Discovery of Periodic Patterns in Spatiotemporal Sequences

Huiping Cao, Nikos Mamoulis, *Member*, *IEEE*, and
David W. Cheung, *Member*, *IEEE Computer Society*

Abstract—In many applications that track and analyze spatiotemporal data, movements obey periodic patterns; the objects follow the same routes (approximately) over regular time intervals. For example, people wake up at the same time and follow more or less the same route to their work everyday. The discovery of hidden periodic patterns in spatiotemporal data could unveil important information to the data analyst. Existing approaches for discovering periodic patterns focus on symbol sequences. However, these methods cannot directly be applied to a spatiotemporal sequence because of the fuzziness of spatial locations in the sequence. In this paper, we define the problem of mining periodic patterns in spatiotemporal data and propose an effective and efficient algorithm for retrieving maximal periodic patterns. In addition, we study two interesting variants of the problem. The first is the retrieval of periodic patterns that are frequent only during a continuous subinterval of the whole history. The second problem is the discovery of periodic patterns, whose instances may be shifted or distorted. We demonstrate how our mining technique can be adapted for these variants. Finally, we present a comprehensive experimental evaluation, where we show the effectiveness and efficiency of the proposed techniques.

Index Terms—Data mining, periodic patterns, spatiotemporal data.

1 Introduction

The efficient management of spatiotemporal data has gained much interest during the past few years [14], [16], [6], [15], mainly due to the rapid advancement in telecommunications (e.g., GPS, Cellular networks, etc.), which facilitate the collection of large data sets of such information. Management and analysis of moving object trajectories are challenging due to the vast amount of collected data and novel types of spatiotemporal queries.

In many applications, the movements obey periodic patterns, i.e., the objects follow the same routes (approximately) over regular time intervals. Objects that follow approximate periodic patterns include transportation vehicles (buses, boats, airplanes, trains, etc.), animals, mobile phone users, etc. For example, Bob wakes up at the same time and then follows, more or less, the same route to his work everyday.

The problem of discovering periodic patterns from historical object movements is very challenging. Usually, the patterns are not explicitly specified, but have to be discovered from the data. The patterns can be thought of as (possibly noncontiguous) sequences of object locations that reappear in the movement history periodically. In addition, since we do not expect an object to visit *exactly* the same location at every time instant of each period, the patterns are not rigid but differ slightly from one occurrence to the next. The approximate nature of patterns in the spatiotemporal domain increases the complexity of the mining tasks. We need to discover, along with the patterns, a flexible description of how they variate in space and time. Previous approaches have studied the extraction of patterns from long event sequences [7], [10]. We identify the difference between the two problems and propose novel techniques for mining periodic patterns from a large historical collection of object movements.

In practice, periodic patterns may not be frequent in the whole sequence. For instance, assume that Bob changes his route to work after being transferred from department A to department B. In this case, his route to department A is frequent only during the time interval he works there. This motivates us to study the problem of mining frequent patterns and their *validity eras*, i.e., the (maximal) time ranges (*eras*) during which these patterns are frequent.

In real applications, pattern occurrences in certain periodic ranges may be shifted or distorted in time. For instance, if Bob wakes up late on a certain day, the movement to his work is shifted on that day (e.g., for 10 minutes). Or, Bob gets up at the usual time, but arrives at the company a little late due to traffic congestion. Although Bob follows the same route (pattern) to the company in the above two cases, the corresponding pattern instances are shifted and/or distorted. In this paper, we extend the baseline pattern mining technique to include in the counting of a pattern's frequency its shifted or distorted instances.

The contributions of this paper are: 1) a new model of partial periodic pattern discovery in spatiotemporal data, 2) an effective and efficient method for discovering the periodic patterns from a long movement history, and 3) techniques that extend the mining approach to identify variants of the periodic patterns: era patterns and shifted/distorted patterns. The rest of the paper is organized as follows: In Section 2, we review work related to the problem under study. The baseline periodic pattern mining problem is formally defined in Section 3. We describe the several approaches presented in [11] and an additional time-efficient technique in Section 4. Section 5 formally defines

• *The authors are with the Department of Computer Science, The University of Hong Kong, Pokfulam Road, Hong Kong.*
E-mail: {hpcao, nikos, dcheung}@cs.hku.hk.

Manuscript received 5 Aug. 2005; revised 1 Apr. 2006; accepted 3 Oct. 2006; published online 19 Jan. 2007.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0296-0805.
Digital Object Identifier no. 10.1109/TKDE.2007.1002.

the problem variants and provides solutions for them. We evaluate the effectiveness and efficiency of the proposed methods experimentally in Section 6. Finally, Section 7 concludes this paper.

2 Related Work

The problem of mining sequential patterns from transactional databases has attracted a lot of interest since Agrawal and Srikant introduced it in [2]. Each transaction contains a set of items that are bought by some customer, and the database consists of ordered lists of transactions. For example, $\langle(a,b),(a,c),(b)\rangle$ is a sequence containing three transactions (a,b), (a,c), and (b). Given such a database, the sequential pattern mining problem is to find ordered lists of itemsets that appear in sequences with high frequency. For instance, $\langle(b),(a),(b)\rangle$ is a pattern which is supported by the above sequence. The original sequential pattern mining problem does not consider the periodicity character of a transaction sequence.

Periodicity has only been studied in the context of time-series databases. Indyk et al. [9] address the following problem: Given a long sequence $\mathcal{S}$ and a period T, the aim is to discover the most representative trend that repeats itself in $\mathcal{S}$ every T timestamps. Exact search might be slow; thus, [9] proposes an approximate technique based on sketches. However, the discovered trend for a given T is only one and spans the whole periodic interval. In [12], the problem of finding association rules that repeat themselves in every period of a data sequence is addressed. Elfeky et al. in [4] tackle the problem of periodicity detection on a series of nominal data, focusing on the automatic detection of the period.

The discovery of multiple partial periodic patterns that do not appear in every periodic segment is first studied in [8]. Such a pattern is in the form of $p_0, p_1, \ldots, p_{T-1}$, where T is the given period, each $p_j\ (0 \leq j < T)$ can be an element (e.g., event type) or a wildcard *, which matches any element in the sequence. The pattern may not repeat itself in every period, but it must appear at least min_sup times (a user-defined parameter). A version of the well-known a priori algorithm [1] is adapted for the problem of finding such patterns. In [7], a faster mining method for this problem is proposed, which uses a tree structure, the max-subpattern tree, to count the support of multiple patterns at two database scans. Specifically, during the first pass, the set F_1 of all frequent patterns with one non-* element is identified (e.g., $F_1 = \{\mathrm{a}^{****}, {}^*\mathrm{b}^{***}, {}^{**}\mathrm{c}^{**}\}$). The max-subpattern tree is rooted at a candidate max-pattern C_{max}, which is the maximal combination of all the patterns in F_1 (e.g., $C_{max} = \mathrm{abc}^{**}$). A node at level l of the tree (e.g., node ${}^*\mathrm{bc}^{**}$ at level 2) has l non-* elements and l children at the level below (e.g., ${}^*\mathrm{b}^{***}$ and ${}^{**}\mathrm{c}^{**}$), which have one more * in their patterns. Each node contains a counter for the exact occurrences of its associated pattern. During the second data pass, each period segment is *inserted* into the tree, and the counters of the maximal patterns that appear in the segment are increased. Therefore, the support of a pattern associated with a node is the sum of the counters along the path from the root to that node. Finally, the tree is used by a priori to extract the frequent patterns.

Given an event sequence, Yang et al. in [20] study the problem of finding asynchronous patterns, which appear in at least a minimum number, min_rep, of consecutive periodic intervals, and groups of such intervals are allowed to be separated by at most a time interval threshold, max_dis. This model is quite similar to mining patterns and their validity eras, which we study in this paper; however, we note two significant differences. First, we apply mining on sequences of locations in a continuous space, whereas [20] deals with sequences of categorical (event) data. Second, we do not use parameters min_rep and min_dis to restrict the definition of eras, but use only one parameter (Section 5), considering the ratio of the periodic intervals that contribute to a pattern and the total periodic intervals in a sequence segment.

Ma and Hellerstein et al. in [10] study the problem of finding sets of events that appear together periodically. In each qualifying period, the set of events may not appear in exactly the same positions, but their occurrences may be shifted or disrupted due to the presence of noise. However, this work does not consider the order of events in such patterns. On the other hand, it addresses the problem of mining patterns and their periods automatically. Yang et al. also study the mining of surprising periodic patterns from event sequences in [21]. They propose a new metric, *information gain*, to validate the usefulness of a pattern. Further, in [22], this work is extended for partial periodic patterns with gap penalties.

All work above assumes that the elements in the sequence are categorical; thus, the occurrences of elements and patterns can be counted by incrementing a counter every time they are observed in the sequence. However, this basic counting technique may not directly be applied to a spatiotemporal sequence since each spatial location in such sequences is in the form of spatial coordinates and does not typically repeat itself exactly. Chiu et al. [3] discretize real-valued time series prior to mining and then identify the most common subsequences in them. The mined patterns are not essentially periodic and they are contiguous (i.e., there are no wildcards).

Previous work on *spatiotemporal data* mining focuses on two types of patterns: 1) frequent movements of objects over time and 2) evolution of natural phenomena, such as forest coverage. Tsoukatos and Gunopulos [17] study the discovery of frequent patterns related to changes of natural phenomena (e.g., temperature changes) in spatial regions. In general, there is limited work on spatiotemporal data mining, which has been treated as a generalization of pattern mining in time-series data (e.g., see [17], [13]). The locations of objects or the changes of natural phenomena over time are converted to categorical values. For instance, we can divide the map into spatial regions and replace the location of the object at each timestamp by the region-id where it is located. Similarly, we can model the change of temperature in a spatial region as a sequence of temperature values. Continuous domains of the resulting time-series data are discretized prior to mining. In the case of multiple moving objects (or time series), trajectories are typically concatenated to a single long sequence. Then, an algorithm that discovers frequent subsequences in a long sequence (e.g., [23]) is applied. To our knowledge, there is no prior work on discovering periodic patterns in spatiotemporal data.

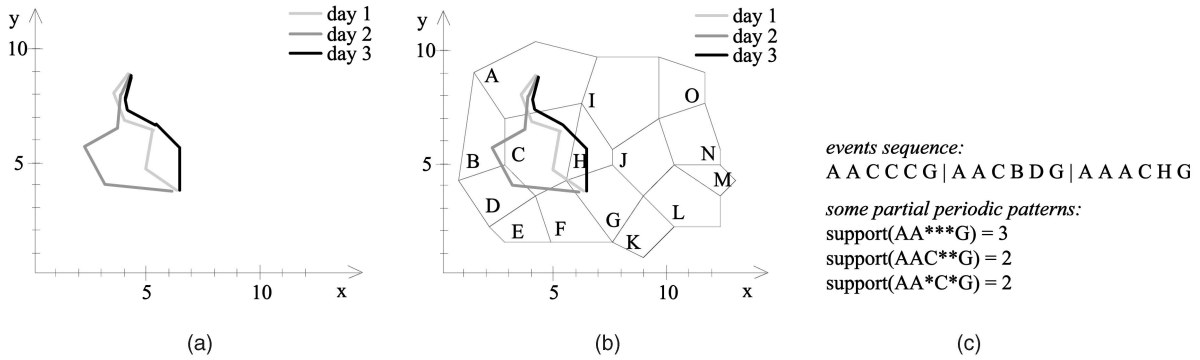

Fig. 1. Periodic patterns with respect to predefined spatial regions. (a) An object's movement. (b) A set of predefined regions. (c) Event-based patterns.

3 Periodic Patterns in Object Trajectories

This section defines the problem of mining periodic patterns in spatiotemporal data. Before the definition, we first motivate our research by discussing why previous work on event sequences is not expected to perform well when applied on object trajectories.

In our model, we assume that the locations of objects are sampled over a long history. In other words, the movement of an object is tracked as an n-length sequence $\mathcal{S}$ of spatial locations, one for each timestamp in the history, of the form $\{(l_0, t_0), (l_1, t_1), \ldots, (l_{n-1}, t_{n-1})\}$, where l_i is the object's location at time t_i. If the difference between consecutive timestamps is fixed (locations are sampled every regular time interval), we can represent the movement by a simple sequence of locations l_i (i.e., by dropping the timestamps t_i, since they can be implied). Each location l_i is expressed in terms of spatial coordinates. Fig. 1a, for example, illustrates the movement of an object in three consecutive days (assuming that it is tracked only during specific hours, e.g., working hours). We can model it with sequence $\mathcal{S} = \{\langle 4, 9\rangle, \langle 3.5, 8\rangle, \ldots, \langle 6.5, 3.9\rangle, \langle 4.1, 9\rangle, \ldots\}$. Given such a sequence, a minimum support min_sup $(0 < min\_sup \leq 1)$, and an integer T, called *period*, our problem is to discover movement patterns that repeat themselves every T timestamps. A discovered pattern P is a T-length sequence of the form $r_0 r_1 \ldots r_{T-1}$, where r_i is a *spatial region* or the special character *, indicating the whole spatial universe. For instance, pattern $\mathrm{AB^*C^{**}}$ implies that, at the beginning of the cycle, the object is in region A, at the next timestamp, it is found in region B, then it moves irregularly (it can be anywhere), then it goes to region C, and after that, it can go anywhere, until the beginning of the next cycle, when it can be found again in region A. The patterns are required to be followed by the object in at least α $(\alpha = min\_sup \cdot \lfloor \frac{n}{T} \rfloor)$ periodic intervals in $\mathcal{S}$.

Existing algorithms for mining periodic patterns (e.g., [7]) operate on event sequences and discover patterns of the above form. However, in this case, the elements r_i of a pattern are events (or sets of events). As a result, we cannot directly apply these techniques for our problem, unless we treat the exact locations l_i as discrete categorical values. Nevertheless, it is highly unlikely that an object repeats an identical sequence of $\langle x, y\rangle$ locations precisely. Even if the spatial route is precise, the location transmissions at each timestamp are unlikely to be perfectly synchronized. Thus, the object does not reach the same location at the same time every day, and as a result, the sampled locations at specific timestamps (e.g., at 9:00 a.m. sharp, every day), are different. In Fig. 1a, for example, the first daily locations of the object are very close to each other; however, they are treated differently by a straightforward mining algorithm.

One way to handle the noise in object movements is to replace the exact locations of the objects by the regions (e.g., districts, mobile communication cells, or cells of a synthetic grid) which contain them. Fig. 1b shows an example of an area's division into such regions. Sequence $\{\mathrm{A}, \mathrm{A}, \mathrm{C}, \mathrm{C}, \mathrm{C}, \mathrm{G}, \mathrm{A}, \ldots\}$ can now summarize the object's movement and periodic pattern mining algorithms, like [7], can directly be applied. Fig. 1c shows three (closed) discovered patterns for $T = 6$ and $min\_sup = \frac{2}{3}$. A disadvantage of this approach is that the discovered patterns may not be very descriptive if the space division is not very detailed. For example, regions A and C are too large to capture in detail the first three positions of the object in each periodic instance. On the other hand, with detailed space divisions, the same (approximate) object location may span more than one different region. For example, in Fig. 1b, observe that the third object positions for the three days are close to each other; however, they fall into different regions (A and C) at different days. Therefore, we are interested in the *automated* discovering of patterns *and their descriptive regions*. Before we present solutions for this problem, we first define it formally.

Problem definition. *Let $\mathcal{S}$ be a sequence of n spatial locations $\{l_0, l_1, \ldots, l_{n-1}\}$, representing the movement of an object over a long history. Let $T \ll n$ be a user-specified integer called period (e.g., day, week, and month). A* periodic segment *s is defined by a subsequence $l_i l_{i+1} \ldots l_{i+T-1}$ of $\mathcal{S}$, such that i modulo $T = 0$. Thus, segments start at positions $0, T, \ldots, (\lfloor \frac{n}{T} \rfloor - 1) \cdot T$, and there are exactly $m = \lfloor \frac{n}{T} \rfloor$ periodic segments in $\mathcal{S}$.*[1] *Let s^j denote the segment starting at position $l_{j \cdot T}$ of $\mathcal{S}$, for $0 \leq j < m$, and let $s_i^j = l_{j \cdot T + i}$, for $0 \leq i < T$.*

Definition 1. *A* **periodic pattern** *P is defined by a sequence $r_0 r_1 \ldots r_{T-1}$ of length T, such that r_i is either a spatial region or *. The* length *of a pattern P is the number of non-* regions in P.*

1. If n is not a multiple of T, then the last n modulo T locations are truncated, and the length n of sequence $\mathcal{S}$ is reduced accordingly.

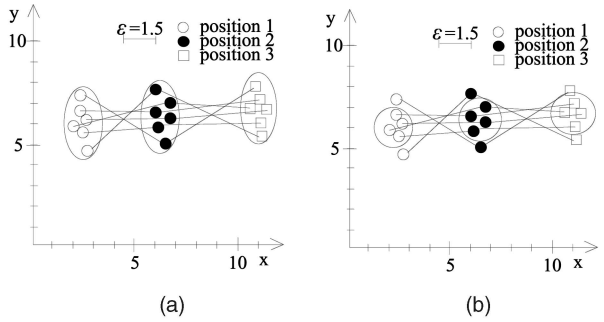

Fig. 2. Redundancy of patterns. (a) A valid pattern. (b) A redundant pattern.

A segment s^j is said to *comply with* P, if for each $r_i \in P$, $r_i =^*$, or s_i^j is *inside* region r_i.

Definition 2. *The* **support** $|P|$ *of a pattern* P *in* $\mathcal{S}$ *is defined by the number of periodic segments in* $\mathcal{S}$ *that comply with* P.

We sometimes use the same symbol P to refer to a pattern and the set of segments that comply with it. Let min_sup be a fraction in the range (0,1] (*minimum support*). A pattern P is *frequent* if its support is larger than $min\_sup \cdot m$.

A problem with the definition above is that it imposes no control over the density of the pattern regions r_i. In other words, if the pattern regions are too relaxed (e.g., each r_i is the whole map), the pattern may always be frequent. Therefore, we impose an additional constraint as follows: Let $\mathcal{S}^P$ be the set of segments that comply with a pattern P. Then, each region r_i of P is *valid* if the set of locations $R_i^P := \{s_i^j | s^j \in \mathcal{S}^P\}$ forms a *dense cluster*. To define a dense cluster, we borrow the definitions from [5] and use two parameters ϵ and $MinPts$. A point p in the spatial data set R_i^P is a *core* point if the circular range centered at p with radius ϵ contains at least $MinPts$ points. If a point q is within distance ϵ from a core point p, it is assigned in the same cluster as p. If q is a core point itself, then all points within distance ϵ from q are assigned in the same cluster as p and q. If R_i^P forms a single, dense cluster with respect to some values of parameters ϵ and $MinPts$, we say that region r_i is valid. If all non-* regions of P are valid, then P is a valid pattern. We are interested in the discovery of valid patterns only. In the following, we use the terms *valid region* and *dense cluster* interchangeably, i.e., we often use the term dense region to refer to a spatial dense cluster and the points in it.

Fig. 2a shows an example of a valid pattern, if $\epsilon = 1.5$ and $MinPts = 4$. Each region at positions 1, 2, and 3 forms a single, dense cluster and is therefore a dense region. Notice, however, that it is possible that two valid patterns P and P' of the same length 1) have the same * positions, 2) every segment that complies with P', complies with P, and 3) $|P'| < |P|$. In other words, P *implies* P'. For example, the pattern of Fig. 2a implies the one of Fig. 2b (denoted by the three circles). A frequent pattern P' is *redundant* if it is implied by some other frequent pattern P.

Definition 3. *The* **mining periodic patterns problem** *searches for all valid periodic patterns* P *in* $\mathcal{S}$*, which are frequent and nonredundant with respect to a minimum support* min_sup.

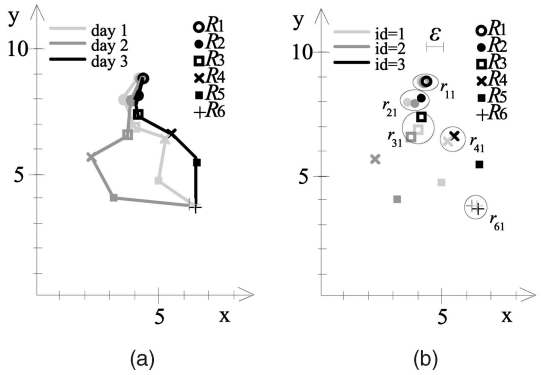

Fig. 3. Locations and regions per periodic offset. (a) T-based decomposition. (b) Dense clusters in R_is.

For simplicity, we use *frequent pattern* to refer to a valid, nonredundant frequent pattern.

4 Mining Periodic Patterns

In this section, we present techniques for mining frequent periodic patterns and their associated regions in a long history of object trajectories. We first address the problem of finding frequent 1-patterns (i.e., of length 1). Then, we propose two methods to find longer patterns: a bottom-up, level-wise technique, denoted by STPMine1 (SpatioTemporal periodic Pattern Min(e)ing 1), and a faster top-down approach, referred to as STPMine2. Finally, we present a simplified version of the top-down approach, which solves the problem approximately, but it is very efficient.

4.1 Obtaining Frequent 1-Patterns

Including automatic discovery of regions in the mining task does not allow for the direct application of techniques that find patterns in sequences (e.g., [7]) as discussed. In order to tackle this problem, we propose the following methodology: We divide the sequence $\mathcal{S}$ of locations into T spatial data sets, one for each offset of the period T. In other words, locations $\{l_i, l_{i+T}, \ldots, l_{i+(m-1)\cdot T}\}$ go to set R_i, for each $0 \leq i < T$. Each location is tagged by the id $j \in [0, \ldots, m-1]$ of the segment that contains it. Fig. 3a shows the spatial data sets obtained after decomposing the object trajectory of Fig. 1a. We use a different symbol to denote locations that correspond to different periodic offsets and different colors for different segment-ids.

Observe that a dense cluster r in data set R_i corresponds to a frequent pattern, having * at all positions and r at position i. Fig. 3b shows examples of five clusters discovered in data sets R_1, R_2, R_3, R_4, and R_6. These correspond to five 1-patterns (i.e., $r_{11}{}^{*****}$, $^*r_{21}{}^{****}$, etc.). In order to identify the dense clusters for each R_i, we can apply a density-based clustering algorithm like DBSCAN [5]. Clusters with less than α ($\alpha = min\_sup \cdot m$) points are discarded, since they are not frequent 1-patterns according to our definition. Clustering is quite expensive and it is a frequently used module of the mining algorithms, as we will see later. DBSCAN [5] has quadratic cost to the number of clustered points, unless an index (e.g., R-tree) is available. Since R-trees are not available for every arbitrary set of points to be clustered, we use an efficient hash-based method. For the sake of readability, we include the details of this method in the Appendix.

Algorithm STPMine1($\mathcal{L}_1, T, min\_sup$);
1). k:=2;
2). **while** ($\mathcal{L}_{k-1} \neq \emptyset \wedge k < T$)
3). $\mathcal{L}_k$:=$\emptyset$;
4). **for each** pair of patterns $(P_1, P_2) \in \mathcal{L}_{k-1}$
5). such that P_1 and P_2 agree on the first $k-2$
6). and have different $(k-1)$-th non-* position
7). P_{cand}:=**candidate_gen**(P_1, P_2);
8). **if** ($P_{cand} \neq null$) **then**
9). P_{cand}:=$P_1 \bowtie_{P_1.sid=P_2.sid} P_2$; //segment-id join
10). **if** ($|P_{cand}| \geq min\_sup \cdot m$) **then**
11). **validate_pattern**($P_{cand}, \mathcal{L}_k, min\_sup$);
12). k:=$k+1$;
13). **return** $\mathcal{P}$:=$\bigcup \mathcal{L}_k, \forall 1 \leq k < T$;

Fig. 4. Level-wise pattern mining.

4.2 A Level-Wise, Bottom-Up Approach

Starting from the discovered 1-patterns (i.e., clusters for each R_i), we apply a variant of the level-wise Apriori-TID algorithm [1] to discover longer ones, as shown in Fig. 4. The input of our algorithm is a collection $\mathcal{L}_1$ of frequent 1-patterns, discovered as described in Section 4.1; for each R_i, $0 \leq i < T$, and each dense region $r \in R_i$, there is a 1-pattern in $\mathcal{L}_1$. Pairs $\langle P_1, P_2 \rangle$ of $(k-1)$-patterns in $\mathcal{L}_{k-1}$, with their first $k-2$ non-* regions in the same position and different $(k-1)$th non-* position, create candidate k-patterns (lines 4-6). For each candidate P_{cand}, we perform a segment-id join between P_1 and P_2, and if the number of segments that comply with both patterns is at least $min\_sup \cdot m$, we run a pattern validation function to check whether the regions of P_{cand} are still clusters. After the patterns of length k have been discovered, we find the patterns at the next level, until there are no more patterns at the current level, or there are no more levels.

In order to facilitate fast and effective candidate generation, we use the MBRs (i.e., *minimum bounding rectangles*) of the pattern regions. For each common non-* position i, the intersection of the MBRs of the regions for P_1 and P_2 must be nonempty; otherwise, a valid superpattern cannot exist. The intersection is adopted as an approximation for the new pattern P_{cand} at each such position i. During candidate pruning, we check for every $(k-1)$-subpattern of P_{cand} if there is at least one pattern in $\mathcal{L}_{k-1}$, which agrees in the non-* positions with the subpattern and the MBR-intersection with it is nonempty at all those positions. In such a case, we accept P_{cand} as a candidate pattern. Otherwise, we know that P_{cand} cannot be a valid pattern, since some of its subpatterns (with common space covered by the non-* regions) are not included in $\mathcal{L}_{k-1}$.

Function **validate_pattern** takes as input a k-candidate pattern P_{cand} and computes a number of actual k-patterns from it. The rationale is that the points at all non-* positions of P_{cand} may not form a cluster anymore after the join of P_1 and P_2. Thus, for each non-* position of P_{cand}, we recluster the points. If, for some position, the points can be grouped to more than one cluster, we create a new candidate pattern for each cluster and validate it. Note that, from a candidate pattern P_{cand}, it is possible to generate more than one actual pattern eventually. If no position of P_{cand} is split to multiple clusters, we may need to recluster the non-* positions of P_{cand}, since some points (and segment-ids) may be eliminated during clustering at some position.

To illustrate the algorithm, consider the 2-patterns $P_1 = r_{1x}r_{2y}{}^*$ and $P_2 = r_{1w}{}^*r_{3z}$ of Fig. 6a. Assume that $MinPts = 4$ and $\epsilon = 1.5$. The two patterns have a common first non-* position and $MBR(r_{1x})$ overlaps $MBR(r_{1w})$. Therefore, a candidate 3-pattern P_{cand} is generated. During candidate pruning, we verify that there is a 2-pattern with non-* positions 2 and 3 which is in $\mathcal{L}_2$. Indeed, such a pattern can be spotted at the figure (see the dashed lines). After joining the segment-ids in P_1 and P_2 at line 9 of STPMine1, P_{cand} contains the trajectories shown in Fig. 6b. Notice that the locations of the segment-ids in the intersection may not form clusters anymore at some positions of P_{cand}. This is why we have to call **validate_pattern**, in order to identify the valid patterns included in P_{cand}. Observe that the segment-id corresponding to the lowermost location of the first position is eliminated from the cluster as an outlier. Then, while clustering at position 2, we identify two dense clusters, which define the final patterns $r_{1a}r_{2b}r_{3c}$ and $r_{1d}r_{2e}r_{3f}$.

Although the algorithm of Fig. 4 can find all partial periodic patterns correctly, it can be very slow due to the huge number of region combinations to be joined. If the actual patterns are long, all their subpatterns have to be computed and validated. In addition, a potentially huge number of candidates need to be checked and evaluated. In this section, we propose a top-down method that can discover long patterns more efficiently,

After applying clustering on each R_i (as described in Section 4.1), we have discovered the frequent 1-patterns with their segment-ids. The first phase of the STPMine2 algorithm (Fig. 8) replaces each location in $\mathcal{S}$ with the cluster-id it belongs to or with an *empty* value (e.g., *) if the location belongs to no cluster. For example, assume that we have discovered clusters $\{r_{11}, r_{12}\}$ at position 1, $\{r_{21}\}$ at position 2, and $\{r_{31}, r_{32}\}$ at position 3. A segment $\{l_1, l_2, l_3\}$, such that $l_1 \in r_{12}$, $l_2 \notin r_{21}$, and $l_3 \in r_{31}$, is transformed to subsequence $\{r_{12}{}^*r_{31}\}$. Therefore, the original spatiotemporal sequence $\mathcal{S}$ is transformed to a symbol sequence $\mathcal{S}'$.

Now, we could use the mining algorithm of [7] to discover all frequent patterns of the form $r_0r_1 \ldots r_{T-1}$ fast, where each r_i is a cluster in R_i or *. However, we do not know whether the results of the sequence-based algorithm are actual patterns, since the contents of each non-* position may not form a cluster. For example, $\{r_{12}{}^*r_{31}\}$ may be frequent, however, if we consider only the segment-ids that qualify this pattern, r_{12} may no longer be a cluster or may form different actual clusters (as illustrated in Fig. 6). We call the patterns P' which can be discovered by the algorithm of [7] *pseudopatterns*, since they may not be valid.

To discover the actual patterns, we apply some changes in the original algorithm of [7]. While creating the max-subpattern tree, we store with each tree node the segment-ids that correspond to the pseudopattern of the node. In this way, one segment-id goes to exactly one node of the tree. However, $\mathcal{S}$ could be too large to manage in memory. In

function validate_pattern(P_{cand}, $\mathcal{L}_k$, min_sup);
1). $split$:=false; $prev\_size$:= $|P_{cand}|$
2). **for each** non-$*$ position i of P_{cand}
3). cluster points of R_i with $sid \in P_{cand}$;
4). **if** (more than one clusters with size $\geq min\_sup \cdot m$) **then**
5). $split$:=true;
6). **for each** cluster r with size $\geq min\_sup \cdot m$
7). P'_{new}:=$\{sid \mid sid \in r\}$;
8). **validate_pattern**(P'_{new}, $\mathcal{L}_k$, min_sup);
9). **else** P_{cand}:=segment-ids in updated cluster r;
10). **if** ($\neg split \wedge |P_{cand}| \geq min\_sup \cdot m$) **then**
11). **if** ($|P_{cand}| < prev\_size$) **then**
12). **validate_pattern**(P_{cand}, $\mathcal{L}_k$, min_sup);
13). **else** $\mathcal{L}_k$:=$\mathcal{L}_k \cup P_{cand}$;

Fig. 5. Validating a new pattern.

order to alleviate this problem, while scanning $\mathcal{S}$, for every segment s we encounter, we perform the following operations:

- First, we insert the segment to the max-subpattern tree, as in [7], increasing the counter of the candidate pseudopattern P' that s corresponds to after the transformation. An example of such a tree is shown in Fig. 7. This node can be found by finding the (first) maximal pseudopattern that is a superpattern of P' and following its children, recursively. If the node corresponding to P' does not exist, it is created (together with any nonexistent ancestors). Notice that the dotted lines are not implemented and not followed during insertion (thus, we materialize the tree instead of a lattice). For instance, for a segment with $P' = \{^*r_{21}r_{31}\}$, we increase the counter of the corresponding node at the second level of the tree.
- Second, we insert an entry $\langle P'.id, s.sid \rangle$ to a file F, where $P'.id$ is the id of the node of the lattice that corresponds to pseudopattern P' and $s.sid$ is the id of segment s. At the end, file F is sorted on $P'.id$ to bring together segment-ids that comply with the same (maximal) pseudopattern. For each pseudopattern with at least one segment, we insert a pointer to the file position, where the first segment-id is located. Nodes of the tree are labeled in breadth-first search order for reasons we will explain shortly.

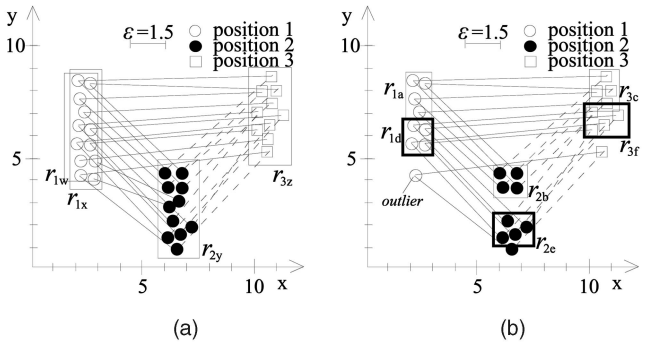

Fig. 6. Example of STPMine1. (a) 2-patterns. (b) Generated 3-patterns.

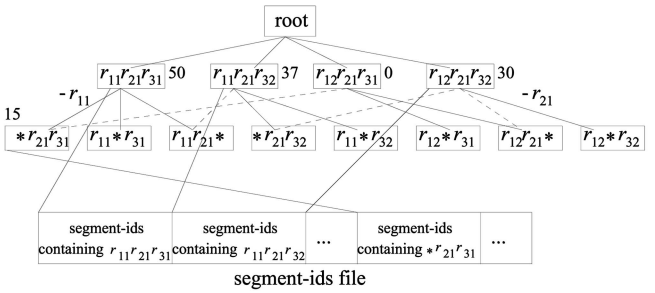

Fig. 7. Example of max-subpattern tree.

Instead of finding frequent patterns in a bottom-up fashion, we traverse the tree in a top-down, breadth-first order. For every pseudopattern with at least $min\_sup \cdot m$ segment-ids, we apply the **validate_pattern** function in Fig. 5 to discover potentially valid patterns. All segment-ids that belong to a discovered pattern are removed from the current pseudopattern. The rationale is that we are interested in patterns that are not spatially contained in some superpattern, so we use only those segment-ids that are not included in a pattern to verify its subpatterns.

Thus, after scanning the first level of the lattice, we may have discovered some patterns, and we may have shrunk segment-id lists of the pseudopatterns. Then, we move to the next level of the lattice. The support of a pseudopattern P' at each level is the recorded support of P' plus the supports of all its superpatterns (recall that a segment-id is assigned to the *maximal* pattern it complies with). The supports of the superpatterns can be immediately accessed from the lattice. If the total support of the candidate is at least $min\_sup \cdot m$, then the segment-ids have to be loaded for application of **validate_pattern**. The segment-ids of a superpattern may already be in memory from previous level executions. If not, they are loaded from the file F. After validation, only the disqualified segment-ids are kept to be used at lower level patterns. Traversal continues until

Algorithm STPMine2($\mathcal{L}_1$, T, min_sup);
1). build max-subpattern tree $\mathcal{T}$ and pattern-file F;
2). sort F on $P'.id$ and connect it to the nodes of $\mathcal{T}$;
3). **for** k:=T down to 2
4). **for each** pattern P' at level k of $\mathcal{T}$
5). $|P'|$:=$P'.counter$ +$\sum_{P'' \supset P', length(P'')=k+1} |P''|$;
6). **if** ($|P'| \geq min\_sup \cdot m$) **then**
7). P_{cand}:=$\bigcup_{P'' \supseteq P'} P''.sids$;
8). **validate_pattern**(P_{cand}, $\mathcal{L}$, min_sup);
9). **if** ($\mathcal{P}$ has changed) **then**
10). remove from P' those $sids$ in new patterns of $\mathcal{P}$;
11). **if** (unassigned $sids$ less than $min\_sup \cdot m$) **then**
12). **return** $\mathcal{P}$;
13). **return** $\mathcal{P}$;

Fig. 8. Top-down pattern mining.

there are no more patterns or it is not possible to find more patterns at lower levels of the lattice.

The fact that segment-ids are clustered in F according to the breadth-first traversal of the lattice minimizes random accesses and restricts the number of loaded blocks to memory. The segment-ids for a superpattern remain in memory to be used at lower level validations. If we run out of memory, the segment-ids of the uppermost lattice levels are rewritten to disk, but this time, possibly to a smaller file if there were some deletions.

A pseudocode for STPMine2 is shown in Fig. 8. Initially, the tree and the segment-ids file are created and linked. Then, for each level, we find the support of a pseudopattern $|P'|$ at level k by accessing only the supports of its superpatterns $P'' \supset P$ at level $k+1$, since we are accessing the tree in breadth-first order. If $|P'| \geq min\_sup \cdot m$, we validate the pattern as in STPMine1, and if some pattern is discovered, we remove from P' all those segment-ids that comply with the discovered pattern. Thus, the number of segment-ids decreases as we go down the levels of the tree, until it is not possible to discover any more patterns, or there are no more levels. Notice that the patterns discovered here are only maximal, as opposed to STPMine1, which discovers *all* frequent patterns. However, we argue that maximal patterns are more useful, compared to the huge set of all patterns. In addition, as we show in the experimental section, STPMine2 is much faster than STPMine1 for data, which contain long patterns.

4.3 A Simplified Algorithm: STPMine2-V2

In our definition, a pattern P is valid if 1) its frequency exceeds $min\_sup \cdot m$ and 2) the locations in R_i^P form a single dense cluster for all non-^* positions in P. Property 2) incurs a high computational burden to the mining algorithms, since it must be validated for every candidate pattern. Repetitive applications of the clustering algorithm and maintenance of the segment-ids that comply with each node of the max-subpattern tree are required.

In this section, we discuss a simplified version of our mining algorithm which considers the second property only in the discovery of frequent 1-patterns. In other words, after computing the dense regions at each R_i, we do not revalidate any clusters anymore. As a result, we need not use the segment-id lists at each node, but simply consider their counters to measure the pattern frequency. This mining technique is identical to STPMine2, excluding the validation and reassignment of segment-ids; thus, we call it STPMine2-V2.

Note that STPMine2-V2 is more inaccurate compared to STPMine2, since it may discover patterns that are not valid according to the definition by merging shorter actual patterns. In addition, the regions that define the patterns discovered by STPMine2-V2 are identical to the regions of the clusters forming frequent 1-patterns (e.g., the region refinement of the example in Fig. 6 is not performed). On the other hand, STPMine2-V2 is expected to be significantly faster than STPMine2. In Section 6, we validate the benefits and disadvantages of this simplification.

4.4 Performance Analysis

4.4.1 Time

Let the length of the maximal pattern be ℓ_{max}. STPMine1 needs to scan the data sequence ℓ_{max} times, verifying at the lth level all l-patterns. STPMine2 and STPMine2-V2 only need to scan the sequence two times: first to compute the frequent 1-patterns and then to construct the max-subpattern tree(s). For verifying an l-pattern P, both STPMine1 and STPMine2 must perform clustering l times, once for each non-^* position of P. Clustering has typically linear cost, as discussed in the Appendix. STPMine2-V2 saves time compared with STPMine2 since it just needs to calculate the support of a pattern, but does not recluster the points at each non-^* position; however, both methods have the same (linear) asymptotic performance.

4.4.2 Space

As far as the space is concerned, STPMine1 generates and validates candidates level-by-level, so its space complexity depends on the maximum number of candidates at a level. This typically corresponds to the number of candidates in the middle of the lattice of the examined space. In the worst case, if ℓ_{max} is the longest pattern length, the number of $\frac{\ell_{max}}{2}$-candidates is the number of cluster combinations at $\frac{\ell_{max}}{2}$ non-^* positions. This number can be estimated after the frequent 1-patterns have been extracted and it is in the same order as the space required to store the max-subpattern tree(s) of STPMine2 and STPMine2-V2. The space required to store these trees has been analyzed in [7]. In summary, all three methods have the same worst-case space complexity, but as discussed above, STPMine2 and STPMine2-V2 are much more time-efficient than STPMine1.

5 Variants of Periodic Patterns

As discussed in Section 1, the patterns followed by objects can be frequent only in some intervals of the whole movement history. In this section, we study the identification of periodic patterns and their associated validity eras, i.e., the time range(s) in which these patterns are frequent. In addition, we study the problem where pattern occurrences in certain time ranges may be shifted or distorted in time; in this case, mining is also adapted to consider such instances when computing the frequency of a pattern.

5.1 Patterns with Validity Eras

Let $\mathcal{S}$ be the trajectory of a moving object and T be a period. Based on them, we can define a set of m segments of $\mathcal{S}$ which are candidate pattern instances, as discussed in Section 3. For instance, segment s^j spans T consecutive locations in $\mathcal{S}$, starting from $l_{j \cdot T}$.

Definition 4. *An* **era** $[b, e]$ *is the subsequence of* $\mathcal{S}$*, from the beginning of segment* s^b *until the end of* s^e*. The* **time span** *of the era* $[b, e]$ *is* $e - b + 1$.

Era $[b, e]$ is a *superset* of era $[b', e']$ iff $b \leq b'$ and $e \geq e'$; accordingly, $[b', e']$ is a *subset* of $[b, e]$.

Definition 5. *A periodic pattern with a validity era, abbreviated as* **era pattern**, *refers to a periodic pattern associated with some era,* $P = r_0 r_1 \ldots r_{T-1}[b, e]$.

getValidEra($\mathbb{Q}_{era}, \mathcal{S}^P_{sid}, min\_sup$); //$\mathbb{Q}_{era}$: queue with eras need to be checked against P

1). **while** ($\mathbb{Q}_{era}$ is not empty)
2). $[b, e]$:= remove the first era from $\mathbb{Q}_{era}$ (with maximal time span);
3). **if** $((e - b + 1) < MinPts)$ **then break;**
4). $\mathcal{S}^P_{sid}[b, e]$:= subset of $\mathcal{S}^P_{sid}$ in the range $[b, e]$;
5). **if** $(|\mathcal{S}^P_{sid}[b, e]| \geq min\_sup \cdot (e - b + 1))$ **then output** $[b, e]$ as a valid era;
6). **else** add eras $[b + 1, e]$ and $[b, e - 1]$ into $\mathbb{Q}_{era}$;

Fig. 9. Get valid era for candidate pattern P.

Example. In Fig. 1c, the era of subsequence AACBDG AAACHG is [1, 2] ($T = 6$), whereas the era of the whole sequence is [0, 2]. Examples of era patterns are $\mathrm{AA}^{***}\mathrm{G}[0, 2]$ and $\mathrm{AAC}^{**}\mathrm{G}[0, 1]$.

Recall that $\mathcal{S}^P$ is the set of segments that comply with a pattern P. We use b_{min} and e_{max} to represent the minimum and maximum segment-ids in $\mathcal{S}^P$, respectively. Given $[b, e]$, a subset of $[b_{min}, e_{max}]$, let $\mathcal{S}^P_{[b,e]}$ contain all the segments in $\mathcal{S}^P$ with segment-ids in $[b, e]$, and $|\mathcal{S}^P_{[b,e]}|$ denote the number of segments in $\mathcal{S}^P_{[b,e]}$.

Definition 6. *Given min_sup and $MinPts$, the era pattern $P[b, e]$ is a* **valid era pattern** *if $|\mathcal{S}^P_{[b,e]}| \geq MinPts$ and*[2]

$$\frac{|\mathcal{S}^P_{[b,e]}|}{e - b + 1} \geq min\_sup.$$

Definition 7. *Consider two era patterns $P = r_0 r_1 \ldots r_{T-1}[b, e]$ and $P' = r'_0 r'_1 \ldots r'_{T-1}[b', e']$. P is a* **superpattern** *of P' if 1) $r_i = r'_i$ or $r'_i = *$ for $0 \leq i < T$ and 2) $[b, e]$ is a superset of $[b', e']$.*

In practice, $\mathcal{S}^P_{[b,e]}$ may be a subset of $\mathcal{S}^{P'}_{[b',e']}$.

Example. Consider three valid patterns $P = r_0 r_1 r_2 r_3{}^*[0, 100]$, $P' = r_0 r_1 r_2{}^{**}[0, 100]$, and $P'' = r_0 r_1 r_2 r_3{}^*[20, 100]$. Let $\mathcal{S}^P_{[0,100]}$ contain segments with ids $\{0, 20, 21, \ldots, 98, 100\}$, and $\mathcal{S}^{P'}_{[0,100]}$ contain one more segment with id 99. Although $\mathcal{S}^P_{[0,100]}$ is a subset of $\mathcal{S}^{P'}_{[0,100]}$, P is a superpattern of P' according to the definition. Similarly, P is a superpattern of P'' as well. However, P'' is not a superpattern of P', since the second condition of Definition 7 is not satisfied.

An era pattern P is *maximal* if it has no proper valid superpattern. Below is a formal definition of the problem studied in this section.

Definition 8. *The* **mining era patterns problem** *aims to find all the* maximal valid era patterns, *given a sequence $\mathcal{S}$, a period T, a minimum support min_sup ($0 < min\_sup \leq 1$), and cluster parameters ϵ and $MinPts$.*

5.1.1 Discovering Patterns and Their Validity Eras

We adapt the STPMine2-V2 algorithm (see Section 4.3), which we found the most efficient in our experimental study. As in Section 4, we follow two steps: detecting valid 1-patterns and discovering the patterns with longer length. We start by discovering the dense cluster(s), $r_i(\mathrm{s})$, from R_i for each period offset i. By setting the $i\mathrm{th}$ position to be r_i and all the other period positions to be *, we get a candidate 1-pattern. All these candidates are put in a set C_1 for computing their validity eras. In addition, $\mathcal{S}$ is replaced by $\mathcal{S}'$, a sequence of spatial regions and noise *.

Recall that $\mathcal{S}^P$ contains the segments complying with a candidate 1-pattern P. Let $\mathcal{S}^P_{sid}$ denote the set of segmentids in $\mathcal{S}^P$. To compute eras for a candidate pattern P in C_1, we run the algorithm in Fig. 9. The goal is to find the eras with the maximum time spans that render the pattern frequent. Parameter $\mathbb{Q}_{era}$ is a FIFO queue containing all the candidate eras that need to be validated for a pattern P. Initially, it contains only one era $[b, e]$, where b and e are the minimum and maximum values in $\mathcal{S}^P_{sid}$. If the era with the maximum time span does not make the pattern frequent, its two greatest subsets are inserted into $\mathbb{Q}_{era}$, and the algorithm continues until a valid era for the pattern is found or no interval of length greater than $MinPts$ exists (line 3). This algorithm is not restricted for 1-patterns, but could also be used to identify validity eras for patterns with arbitrary lengths.

Note that the algorithm may output multiple (maximal) validity eras for a given pattern. In order to avoid exploding the space of potential solutions, we choose to terminate it when the first era is output. Since the contents of $\mathbb{Q}_{era}$ are in descending order of their time spans, the first interval to be output is guaranteed to be the longest. Alternatively, we may collect all maximal eras and pick a subset that consists of maximal nonoverlapping intervals. This allows us to detect a pattern which is frequent in different segments of the history.

For finding the longer era patterns, we adapt STPMine2-V2 to a new algorithm, which we call *EPMine* (Era periodic Pattern Min(e)ing). Next, we describe how to compute the candidate max-subpatterns, build max-subpattern trees, and derive valid patterns from them.

A max-subpattern is formed by combining the valid 1-patterns, as discussed in Section 4. To determine the era of a max-subpattern P, we take the *union* of the eras from the 1-patterns that define P. The union of a set of eras $\{[b_1, e_1], [b_2, e_2], \ldots [b_k, e_k]\}$ is defined by $[\min_{i=1}^k b_i, \max_{i=1}^k e_i]$. In addition, we require that the eras of the 1-patterns that form a max-subpattern P have a nonempty intersection; otherwise, there can be no valid instance of P. For example, for three 1-patterns: $\mathrm{a}^{**}[0, 10]$, $^*\mathrm{b}^*[1, 9]$, and $^{**}\mathrm{c}[2, 11]$, the max-subpattern is $\mathrm{abc}[0, 11]$. The computation of candidate

2. We use the first condition since we need to have at least $MinPts$ points in each valid region for a pattern P.

max-subpatterns requires one scan of the 1-patterns if these are ordered by validity time.

After forming the candidate max-patterns, EPMine builds the max-subpattern tree for each of them. Sequence $\mathcal{S}'$ is then scanned and each segment is inserted into the trees whose era contains the corresponding segment-id. Valid era patterns are derived from a max-subpattern tree by scanning it in a breadth-first order; for each candidate pattern P, the set $\mathcal{S}^P_{sid}$ is extracted, the initial era $[b, e]$ is obtained by the minimum and maximum sid in $\mathcal{S}^P_{sid}$, and the algorithm of Fig. 9 is eventually run.

5.2 Shifted and Distorted Patterns

Recall that s^j denotes a segment starting at position $j \cdot T$. Given a tolerance integer τ $(0 \leq \tau \leq \lfloor T/2 \rfloor)$, a segment starting at position $j \cdot T + d$, $-\tau \leq d \leq \tau$, is denoted by $s^j[d]$ (note that $s^j[0] = s^j$).

Definition 9. *Given a sequence $\mathcal{S}$ and an integer τ, a segment $s^j[d]$, $-\tau \leq d \leq \tau$, is a* **shifted pattern instance** *of a pattern P if it complies with P, i.e., P's occurrence in $\mathcal{S}$ is shifted at most τ timestamps forward or backward from its expected position $j \cdot T$.*

Example. Let $T = 5$ and $\mathcal{S}' = r_0r_1r_2r_3r_4r_0r_0r_1r_4r_3r_2r_0r_1r_3r_3$ be the transformed sequence after replacing the locations in $\mathcal{S}$ by spatial regions. The pattern $r_0{r_1}^*{r_3}^*$ has one nonshifted instance, s^0, starting at position 0, and two shifted pattern instances, $s^1[1]$ and $s^2[1]$, starting at positions 6 $(1 \cdot T + 1)$ and 11 $(2 \cdot T + 1)$.

There are cases, where the pattern instances are not simply shifted, but they are distorted.

Definition 10. *A segment $s^j[d]$, $-\tau \leq d \leq \tau$, is a* **distorted instance** *for a pattern $P = r_0r_1 \ldots r_{T-1}$ with length P_{len}, with respect to τ, if there exist P_{len} ordered locations in $s^j[d]$ such that 1) these locations follow the order of non-* elements in P, and 2) for every non-* element in P, its period offset differs at most τ from the period offset of its related location in $s^j[d]$.*

Example. Consider a segment $s^0 = l_0l_1l_2l_3l_4$ and let $\tau = 1$. If $l_1 \in r_0$, $l_2 \in r_2$, and $l_4 \in r_3$, s^0 is a distorted instance of pattern $P = {r_0}^*r_2{r_3}^*$.

Two pattern instances (segments) *overlap* if they have some locations in common. For example, $s^0[1] = l_1l_2l_3l_4l_5$ overlaps with $s^1 = l_5l_6l_7l_8l_9$ since they have l_5 in common.

Definition 11. *If a pattern P has more than $min\_sup \cdot m$ (shifted/distorted) pattern instances in $\mathcal{S}$, such that no two instances overlap, then P is a frequent pattern with shifted/distorted instances. Given a sequence $\mathcal{S}$, minimum support min_sup $(0 < min\_sup \leq 1)$, cluster parameters ϵ and $MinPts$, and maximum shifting/distortion parameter τ $(0 \leq \tau \leq \lfloor T/2 \rfloor)$, the* **problem of discovering shifted/distorted patterns** *aims at finding all frequent patterns with shifted/distorted instances from $\mathcal{S}$.*

5.2.1 Mining Patterns with Shifted and/or Distorted Instances

As discussed in Section 4.1, 1-patterns can be mined after we divide the sequence $\mathcal{S}$ of locations into T data sets and apply clustering to each of them. In order to consider shifted/distorted pattern instances in this process, for an object location at offset position i, instead of generating a single point in the corresponding data set R_i, we generate a point at all τ-neighbor positions

$$R_{(i-\tau) \bmod T}, R_{(i-\tau+1) \bmod T}, \ldots, R_{(i+\tau) \bmod T}.$$

Consider, for instance, the fifth position of day 1 in Fig. 3a and assume that $\tau = 1$. Instead of generating a single "□" point at that location, we generate one "□" point (to file R_5), one "+" point (to file R_4), and one "×" point (to file R_6). In other words, there is a data replication with a factor $2 \cdot \tau + 1$; however, this ensures that shifted/distorted patterns are counted in the supports of the actual positions.

We adopt STPMine2-V2 to *SPMine* (Shifted/distorted Pattern Min(e)ing) in Fig. 10 to facilitate the counting of longer (shifted/distorted) pattern instances. SPMine also works in a top-down manner, starting the pattern validation from the max-subpattern P_{max} and continuing down to patterns of shorter lengths level-by-level. SPMine does not utilize the max-subpattern tree, but it still generates max-subpatterns by combining the frequent 1-patterns which have non-* elements at different period offsets. For example, from 1-patterns ${}^*r_1{}^{***}$, ${}^{***}r_3{}^*$, and ${}^{****}r_4$, we get the max-subpattern ${}^*r_1{}^*r_3r_4$.

The pseudocode of Fig. 10 describes how to extract frequent patterns from P_{max}. We examine the subpatterns of P_{max} level-by-level. For each candidate subpattern P, which is formed by a set of clusters, one for each non-* position i, we initialize a pointer p_i to the first point in each cluster r_i. Then, we perform a merge-join by synchronously scanning the contents of the clusters and attempting to find shifted/distorted pattern instances from the sets of points currently indexed by each p_i (lines 6–15). Given the current pointer positions, if the set of locations is a valid shifted/distorted pattern instance, then we increase all pointers, as we do not want to count more instances that share locations with the current one. Otherwise, if there is a pair of points with identical locations (which have been clustered to different offsets due to replication), we increase the pointer in the

Algorithm SPMine(P_{max}, T, min_sup);
1). **for** l:= ℓ_{max} down to 2 //ℓ_{max} is the length of P_{max}
2). **for** every subpattern of P_{max} with length l and with no frequent superpattern;
3). $|P|$:= 0; //$|P|$ is the support of P (Definition 2)
4). **for** each non-* position i of P with cluster r_i
5). p_i:=first point in r_i;
6). **if** ($\{p_1, p_2, \ldots, p_l\}$ is not a valid instance of P) **then**
7). **if** ($p_i = p_j$, for some $i < j$) **then**
8). p_j:= next point in r_j;
9). **else** j:=$\{i : p_i$ has the smallest timestamp$\}$;
10). p_j:= next point in r_j;
11). **else** //valid pattern instance
12). $|P|$:=$|P| + 1$;
13). **for** each non-* position i of P with cluster r_i
14). p_i:= next point in r_i;
15). **if** (more points in all r_i) **then goto** line 6;
16). **if** ($|P| \geq min\_sup \cdot m$) **then** report P;

Fig. 10. Shifted/distorted pattern mining.

cluster which corresponds to the largest offset (lines 7-8). If there is no such pair of identical points, we increase the pointer with the smallest timestamp (lines 9-10). Finally (line 16), we report the pattern if it is found to be frequent. Note that we do not discover patterns whose superpatterns are frequent in order to improve the scalability of the method.

Example. Assume that we run SPMine to retrieve the distorted patterns and initially get clusters $r_0 := \{l_0, l_5, l_{10}\}$ and $r_1 := \{l_1, l_5, l_6, l_7, l_{11}\}$. Consider a candidate pattern $P = r_0 {r_1}^{***}$, and let $\tau = 1$. (l_0, l_1) is the first point-id pair from the two sets, falling into the same segment, so they contribute 1 to $|P|$. The next pair, (l_5, l_5), contains identical ids so it does not contribute to P. Keeping $p_0 = l_5$ from r_0 unchanged, we get the next location, $p_1 = l_6$, from r_1. The current locations (l_5, l_6) form a segment and add 1 to P's frequency. Then, we proceed to location pair (l_{10}, l_7) (not an instance) and, finally, continue to the contributing join pair (l_{10}, l_{11}).

Because of the replication effect, SPMine may generate redundant candidates. In order to alleviate this problem, we can *weigh* the replicated points with a number antiproportional to their distance from their actual temporal positions, in order to penalize distortion and increase accuracy. In counting the 1-patterns after clustering, a non-^* element which is shifted τ positions from its expected period offset is given the support $1 - \tau \times w$, where w can be any value in (0,1) depending on how much the user wants to take into account the shifted/distorted pattern instances. Consider the example of Fig. 3 and let $\tau = 1$ and $w = 0.5$. For counting the support of ${}^{****}r_5{}^*$, we give, for each exact "□" point, a weight 1, but for "+" and "×" points, only a weight 0.5; these are approximate and should be treated with reduced significance in counting. When counting the occurrences of an l-pattern P $(l \geq 2)$, we add for each pattern instance the maximal weight of all the non-^* elements in it. We denote this weighted variant of SPMine by SPMine-w, while we use SPMine-b to refer to the original method with $w = 0$.

6 Experimental Evaluation

We implemented and evaluated the mining techniques presented in the paper. The language used was C++ and the experiments were performed on a Pentium III 700MHz workstation with 1GB of memory, running Unix. Because of the lack of real data, we generated synthetic data that simulate periodic movements. We introduce our synthetic data generator in Section 6.1 and show the effectiveness and efficiency test results in Section 6.2 and Section 6.3.

Setting the mining parameters. We assume that the period T is known by the user and given as an input parameter. In many applications (including Bob's daily activities example mentioned in Section 1), this is a realistic assumption. The automatic derivation of T from the data is an issue, which is out of the scope of this paper. We note that current periodicity detection algorithms (e.g., [4]) may not be applicable to our problem, since these methods apply to a priori discretized data. In addition, if the actual period is no greater than τ compared to T, the shifted/distorted mining variant could be used to discover the patterns. In the future, we plan to study their adaptation for our problem. The two clustering parameters, $MinPts$ and ϵ, used to control the density of a region can generally be determined by the sampling method proposed in [5]. In our experiments, we work with synthetically generated data, for which ϵ and $MinPts$ can be derived from the parameters of the data generator.

6.1 Synthetic Data Generator

In order to test the effectiveness and efficiency of the techniques under various conditions, we designed a generator for long object trajectories, which exhibit periodicity according to a set of parameter values. These parameters are the length n of the time history (in timestamps), the period T, the length ℓ of the maximal frequent patterns followed by the object $(\ell \leq T)$, and a probability f for a periodic segment in the object's movement to comply with no hidden patterns (i.e., the movement during this segment is irregular).

Before generating the movement, the approximate regions for the maximal periodic patterns are determined. Let P be a generated pattern. A random circular route is generated in space, and for each non-^* position i in P, a spatial location l_{P_i} (i.e., point) on that route is determined, such that the distance between two non-^* positions on the route is proportional to their temporal distance in the pattern. Afterwards, the movement of the object is generated. For every periodic segment s, we determine whether s should be a noise (i.e., irregular) segment or not, given the probability f.

If s is a regular segment, a random maximal pattern P is selected, and the object's movement is generated as follows: If the next segment location to be generated corresponds to a non-^* position i of P, the location l_i is generated randomly and within a distance E from the spatial location l_{P_i} of the non-^* position. E ranges from 0 to 2 percent of the map size. Otherwise (i.e., l corresponds to a * position), l_i is generated randomly, but such that the movement is *targeted* to the next periodic location. In other words, 1) l_i *moves* with respect to the previous segment location l_{i-1} toward the next non-^* position j, and 2) its distance from the previous location l_{i-1} is the spatial distance between l_{i-1} and l_{P_j} divided by $j - i + 1$, i.e., the temporal distance between these two positions. In order to prevent regular movements, both the distance and direction angle are distorted. Specifically, we add to the angle (in radians) a random number in $[-1, 1]$ and the distance is multiplied by a number between [1.5, 0.8].[3] If s is a noise segment, the object can move everywhere in space. The movement is determined by a random direction angle (with respect to the previous location), and a random distance in $[0, maxwalk]$, where $maxwalk$ is used to control the maximum *walking* distance of the object between two timestamps. In order to avoid extreme jumps, after half of the movements in a noise segment, the rest are generated to *target* to the next periodic position, using the method described above.

For generating the era patterns, we add to the generator one more parameter E_n to determine the number of hidden era patterns. Given T, the generator first produces E_n patterns. Given the length n of the desired sequence $\mathcal{S}$, an era pattern is hidden in a subsequence of $\mathcal{S}$, each of

3. These values were tuned to match realistic object movements and at the same time to disallow falsely generated periodic patterns.

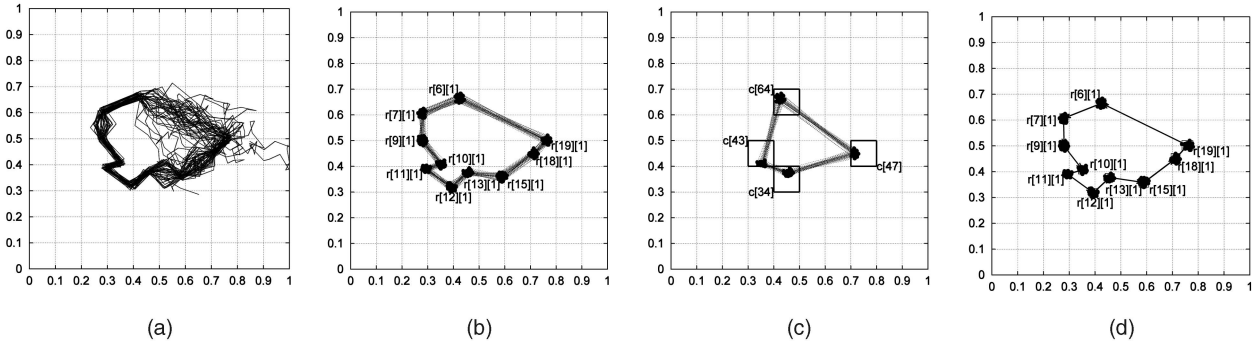

Fig. 11. Effectiveness comparison. (a) Input file. (b) P by STPMine. (c) A grid-based P'. (d) P by STPMine2-V2.

which contains approximately $\frac{n}{E_n}$ consecutive locations. In generating a subsequence that covers one era pattern, a segment in it contributes to its hidden era pattern with probability $1 - f$.

To generate shifted/distorted pattern instances, we divide the generated segments that comply with some hidden pattern into $\tau + 1$ partitions, and the segments in partition i $(0 \leq i \leq \tau)$ are shifted/distorted i timestamps forward or backward after a coin-flip.

6.2 Effectiveness

The first experiment demonstrates the effectiveness of the baseline mining techniques proposed above in Section 4. We generated a small data set, with $n = 1,000$ (i.e., there are only 1,000 locations in the object's trajectory). T is set to 20, and the object follows a single periodic pattern P at 39 out of 50 segments, whereas the movement is irregular in 11 segments. Fig. 11a shows the object trajectory, where the periodic movement can roughly be observed. For this data set, $\ell = 10$, i.e., there are 10 non-* positions in P. Fig. 11b shows the maximal frequent pattern P of length 10, successfully discovered by STPMine1 and STPMine2 when $min\_sup = 0.6$. The non-* positions are 6, 7, 9, 10, 11, 12, 13, 15, 18, and 19. We plot the object's movement, interpolated using only the non-* positions. The discovered pattern is identical to the generated one. The dense regions are successfully detected by the clustering module, and the spatial extents of the pattern are minimal.

We also developed and tested a technique that applies directly the data mining algorithm for event sequence data [7]. The space is divided using a regular $M \times M$ grid. Then, each location of $\mathcal{S}$ is transformed to the cell-id which encloses the location. For instance, if we assume that all locations are in a unit [0, 1] × [0, 1] space, a location $l = \langle x, y \rangle$ is transformed to a cell with id $\lfloor y \cdot M \rfloor \cdot M + \lfloor x \cdot M \rfloor$. Then, we use the algorithm of [7] to find partial patterns that are described by cell-ids. We call this the *grid-based* mining method. The time and space complexity of this method is asymptotically the same to that of STPMine2-V2 (analyzed in Section 4.4); the *grid-based* algorithm only saves the (linear) cost of applying clustering for identifying the frequent 1-patterns. However, as shown later, STPMine2-V2 is much more effective. Fig. 11c shows a maximal pattern P' discovered by this grid-based technique when using a 10×10 grid. P' has the largest length among all discovered patterns; however, it is only 4 (whereas the actual pattern P has 10 non-* positions). The non-* positions of P' are 6, 10, 13, and 18, captured by cells c_{64}, c_{43}, c_{34}, and c_{47}, respectively. We repeated the experiment using different grid granularities; for a 20 × 20 grid, no pattern is generated, whereas with a 5 × 5 grid, we get a maximal 9-pattern, which, however, is not very descriptive as the cells are very big. Thus, with a grid with fine granularity, frequent regions which span multiple cells cannot be identified (e.g., the cluster r [19], [1] is split between cells c_{47} and c_{57} and neither of these cells has higher support than $min\_sup \cdot m$), whereas, with a grid of low granularity, the patterns are formed by very large regions. From this small example, we can see the importance of discovering the periodic patterns and *their descriptive regions* effectively.

STPMine2-V2 also finds the maximal pattern with length 10 shown in Fig. 11d. This pattern has the same non-* positions as that in Fig. 11b, and the region for each non-* position is represented with the MBR (Minimum Bounding Rectangle) of its associated initial cluster. Thus, STPMine2-V2 retrieves comparative results to STPMine1 and STPMine2 in finding the descriptive regions and patterns. As discussed before, STPMine 1 and STPMine2 identify the same maximal patterns which are used to generate data. STPMine2-V2 finds patterns similar to that of STPMine2 except that the non-* regions are a little larger (i.e., a little less descriptive) than the more accurate ones discovered by STPMine2 (see Fig. 6).

Fig. 12 shows the effectiveness of EPMine in discovering patterns and their valid eras. For generating the data file in Fig. 12a, we set parameters $T = 10$, $E_n = 2$, $min\_sup = 0.8$, and the total number of segments to 20. Given $min\_sup = 0.8$, we could find the two patterns hidden in the sequence with validity eras [0, 9] and [10, 19], respectively.

We now compare the effectiveness of SPMine-b and SPMine-w in finding shifted/distorted patterns on two generated data sets. Table 1 shows the length of patterns found by each of these two methods. Table 1a displays the result for a small data set to generate which we set $T = 10$, $n = 25K$, and maximal pattern length $\ell = 8$. In most cases, both SPMine-b and SPMine-w could find the hidden pattern used to generate the sequence. However, SPMine-w sometimes misses some non-* positions. For example, when $\tau = 4$, it can only find patterns of length 7, which are shorter than the hidden patterns (8). This problem is more obvious in Table 1b, which shows the result on a big data set, for which the generation parameters are $n = 1M$, $T = 50$, and the maximal pattern length $\ell = 5$. When $\tau = 3, 4, 5$, SPMine-w finds patterns shorter than the hidden maximal pattern while SPMine-b could find the generated hidden patterns.

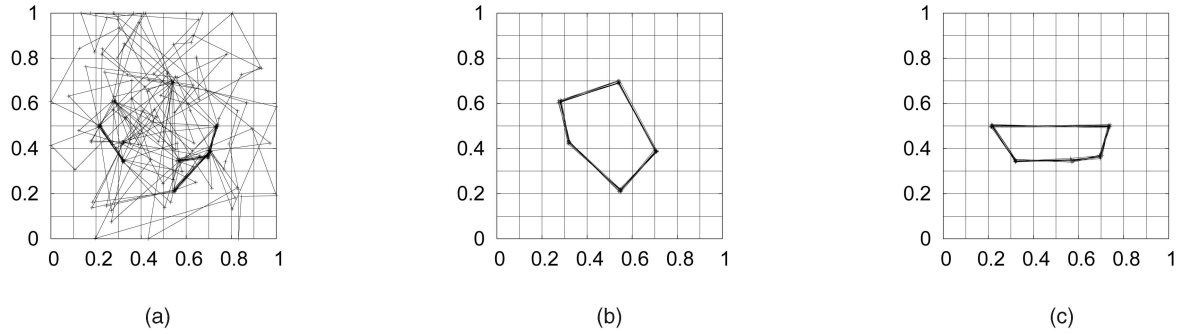

Fig. 12. Example of mining patterns and their valid eras. (a) Data file. (b) Pattern 1. (c) Pattern 2.

6.3 Efficiency

In the next set of experiments, we validate the efficiency of the proposed techniques under various data settings. First, we compare the costs of the (ineffective) grid-based method, STPMine1, STPMine2, and STPMine2-V2 as a function of the length of the maximal hidden pattern. We generated a sequence $\mathcal{S}$ of $n = 1M$ object locations, and set $T = 100$ and $min\_sup = 0.7$. For this and subsequent experiments, we used $\epsilon = 0.005$ and $MinPts = 200$ in the clustering module.

Fig. 13a plots the results. Naturally, the grid-based approach is the fastest method since it performs no clustering and no refinement of the discovered regions. However, as shown in the previous section, it misses the long patterns in all tested cases. Moreover, its efficiency is due to the fact that a large fraction of actual 1-patterns are missed and the search space is pruned. STPMine1 is very slow when the hidden patterns are long. Like most bottom-up mining techniques, it suffers from the huge number of candidates that need to be generated and validated, and, therefore, it is inapplicable for the tested cases where the hidden patterns have more than 10 non-* positions. STPMine2 is very efficient and scales well because it uses the first phase to identify fast large patterns that are potentially useful. Even when reclustering fails for the maximal candidate patterns, the actual patterns are discovered usually only after few hops down the max-subpattern tree. Observe that, even though STPMine2 performs clustering a large number of times, it is not significantly slower than the ineffective grid-based approach. Interestingly, it outperforms the grid-based method when there is a single hidden pattern with length equal to T. In this case, the grid method spans many actual clusters between grid cells and splits the actual pattern to multiple maximal frequent patterns, the support of which is expensive to count in the large lattice. STPMine2-V2 is faster than the original version STPMine2 because it does not need to perform the reclustering. Furthermore, the difference in their execution time rises when the maximal pattern length increases since, for getting maximal patterns with longer length, STPMine2 takes more time in the process of reclustering. In addition, with the increase of the maximal pattern length, the execution time of STPMine2-V2 goes down slightly while that of STPMine2 rises a little. This is because more time is used in the initial cluster process to generate frequent 1-patterns when the length of maximal pattern is shorter. We use five (10) and 100 (80) to test the effect of extremely (very) short and long pattern lengths on the performance, while 50 represents the moderate pattern length.

In the next experiment, we test the effects of period length on the same database size, but with different values of T. The length of the maximal hidden pattern is $0.5 \cdot T$ in all cases. Again, $n = 1M$ and $min\_sup = 0.7$. Fig. 13b compares the costs of the grid-based approach, STPMine2. and STPMine2-V2; we do not include the cost of STPMine1 since this method is very slow for long patterns. The figure shows that the costs of the three methods are almost invariant to T for a constant database size n. If T is small, then there are few, but large files to be clustered by STPMine1. On the other hand, for large T, there are many, but small R_i to be clustered.

We also test the scalability to the length n of the spatiotemporal sequence $\mathcal{S}$. Fig. 13c shows the costs of STPMine2, STPMine2-V2, and the grid-based approach as a function of n, when $T = 100$ and the maximal pattern length is 50.[4] Observe that all methods are scalable since the database size is only linearly related to the cost of finding and validating the maximal patterns. STPMine2-V2 shows better performance because of the reasons we mentioned already. In summary, STPMine2 and STPMine2-V2 are effective and efficient techniques for mining periodic

TABLE 1
Effectiveness Comparison for Shifted/Distorted Pattern Mining Methods

τ	Pattern length	
	SPMine-b	SPMine-w
1	8	8
2	8	8
3	8	8
4	8	7
5	8	8

(a)

τ	Pattern length	
	SPMine-b	SPMine-w
1	5	5
2	5	5
3	5	4
4	5	4
5	5	4

(b)

(a) Small data set. (b) Big data set.

4. Trajectories with millions of positions can be commonly tracked by sampling very frequent intervals (e.g., seconds) over a long history (e.g., months).

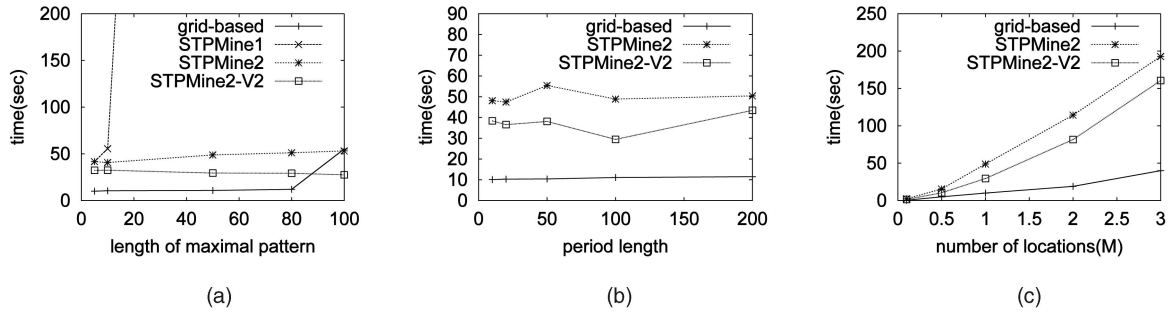

Fig. 13. Efficiency test. (a) Cost versus max-subpattern length. (b) Cost versus period length. (c) Cost versus database size.

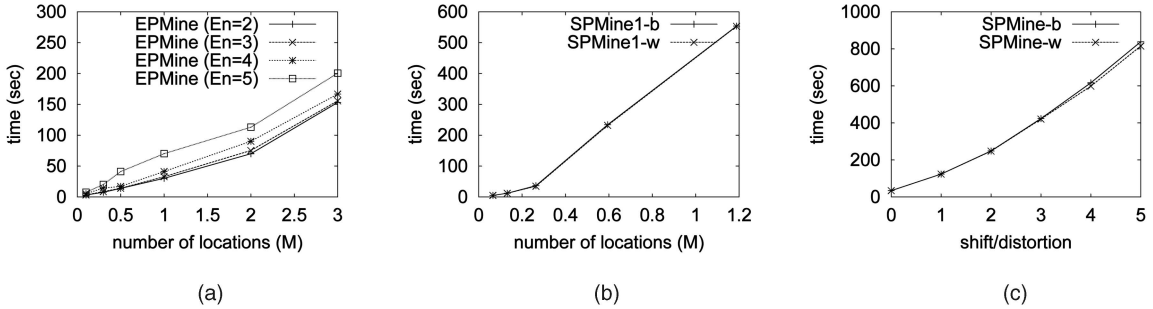

Fig. 14. Efficiency test of era pattern and shifted/distorted pattern mining. (a) EPMine: Cost versus database size. (b) SPMine: Cost versus database size. (c) SPMine: Cost versus τ.

patterns and their accurate descriptive regions in spatiotemporal data.

Fig. 14 shows the performance of the era pattern mining method and the shifted/distorted pattern mining approaches. Fig. 14a demonstrates the scalability of EPMine. For this test, we set T be 100 and vary the number of locations n in the sequence from 100K to 3M. The running time is plotted with different E_n (the number of era patterns). All maximal patterns in the generated data set could be found. It is clear that EPMine scales linearly to the number of locations for different E_n, which is compatible with the results in Fig. 13c. In addition, for the same n, the running time slightly increases with E_n because more candidates need to be validated.

Fig. 14b illustrates the scalability of the shifted/distorted pattern mining methods. In this experiment, we fix $T = 100$, $\tau = 2$, and vary n from 50K to 1M. Note that SPMine-w and SPMine-b have a similar performance because the weighted counting reduces the candidate support only a little. Fig. 14c demonstrates how the tolerance parameter τ affects the mining time when $n = 1M$ and $T = 50$. The time increases almost linearly with τ, since the clustered locations increase by a factor of $2 \cdot \tau + 1$ and the maximal candidate patterns are longer for bigger τ.

7 Conclusion

In this paper, we studied the discovery of periodic patterns from a long spatiotemporal sequence. We identified the differences of the problem in comparison to mining periodic patterns from event sequences and described effective and efficient algorithms for solving it. Our methods employ spatial clustering to retrieve frequent 1-patterns and adapt bottom-up and top-down mining techniques for longer patterns. In addition to the baseline problem, we defined and solved two practicable variants. The first is the discovery of periodic patterns that are not frequent in the whole time span of the sequence, but only in a time interval, called validity era, which is to be discovered automatically. To solve this problem, we adjust the definition of periodic patterns to be associated with a maximal validity interval and we adapt the mining algorithms to identify the validity eras for patterns while counting their supports. The second mining variant counts shifted or distorted instances of patterns. We redefined frequent 1-patterns to consider such instances and refined the mining algorithm to discover longer patterns.

Topics for future work include the automatic discovery of the period T related to frequent periodic patterns and the discovery of patterns with distorted period lengths. For instance, the movement of an object may exhibit periodicity, however, the temporal length of the period may not be fixed but could vary between pattern instances. Public transportation vehicles may have this type of periodicity, since during heavy traffic hours, a cycle can be longer that usual. Building indices based on distorted and shifted patterns is also an interesting direction for future work.

Appendix

Our mining algorithms apply density-based clustering to identify the spatial regions that are components of the mined patterns. We devised an efficient hash-based implementation of DBSCAN [5], which typically achieves linear performance. The pseudocode of this method is shown in Fig. 15. Given clustering parameters ϵ and $MinPts$, we partition the space using a regular grid of $\frac{\epsilon}{\sqrt{2}} \times \frac{\epsilon}{\sqrt{2}}$ cells and hash each point to be clustered into the cell that contains it.

Algorithm gridDBSCAN(set of points $\mathcal{P}$);

1). hash $\mathcal{P}$ to an $\epsilon/\sqrt{2} \times \epsilon/\sqrt{2}$ grid;
2). **for** each cell C_i of the grid
3). **if** ($|C_i| \geq MinPts$) **then** //dense cell
4). merge C_i with prev. dense cells, if applicable;
5). **for** each cell C_i of the grid
6). **if** ($|C_i| < MinPts$) **then** //sparse cell
7). **for** each point $p \in C_i$ unassigned to a cluster
8). update $p.sup$ from C_i's ϵ-neighbor sparse cells;
9). **for** each ϵ-neighbor dense cell C_j of C_i
10). **if** ($p.sup \geq MinPts$) **then**
11). check potential merging with C_j's cluster;
12). **else** update $p.sup$ by checking points in C_j;
and potentially link or merge p with C_j;
13). **if** ($p.sup \geq MinPts$) **then**
14). **if** (unassigned(p)) **then**
15). create new cluster for p;
16). expand p's cluster from ϵ-neighbors in C_i and next cells;

Fig. 15. Grid-based clustering algorithm.

In its first phase, the algorithm performs a pass over the cells and uses their hash counters to determine whether a cell C_i is *dense* (i.e., it contains at least $MinPts$ points) or not. A dense cell C_i is always a part of a cluster, since the maximum possible distance between any two points in it is at most ϵ (the diagonal of the cell). Therefore, any point there is a core point, based on the definition of [5]. The pass of lines 2-4 finds all pairs (C_i, C_j) of dense cells with distance no greater than ϵ to each other and checks whether there is at least a pair of points $(p_i, p_j), p_i \in C_i, p_j \in C_j$, such that $dist(p_i, p_j) \leq \epsilon$. In that case, the corresponding clusters are merged, because p_i and p_j are both core points and one is in the ϵ-neighborhood of the other. Consider, for example, cell C_{26} in the grid of Fig. 16 and assume that $MinPts = 4$. All cells within the bold line are ϵ-neighbors of C_{26} (i.e., they could contain points within ϵ distance from a point in C_{26}). Since C_{26} is dense (it has four points), all ϵ-neighbor cells before it (the shaded cells in the figure) are examined for potential merging with C_{26}, if they are also dense. During this process, C_{24} and C_{26} are merged to the same cluster.

In the second phase (lines 5-16), the algorithm again scans the cells, treating this time sparse ones (i.e., cells with $|C_i| < MinPts$). For each point p in a sparse cell C_i, we first compute the number $p.sup$ of p's ϵ-neighbors in C_i and in sparse ϵ-neighbor cells of C_i. If $p.sup \geq MinPts$, we already know that p is a core point. Next, we check the dense ϵ-neighbors of C_i. For each such cell C_j, if p is already known to be a core point and we could find a point $p' \in C_j$ such that $dist(p, p') \leq \epsilon$, we add p and its ϵ-neighborhood points in the cluster of C_j. If p is not yet known to be a core point (i.e., $p.sup < MinPts$), then we scan cell C_j and increase $p.sup$ as p's neighbors are found in C_j, until no more points exist in C_j or p becomes a core point. As soon as p is known to be a core point, merging is performed with the cluster of C_j. If p is a found to be an ϵ-neighbor of a dense cell, but it is not a core point yet, then we link p as *density-reachable* from C_j and include it into C_j's cluster. If p is later found to be a core point, C_j's cluster is merged with any other clusters close to p and the ϵ-neighborhood of p. Finally (lines 13-16), if p is found to be a core point, p's cluster is expanded from the next points in C_i (and succeeding cells) that are ϵ-neighbors of p, like in the original DBSCAN algorithm. A subtle point to note is that, once we start expanding the cluster that includes point p (line 16), we proceed working with points and dense cells related to that cluster only. When the whole cluster is identified, we return to point p and process the next unassigned point in C_i or succeeding cells.

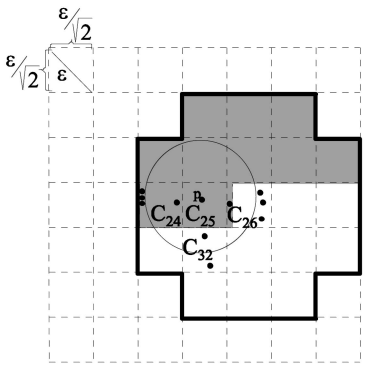

Fig. 16. Clustering example.

Fig. 16 exemplifies the functionality of the algorithm. Assume that $MinPts = 4$. As discussed, cells C_{24} and C_{26} are identified as dense and they are merged in the first phase of the algorithm (lines 2-4). In the second phase of the algorithm, when sparse cell C_{25} is examined, we find p, with initial $p.sup = 1$. Then (line 8), we update $p.sup = 2$ by searching the ϵ-neighbor cells of C_{25} that are sparse (i.e., C_{32}). Next, we check dense ϵ-neighbor cells of C_{25}, starting from C_{24}. We find a point there in p's neighborhood and update $p.sup = 3$. Since p is not yet a core point, it is just linked to the cluster of C_{24} (as density-reachable). After we check C_{26}, we find another neighbor of p there, thus $p.sup = 4$ and now p becomes a core point. Now, if we know that the cluster containing C_{24}, C_{26}, and p has one more point in C_{32}, then we move to C_{32} to process that point and expand the cluster as necessary (line 16).

Our method achieves the same result as that of DBSCAN, while being much faster. The original DBSCAN algorithm has worst-case $O(n^2)$ cost, since finding the ϵ-neighborhood of any point requires a scan of the database. The cost can be reduced to $O(n \log n)$ if a spatial index facilitates neighborhood retrieval. Such indices do not exist for the arbitrary sets of points that are clustered by the mining algorithm. Of course, an index could be built on-the-fly before clustering, but our grid-based method avoids this. It requires one scan of the data to create the grid-based partitions. Then, the dense cells are merged at a single pass and many computations are saved, since we know (without any search) that all points in such cells are core points. Finally, sparse cells are handled at a single pass of the database, since ϵneighborhoods are efficiently found from the neighboring cells of the current point. In practice, the cost of our method is linear to the database size.

Grid-based clustering has also been used by STING [18] and STING+ [19], albeit the aim of these methods is to provide a data summary for fast (approximate) range query evaluation. These algorithms split the space into cells and use a hierarchical structure to organize them; the points in a cell are put to a cluster only if the density of the cell is no less than $\frac{MinPts}{\pi\epsilon^2}$. The points in the sparse cells are not considered at all. The clusters of STING are similar to those of DBSCAN only when the granularity of the bottom-level cells is close to zero. Our method is essentially different from STING, since it is merely a grid-based efficient implementation of DBSCAN.

ACKNOWLEDGMENTS

This work was supported by grant HKU 7142/04E from Hong Kong RGC.

REFERENCES

[1] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," *Proc. Very Large Data Bases Conf.,* pp. 487-499, 1994.
[2] R. Agrawal and R. Srikant, "Mining Sequential Patterns," *Proc. Int'l Conf. Data Eng.,* pp. 3-14, 1995.
[3] B. Chiu, E. Keogh, and S. Lonardi, "Probabilistic Discovery of Time Series Motifs," *Proc. Ninth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 493-498, 2003.
[4] M.G. Elfeky, W.G. Aref, and A.K. Elmagarmid, "Periodicity Detection in Time Series Databases," *IEEE Trans. Knowledge and Data Eng.,* vol. 17, no. 7, pp. 875-887, July 2005.
[5] M. Ester, H.P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," *Proc. ACM Second Int'l Conf. Knowledge Discovery and Data Mining,* pp. 226-231, 1996.
[6] M. Hadjieleftheriou, G. Kollios, V.J. Tsotras, and D. Gunopulos, "Efficient Indexing of Spatiotemporal Objects," *Proc. Eighth Int'l Conf. Extending Database Technology,* pp. 251-268, 2002.
[7] J. Han, G. Dong, and Y. Yin, "Efficient Mining of Partial Periodic Patterns in Time Series Database," *Proc. Int'l Conf. Data Eng.,* pp. 106-115, 1999.
[8] J. Han, W. Gong, and Y. Yin, "Mining Segment-Wise Periodic Patterns in Time-Related Databases," *Proc. Int'l Conf. Knowledge Discovery and Data Mining (KDD '98),* pp. 214-218, 1998.
[9] P. Indyk, N. Koudas, and S. Muthukrishnan, "Identifying Representative Trends in Massive Time Series Data Sets Using Sketches," *Proc. Very Large Data Bases Conf.,* pp. 363-372, 2000.
[10] S. Ma and J.L. Hellerstein, "Mining Partially Periodic Event Patterns with Unknown Periods," *Proc. 17th Int'l Conf. Data Eng. (ICDE '01),* pp. 205-214, 2001.
[11] N. Mamoulis, H. Cao, G. Kollios, M. Hadjieleftheriou, Y. Tao, and D. Cheung, "Mining, Indexing, and Querying Historical Spatiotemporal Data," *Proc. 10th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* 2004.
[12] B. Özden, S. Ramaswamy, and A. Silberschatz, "Cyclic Association Rules," *Proc. Int'l Conf. Data Eng.,* pp. 94-101, 1998.
[13] W.-C. Peng and M.-S. Chen, "Developing Data Allocation Schemes by Incremental Mining of User Moving Patterns in a Mobile Computing System," *IEEE Trans. Knowledge and Data Eng.,* vol. 15, no. 1, pp. 70-85, Jan./Feb. 2003.
[14] D. Pfoser, C.S. Jensen, and Y. Theodoridis, "Novel Approaches in Query Processing for Moving Object Trajectories," *The VLDB J.,* pp. 395-406, 2000.
[15] Y. Tao, G. Kollios, J. Considine, F. Li, and D. Papadias, "Spatio-Temporal Aggregation Using Sketches," *Proc. Int'l Conf. Data Eng.,* pp. 449-460, 2004.
[16] Y. Tao and D. Papadias, "MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries," *Proc. Very Large Data Bases Conf.,* pp. 431-440, 2001.
[17] I. Tsoukatos and D. Gunopulos, "Efficient Mining of Spatiotemporal Patterns," *Proc. Symp. Advances in Spatial and Temporal Databases,* pp. 425-442, 2001.
[18] W. Wang, J. Yang, and R.R. Muntz, "Sting: A Statistical Information Grid Approach to Spatial Data Mining," *Proc. Very Large Data Bases Conf.,* pp. 186-195, 1997.
[19] W. Wang, J. Yang, and R.R. Muntz, "Sting+: An Approach to Active Spatial Data Mining," *Proc. Int'l Conf. Data Eng.,* 1999.
[20] J. Yang, W. Wang, and P.S. Yu, "Mining Asynchronous Periodic Patterns in Time Series Data," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 275-400, 2000.
[21] J. Yang, W. Wang, and P.S. Yu, "Infominer: Mining Surprising Periodic Patterns," *Proc. Seventh Int'l Conf. Knowledge Discovery and Data Mining (KDD '01),* pp. 395-400, 2001.
[22] J. Yang, W. Wang, and P.S. Yu, "Infominer+: Mining Partial Periodic Patterns with Gap Penalties," *Proc. Second IEEE Int'l Conf. Data Mining (ICDM '02),* pp. 725-728, 2002.
[23] M.J. Zaki, "SPADE: An Efficient Algorithm for Mining Frequent Sequences," *Machine Learning,* vol. 42, no. 1/2, pp. 31-60, 2001.

Huiping Cao received the bachelor's and master's degrees in information technology from the People's University of China, China, in 1999 and 2002, respectively. She is currently a PhD candidate in the Department of Computer Science at the University of Hong Kong. Her research interests include data mining and data engineering.

Nikos Mamoulis received the diploma in computer engineering and informatics in 1995 from the University of Patras, Greece, and the PhD degree in computer science in 2000 from the Hong Kong University of Science and Technology. Since September 2001, he has been an assistant professor in the Department of Computer Science at the University of Hong Kong. In the past, he has worked as a postdoctoral researcher at the Centrum voor Wiskunde en Informatica (CWI), the Netherlands. His research interests include complex data management, data mining, advanced indexing and query processing, and constraint satisfaction problems. He has published more than 60 articles in reputable international conferences and journals and has served on the program committees of major database and data mining conferences. He is a member of the IEEE.

David Wai-lok Cheung received the BSc degree in mathematics from the Chinese University of Hong Kong, and the MSc and PhD degrees in computer science from Simon Fraser University, Canada, in 1985 and 1989, respectively. From 1989 to 1993, he was a member of the scientific staff at Bell Northern Research, Canada. Since 1994, he has been a faculty member in the Department of Computer Science at The University of Hong Kong. He is also the director of the Center for E-Commerce Infrastructure Development. His research interests include data mining, data warehouse, XML technology for e-commerce, and bioinformatics. Dr. Cheung was the program committee chairman of the Fifth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD '01), the program cochair of the Ninth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD '05), and the conference chair of the 11th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD '07). Dr. Cheung is a member of the ACM and the IEEE Computer Society.