7	**97.7**	79.7	96.9
10000	1	**97.4**	90	90.4	**90.8**	85.4	88.1	**92.8**	84.5	88.2	**95.6**	84.2	89.5	81.8	76.4	**82.8**	**91.8**	76.5	87.6
	2	**98.2**	91.6	92.6	**93.2**	90.5	92.7	**95.0**	89.3	92.4	**98.0**	91.1	95	89.9	86.9	**90.7**	**95.8**	80.6	93.3
	5	**99.0**	96.8	97.1	**96.2**	94.4	96.2	**97.9**	89.6	96.8	**99.1**	95.8	98	**95.3**	90.1	95.3	**97.3**	76.4	96.6

Table 1. Synthetic Dataset Results (*CTO*=The Proposed Algorithm *CTODA*, *BL1*=Consecutive Baseline, *BL2*=No-gaps Baseline) for Outlier Degree=0.5 and 0.8, i.e., Outliers follow Base Pattern for 3/6 and 5/6 Timestamps respectively.

values. We compare with two baselines: *Consecutive (BL1)* and *No-gaps (BL2)*. Often times evolution is studied by considering pairs of consecutive snapshots of a dataset, and then integrating the results across all pairs. One can use such a method to compute outliers across each pair of consecutive snapshots and then finally combine results to get an outlier score across the entire time duration. We capture this in our *BL1* baseline. Note that we consider only those configurations which are of length-2 and which contain consecutive timestamps in this baseline. Thus, *BL1* will mark an object as outlier if its evolution across consecutive snapshots is much different from observed length-2 patterns (with no gaps). For the *BL2* baseline, we consider all configurations corresponding to patterns of arbitrary length without any gaps. Note that this baseline simulates the trajectory outlier detection scenario with respect to our framework. Recall that our method is named as *CTODA*.

We change the number of objects from 1000 to 5000 to 10000. We vary the percentage of outliers injected into the dataset as 1%, 2% and 5%. The outlier degree is varied as 0.5, 0.6 and 0.8 (i.e., 3, 4 and 5 timestamps). Finally, we also use different number of full-length patterns ($|F| = 5, 10, 15$), i.e., $|P| = 285, 570, 855$, to generate different datasets. Table 1 shows the results in terms of precision when the number of retrieved outliers equals to the actual number

of injected outliers for γ=0.5 and 0.8. Average standard deviations are 3.11%, 4.85% and 3.39% for *CTODA* and the two baselines respectively. Results with γ=0.6 are also similar; we do not show them here for lack of space. On an average *CTODA* is 7.4% and 2.3% better than the two baselines respectively.

The reasons why the proposed *CTODA* method is superior are as follows. Consider a soft sequence $\langle S_{1_o}, S_{2_o}, S_{3_o} \rangle$. Both the soft patterns $\langle S_{1_o}, S_{2_o} \rangle$ and $\langle S_{2_o}, S_{3_o} \rangle$ might be frequent while $\langle S_{1_o}, S_{2_o}, S_{3_o} \rangle$ might not be frequent. This can happen if the sequences which have the pattern $\langle S_{1_o}, S_{2_o} \rangle$ and the sequences with the pattern $\langle S_{2_o}, S_{3_o} \rangle$ are disjoint. This case clearly shows why our method which computes support for patterns of arbitrary lengths is better than baseline *BL1* which considers only patterns of length two with consecutive timestamps. Now let us show that gapped patterns can be beneficial even when we consider contiguous patterns of arbitrary lengths. Consider two soft gapped patterns $p = \langle P_{1_p}, P_{2_p}, P_{4_p} \rangle$ and $q = \langle P_{1_q}, P_{3_q}, P_{4_q} \rangle$ such that $P_{1_p} = P_{1_q}$ and $P_{4_p} = P_{4_q}$. Now p might be frequent while q is not. However, this effect cannot be captured if we consider only contiguous patterns. This case thus shows why our approach is better than *BL2*.

We ran our experiments using a 2.33 GHz Quad-Core Intel Xeon processor. On an average, the proposed algorithm takes 83, 116 and 184 seconds for N=1000, 5000 and 10000 respectively. Of this 74, 99 and 154 seconds are spent in pattern generation while the remaining time is spent in computing outliers given these patterns.

4.2 Real Datasets

Dataset Generation

We perform experiments using two real datasets: *GDP* and *Budget*.

***GDP*:** The *GDP* dataset consists of (Consumption, Investment, Public Expenditure and Net Exports) components for 89 countries for 1982-91 (10 snapshots)[3].

***Budget*:** The *Budget* dataset consists of (Pensions, Health Care, Education, Defense, Welfare, Protection, Transportation, General Government, Other Spending) components for 50 US states for 2001-10 (10 snapshots)[4].

Results on Real Datasets

CTOutliers are objects that break many temporal community patterns. We will provide a few interesting case studies for each dataset and explain the intuitions behind the identified outliers on how they deviate from the best matching patterns.

***GDP*:** We find 3682 patterns when minimum support is set to 20% for the 89 countries. The top five outliers discovered are Uganda, Congo, Guinea, Bulgaria and Chad, and we provide reasonings about Uganda and Congo as examples to support our claims as follows. National Resistance Army (NRA) operating under the leadership of the current president, Yoweri Museveni came to power in 1985-86 in **Uganda** and brought reforms to the economic policies. Uganda

[3] http://www.economicswebinstitute.org/ecdata.htm

[4] http://www.usgovernmentspending.com/

showed a sudden change of (45% consumption and 45% net exports) to (80% consumption and 1-2% net exports) in 1985. Unlike Uganda, other countries like Iceland, Canada, France with such ratios of consumption and net export maintained to do so. Like many other countries, **Congo** had 45-48% of its GDP allocated to consumption and 36-42% of GDP for government expenditure. But unlike other countries with similar pattern, in 1991, consumption decreased (to 29% of GDP) but government expenditure increased (56% of GDP) for Congo. This drastic change happened probably because opponents of the then President of Congo (Mobutu Sese Seko) had stepped up demands for democratic reforms.
***Budget*:** We find 41545 patterns when minimum support is set to 20% for the 50 states. The top five outliers discovered are AK, DC, NE, TN and FL, and the case on AK is elaborated as follows. For states with 6% pension, 16% healthcare, 32% education, 16% other spending in 2006, it has been observed that healthcare increased by 4-5% in 2009-10 while other spending decreased by 4%. However, in the case of **Arkansas (AK)** which followed a similar distribution in 2006, healthcare decreased by 3% and other spending increased by 5% in 2009-10. More details can be found in Fig. 3. The right figure shows the distribution of expenses for AK for the 10 years, while the left part shows similar distribution averaged over 5 states which have a distribution very similar to AK for the years 2004-09. One can see that Arkansas follows quite a different distribution compared to the five states for other years especially for 2002.

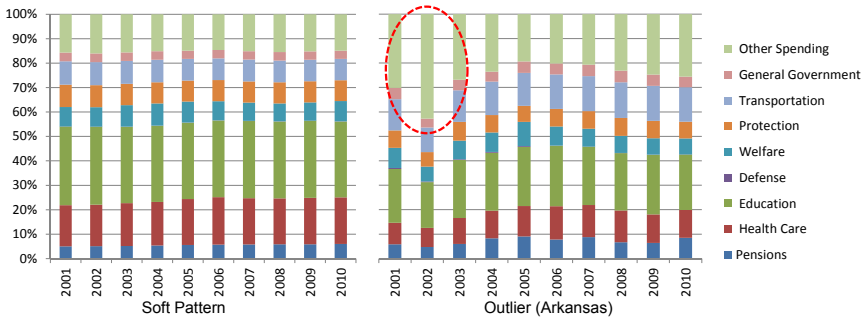

Fig. 3. Average Trend of 5 States with Distributions close to that of AK for 2004-09 (Left – Pattern), Distributions of Budget Spending for AK (Right – Outlier)

In summary, the proposed algorithm is highly accurate in identifying injected outliers in synthetic datasets and it is able to detect some interesting outliers from each of the real datasets.

5 Related Work

Outlier (or anomaly) detection [10,18] is a very broad field and has been studied in the context of a large number of application domains. Outliers have been dis-

covered in high-dimensional data [1], uncertain data [3], stream data [4], network data [13] and time series data [11].

Temporal Outlier Detection

Recently, there has been significant interest in detecting outliers from evolving datasets [4,14,15], but none of them explores the outliers with respect to communities in a general evolving dataset. Outlier detection methods for data streams [2,7] have no notion of communities. *CTOutlier* detection could be considered as finding outlier trajectories across time, given the soft sequence data. However as discussed in Sec. 1, there are major differences, making trajectory outlier detection techniques unsuitable for the task.

Community Outlier Detection

Community outlier detection has been studied for a static network setting [13] or for a setting of two network snapshots [17], but we develop an algorithm for a general evolving dataset with multiple snapshots. Group (community) identification in evolving scenarios has been studied traditionally in the context of hard patterns [16,20], while we discover soft patterns capturing *subtle community* evolution trends.

Thus, though significant literature exists both for temporal as well as community outlier detection, we discover novel outliers by considering both the temporal and the community dimensions together.

6 Conclusions

In datasets with continuously evolving values, it is very important to detect objects that do not follow normal community evolutionary trend. In this paper, we propose a novel concept of an outlier, denoting objects that deviate from temporal community norm. Such objects are referred to as Community Trend Outliers (*CTOutliers*), and are of great practical importance to numerous applications. To identify such outliers, we proposed an effective two-step outlier detection algorithm *CTODA*. The proposed method first conducts soft pattern mining efficiently and then detects outliers by measuring the objects' deviations from the normal patterns. Extensive experiments on multiple synthetic and real datasets show that the proposed method is highly effective and efficient in detecting meaningful community trend outliers. In the future, we plan to further our studies on evolutionary outlier detection by considering various evolution styles in different domains.

Acknowledgements Research was sponsored in part by the Cyber Security project (W911NF-11-2-0086) and NSCTA project (W911NF-09-2-0053) of U.S. Army Research Laboratory, NSF IIS-0905215, MIAS, a DHS-IDS Center for Multimodal Information Access and Synthesis at UIUC and U.S. Air Force Office of Scientific Research MURI award FA9550-08-1-0265. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

1. C. C. Aggarwal and P. S. Yu. Outlier Detection for High Dimensional Data. *SIGMOD Records*, 30:37–46, May 2001.
2. C. C. Aggarwal. On Abnormality Detection in Spuriously Populated Data Streams. In *SDM*, pages 80–91. 2005.
3. C. C. Aggarwal and P. S. Yu. Outlier Detection with Uncertain Data. In *SDM*, pages 483–493, 2008.
4. C. C. Aggarwal, Y. Zhao, and P. S. Yu. Outlier Detection in Graph Streams. In *ICDE*, pages 399–409. 2011.
5. R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *VLDB*, pages 487–499. 1994.
6. J. Alon, S. Sclaroff, G. Kollios, and V. Pavlovic. Discovering Clusters in Motion Time-Series Data. In *CVPR*, pages 375–381. 2003.
7. F. Angiulli and F. Fassetti. Detecting Distance-based Outliers in Streams of Data. In *CIKM*, pages 811–820. 2007.
8. A. Basharat, A. Gritai, and M. Shah. Learning Object Motion Patterns for Anomaly Detection and Improved Object Detection. In *CVPR*. pages 1–8. 2008.
9. T. Bernecker, H. P. Kriegel, M. Renz, F. Verhein, and A. Zuefle. Probabilistic Frequent Itemset Mining in Uncertain Databases. In *KDD*, pages 119–128. 2009.
10. V. Chandola, A. Banerjee, and V. Kumar. Anomaly Detection: A Survey. *ACM Computing Surveys*, 41(3), 2009.
11. A. J. Fox. Outliers in Time Series. *Journal of the Royal Statistical Society. Series B (Methodological)*, 34(3):350–363, 1972.
12. J. Gao, F. Liang, W. Fan, Y. Sun, and J. Han. Graph-based Consensus Maximization among Multiple Supervised and Unsupervised Models. In *NIPS*, pages 585–593. 2009.
13. J. Gao, F. Liang, W. Fan, C. Wang, Y. Sun, and J. Han. On Community Outliers and their Efficient Detection in Information Networks. In *KDD*, pages 813–822, 2010.
14. Y. Ge, H. Xiong, Z. hua Zhou, H. Ozdemir, J. Yu, and K. C. Lee. Top-Eye: Top-K Evolving Trajectory Outlier Detection. In *CIKM*, pages 1733–1736, 2010.
15. A. Ghoting, M. E. Otey, and S. Parthasarathy. LOADED: Link-Based Outlier and Anomaly Detection in Evolving Data Sets. In *ICDM*, pages 387–390, 2004.
16. J. Gudmundsson, M. Kreveld, and B. Speckmann Efficient Detection of Motion Patterns in Spatio-temporal Data Sets. In *GIS*, pages 250–257. 2004.
17. M. Gupta, J. Gao, Y. Sun, and J. Han. Integrating Community Matching and Outlier Detection for Mining Evolutionary Community Outliers. In *KDD*, To appear, 2012.
18. V. J. Hodge and J. Austin. A Survey of Outlier Detection Methodologies. *AI Review*, 22(2):85–126, 2004.
19. J.-G. Lee, J. Han, and X. Li. Trajectory Outlier Detection: A Partition-and-Detect Framework. In *ICDE*, pages 140–149. 2008.
20. Z. Li, B. Ding, J. Han, and R. Kays. Swarm: Mining Relaxed Temporal Moving Object Clusters. In *VLDB*, pages 723–734. 2010.
21. M. Muzammal and R. Raman. Mining Sequential Patterns from Probabilistic Databases. In *PAKDD*, pages 210–221. 2011.
22. D. Pelleg and A. W. Moore. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In *ICML*, pages 727–734. 2000.
23. Y. Sun, J. Tang, J. Han, M. Gupta, and B. Zhao. Community Evolution Detection in Dynamic Heterogeneous Information Networks. In *KDD-MLG*, pages 137–146. 2010.