THE NATIONAL UNIVERSITY
of SINGAPORE

School of Computing
Lower Kent Ridge Road, Singapore 119260

TR20/05

Efficient Mining of Dense Periodic Patterns in Time Series Database□□

Chang SHENG, Wynne HSU and Mong Li LEE

October 2005

Technical Report

Foreword

This technical report contains a research paper, development or tutorial article, which has been submitted for publication in a journal or for consideration by the commissioning organization. The report represents the ideas of its author, and should not be taken as the official views of the School or the University. Any discussion of the content of the report should be sent to the author, at the address shown on the cover.

JAFFAR, Joxan
Dean of School

Efficient Mining of Dense Periodic Patterns in Time Series Database

Chang Sheng Wynne Hsu Mong Li Lee
School of Computing, National University of Singapore
{shengcha, whsu, leeml}@comp.nus.edu.sg

Abstract

Existing techniques to mine periodic patterns in time series data are focused on discovering full-cycle periodic patterns from an entire time series. However, many useful partial periodic patterns are hidden in long and complex time series data. In this paper, we aim to discover the partial periodicity in local segments of the time series data. We introduce the notion of character density to partition the time series into variable-length fragments and to determine the lower bound of each character's period. We propose a novel algorithm, called DPMiner, to find the dense periodic patterns in time series data. The algorithm makes use of an Apriori-like property to prune the search space. Experimental results on both synthetic and real-life datasets demonstrate that the proposed algorithm is effective and efficient to reveal interesting dense periodic patterns.

1 Introduction

Time series databases contain data that evolve over time. Research in time series databases focuses on discovering different types of patterns, e.g., periodic association rule [13], sequential patterns [1, 15, 14, 4], partial periodic patterns [8, 17], surprising periodic patterns [18, 19], and so on. Many of these patterns involve periodicity detection. Two kinds of periodicity detections exist: full-cycle periodicity and partial periodicity [9]. In full-cycle periodicity, every point in the time series contributes to part of the cycle (e.g. the season cycle of the year). In partial periodicity, only a portion of the time series data are essential to the mining results.

Recent research focuses on partial periodicity detection [9, 17, 18, 16, 20, 2]. Most of these works suffers from the assumption that the period is known before. Unfortunately, this assumption is inadequate in most real-life applications. To tackle this problem, a linear distance-based algorithm [11] and convolution-like formulas [3, 5, 6] are devised for discovering possible periods. While they alleviate the constrain of known periods assumption, they suffer from two drawbacks.

The first drawback is that the periodic patterns must be discovered from the entire time series. This is actually inadequate in some applications where the periodic patterns occur only within small segments of the time series. Consider the following example: Tom is an employee who drives his car to work every day. However, his route may change from time to time, because of traffic congestion problem. For the first time interval, say, one month, he

follows the route of "home→BLOCK A→BLOCK D→company"; In the second time interval(the second month), he follows the route of "home→BLOCK B→BLOCK K→company"; and in the third interval(including the third and fourth months), he changes to the route "home→BLOCK C→BLOCK F→company". Existing periodicity detection algorithms are unable to discover his travelling habits since they are present in only a third of the entire three months period.

The second drawback is that they treat all potential periods as equal. As a result, they need to spend much time searching for patterns of all possible periods. For example, the algorithm in [6] validate all the periods from 2 to $n/2$ in a time series of length n. Yet we observe that not all of periods are interesting to the users. In general, the users may be interested in a narrow range of periods, or prefer to patterns of short periods than patterns of multiple period.

In this paper, we develop a new periodicity detection algorithm to efficiently discover short period patterns that may exist in only a limited range of the time series. We refer to these patterns as the *dense periodic patterns*. Our contributions are summarized as follows:

1. We introduce the notion of dense periodic patterns where the periodicity is focused on part of time series. To the best of our knowledge, this is the first work that deals with localized segments periodic patterns.
2. We design a pruning strategy to limit the search space to just the feasible periods.
3. We develop a dense periodic pattern mining algorithm called DPMiner. The results of experiments on both real-life and synthetic datasets indicate that DPMiner is both scalable and efficient.

The rest of the paper is organized as follows. Section 2 gives a survey of related work. Formal notations and definitions are presented in Section 3. Section 4 describes the basic idea behind the proposed algorithm. Section 5 gives the details of the algorithm DPMiner for mining dense periodic patterns. In Section 6, we study the performance of DPMiner. Finally, we summarize our findings and contributions in Section 7.

2 Related Work

Han et.al [9] first introduce the concept of partial periodic patterns and propose an efficient top-down approach to mine such patterns by utilizing a novel structure called *max-subpattern tree* to facilitate the counting of candidate patterns in [8]. Aref et.al [2] further extend Han's work to the problem of incremental periodic pattern mining. According to the incremental time series, the max-pattern is dynamically updated, and accordingly the max-subpattern tree is tuned to guarantee its coherence to the whole time series. However, both these works assume that the periods are known in advance.

Wang et.al [16] propose a meta-pattern model to extend the representation of partial periodic patterns by encapsulating patterns and meta-patterns together through a hierarchical organization. Although this approach is flexible to capture the hierarchical periodic patterns, it suffers from the exponential growth of candidate meta-patterns.

Yang et.al [17] address the error tolerance problem in time series. They find the longest valid subsequence for every 1-pattern and employ a bottom-up level-wise approach to generate the subsequences for i-patterns, $i > 1$. But the bottom-up strategy requires multiple scans for mining the long period patterns. To mine patterns from

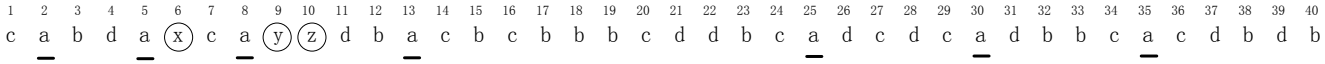


Figure 1. Example of a Time Series T

spatio-temporal data, Mamoulis et.al [12] propose an algorithm by incorporating clustering techniques and max-subpattern tree structure. They use a grid plane and the mapping scheme to transform clusters into the traditional string-like 1-patterns.

All the works above assume that the periods of the patterns are known and their mining algorithms are designed to be directed by the known periods. However, in many real-life applications, the periods are often not known. For such situations, these mining algorithms will need to iterate through all the possible periods in order to discover patterns of unknown periods. This is expensive and more efficient algorithms to mine patterns with unknown periods is needed.

Ma et.al [11] devise a linear distance-based algorithm for discovering possible periods. However, this algorithm may miss some valid periods since it considers only the adjacent intervals. For example, suppose a symbol s occurs in a time series in positions $2, 7, 9, 11, 14, 16$. Then 7 will be a potential period for the symbol s since it occurs in positions $2, 9, 16$. But the algorithm will miss the period 7 for s because it only considers the intervals $5, 2, 2, 3, 2$.

Other works such as [3, 5, 6] employ a convolution-like formula to validate all the periods from 2 to $n/2$ in a time series of length n. This approach can discover all the periodic patterns, but it does not take advantage of the fact that in general, short periods are preferred since they are more accurate and informative compared to the longer ones. For example, a user will typically prefer patterns of period 5, instead of patterns of periods which are multiples of 5, e.g. 10 and 15.

3 Background

In this section, we will give the definitions and notations used.

Given a finite alphabet $\sum$, we define a **time series** T as a sequence of characters $c_1 \cdots c_i \cdots c_n$ where $c_i \in \sum$.

An **itemset** I is a set of zero or more non-repeated characters denoted as $\{c^*\}$, where $c \in \sum$. Examples of itemsets are $\{a, b\}$ and $\{a\}$. For simplicity, we write $\{a\}$ and a interchangeably. An empty itemset, denoted as $*$, refers to the itemset with no symbol in $\sum$.

A **pattern** $P = (I_1 I_2 \ldots I_p)$, is an ordered sequence of itemsets. The **length** of a pattern $|P|$ is the number of non-$*$ itemsets in P. For example, $(*\{b, c\} * *)$ is a pattern of length 1 and $(ab * *)$ is a pattern of length 2.

An i**-pattern** refers to a pattern that contains exactly i characters in $\sum$. For example, $(\{b, c\}*)$ is a 2-pattern and $(abc*)$ is a 3-pattern.

A pattern $(I'_1 I'_2 \ldots I'_p)$ is said to be a **subpattern** of the pattern $(I_1 I_2 \ldots I_p)$ if both patterns have the same length and $I'_i \subseteq I_i$ for all i, $0 \leq i \leq p$. For example, $(a * b * e)$ and $(* * \{b, c, d\} * e)$ are two subpatterns of the pattern $(a * \{b, c, d\} * e)$.

A time series $T = c_1 \cdots c_n$ of n characters can be partitioned into a set of **segments** S_i of length k, where $0 \leq i < \lfloor \frac{n}{k} \rfloor$ and $S_i = c_{i|k|+1} \ldots c_{i|k|+k}$. Given a pattern $P = (I_1 I_2 \ldots I_k)$ of period k and a segment $S_i = c_{i1}, \ldots, c_{ik}$, we say that P **matches** S_i (or S_i **supports** P) if and only if, for each position $j (1 \leq j \leq k)$, either

$I_j = *$ or $c_{ij} \in I_j$.

A segment is called a **frequent segment** of a pattern P if the segment supports P, otherwise, it is an infrequent segment. The **frequency_count** of a pattern P is the total number of frequent segments in the time series. The **confidence** of a pattern P is defined as the ratio of frequency_count over the total segment count, that is,

$$confidence(P) = \frac{frequency\_count(P)}{m},$$

where $m = \lfloor \frac{n}{k} \rfloor$.

Given the minimum confidence min_conf, we say that a pattern P is **frequent** in the time series if its confidence is greater than or equal to the minimal confidence, $confidence(P) \geq min\_conf$.

4 Dense Periodicity

Density is a familiar concept in the area of data mining and has been widely used in many density-based clustering methods [7, 10]. The traditional definition of density relates to the spatial distance. To adapt to the requirement of time series, we define the concept of density of a symbol in the alphabet $\sum$.

4.1 Density in Time Series

The **distance** between any two characters, say c_i and c_j, in the time series $T = c_1 \cdots c_n$, is defined to be $|i - j|$. For example, in Figure 1, the distance between x and y is $|9 - 6| = 3$, and the distance between y and z is $|10 - 9| = 1$.

DEFINITION 4.1 *Given a time series $T = c_1 c_2 \cdots c_n$, and a maximum distance, d_{max}, two characters c_i and c_j are said to be **directly density-reachable** if c_i and c_j are of the same symbol in $\sum$ and there is no other character in the segment $c_{i+1} \cdots c_{j-1}$ that is of the same symbol. Furthermore, the distance between c_i and c_j is less than or equal to d_{max}.*

Consider again Figure 1. Suppose d_{max}=10. Then the two characters of the symbol 'a' in positions 2 and 5 are directly density-reachable because their distance is 3, which is less than d_{max}. On the other hand, the two characters of the symbol 'a' in position 13 and 25 are not directly density-reachable because their distance is 12, which exceeds d_{max}.

DEFINITION 4.2 *For a symbol $s \in \sum$, we extract an ordered sequence of characters of the symbol s from time series T. If any two neighboring characters in this sequence are directly density-reachable, and there is no other character of the same symbol s in T that can be directly density-reachable to any character in this ordered sequence, we say that this ordered sequence is a **closure set** for s, denoted as Cl_s.*

In Figure 1, the sequence of characters of the symbol a is $(a_2, a_5, a_8, a_{13}, a_{25}, a_{30}, a_{35})$. We observe that the distance between a_{13} and a_{25} is greater than d_{max}=10 whereas the distance between all other neighboring pairs are less than d_{max}. In other words, we can form two closure sets for the symbol a: $\{a_2, a_5, a_8, a_{13}\}$ and $\{a_{25}, a_{30}, a_{35}\}$.

DEFINITION 4.3 *Let bpos and epos be the beginning and ending positions of the first and last characters in the closure set of symbol s. A **dense fragment** of s in the time series T is the continuous sequence of characters in T from position bpos to position epos, denoted as $F_{s,(bpos,epos)}$.*

DEFINITION 4.4 *The **dense fragment set** of s in the time series T is the set of all the dense fragments of s, denoted as FS_s.*

We calculate the length of a dense fragment $F_{s,(bpos,epos)}$ to be $epos-bpos$, denoted as $|F_{s,(bpos,epos)}|$. Similarly, the $|FS_s|$ is the sum of the length of all the dense fragments. For example, in Figure 1, the dense fragment set FS_a for symbol a includes two dense fragments, $F_{a,(2,13)}$ and $F_{a,(25,35)}$. $|F_{a,(2,13)}| = 11$, $|F_{a,(25,35)}| = 10$, and $|FS_a| = 11 + 10 = 21$. It is obvious that the length of a dense fragment set of any symbol is less than $|T|$.

4.2 Lower Bound Period in Fragments

In this section, we derive a lower bound on the period as implied by the symbol's density.

Traditionally, for a time series data with unknown periods, the mining algorithm needs to iteratively inspect all the possible periods from 2 to half of the length of the time series data. This is very expensive.

Here, we show that for a given d_{max} (the maximum allowable distance between two directly density-reachable characters) and min_conf (the minimum confidence), we can deduce a lower bound period for all the possible 1-patterns containing the symbol $s \in \sum$.

THEOREM 4.1 (Lower Bound Period) *For a given symbol $s \in \sum$, we can obtain the closure set of s, denoted as Cl_s, and a dense fragment of s, denoted as $F_{s,(bpos,epos)}$. The lower bound period of all possible 1-patterns containing symbol s in $F_{s,(bpos,epos)}$ is equal to $\frac{|F_{s,(bpos,epos)}| \times min\_conf \times d_{max}}{|Cl_s| \times d_{max} - |F_{s,(bpos,epos)}| \times (1-min\_conf)}$.*

PROOF: From the definition of closure set, we know that Cl_s contains $|Cl_s|$ occurrences for symbol s in the fragment $F_{s,(bpos,epos)}$. In order to determine the minimum period, we require that the patterns within this period be frequent. With this in mind, we partition Cl_s into two sets. The first set, called $Type-I$ characters, contain characters that occur in the frequent segments (whose count determine whether a pattern is a frequent pattern). The second set, called $Type-II$ characters, contains the remaining characters that occur in the infrequent segments. Clearly, we have

$$|Cl_s| = Count_{Type-I} + Count_{Type-II}$$

Since the period of a frequent pattern is determined by the characters in the set $Type-I$, by assigning as many characters as possible to $Type-I$, we hope to increase the density of this symbol within the frequent segments, thus leading to the minimum period. More specifically, we have the count in $Type-I$:

$$Count_{Type-I} = \lfloor \frac{|F_{s,(bpos,epos)}|}{period_{min}} \rfloor \times min\_conf,$$

where $period_{min}$ is the minimal period that we want to compute.

Once we have obtained the count of characters in $Type-I$, the count of the characters in the set $Type-II$ is computed by spreading the remaining characters as widely as possible, i.e. d_{max}.

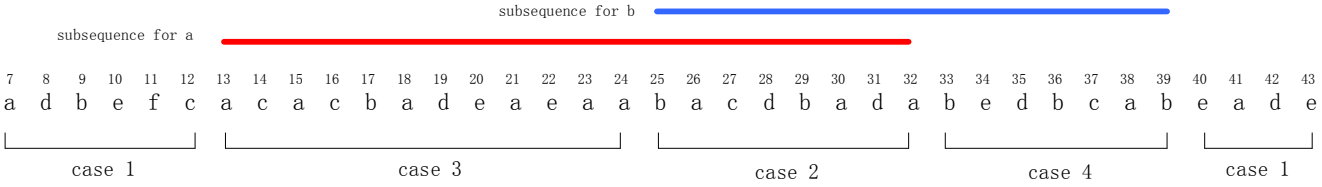


Figure 2. Fragments for Symbols 'a' and 'b'

$$Count_{Type-II} = \frac{|F_{s,(bpos,epos)}| - Count_{Type-I} \times period_{min}}{d_{max}}.$$

Integrating the formula of $Count_{Type-I}$, we have

$$Count_{Type-II} \approx \frac{|F_{s,(bpos,epos)}| - |F_{s,(bpos,epos)}| \times min\_conf}{d_{max}}.$$

Finally, by combining the formulae of $Count_{Type-I}$ and $Count_{Type-II}$, the minimal period can be computed as follows:

$$period_{min} = \frac{|F_{s,(bpos,epos)}| \times min\_conf \times d_{max}}{|Cl_s| \times d_{max} - |F_{s,(bpos,epos)}| \times (1 - min\_conf)}$$

For example, in Figure 1, suppose we let min_conf=0.8 and d_{max}=10, we can easily compute the minimum period of fragment $F_{a,(2,13)}$ as follows:

$period_{min} = \frac{12 \times 0.8 \times 10}{4 \times 10 - 12 \times 0.2} \approx 2.55$.

The minimum period obtained indicates that it is not possible to find frequent patterns containing the symbol a within a period of 2. A closer look reveals that with a specified period of 2, $F_{a,(2,13)}$ contains 6 segments, in which it requires at least $\lceil 6 \times 0.8 \rceil = 5$ a in the frequent segments. However, we only have $|Cl_a| = 4$ a. In other words, the 1-patterns of period 2 containing 'a'(i.e, $(a*)$ and $(*a)$) are impossible to be frequent in $F_{a,(2,13)}$.

This theorem allows us to prune the search space of all the periods that are less than the lower bound computed.

4.3 Density-Based Pruning

Having found the lower bound periods of the 1-patterns that allow us to prune the search space of all those periods that are less than the lower bound computed, we now need to extend the pruning to the k-patterns, $k > 1$. For simplicity, we assume that the entire time series is divided into segments of period p. Our goal is to identify all the promising high density regions and perform mining only in these regions.

Consider the time series in Figure 2 with min_conf = 0.7 and d_{max} = 5. There are two overlapping fragments: $F_{a,(13,32)}$ is a dense fragment for symbol a of lower bound period 2, and $F_{b,(25,39)}$ is a dense fragment for symbol b of lower bound period 3. The overlapped portion, from 13 to 39, can be divided into three parts: (13,24) which only belongs to $F_{a,(13,32)}$, (25,32) belongs to both $F_{a,(13,32)}$ and $F_{b,(25,39)}$, and (33,39) which only belongs to $F_{b,(25,39)}$.

Let us first consider the case of period 2. Here, the 2-patterns (i.e, (ab) or (ba)) are not likely to exist in $F_{b,(25,39)}$, because the lower bound period for $F_{b,(25,39)}$ is 3. As for(13,24), we need to inspect whether there are other b's fragments with lower bound of 2 that overlap with it. In other words, we can eliminate (25, 39) from the mining of 2-patterns with period 2.

For period 3 or more, we have four cases.

- **Case 1**. The partial time series does not belong to the fragments of symbol a nor the fragments of symbol b. In other words, the distance between any two arbitrary a or b is more than d_{max}. This implies that both a and b occur sparsely, and can be pruned.

- **Case 2**. This is the overlapping region. Here, it is certainly possible to find the frequent segments that support patterns consisting of a and b.

- **Case 3**. In this case, the time series belong to a's fragments but do not belong to any of the b's fragments. Due to the low density of b, it is unlikely to include the frequent segments which support the patterns containing both symbols except nearer to the overlapped region. By extending the overlapped regions (Case 2) slightly on both ends (see Figure 2), we can avoid searching the remaining regions as they are not likely to contain the 2-patterns.

- **Case 4**. In this case, the time series belong to b's fragments but do not belong to any of the a's fragments. The analysis of this case is identical to Case 3.

The above cases can be easily extended to three or more symbols. In other words, to check whether a pattern P is frequent, we only need to check the overlapping dense regions of all the itemsets in P.

We will now formally define the pruning strategy.

DEFINITION 4.5 *Given a period p and an itemset $I = \{c_1, c_2, \ldots, c_m\}$, the **dense region** of I with period p, $DR(I, p)$, is defined as the union of all the dense fragments of c_i, $1 \leq i \leq m$, where the lower bound period of each dense fragment is less than or equal to p.*

The dense region of the empty itemset $*$ is defined to be the entire time series. Note that all the merged dense fragments must have the lower bound less than or equal to p so that we make sure these fragments to satisfy the minimal density requirement.

DEFINITION 4.6 *For a pattern P with period p, $(I_1, \ldots, I_p)$, the **dense interval** of P, $DI(P, p)$, is defined to be the intersection of all the dense regions of its itemset members, namely $DI(P, p) = DR(I_1, p) \cap \ldots \cap DR(I_p, p)$.*

Given a period p, alphabet $\sum$, and the dense fragments of all symbols in $\sum$, we detect the frequent 1-patterns from these dense fragments. To mine the k-patterns, $k > 1$, we only need to check the intersection regions, namely the dense intervals of the k-pattern.

5 DPMiner

In this section, we show how the proposed density based pruning strategy can be incorporated into our mining algorithm DPMiner (Dense Periodic pattern Miner). The outline of DPMiner is described in Algorithm 1. The algorithm mines dense periodic patterns in two phases.

The first phase (Steps 1 to 5 in Algorithm 1) scans the time series once to obtain the dense fragments for each symbol s in $\sum$. To avoid having too many short and trivial fragments, a parameter $\mu \in (0, 1]$ is used so that only the fragments whose lengths exceed $\mu\times$ length of time series are accepted as dense fragments.

The second phase (Steps 6 to 12 in Algorithm 1) utilizes a top-down method that is similar to the one proposed in [8]. The work in [8] define the *max-pattern*, P_{max}, to be the maximal pattern which can be obtained by merging all frequent 1-patterns F_1. For example, if the F_1={a***, c***, *b**, ***d}, the max-pattern will be {a, c}b*d.

Given the max-pattern P_{max}, a subpattern of P_{max} is *hit* by a segment S_i if it is the maximal subpattern of P_{max} in S_i. For example, if P_{max}={a, c}b*d, the subpattern $cb*d$ is hit by segment $S_i = cbdd$.

A data structure called *max-subpattern tree* is designed to facilitate the registration of the hit subpatterns and their counts. The root node of the max-subpattern tree is the max-pattern P_{max}. Each subpattern of P_{max} with one non-* letter missing is a direct child node of the root. The tree is constructed by the recursive insertion of the max-subpatterns. If the node of the corresponding max-subpatterns already exists, the hit count of that node is increased by 1; otherwise, a new node is created.

The top-down algorithm can be summarized in three steps. The first step finds the set of frequent 1-patterns F_1 by scanning the time series once, and thereby forming the max-pattern, P_{max}. The second step constructs the max-subpattern tree by scanning the time series once again. The last step is the mining procedure from which the frequent patterns are derived according to the constructed max-subpattern tree.

Algorithm 1 DPMiner

Input: A time series $T = c_1 c_2 \ldots c_n$
Alphabet $\sum$
Maximal distance of two characters in fragments d_{max}
Fragment length coefficient μ
Periodicity threshold min_conf.

Output: Periodic patterns for T.
{The periods of the output patterns are in the range of $[2, d_{max}]$}

1: Scan the time series once to find the fragment set S_s for each symbol $s \in \sum$;
2: **for** each symbol $s \in \sum$ **do**
3: Delete the fragments of length less than $\mu \times |T|$ from S_s;
4: Compute the lower bound period of every fragment in S_s;
5: **end for**
6: **for** period p= 2 to d_{max} **do**
7: For symbol s, discover the frequent 1-patterns, $|F1_s|$, from S_s;
8: Merge all $|F1_s|$, $s \in \sum$, to obtain max-pattern P_{max};
9: Let P_{max} with period p is the root node of max-subpattern tree R, compute the dense support $DS(R, p)$;
10: Scan $DS(P, p)$ to construct R;
11: Traverse R to output the frequent patterns.
12: **end for**

However, instead of scanning the entire time series as is done in the algorithm in [8], the proposed Algorithm 1 only scans the union of the dense regions of the corresponding root nodes' itemsets.

DEFINITION 5.1 *For a max-subpattern tree R taking the max-pattern P_{max} with period p, $(I_1, \ldots, I_p)$, as its root node, the **dense support** of tree R, $DS(R, p)$, is defined to be the union of all the dense regions of its root node's itemsets, namely $DS(R, p) = DR(I_1, p) \cup \ldots \cup DR(I_p, p)$.*

The dense support of tree R thoroughly covers all segments supporting the max-pattern or its subpatterns. In

other words, only those segments in $DS(R)$ are useful for the construction of max-subpattern tree. This allows us to prune away many unnecessary segments.

We also make modifications to the structure of the max-subpattern tree as follows. Besides the hit count in each node, we also calculate and store the patterns' dense fragments. An example of the modified tree is shown in Figure 3.

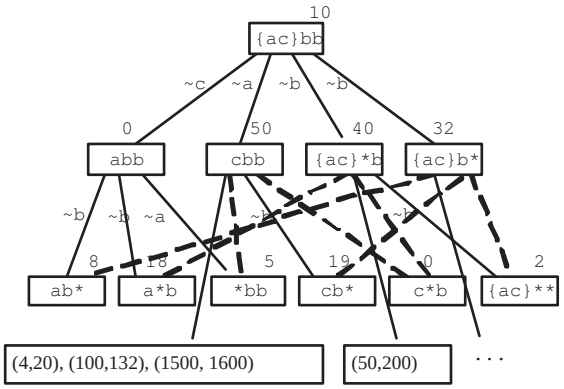


Figure 3. Example of a Max-Subpattern Tree

Let us now discuss how to build the max-subpattern tree and how we can mine patterns from the tree.

We first compute the dense support of a max-subpattern tree R according to its max-pattern and Definition 5.1. Note that we only scan the the dense support of tree R to build its max-subpattern tree, instead of the time series (this is also a reason why DPMiner gives better performance). For every segment in dense support of R, we check whether it supports the max-pattern P_{max} or any subpattern of max-pattern it. If yes, this maximal subpattern P' will be found by starting from the root node P_{max} and along the branches. If the node of P' is found, its count is increased by 1. Otherwise, a new node of P' is created (together with any non-existent ancestors) and the dense interval for P' is computed accordingly.

Consider the example in Figure 3. Let a segment cbd be in dense support of R. Then it supports the subpattern $(cb*)$ since $cbd \wedge P_{max} = cbd \wedge (\{ac\}bb) = (cb*)$. This subpattern will be found along the route $(\{ac\}bb) \rightarrow (cbb) \rightarrow (cb*)$ or $(\{ac\}bb) \rightarrow (\{ac\}b*) \rightarrow (cb*)$.

Analysis. Phase 1 in Algorithm 1 scans the time series once. Step 7 in phase 2 scans $min\{|T|, |S_{c_1}| + \ldots + |S_{c_i}| + \ldots + |S_{c_k}|\}$ characters, where T is the time series and $c_i \in \sum$. Step 10 scans $|DS(R, p)|$ characters, which is also less than $|T|$. The total number of scanned characters for an iteration in phase 2 is $\alpha \times |T|$, where $\alpha \leq 2$ and α is bounded by the time series. For a range of $k(= d_{max})$ period values, we scan $k \times \alpha \times |T|$ characters. Thus, the total number of characters scanned in DPMiner is $(1 + k \times \alpha) \times |T|$, for mining periodic patterns of k periods.

6 Experiment Evaluation

In this section, we give the results of the experiments that are carried out to evaluate the performance of DP-Miner. We implement the DPMiner algorithm in Java. The experiments are performed on a Pentium 4 3Ghz PC with 1GB of memory, running Windows XP. Both synthetic datasets and real-life datasets are used.

Synthetic Datasets: We use the Elfeky's program [1] to generate the synthetic time series data. By adjusting different parameters such as data distribution (uniform and normal), length of time series, period, and alphabet size, we obtain two synthetic datasets

1. DATA-6-10000 is a time series satisfying normal distribution with 6 symbols and 10000 characters;
2. DATA-21-75000 is a time series satisfying normal distribution with 21 symbols, and it consists of 75000 characters.

Real-life Dataset: The PACKET time series dataset[2] serves as the real-life dataset in our experiment studies. This dataset records the packet Round Trip Time (RTT) delay. We discretize the values into 360K-length characters of alphabet 26.

6.1 Sensitivity Experiments

The dense fragments are significant to the final mining result. If the fragment set for a symbol cover too many segments, it is not easy to satisfy the minimal confidence requirement. The extreme case is the entire time series, in which the mining performance and discovered patterns are the same to the previous max-subpattern tree hit set (MTHS) method [8]. Otherwise, the dense fragments should not be extremely short to avoid the patterns within them to be trivial. We will observe the effect of two parameters, namely the maximum distance d_{max} and the minimal length of dense fragment μ, on the running time of DPMiner, and select the best μ value to ensure adequate dense fragments.

We first vary the value of d_{max} from 10 to 120, and fix the value of μ at 0.01. Figure 4 shows the time taken by DPMiner on the PACKET dataset. We observe that d_{max} has a direct effect on the runtime, which can explain that d_{max} affect the length of dense fragments as follows. According to Definition 4.1 and 4.2, the lower the d_{max} value, the more characters are included in the closure set. This leads to longer dense fragments, and the dense support of max-subpattern tree covers more segments. Hence, we will need more time to search for patterns.

μ	Period=3	Period=5	Period=7
0.001	0	0	0
0.005	1	3	2
0.01	1	4	4
0.015	0	1	1
0.02	0	0	0
0.025	0	0	0
0.03	0	0	0

Table 1. Number of Patterns Found for Varying Values of μ

Next, we set the value of d_{max} to 30, and vary the value of μ from 0.001 to 0.03. μ is the parameter to control the minimal length of dense fragment, and only the dense fragments whose lengths exceed $\mu \times |T|$ are accepted

[1] We get code from http://www.cs.purdue.edu/ mgelfeky/Source.
[2] http://www.cs.ucr.edu/ eamonn/TSDMA/packet.data

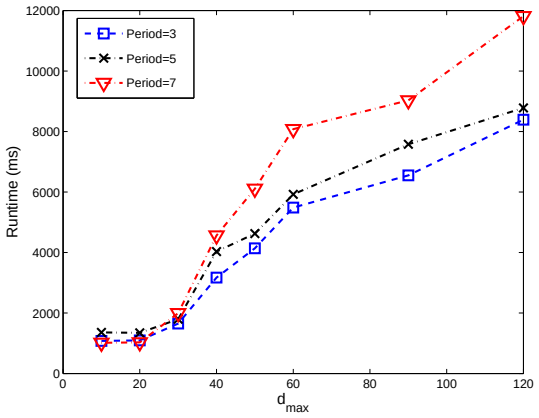


Figure 4. Experiment to Find Optimal Value for d_{max} (PACKET dataset)

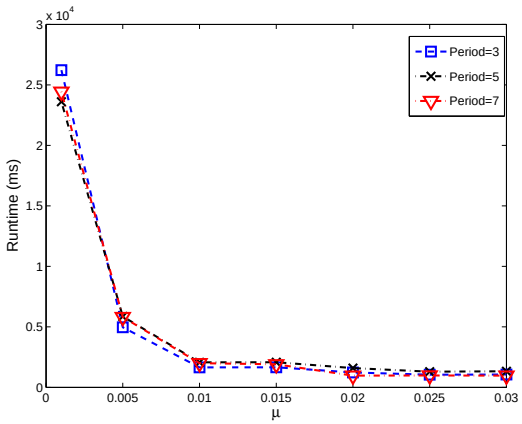


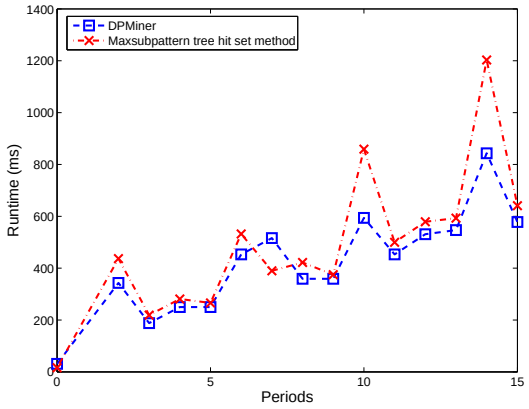
Figure 5. Experiment to Find Optimal Value for μ (PACKET dataset)

as dense fragments, where T is the time series. Figure 5 shows the runtime of DPMiner on the PACKET dataset. We observe that the smaller the value of μ, a lot of dense fragments will be generated, and more time is needed to search for patterns. On the other hand, a large μ value will lead to limited frequent patterns, as shown in Table 1. This is because a large μ value will cause no fragment to be accepted as dense fragment. By considering both running time and the number of patterns discovered, we find the μ=0.01 yields the optimal performance. For the rest of the experiments, we set $\mu = 0.01$.

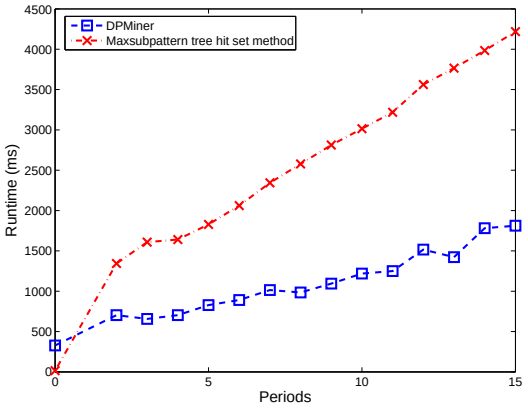
6.2 Efficiency

Next, we run our algorithm on the three datasets and compare its running time with MTHS method. We set $d_{max} = 30$ and $\mu = 0.01$. Figure 6 shows the experimental results. Note that the period '0' on the axis y denotes the time needed for the initialization phase.

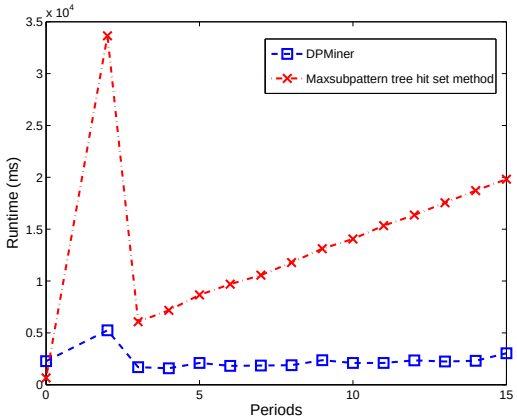
Figure 6(a) shows that DPMiner has nearly the same performance as MTHS method for the DATA-6-10000 dataset. Figures 6(b) and 6(c) show that DPMiner outperforms MTHS method for the datasets DATA-21-75000



(a) DATA-6-10000



(b) DATA-21-75000



(c) PACKET

Figure 6. Time Comparison of Two Algorithms

Period	Pattern	Confidence	Dense Interval
2	be	0.40938404	[9002, 12998][21002, 25997]
3	eea	0.3804841	[8, 2993][9460, 10436]
4	**be	0.31823775	[2, 2174][2216, 2998][9002, 12998][21002, 25997][59193, 60040][60075, 60997]
5	d*ebc	0.38138783	[21004,25995][54004,55010][56939,58454][59193,60040][60075, 60994]
5	d*ebc	0.38138783	[21004, 25995][54004, 55010][56939,58454][59193, 60040][60075, 60994]

Table 2. Patterns in DATA-21-75000

Period	Pattern	Confidence	Dense Interval
2	aa	0.30474216	[148187, 155084][181413, 186097][300522, 304372]
3	*aa	0.3124757	[148187, 155084][181413, 186097][300522, 304372]
4	aa**	0.3103627	[148187, 155084][181413, 186097][300522, 304372]
5	aa***	0.3001943	[148187, 155084][181413, 186097][300522, 304372]
5	*aa**	0.30829015	[148187, 155084][181413, 186097][300522, 304372]
5	***aa	0.32901555	[148187, 155084][181413, 186097][300522, 304372]
5	*a*a*	0.31606218	[148187, 155084][181413, 186097][300522, 304372]
6	a**a**	0.30120483	[148187, 155084][181413, 186097][300522, 304372]
6	***aa*	0.31053245	[148187, 155084][181413, 186097][300522, 304372]
6	****aa	0.34317917	[148187, 155084][181413, 186097][300522, 304372]

Table 3. Patterns in PACKET

and PACKET.

It is because the dense support of the max-subpattern tree is greatly affected by the alphabet size of the dataset. For time series data with small alphabet size, the dense support of the tree is equivalent to the entire dataset. In this case, DPMiner will hardly prune any data. On the contrary, for time series data composed of a large alphabet, DPMiner can successfully omit a lot of meaningless data since only a few of characters of alphabet are the itemsets of max-pattern. In other words, only their dense fragments are useful for building the max-subpattern tree. For the reason, DPMiner exhibited better performance on DATA-21-75000 (alphabet size is 21) and PACKET (alphabet size is 26).

6.3 Effectiveness

We also compare the patterns discovered by DPMiner and MTHS method. Besides $d_{max} = 30$ and $\mu = 0.01$, we set $min\_conf = 0.3$. Table 2 shows the patterns found by DPMiner with the period range from 2 to 5 on the synthetic dataset DATA-21-75000. Table 3 shows the patterns found by DPMiner with the period range from 2 to 6 on real data PACKET. We do not provide the patterns found by MTHS method because it does not discover any frequent patterns on these two datasets.

We observe that DPMiner can discover the dense periodic patterns as well as their density range. The patterns in DATA-21-75000 are expected since we generate this dataset only in these ranges. The patterns mined from the PACKET dataset are meaningful to keep track of the network traffic.

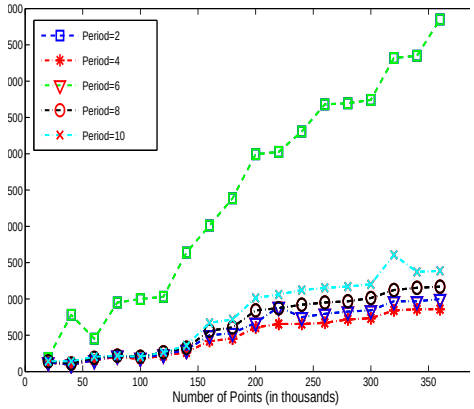


Figure 7. Runtime on Varying Dataset Size

6.4 Scalability

Finally, we study the scalability of DPMiner by varying the length of the time series data from 20K to 360K. We also set $d_{max} = 30$ and $\mu = 0.01$. Figure 7 shows that the execution time of DPMiner is linearly proportional to the size of the time series.

7 Conclusion and Future Works

In this paper, we have defined the problem of mining dense periodic patterns. We introduced the concepts of density and fragment, and a strategy for pruning the search space. We have developed a mining algorithm called DPMiner to discover the dense periodic patterns. The experiments carried out on both synthetic and real-life datasets prove the effectiveness and efficiency of DPMiner. The results also show that DPMiner outperforms the existing max-subpattern tree hit set (MTHS) method, especially on the large alphabet datasets.

Future research direction include investigating error tolerance problems whereby the insertion or deletion affect the positions of the characters in the time series, and extending DPMiner to handle more complex data such as spatio-temporal data.

References

[1] R. Agrawal and R. Srikant. Mining sequential patterns. In *International Conference on Database Engineering*, pages 3–14. ieee, 1995.

[2] W. G. Aref, M. G. Elfeky, and A. K. Elmagarmid. Incremental, online, and merge mining of partial periodic patterns in time series databases. 2004.

[3] C. Berberidis, W. G. Aref, M. J. Atallah, I. P. Vlahavas, and A. K. Elmagarmid. Multiple and partial periodicity mining in time series databases. In *ECAI*, pages 370–374, 2002.

[4] H. Cheng, X. Yan, and J. Han. Incspan: incremental mining of sequential patterns in large database. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 527–532, New York, NY, USA, 2004. ACM Press.

[5] M. G. Elfeky, W. G. Aref, and A. K. Elmagarmid. Using convolution to mine obscure periodic patterns in one pass. In *EDBT*, pages 605–620, 2004.

[6] M. G. Elfeky, W. G. Aref, and A. K. Elmagarmid. Periodicity detection in time series databases. 2005.

[7] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, pages 226–231, 1996.

[8] J. Han, G. Dong, and Y. Yin. Efficient mining of partial periodic patterns in time series database. In *ICDE*, pages 106–115, 1999.

[9] J. Han, W. Gong, and Y. Yin. Mining segment-wise periodic patterns in time-related databases. In *KDD*, pages 214–218, 1998.

[10] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, 2001.

[11] S. Ma and J. L. Hellerstein. Mining partially periodic event patterns with unknown periods. In *Proceedings of the 17th International Conference on Data Engineering*, pages 205–214, Washington, DC, USA, 2001. IEEE Computer Society.

[12] N. Mamoulis, H. Cao, G. Kollios, M. Hadjieleftheriou, Y. Tao, and D. W. Cheung. Mining, indexing, and querying historical spatiotemporal data. In *KDD '04: Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 236–245, New York, NY, USA, 2004. ACM Press.

[13] B. Ozden, S. Ramaswamy, and A. Silberschatz. Cyclic association rules. In *ICDE*, pages 412–421, 1998.

[14] J. Pei, J. Han, B. Mortazavi-Asl, and et.al. Prefixspan: Mining sequential patterns efficiently by prefix-projected patter growth. In *Proc. 2001 Int. Conf. Data Engineering*, pages 215–224, April 2001.

[15] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In P. M. G. Apers, M. Bouzeghoub, and G. Gardarin, editors, *Proc. 5th Int. Conf. Extending Database Technology, EDBT*, volume 1057, pages 3–17. Springer-Verlag, 25–29 1996.

[16] W. Wang, J. Yang, and P. S. Yu. Meta-patterns: Revealing hidden periodic patterns. In *ICDM*, pages 550–557, 2001.

[17] J. Yang, W. Wang, and P. S. Yu. Mining asynchronous periodic patterns in time series data. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 275–279, New York, NY, USA, 2000. ACM Press.

[18] J. Yang, W. Wang, and P. S. Yu. Infominer: mining surprising periodic patterns. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 395–400, New York, NY, USA, 2001. ACM Press.

[19] J. Yang, W. Wang, and P. S. Yu. Infominer+: Mining partial periodic patterns with gap penalties. In *Proc. of IEEE International Conference on Data Mining*, pages 725–728, December 2002.

[20] W. Yang and G. Lee. Efficient partial multiple periodic patterns mining without redundant rules. In *COMPSAC*, pages 430–435, 2004.