Efficient Discovery of Periodic Patterns over Event Sequences

Chuancong Gao, Jianyong Wang
Tsinghua University, Beijing 100084, China
chuancong@gmail.com, jianyong@tsinghua.edu.cn

ABSTRACT

Frequent periodic pattern mining has been studied a lot in the last decade, with various algorithms already proposed for mining different kinds of periodic patterns, from event sequence, spatio-temporal data, etc.. In this paper, instead of using the Maximal-subpattern Tree structure adopted widely in previous algorithms, we proposed a new algorithm under pattern-growth framework which was widely used in generalized sequential pattern mining. Based on that, we further proposed pruning/checking strategies for mining maximal periodic patterns. We further proposed a hybrid method combining the strength of both the Maximal-subpattern Tree and our proposed maximal pattern mining algorithm, to discover maximal periodic patterns more efficiently. Experimental results show that comparing to existing algorithm MTHS, our frequent periodic pattern mining algorithm is thousands times faster, and the devised maximal periodic pattern mining algorithm is even more than hundreds times faster than the former proposed frequent periodic pattern mining algorithm on some datasets. Comparing to both the two proposed frequent (maximal) pattern mining algorithms, the novel hybrid method for mining maximal patterns shows even greater performance advantage, with the running time further reduced for thousands times. Finally, we proposed a strategy for mining common frequent (maximal) periodic patterns and gave a novel case study on the discovery of common animal migration behaviors from a large real GPS-tracking dataset. Some of the discovered patterns are very interesting, like turkey vultures move to Mexico in Winter, while get back to US in Summer.

1 INTRODUCTION

As one of the important tasks in data mining area, periodic pattern mining has been widely used among a lot of domains, including motion activity detection, market basket analysis, significant motif detection in genome, server load analysis, stock chart evolution, environment analysis, animal behavior research, etc., and recently especially on social network behavior analysis and spatio-temporal data analysis. It has even been used in earthquake analysis in areas where earthquakes occur very often, like California and Japan. Due to its wide application range, lots of research have been conducted on frequent partial periodic [1] pattern mining in the last decade, aiming to find sequential patterns appearing in at least sup_{min} segments in the input single event sequence of time-series dataset, where sup_{min} is a user-specified minimum frequency threshold and an event is a set consisting of different items (features). For example, one frequent periodic pattern may indicate that in five days of a week Tom tends to work from 8 AM to 5 PM, do some exercises in the next one hour, and stay at home watching TV from 7 PM to 9 PM. A method for solving this problem was proposed in [4], by using the Maximal-subpattern Tree to store the maximal-candidate pattern of each segment. Since then on, other research have also been conducted on periodic pattern mining, with most of them relying on the Maximal-subpattern Tree or its variants.

On the other hand, generalized sequential pattern mining has also been studied a lot. The problem tries to find frequent patterns appearing in at least sup_{min} input sequences from the dataset, instead of mining on a long single input event sequence in the problem of periodic pattern mining. As one of the most efficient sequential pattern mining algorithms, PrefixSpan [11] was proposed by adopting a horizontal data representation and a pattern-growth paradigm with longer sequential patterns extended from current prefix pattern and local frequent items.

Inspired by generalized sequential pattern mining, instead of generating candidate patterns and calculating their support values on the Maximal-subpattern Tree like [4], in this paper we propose a novel algorithm with the idea of viewing each segment as an input sequence and applying pattern-growth-based framework devised specifically for periodic pattern mining to mine the complete set of frequent periodic patterns.

We further propose some novel pattern checking and pruning strategies for frequent maximal periodic pattern mining, where a pattern is called frequent maximal if and only if it does not have any frequent proper superpattern. Since the set of frequent maximal patterns is only a very small subset of the complete set of frequent patterns, they are much easier to be browsed and analyzed, and the required mining time may also be reduced significantly on some of the datasets. To further improve its efficiency, we further propose another novel hybrid method (call Maximal-Hybrid) by combining both the Maximal-subpattern Tree and the pattern-growth framework, with the purpose of filtering out segments which could not cover any maximal pattern.

People often concern about the mining of frequent periodic pattern on single event sequence, however, sometimes they also would like to know whether those patterns are common in several event sequences simultaneously. Hence we also propose a novel method on mining common periodic patterns, which is based on our proposed maximal periodic pattern mining algorithms Maximal-Growth and Maximal-Hybrid. A detailed case study is also provided on detecting common migration patterns of turkey vultures using the proposed algorithm, in order to discover interesting patterns representing the behaviors of the whole community. Some of the interesting common periodic patterns have been found, like migrating to warm place in Winter, and moving back in Summer.

Our contributions are summarized in the following:

- We propose a novel frequent periodic pattern mining algorithm based on the pattern-growth framework used previously in generalized sequential pattern mining. To our

[1]In the following we call partial periodic as periodic, where each pattern appears in some of the segments. While in some other papers, periodic refers to full periodic that each pattern are required to appear in every segments.

best knowledge, it is the first attempt to use pattern-growth framework on periodic pattern mining.

- Novel and effective pruning and checking strategies have been devised for maximal periodic pattern mining. A more efficient hybrid method is further proposed, with the efficiency of maximal pattern mining improved significantly. So far as we know, those are the first algorithms specifically designed for mining the complete set of frequent maximal/closed periodic patterns on event sequence data.
- A novel case study is provided showing how to mine common animal migration behaviors using a method based on maximal pattern mining, with some interesting migration patterns discovered on a real animal tracing dataset. We believe this is the first attempt on common periodic pattern mining.

2 RELATED WORK

2.1 Periodic Pattern Mining

The problem of frequent partial periodic pattern mining was first studied by [5] with the Apriori [1] property adopted. The authors further propose the Maximal-subpattern Hit Set method in [4], with the adoption of a novel data structure called Maximal-subpattern Tree which stores the hit counter of the maximal-candidate pattern in each segment, and the intermediate patterns from the root maximal-candidate pattern. Each frequent periodic pattern is generated from shorter patterns, with the support value summed from nodes containing it by traversing part of the tree. [6] discusses how to discover the most representative periodic trends in time-series database. [9] studies the problem of shifted or disrupted events with the presence of noise, while [18] studies asynchronous pattern mining. [16] studies surprising periodic pattern mining from event sequence. The authors further studied partial periodic pattern mining with gap penalties in [17]. In [14] a new algorithm is devised to mine some high-level patterns. Incremental mining of periodic patterns has been discussed in [2]. [19] also studies mining periodic patterns with gap requirement. Based on [4], [12] further proposes a density based pruning strategy for dense datasets.

Periodic pattern mining has also been studied on spatio-temporal data recently, where the input is no longer an event sequence but instead a set of data points with time stamp and location information. [10] works on the problem of periodic pattern mining in spatio-temporal data, with both bottom-up and top-down algorithms proposed respectively, by clustering data points covered by the current pattern into different regions and mapping each region to a single item event and then applying periodic pattern mining algorithms on event sequence. They further extend their work in [3]. Recently, [7] also studies the problem of periodic pattern mining for moving objects. Instead of using traditional pattern mining technology, it proposes to use hierarchical clustering on segments to mine periodic behaviors.

2.2 Generalized Sequential Pattern Mining

The currently most popular algorithm PrefixSpan [11] adopts a horizontal data representation and a pattern-growth paradigm, and traverses longer sequential patterns which are extended from the current prefix pattern. Some algorithms for closed sequential pattern mining have also been proposed. CloSpan [15] adopts a frequent prefix subtree-sharing data structure to facilitate the projected database pruning. While BIDE [13] proposes very effective search space pruning and pattern closure checking strategies to enhance the algorithm performance. As for maximal sequential pattern mining, [8] proposes a method using multiple samples.

3 PRELIMINARIES

3.1 Periodic Pattern

A sequence S is defined as an ordered list of events, where each event represents the set of different features collected at a time stamp. Hence, given a set of all the possible items $\mathbb{I}$ we could represent the sequence S as $e_1 e_2 \cdots e_i \cdots e_n$, where $e_i \subseteq \mathbb{I}$ for $1 \leq i \leq n$. We also use $T[i]$ to denote the ith event of T. As in other studies, we also adopt the "do not care" symbol $*$. Different from previous studies treating $*$ as a special event matching any event, here we denote $*$ as an event containing no item, that is $* \equiv \emptyset$. Now for any event e_i we all have $* \subseteq e_i \subseteq \mathbb{I}$. If an event contains no item, we represent it with $*$.

Given a period p, a pattern T is defined as a non-empty sequence (that is at least one of its events is not $*$) with the length of p. Pattern T' is said to be contained by T if and only if T and T' have the same period and for each event $e_i{}'$ of T' and event e_i of T at the corresponding position i $(1 \leq i \leq p)$ we all have $e_i{}' \subseteq e_i$, represented as $T' \sqsubseteq T$ or $T' \sqsubset T$ when $T' \neq T$. If T' is contained by T, we call T' the subpattern of T and T the superpattern of T'.

Under the period p, sequence $S = e_1 e_2 \cdots e_n$ [2] could be divided into a set of segments $\mathbb{G} = \{G_1, G_2, \cdots, G_m\}$ where each segment $G_i \in \mathbb{G}$ $(1 \leq i \leq |\mathbb{G}|)$ is a sequence of length p and $G_i = e_{l+1} e_{l+2} \cdots e_{l+p}$, with $l = (i-1) \times p$. The absolute support value (occurrence frequency) of pattern T with respect to sequence S now could be defined as $|\{G_i | G_i \in \mathbb{G} \ and \ T \sqsubseteq G_i\}|$, denoted by sup_T^S or simply sup_T when S is clear in context. We also call $sup_T / |\mathbb{G}|$ the relative support value of T. When context is clear, both absolute and relative support could be called support and denoted by sup_T. A pattern T is said to be frequent if and only if $sup_T \geq sup_{min}$, where sup_{min} is a user-specified (absolute or relative) minimum support threshold.

Now we could derive the definitions of a frequent closed pattern and a frequent maximal pattern. A frequent pattern T is said to be frequent closed if and only if $\nexists T'$ such that $T \sqsubset T'$ and $sup_T = sup_{T'}$. While a frequent pattern T is said to be frequent maximal if and only if $\nexists T'$ such that $T \sqsubset T'$ and T' is frequent. Note that according to the definitions a frequent maximal pattern is also a frequent closed pattern.

Example 3.1. Under the period of 4, sequence $\{a\ c\} * b * a * \{b\ c\}\ c * a\ b\ c$ [3] could be divided into 3 segments $\{a\ c\} * b *$, $a * \{b\ c\}\ c$, and $* a\ b\ c$. Given $sup_{min} = 2$, there are five frequent periodic patterns: $a * b * : 2$, $a * * * : 2$, $* * b\ c : 2$, $* * b * : 3$, $* * * c : 2$, with the number after ":" denoting the support value. Among those there are three frequent closed patterns: $a * b * : 2$, $* * b\ c : 2$, and $* * b * : 3$, and two frequent maximal patterns: $a * b * : 2$ and $* * b\ c : 2$.

[2]Here we assume $n \mod p = 0$. If not, we could firstly append several $*$ events to the end of S to satisfy this condition.

[3]The braces of events containing only one item are omitted for clarity.

Here we define the event-size of T as the number of non-$*$ events in T, denoted by $|T|_e$. We also define the item-size of T as the number of all the items in T, denoted by $|T|_i$. For example, given $T = *\{ac\}b*$ we have $|T|_e = 2$ and $|T|_i = 3$. When $|T|_e = l$ we say T is l-event-size, and when $|T|_i = l$ we say T is l-item-size.

To simplify the expression, we also define the "$-$" operation on periodic pattern. Given pattern T and T' with the same period of p, we have $T - T' = (T[1] - T'[1])(T[2] - T'[2]) \cdots (T[i] - T'[i]) \cdots (T[p] - T'[p])$. For example, we have $* \{a\, c\}\, b - a\, c\, * = *\, a\, b$. Operation "$\uplus$" is also defined. Given a pattern $T = e_1 e_2 \cdots e_p$ and a tuple $(pos, item)$, we define $T \uplus (pos, item) = e_1 e_2 \cdots e_{pos-1} (e_{pos} \cup \{item\}) e_{pos+1} \cdots e_p$. For example, we have $* \{a\, c\}\, b \uplus (3, c) = * \{a\, c\}\, \{b\, c\}$.

3.2 Maximal-subpattern Tree

Since firstly proposed in [4], Maximal-subpattern Tree structure has been widely used in periodic pattern mining, to help keep the hit count of each maximal candidate patterns contained in segments and the formed maximal pattern lattice.

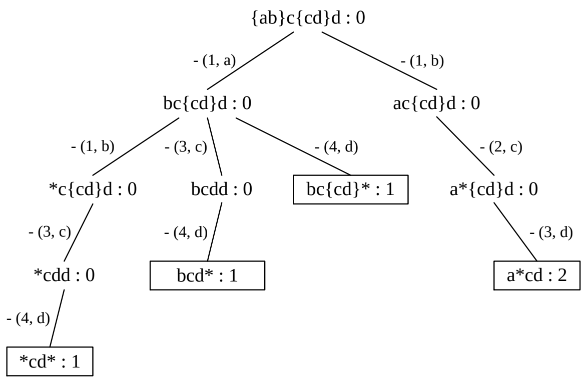
Maximal-subpattern Tree uses the maximal candidate pattern T_{max} as the root node. Each event in T_{max} is formed with the union of all the frequent items at the corresponding position. If none of the item is frequent at that position, the event is represented with $*$. To add a segment G, we first get the maximal candidate pattern of G, which is the intersection G' of G and T_{max}, in order to eliminate all the non-frequent items appeared in G. If G' is non-empty (with at least one non-$*$ event), we add it and all its ancestors to the tree by keep adding patterns with one non-$*$ item not appeared in G' removed from T_{max} in order at each step and set its hit counter as 1, until we reach G'. If a pattern already exists in the tree, we increase its hit counter by 1.

Example 3.2. Figure 1 gives an example of a Maximal-subpattern Tree with an input sequence $a\ b\ c\ d\ b\ c\ \{c\ d\}\ *\ c\ a\ b\ *\ a\ *\ c\ d\ b\ c\ d\ b\ *\ c\ d\ c$ of 24 events under the period of 4. After inserting the 6 segments $a\ b\ c\ d$, $b\ c\ \{c\ d\}\ *$, $c\ a\ b\ *$, $a\ *\ c\ d$, $b\ c\ d\ b$, and $*\ c\ d\ c$ by intersecting with the maximal candidate pattern $T_{max} = \{a\ b\}\ c\ \{c\ d\}\ d$, we have 5 non-empty intersected segments $a\ *\ c\ d$, $b\ c\ \{c\ d\}\ *$, $a\ *\ c\ d$, $b\ c\ d\ *$, and $*\ c\ d\ *$ inserted.

Taking inserted segment $T = b\ c\ d\ *$ for example, we first calculate the difference pattern $T_{diff} = T_{max} - T = a\ *\ c\ d$. Then we keep removing the non-$*$ items of T_{diff} in order, that is $(1, a)$, $(3, c)$, and $(4, d)$, with the numbers in tuples denoting corresponding event positions, and insert pattern $b\ c\ \{c\ d\}\ d$, $b\ c\ d\ d$, and $b\ c\ d\ *$ into tree respectively. Finally, we insert T into tree and set its hit counter to 1. Note that the number after ":" denotes the hit counter, not the support value.

4 FREQUENT PERIODIC PATTERN MINING

In this section we will discuss the details on our proposed frequent periodic pattern mining algorithm, which is based on pattern-growth framework instead of the Maximal-subpattern Tree structure. We also give a brief overview of the MTHS method in the supplementary file.



Rectangle denotes the inserted segments.

Figure 1: An Example of Maximal-subpattern Tree

4.1 Pattern-growth Framework for Periodic Pattern Mining

If we view each segment in the single input event sequence as an individual input sequence. We could transform the original periodic pattern mining problem to the generalized sequential pattern mining problem, which mines sequential patterns from multiply input sequences, except that we now have the restriction on the fixed position of each matched event. Hence we can apply a specially devised variety of the widely-used pattern-growth Framework [11], combining with some specially designed pattern pruning/checking strategies, to mine frequent (all/closed/maximal) periodic patterns.

Prior to discussing the algorithm details, we first define some basic concepts working on periodic pattern mining.

Definition 4.1 (Projected Pattern). Given a pattern T and a segment G under the period of p, the projected pattern of T with respect to G is defined as $e_1 e_2 \cdots e_p$, where $e_i = *$ when $1 \le i \le k-1$ and $e_i = G[i] - T[i]$ when $k \le i \le p$, with k as the position of the last non-$*$ event.

For example, given $T = b\ c\ c\ *$ and $G = b\ c\ \{c\ d\}\ *$, the projected pattern of T with respect to G is $*\ *\ d\ *$.

Definition 4.2 (Projected Database). Given a pattern T and a set of segments $\mathbb{G}$ under the period of p, the projected database of T with respect to $\mathbb{G}$ is defined the set of non-empty projected patterns of T with respect to each segment in $\mathbb{G}$, denoted by $projDB_T^{\mathbb{G}}$ or simply $projDB_T$ when $\mathbb{G}$ is clear in context.

Note that each projected pattern in projected database is required to be non-empty, to make sure there is at least one item left to help extend the current prefix pattern.

Now we could give the framework of our algorithm. Starts from a prefix pattern T, we first compute its projected database $projDB_T$. For each event position we calculate the set of local frequent items in $projDB_T$ and use each of them to extend the current prefix pattern. Then we continue to work on the newly extended pattern. During this progress, all the frequent patterns are outputted.

Example 4.3. Figure 2 illustrates an example of the pattern search space, on the five segments: $a \ * \ c \ d$, $b \ c \ \{c \ d\} \ *$, $a \ * \ c \ d$, $b \ c \ d \ *$, and $* \ c \ d \ *$, previously used in Example 3.2. Each node represents a frequent periodic pattern, with totally 14 frequent patterns discovered. All the patterns are organized in lexicographic order, with one more item added in each next level. We could see that starting from the empty pattern, the algorithm keeps extending the current prefix with one more local frequent item, until there is no more local frequent items in the projected database of the current prefix pattern.

The advantages of our method is obvious. By extending the current prefix pattern with one more local frequent item, the algorithm guarantees only frequent patterns would be generated and only once for each pattern. While for MTHS , not only frequent but also infrequent patterns could be generated. There are also chances that one pattern is generated more than once. In addition the very inefficient support calculation in MTHS is also avoided. The experimental results provided in Section 7 show that our algorithm is more than thousands of times faster than MTHS, especially on large dataset with longer period.

5 FREQUENT MAXIMAL PERIODIC PATTERN MINING

Due to the fact that the number of frequent periodic patterns could easily exceed more than ten million in many of the datasets with a low minimum support threshold, in many cases frequent periodic patterns could not be used directly in data analysis. Instead often we need only a compact subset, which maintains most of the desired patterns. Hence we further study efficient algorithms for mining frequent maximal patterns in this section.

Frequent maximal patterns form the most compact pattern set, by storing only those whose superpatterns are all infrequent. All the frequent patterns could be derived by enumerating all the subpatterns of the frequent maximal patterns, in the cost of losing all the support value information. However, comparing its rather small pattern number with all the frequent one's, frequent maximal patterns have significant advantages in both mining and processing. Figure 2 also gives some examples of frequent maximal pattern.

5.1 Pruning and Checking Strategy

We first introduce some theorems and lemmas. Due to space limit, we put all the proofs in the supplementary file.

THEOREM 5.1. *Pattern T would not be a frequent maximal pattern if there exists a tuple $(pos, item)$, where $|\{projPatt \mid projPatt \in projDB_T$ and $item \in projPatt[pos]\}| \geq sup_{min}$.*

LEMMA 5.2 (FORWARD CHECKING). *Pattern T would not be a frequent maximal pattern if there exists a local frequent item.*

Now we could easily check whether a candidate pattern is maximal, by checking whether there is a local frequent item with respect to the current pattern. For other cases, we also devise a backward checking strategy. First we give the definitions of appearance pattern and appearance database.

Definition 5.3 (Appearance Pattern). Given a pattern T and a segment G under the period of p, the appearance pattern of T with respect to G is defined as $G - projPatt_T$, where $projPatt_T$ is the projected pattern of T with respect to G.

For example, given $T = b \ c \ d \ *$ and $G = b \ c \ \{c \ d\} \ a$, the appearance pattern of T with respect to G is $b \ c \ \{c \ d\} \ *$. Since T's projected pattern with respect to G is $projPatt_T = * \ * \ * \ a$, and $G - projPatt_T = b \ c \ \{c \ d\} \ *$.

Definition 5.4 (Appearance Database). Given a pattern T and a set of segments $\mathbb{G}$ under the period of p, the appearance database of T with respect to $\mathbb{G}$ is defined the set of non-empty appearance patterns of T with respect to each segment in $\mathbb{G}$, denoted by $apprDB_T^{\mathbb{G}}$ or simply $apprDB_T$ when $\mathbb{G}$ is clear in context.

THEOREM 5.5 (BACKWARD CHECKING). *Pattern T would not be a frequent maximal pattern if there exists a tuple $(pos, item)$, where $item \notin T[pos]$ and $|\{patt|patt \in apprDB_T$ and $item \in patt[pos]\}| \geq sup_{min}$.*

Unlike Forward Checking which checks whether a new local frequent item could be used to extend the prefix pattern, Backward Checking checks whether a new "backward local frequent" item could be inserted into the original prefix pattern to derive a new frequent prefix pattern. For example for pattern $a \ * \ * \ d$ which could not be detected as non-maximal by Forward Checking due to non-existing local frequent item, we could find it is non-maximal because of the existence of the new backward local frequent item c at the position 3. Note that here backward local frequent means the item appears at least sup_{min} times in the prefix's appearance database. Combining the two checking strategies together, we have our final checking strategy covering all the possible cases.

We also give some properties and theorems about the pattern pruning strategy, which prunes a pattern whose super-patterns all are not frequent maximal and hence reduce the pattern search space.

PROPERTY 1 (EQUIVALENT PROJECTED DATABASES). *Given two patterns T and T', if we have $projDB_T = projDB_{T'}$ then T and T' could be extended to derive the exact same set of super-patterns.*

THEOREM 5.6 (BACKWARD PRUNING). *Pattern T would be safely pruned if there exists a tuple $(pos, item)$, where $item \notin T[pos]$ and $|\{patt|patt \in apprDB_T$ and $item \in patt[pos]\}| = sup_T$.*

For example, the pattern $b \ * \ d \ *$ in Example 4.3 and Figure 2 could be pruned because we could insert a new item c at the position 2, and it appears 2 times at the position 2 in the pattern's appearance database, which is equal to the pattern's support of 2.

The new algorithm looks similar to the proposed frequent periodic pattern mining algorithm, with some modifications combining both the Backward Checking and the Backward Pruning strategies into a same function, while the Forward Checking is performed in the calculation of local frequent items. The previous function used for calculating local frequent items in projected database could be updated to also support calculating "backward local frequent" items in appearance database for backward pruning/checking.

5.2 Hybrid Method for Maximal Periodic Pattern Mining

Instead of using only the pattern-growth framework to mine frequent maximal periodic patterns, here we also propose another

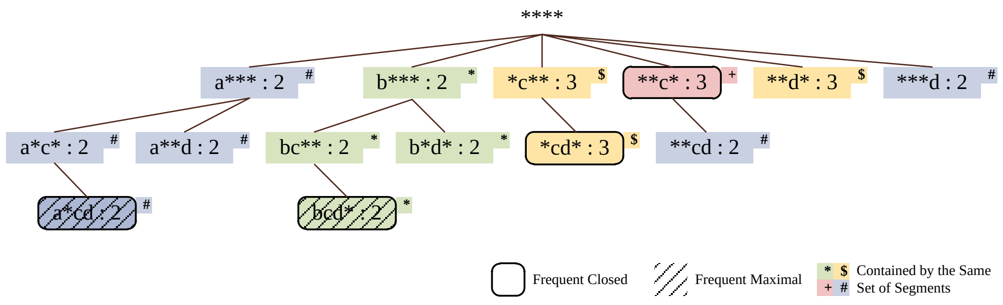


Figure 2: An Example of Pattern Search Space

novel hybrid method by combining both the Maximal-subpattern Tree and the pattern-growth framework.

We first traverse the Maximal-subpattern Tree and divide the visited patterns into two groups $frequent$ and $infrequent$, where $frequent$ contains those frequent patterns whose ancestor patterns are not in $frequent$, and $infrequent$ contains those infrequent patterns with hit counter larger than 0 (and hence only input segments) and no ancestor patterns in $frequent$. Since a frequent pattern's subpatterns are all frequent, we could stop traversing the subtree of a pattern added to $frequent$. Note that we calculate the support value of one pattern by simply adding all the hit counters of both it and all its ancestor patterns in the tree, which is actually a lowest bound of the true support value. Hence for those in $infrequent$, they may actually be frequent.

Now we can apply the frequent maximal pattern mining algorithm proposed in Section 5 on $infrequent$ to find a set of hidden frequent maximal patterns called $ifPatterns$, and merge it with the already discovered set of frequent patterns in $frequent$ called $fPatterns$ to remove duplicate patterns. Note that there is no need to test one pattern in $ifPatterns$ against another in $ifPatterns$, since the mining algorithm guarantees they are frequent maximal patterns in $infrequent$ and none of them is a subpattern of another.

The main idea of the method is to filter out the segments whose subpatterns could not be maximal due to the existence of a frequent superpattern in set $frequent$, from the set of segment to be mined. Hence it can reduce the required mining time a lot.

Here we prove that the hybrid method outputs the complete set of frequent maximal patterns.

THEOREM 5.7 (CORRECTNESS). *The hybrid method outputs the complete set of frequent maximal patterns.*

For example, given the five inserted segments and the Maximal-subpattern Tree in Example 3.2 and Figure 1, we have $infrequent = \{* \ c \ d \ * : 1, b \ c \ d \ * : 1, b \ c \ \{c \ d\} \ * : 1\}$ and $frequent = \{a \ * \ c \ d : 2\}$. By applying the proposed frequent maximal pattern mining algorithm on $infrequent$ we have one maximal pattern $* \ c \ d \ *$ discovered. After merging with the pattern $a \ * \ c \ d$ in $frequent$, we get the two frequent maximal patterns $* \ c \ d \ *$ and $a \ * \ c \ d$ shown in Figure 2.

The disadvantage of the new hybrid method is that the patterns' accurate support values are unavailable. For those in $frequent$, we could only get a value which may be smaller than its support, while for those mined from $infrequent$, their supports only come from the segments in $infrequent$ and may be also smaller than the actually supports. However, given the fact that the hybrid method is hundreds times faster than the previous pattern-growth based algorithm on some of the datasets, this would not be a big concern in situations where the accurate support values are not needed.

6 COMMON PERIODIC PATTERN DISCOVERY FROM MULTI-SEQUENCES

Some times, we need to discovery some periodic patterns which are not only frequent in a single sequence, but also frequent among multiply sequences. (Which we name common frequent (closed/maximal) pattern.) For example, although we could discover some animal moving periodic patterns in one animal's region sequence, they may not represent the behavior of the whole community living together.

If we view each of the frequent maximal periodic patterns mined in each sequence as an input segment, we could transform the common frequent (closed/maximal) periodic pattern mining problem to a normal frequent (closed/maximal) periodic pattern mining problem, and apply the previously proposed frequent (closed/maximal) pattern mining algorithms to solve it. Since the maximal patterns in each sequence represent all the frequent patterns in that sequence.

However, there are also some modifications to be applied, since we treat the maximal patterns from the same sequence as different segments. For example, a common frequent pattern may be contained by both two maximal patterns from one sequence, and its common support would be calculated incorrectly. This would also affect the detecting of whether a pattern is common frequent. To solve this, we need to do the following modifications: 1) For the frequency calculation in all the pattern-growth based algorithms, the frequency is now determined by the number of distinctive segment IDs containing the item, instead of adding a number. 2) For the hybrid maximal pattern mining method, the support of a pattern is calculated now by the number of distinctive segment IDs. We do not need to modify the hit counter of Maximum-subpattern Tree, since the maximal patterns from the same sequence could never be inserted into the same node.

Note that we do not need the support values of the mined maximal patterns in each input sequence, since the definition of a common frequent patterns only concerns whether a pattern is frequent in each sequence. Hence, the previously proposed hybrid method for frequent maximal pattern mining in Section 5.2 is perfect for the task, with its very good efficiency proved by the experimental results in Section 7.

7 EXPERIMENTAL RESULTS

We present the experimental results in this section, including the performance comparison between the MTHS method and the proposed pattern-growth based algorithm All-Growth, the comparison between algorithm All-Growth and the frequent maximal pattern mining algorithms Maximal-Growth and Maximal-Hybrid. We also provide an evaluation on scalability. Note that because the evaluated algorithms all mine the exact same complete set of frequent (maximal) patterns, there is no need to concern about the result accuracy.

Experiments were conducted on a workstation with Intel Xeon CPU E5405 (2.0 GHz) and 10 GB RAM installed, running Microsoft Windows Server 2003 R2 X64. Algorithms are written in C#.

7.1 Datasets

We used two publicly available datasets from the UCI Machine Learning Repository [4]. The first one Synthetic Control Chart (in short *control*) contains 100 cyclic sequences with the same length of 60. We created a long event sequence with the length of $6,000$ by concatenating all the 100 sequences and converting the original continuous values to 5 categorical values with unsupervised discretization applied by Weka [5]. Another dataset we used is called Dodgers Loop Sensor Data (in short *dodgers*) which was collected on a highway near the stadium in every 5 minutes, to try to see unusual traffic after a Dodgers game. There are totally $50,400$ timestamps in the dataset. Similarly, we convert the original continuous values to 5 categorical values with discretization applied by Weka. This dataset is relatively sparse, and contains many missing values represented by the empty event $*$.

7.2 Evaluation on Frequent Periodic Pattern Mining

We compared our pattern-growth framework based frequent periodic pattern mining algorithm (All-Growth) proposed in Section 4.1 with the Maximal-subpattern Tree Hit Set method (MTHS) proposed in [4]. Figure 3 and Figure 4 show the running time comparison results on dataset *control* and *dodgers* respectively, with either varying minimum support or varying period. We could see that our pattern-growth based algorithm All-Growth is significantly faster than the MTHS method in at least two orders. As we introduce in Section A.1, most of the time used by MTHS is in pattern generation and support computation, especially the latter one which needs one scan on the Maximal-subpattern Tree for each pattern. While for All-Growth, patterns are extended from the previous prefixes with local frequent items appended. This is very efficient since only one scan of the projected database is required for calculating the current local frequent items. The support values of the new patterns could also be easily derived since they are equal to the frequencies of the local frequent items.

[4]http://archive.ics.uci.edu/ml/index.html

[5]http://www.cs.waikato.ac.nz/ml/weka/

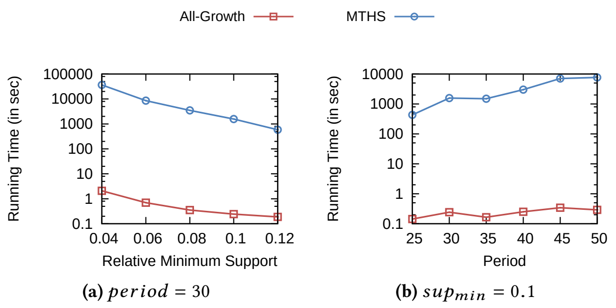


Figure 3: Comparing All-Growth with MTHS (Dataset: *control*)

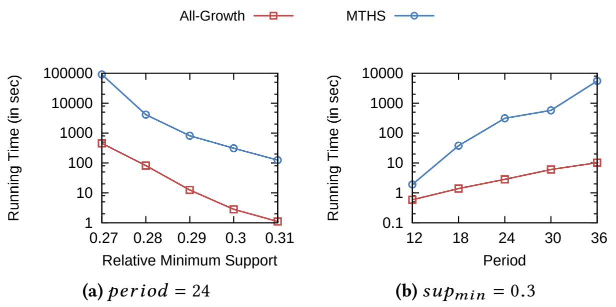
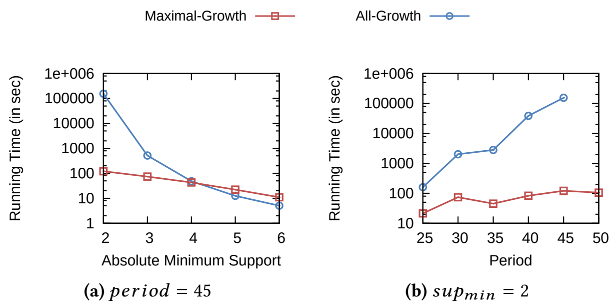


Figure 4: Comparing All-Growth with MTHS (Dataset: *dodgers*)

7.3 Evaluation on Frequent Maximal Periodic Pattern Mining

We also evaluated the performance of the two proposed frequent maximal pattern mining algorithms – the pattern-growth based algorithm Maximal-Growth and the hybrid method Maximal-Hybrid, by comparing with the frequent periodic pattern mining algorithm All-Growth. Note that we did not include other baselines because to our best knowledge currently there is only one study [3] partly related to frequent maximal periodic pattern mining, which is based on the MTHS method. However, its set of mined frequent maximal patterns is not complete. Besides, its efficiency is not good, and in almost all of the cases even slower than the original MTHS method in their evaluation. What is more, the mined patterns are not guaranteed to be frequent maximal.

Figure 5 gives the results comparing Maximal-Growth with All-Growth on dataset *control*. Note that we do not include the results of Maximal-Hybrid, because on this dataset there is no any frequent pattern found in the Maximum-subpattern Tree traversing proposed in Section 5.2, even at the smallest minimum support of 2. Together with the fact that there are also very few segments having the same content, Maximal-Hybrid unfortunately degenerates to Maximal-Growth on this dataset. From the results we could find that the newly proposed pruning methods for maximal periodic pattern mining is very effective, and Maximal-Growth runs significantly faster than All-Growth on this dataset.



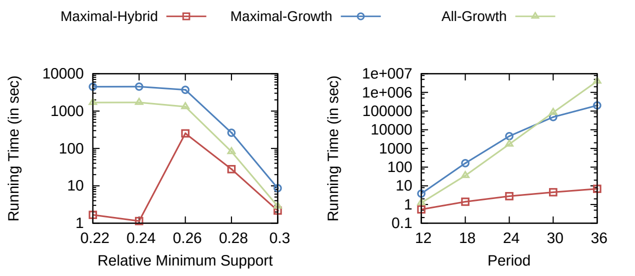
(a) $period = 45$ **(b)** $sup_{min} = 2$

Figure 5: Comparing Maximal-Growth with All-Growth (Dataset: *control*)

Figure 6 shows the results on the sparse dataset *dodgers*. We could observe that: 1) Instead of running thousands times faster than All-Growth on dataset *control*, Maximal-Growth is about 2-3 times slower than All-Growth on this dataset. After analysis on the pattern search space, we found that unlike the dataset *control*, most of the patterns pruned in dataset *dodgers* by Maximal-Growth are located at the bottom of the search space. In other words, the pruned search space is not very large. Hence the time saved by skipping those patterns in the pruned search space is less than the time used in Backward Pruning/Checking itself. However, when period is 30 or larger, we see instead All-Growth runs much more slower than Maximal-Growth, with $sup_{min} = 0.2$. This is because All-Growth begins to generate too many patterns, e.g. $1,073,744,917$ with the period 30, and the pruning strategy begins to show its power. It is even more obvious with a larger period and a smaller minimum support. 2) The newly proposed hybrid algorithm Maximal-Hybrid is significantly faster than both Maximum-Growth and All-Growth, and its running time even drops to about one second at the minimum support of 0.24. When $sup_{min} > 0.24$, there is not any frequent pattern discovered in the Maximal-subpattern Tree's traversing introduced in Section 5.2, hence Maximal-Hybrid works like Maximal-Growth, except it updates the frequencies of local frequent items in a new manner. This improves its performance a lot since the dataset *dodgers* contains a lot of segments with the same content. While when $sup_{min} \leq 0.24$, at least one frequent pattern is discovered in the tree's traversing and a great many of contained input segments are filtered out, and the running time reduces significantly in more than two orders.

7.4 Evaluation on Scalability

We evaluated the scalability of the three algorithms All-Growth, Maximal-Growth, and Maximal-Hybrid, by duplicating the original dataset 2, 4, and 8 times, respectively. The results on the two datasets are shown in Figure 8. We could see that on the two datasets with the increment of the dataset size, the running time increases linearly, which proves the good scalability of the three of our algorithms. Like in Figure 5, we do not include the result of Maximal-Hybrid on dataset *control*, since it has the nearly the same performance as Maximum-Growth.



(a) $period = 24$ **(b)** $sup_{min} = 0.2$

Figure 6: Comparing Maximal-Hybrid and Maximal-Growth with All-Growth (Dataset: *dodgers*)

8 DISCOVERY OF COMMON PERIODIC ANIMAL MIGRATION BEHAVIOR

Here we conduct a novel case study on the discovery of common periodic animal migration patterns. Mining migration patterns from one single golden eagle's tracing data has been studied in [7] recently. However, the pattern discovered in their work could not convincingly represent the behaviors of the whole or at least part of the community living together. Instead we would like to discover some of the common periodic patterns to reveal some of the hidden common behavior patterns.

We used a public real dataset [6] which contains $177,315$ GPS locations of 20 turkey vultures from 2003 to 2010. Before mining those common migration patterns, we first need to calculate the Reference Spots in the given dataset using the method proposed in [7], in order to cluster the location points into different regions where each one could be viewed as different living places of animals. Figure 9 shows the raw data on Google Map on the left with Pink color denoting location points, and the 6 found reference spots on both with the density threshold set as top-5%.

8.1 Details of Data Conversion

Before extracting some patterns, we first need to convert the raw data to the form of event sequences. Since most of the animals migrate every year, here we use a period of one year (or exactly 53 weeks). The following steps give the converting details:

(1) Replace the original $(latitude, longitude)$ location in each data point with the label of the corresponding reference spot. If no reference spot covers the location, replace it with -1. [7]
(2) Generate a sequence for each individual animal with the length of $n \times 53$, where n is the number of years the whole data spans and 53 represents the number of weeks (including

[6]http://www.movebank.org/
[7]We do not use $*$ for not covered locations, because there is no containing relationship between the reference spot and the non-reference spot.

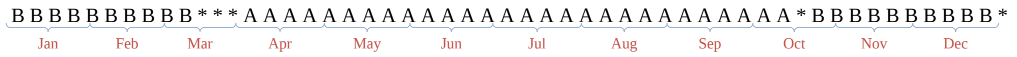
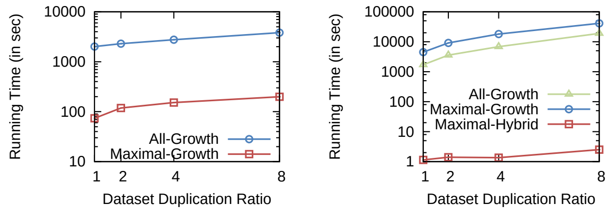
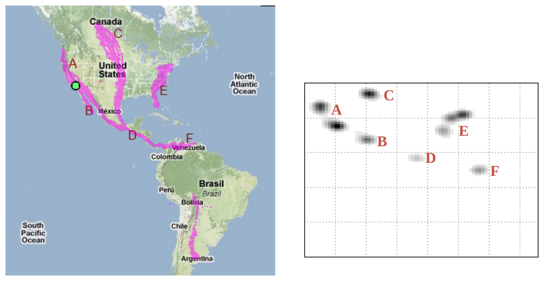

B B B B B B B B B B * * * A A A A A A A A A A A A A A A A A A A A A A A A A A A * B B B B B B B B B B *
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec

Figure 7: One of the Discovered Common Animal Migration Patterns



(a) ***control*** **(sup_{min}=0.01)** **(b)** ***dodgers*** **(sup_{min}=0.24)**

Figure 8: Scalability Evaluation



(a) Raw Data on Google Map **(b) Calculated Reference Spots**

Figure 9: Raw Data and Reference Spots of Turkey Vulture

the last incomplete one) in one year. Initialize each event with $*$.

(3) For each data point, add the label of its reference spot to the event of the corresponding week in that year in the animal's sequence.

8.2 Discovered Common Migration Behaviors

We mined some common frequent periodic patterns on the dataset, by applying the method proposed in Section 6. In total we have 23 common frequent maximal periodic patterns discovered. Though some of them indicate plain moving behaviors, like always staying at a place, some patterns do show interesting behaviors. Among the 23 common maximal patterns, there is one very interesting pattern, shown in Figure 7. Combine with the raw data map, the pattern tells us that some of the turkey vultures migrate from the cold north west states in US to the warm Mexico in almost the middle of October, and get back to home in the next year's end of March.

Some other migration patterns have also been discovered. For example, some of the turkey vultures stay at the Middle of Canada (Place C) from the beginning of June to the beginning of October, then they start to travel using one and a half month. When in the middle of November, they have migrated to Venezuela (Place F) and stay there until at least the middle of next year's March. In the latest next year's start of June they get back to Canada, after a home trip taking the same length of one and a half month.

9 CONCLUSIONS

In this paper, we proposed a novel algorithm mining frequent periodic patterns from event sequence using the pattern-growth framework, which is thousands times faster than algorithm MTHS. Base on the new algorithm, we further proposed a frequent maximal periodic pattern mining algorithm, with novel pruning/checking strategies. Evaluation shows that on some datasets the new algorithm is hundreds times faster than proposed frequent periodic pattern mining algorithm. We also proposed a novel hybrid method combining the Maximal-subpattern Tree and the proposed maximal pattern mining algorithm. Experimental results validate that the new hybrid method is thousands times faster than both the two proposed algorithms. Finally, we give an interesting case study on common animal migration behavior discovery, using a common frequent periodic pattern mining algorithm, and illustrate some very interesting common migration patterns.

REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*'94.
[2] W. Aref, M. Elfeky, and A. Elmagarmid. Incremental, online, and merge mining of partial periodic patterns in time-series databases. *IEEE Trans. Knowl. Data Eng.*, 2004.
[3] H. Cao, N. Mamoulis, and D.Cheung. Discovery of periodic patterns in spatiotemporal sequences. *IEEE Trans. Knowl. Data Eng.*, 2007.
[4] J. Han, G. Dong, and Y. Yin. Efficient mining of partial periodic patterns in time series database. In *ICDE*'99.
[5] J. Han, W. Gong, and Y. Yin. Mining segment-wise periodic patterns in time-related databases. In *KDD*'98.
[6] P. Indyk, N. Koudas, and S. Muthukrishnan. Identifying representative trends in massive time series data sets using sketches. In *VLDB*'00.
[7] Z. Li, B. Ding, J. Han, R. Kays, and P. Nye. Mining periodic behaviors for moving objects. In *KDD*'10.
[8] C. Luo and S. Chung. Efficient mining of maximal sequential patterns using multiple samples. In *SDM*'05.
[9] S. Ma and J. Hellerstein. Mining partially periodic event patterns with unknown periods. In *ICDE*'01.
[10] N. Mamoulis, H. Cao, G. Kollios, M. Hadjieleftheriou, Yufei Tao, and D. Cheung. Mining, indexing, and querying historical spatiotemporal data. In *KDD*'04.
[11] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, Umeshwar Dayal, and M. Hsu. Prefixspan: Mining sequential patterns by prefix-projected growth. In *ICDE*'01.
[12] C. Sheng, W. Hsu, and M. Lee. Mining dense periodic patterns in time series data. In *ICDE*'06.
[13] J. Wang, J. Han, and C. Li. Frequent closed sequence mining without candidate maintenance. *IEEE Trans. Knowl. Data Eng.*, 2007.
[14] W. Wang, J. Yang, and P. Yu. Meta-patterns: Revealing hidden periodic patterns. In *ICDM*'01.
[15] X. Yan, J. Han, and R. Afshar. Clospan: Mining closed sequential patterns in large databases. In *SDM*'03.
[16] J. Yang, W. Wang, and P. Yu. Infominer: mining surprising periodic patterns. In *KDD*'01.
[17] J. Yang, W. Wang, and P. Yu. Infominer+: Mining partial periodic patterns with gap penalties. In *ICDM*'02.

[18] J. Yang, W. Wang, and P. Yu. Mining asynchronous periodic patterns in time series data. *IEEE Trans. Knowl. Data Eng.*, 2003.
[19] M. Zhang, B. Kao, D. Cheung, and K. Yip. Mining periodic patterns with gap requirement from sequences. In *SIGMOD*'05.

A SUPPLEMENTARY

A.1 The Existing Maximal-subpattern Tree Hit Set Method

The Maximal-subpattern Tree Hit Set (called MTHS in short) method was firstly proposed in [4] to solve the problem of mining the complete set of frequent periodic patterns, and it has been the most typical algorithm for mining periodic pattern. Although some other algorithms have also been proposed, most of them are still based on the basic MTHS method. To better compare it with our method, here we give a brief introduction. The method could be summarized in the following steps:

(1) Mine all the frequent 1-event-size patterns and form the root maximal candidate pattern T_{max}.
(2) Construct the Maximal-subpattern Tree.
(3) For $2 \leq i \leq e\_size_{T_{max}}$ do:
   (a) Derive all the candidate i-event-size patterns by joining all the frequent $(i-1)$-event-size patterns.
   (b) For each candidate i-event-size pattern, traverse the Maximal-subpattern Tree to calculate its support value.
   (c) Eliminate all the infrequent candidate patterns. Return if none of the candidate is left.

To derive all the candidates with the event-size of i, every pair of the frequent $(i-1)$-event-size patterns are joined, with the requirement that both the two joining patterns have the first $i-2$ identical (in both the event items and the event position) non-$*$ events, and the last one different (in either the event items or the event position) non-$*$ events.

A tree traverse is required to calculate the support value of a given pattern T, by summing all the hit values of the segments containing T. By removing items not appeared in T recursively from T_{max} from the first position to the last position, and from the first item to the last item in the event in current position, all the segments containing T could be visited. Details could be also be found in [3].

However, the disadvantage of the MTHS method is very obvious. Lots of time is spent each time on the deriving of longer pattern with the event-size increased 1. Besides, the support calculation is also very inefficient, since a large part of the tree has to be scanned every time for every pattern.

A.2 Proofs

Theorem A.1 (Theorem 5.1). *Pattern T would not be a frequent maximal pattern if there exists a tuple $(pos, item)$, where $|\{projPatt \mid projPatt \in projDB_T$ and $item \in projPatt[pos]\}| \geq sup_{min}$.*

Proof. Given pattern T and the tuple $(pos, item)$, we could extend T with $item$ at the position pos to get a new pattern T'. Since we have $|\{projPatt|projPatt \in projDB_T$ *and* $item \in projPatt[pos]\}| \geq sup_{min}$, we could also derive that T' is frequent. According to the definition, T is not a frequent maximal pattern. □

Lemma A.2 (Lemma 5.2, Forward Checking). *Pattern T would not be a frequent maximal pattern if there exists a local frequent item.*

Proof. Assume there is a local frequent item $item$ at the position pos, according to Theorem 5.1 we could know T is not frequent maximal, given the fact that $|\{projPatt \mid projPatt \in projDB_T$ *and* $item \in projPatt[pos]\}| \geq sup_{min}$ is just the definition of local frequent item. □

Theorem A.3 (THeorem 5.5, Backward Checking). *Pattern T would not be a frequent maximal pattern if there exists a tuple $(pos, item)$, where $item \notin T[pos]$ and $|\{patt|patt \in apprDB_T$ and $item \in patt[pos]\}| \geq sup_{min}$.*

Proof. Given pattern T and the tuple $(pos, item)$, we could insert T with $item$ at the position pos to get a new pattern T', since $item \notin T[pos]$. Because $|\{patt|patt \in apprDB_T$ *and* $item \in patt[pos]\}| \geq sup_{min}$, we could also derive that T' is frequent. Hence, T is not a frequent maximal pattern. □

Property 2 (Property 1, Equivalent Projected Databases). *Given two patterns T and T', if we have $projDB_T = projDB_{T'}$ then T and T' could be extended to derive the exact same set of super-patterns.*

Proof. Since T and T' have the same project database, they all could be extended with the same set of local frequent items. Hence they have the same set of extended superpatterns. □

Theorem A.4 (Theorem 5.6, Backward Pruning). *Pattern T would be safely pruned if there exists a tuple $(pos, item)$, where $item \notin T[pos]$ and $|\{patt|patt \in apprDB_T$ and $item \in patt[pos]\}| = sup_T$.*

Proof. Given pattern T and the tuple $(pos, item)$, we could get a new pattern $T' = T \uplus (pos, item)$ by inserting T with $item$ at the position pos. Since $|\{patt|patt \in apprDB_T$ *and* $item \in patt[pos]\}| = sup_T$, we could know that T''s appearance database is equivalent to T's. According to Property 1, we know that both T and T' have the same set of superpatterns after being extended with local frequent items. If we denote the new superpatterns of T and T' by T^* and T'^* grown with the same local frequent item, we always have $T^* \sqsubset T'^*$. Hence both T and its extended superpatterns could not be frequent maximal according to the definition, and we can safely prune T. □

Theorem A.5 (Theorem 5.7, Correctness). *The hybrid method outputs the complete set of frequent maximal patterns.*

Proof. With the complete set of frequent maximal patterns, we could divide it into two disjunctive subsets by whether each pattern is contained by one of the patterns in $frequent$. For those contained by one of the patterns in $frequent$, their superpatterns are in $frequent$. Because they are maximal, there are no frequent superpatterns except themselves. Hence, they all appear in $frequent$. While for those contained by none of the pattern in $frequent$, they are discovered by mining the segments in $infrequent$, since all the segments containing the patterns are in $infrequent$. Hence the merged patterns contain all of the frequent maximal patterns. Since the merging progress filters out those non-maximal patterns, the final output contains only the complete set of frequent maximal patterns. □

A.3 Modification for Closed Periodic Pattern Mining

Here we discuss the algorithm about frequent closed periodic pattern mining. Before giving the details, we first give some theorems about pruning and checking. Since the theorems are modified from those for mining maximal patterns, here we do not duplicate their very similar proofs.

Theorem A.6 (Forward Checking). *Pattern T would not be a frequent closed pattern if there exists a tuple $(pos, item)$, where $|\{projPatt | projPatt \in projDB_T$ and $item \in projPatt[pos]\}| = sup_T$.*

Lemma A.7. *Pattern T would not be a frequent closed pattern if there exists a local frequent item with the frequency of sup_T.*

Theorem A.8 (Backward Checking). *Pattern T would not be a frequent closed pattern if there exists a tuple $(pos, item)$, where $item \notin T[pos]$ and $|\{patt | patt \in apprDB_T$ and $item \in patt[pos]\}| = sup_T$.*

Theorem A.9 (Backward Pruning). *Pattern T would be safely pruned if there exists a tuple $(pos, item)$, where $item \notin T[pos]$ and $|\{patt | patt \in apprDB_T$ and $item \in patt[pos]\}| = sup_T$.*

For example, with the example in Example 4.3 and Figure 2, pattern $b\ *\ d\ *$ would be detected as non-closed by Backward Checking due to the existence of the new "backward local frequent" item c at the position 2. It could also be pruned due to the same reason.

We could find that the checking strategies are similar to those of maximal pattern mining in Section 5, and the pruning strategy is exactly the same as that of maximal pattern mining. For both algorithms, the sets of visited patterns will be exactly the same, and they have the same running time all the time and further the same evaluation results. The derived algorithm for closed periodic pattern mining is similar to that of maximal pattern mining, except that we need to replace the checking strategy with the new one.

A.4 Details of Reference Spots Calculation

We used the latest proposed method in [7] called Reference Spots Calculations to help cluster the original location $(latitude, longitude)$ points into different reference spots, where each reference spot is a region containing much more points than other regions and could be viewed as different living place of animals. The method could be summarized into the following three steps:

(1) Divide the space into a grid of cells.
(2) For each cell c, calculate its density

$$f(c) = \frac{1}{n\gamma^2} \sum_{i=1}^{n} \frac{1}{2\pi} exp(-\frac{|c - loc_i|^2}{2\gamma^2})$$

, where $|c - loc_i|$ is the distance between cell c and the ith data point loc_i. $\gamma = \frac{1}{2}(\delta_x^2 + \delta_y^2)^{\frac{1}{2}} n^{-\frac{1}{6}}$ is a smoothing parameter, where δ_x and δ_y are the standard variances of the x and y coordinates of all the data points.
(3) Select the cells with top-$p\%$ density among all the cells. The reference spots are formed by those contours with the selected cells.