

STAGGER: Periodicity Mining of Data Streams using Expanding Sliding Windows

Mohamed G. Elfeky*
Google Inc.
mgelfeky@google.com

Walid G. Aref Ahmed K. Elmagarmid
Department of Computer Sciences, Purdue University
{aref, ake}@cs.purdue.edu

Abstract

Sensor devices are becoming ubiquitous, especially in measurement and monitoring applications. Because of the real-time, append-only and semi-infinite natures of the generated sensor data streams, an online incremental approach is a necessity for mining stream data types. In this paper, we propose STAGGER: a one-pass, online and incremental algorithm for mining periodic patterns in data streams. STAGGER does not require that the user pre-specify the periodicity rate of the data. Instead, STAGGER discovers the potential periodicity rates. STAGGER maintains multiple expanding sliding windows staggered over the stream, where computations are shared among the multiple overlapping windows. Small-length sliding windows are imperative for early and real-time output, yet are limited to discover short periodicity rates. As streamed data arrives continuously, the sliding windows expand in length in order to cover the whole stream. Larger-length sliding windows are able to discover longer periodicity rates. STAGGER incrementally maintains a tree-like data structure for the frequent periodic patterns of each discovered potential periodicity rate. In contrast to the Fourier/Wavelet-based approaches used for discovering periodicity rates, STAGGER not only discovers a wider, more accurate set of periodicities, but also discovers the periodic patterns themselves. In fact, experimental results with real and synthetic data sets show that STAGGER outperforms Fourier/Wavelet-based approaches by an order of magnitude in terms of the accuracy of the discovered periodicity rates. Moreover, real-data experiments demonstrate the practicality of the discovered periodic patterns.

1. Introduction

Data streams consist of continuous and time sensitive data. Periodicity mining is a tool that helps in predicting

the behavior of data streams. For example, periodicity mining allows a telephone company to analyze telephone calling patterns and predict periods of high and low usage so that proper planning may take place. In this paper, we address the problem of mining such periodic patterns over data streams. We define periodicity mining as the detection of frequent periodic patterns where the periodicity rate (period length) is also unknown. Although one may argue that visual inspection of the time plot of the time series may lead to the identification of potential periodicity rates, huge amounts of data and the evolution nature of time series streams complicate such visual inspection. Moreover, complex applications like stock market do not follow obvious periodicity rates (daily or weekly). A period may span a long rate (e.g, quarterly), which is not obvious to observe visually. Therefore, periodicity mining comprises mainly two steps. The first step, termed *Periodicity Detection*, is to discover potential periodicity rates. The second step, termed *Mining Periodic Patterns*, is to detect the frequent periodic patterns of each discovered periodicity rate. While this two-step process works for traditional time series data [12, 10], it does not apply well to real-time data streams. Data streams differ substantially from traditional time series data. Of particular interest to data mining are the following two differences. First, while traditional time series assume that data elements are synchronized, and so exact mining results can always be obtained; streams data is often lost, stale, or omitted intentionally. Therefore, mining results must be computed with incomplete information. Second, traditional time series assume that applications require no real-time services whereas data streams are real-time by nature, and therefore require online and incremental processing. Recently, Papadimitriou et al. [11] have addressed the problem of periodicity detection (the first step, above) in data streams using Wavelets. In this paper, we propose a complete solution to the periodicity mining problem in data streams, which combines both periodicity detection and mining periodic patterns in a one-pass algorithm. Moreover, in contrast to the techniques proposed in this paper, the Wavelet-based approaches are approximate. The

*Work done while at Department of Computer Sciences, Purdue University

experiments demonstrate that the quality of the proposed techniques are superior in terms of the accuracy of the discovered periodicity rates.

In this paper, we propose to maintain multiple expanding sliding windows that are staggered over the data stream. Using a convolution-based algorithm [4, 5], a sliding window¹ of length w can discover period lengths up to $w/2$. Whereas a small window length is required for early and real-time output, it limits the period lengths that can be discovered. Therefore, we propose STAGGER, a new algorithm that uses the algorithm in [4] as a building block. STAGGER allows multiple small sliding windows to expand in length to cover the whole data stream, i.e., STAGGER *stagger*s the stream with multiple and concurrent expanding sliding windows. Computations are shared among the multiple overlapping windows. Thus, STAGGER is able to produce interactive output as well as to discover a wide range of potential period lengths. Subsequently, we maintain a max-subpattern tree [6] for each potential period length, which discovers the periodic patterns for that length. We propose an approximate incremental technique for building and maintaining max-subpattern trees.

A related problem to online mining of data streams is that of oscillating patterns. Traditional data mining techniques employ a single frequency threshold value. A pattern is reported as frequent only if its frequency is above that threshold. When data arrives continuously, a pattern may oscillate between being frequent and being infrequent. Maintaining a data structure for frequent patterns, e.g., the max-subpattern tree, such an oscillating pattern loses its history information every time it becomes infrequent. When it becomes frequent again, the pattern’s history has already gone without being stored. Losing a pattern’s history deteriorates the accuracy of one-pass online stream mining algorithms. A naïve solution is to keep all patterns’ histories even for those patterns that become infrequent. However, this solution requires an excessive amount of storage. We propose a new approach, termed “*hysteresis*” *thresholding*, that maintains two thresholds. A pattern is considered frequent if its frequency is above the higher threshold value, and is considered infrequent if its frequency is below the lower threshold value. The distance between the two threshold values acts as a period of time during which a pattern is given a chance to stabilize rather than to oscillate. As we demonstrate on the experimental section, the proposed “hysteresis” thresholding approach shows significant performance improvement over that of the traditional thresholding approach.

The contributions of this paper can be summarized as follows:

1. We propose STAGGER, an online incremental algo-

¹A window is defined in terms of number of symbols rather than a time window

rithm that uses expanding sliding-windows in order to discover potential periodicity rates in data streams. STAGGER uses and shares the execution among multiple expanding sliding-windows that are staggered over the data stream, in order to produce interactive output. This way, STAGGER discovers a wide range of potential periodicity rates.

2. We propose a new incremental technique for maintaining the max-subpattern tree that is used for mining periodic patterns in data streams. The proposed technique requires only one pass over the data stream and no reprocessing of data that has been previously seen.
3. We propose using “hysteresis” thresholding for maintaining the mining thresholds. This approach is very relevant to one-pass online stream mining algorithms and is conservative in terms of preserving the history of candidate frequent patterns.

The rest of this paper is organized as follows. In the remaining of this section, we introduce the notation used throughout the paper. We outline STAGGER in Section 2. In Section 3, we present the proposed technique for online periodicity detection. In Section 4, we present the proposed technique for incremental mining of periodic patterns. In Section 5, we evaluate the performance of STAGGER, and compare it to other approaches using both synthetic and real data. Finally, we conclude the paper in Section 6.

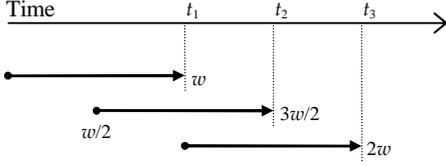
1.1. Notation

A data stream of events is an infinite sequence of timestamped events drawn from a finite set of nominal² event types. An example is an event stream in a computer network that monitors the various events. Let e_i be the event occurring at timestamp i , then the data stream S is represented as $S = e_0, e_1, \dots, e_i, \dots$. Each event type can be denoted by a symbol (e.g., a, b, c). The set of event types can be denoted $\Sigma = \{a, b, c, \dots\}$. Thus, the data stream S can be considered a sequence of infinite length over a finite alphabet Σ .

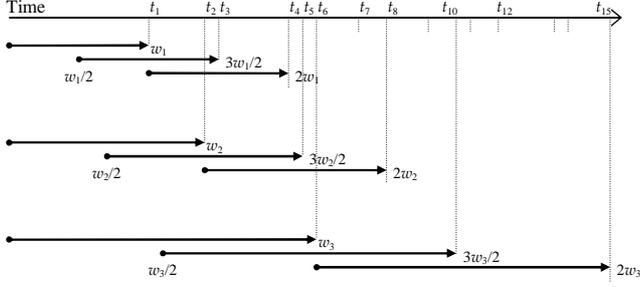
A data stream may also be an infinite sequence of timestamped values collected by a sensor measuring a specific feature. For example, the feature in a data stream for stock prices might be the final daily stock price of a specific company. If we discretize³ the data stream feature values into nominal discrete levels and denote each level (e.g., high, medium, low) by a symbol (e.g., a, b, c), then we can use the same notation above.

²Nominal values are distinguished by name only, and do not have an inherent order or a distance measure.

³The problem of discretizing time series into a finite alphabet is orthogonal to our problem and is beyond the scope of this paper. See [3, 7] for an exhaustive overview of discretization and segmentation techniques, and [9] for streaming implications of discretization techniques.



(a) A window of length w slides $w/2$ positions



(b) Three sliding windows of lengths w_1 , w_2 and w_3

Figure 1. Expanding sliding windows

2. Algorithm Outline

In this section, we outline STAGGER algorithmic behavior. The input is a data stream for which we output its periodic patterns. We are given a periodicity threshold (or two periodicity thresholds if the “hysteresis” thresholding approach is used). Arbitrarily, we select m different windows of lengths $w_1 < w_2 < \dots < w_m$. Notice that w_m can be selected as large as the buffer size that is allocated for buffering the data stream. STAGGER maintains an event-based priority queue where an event is triggered by the arrival of new data in the stream. The events in the priority queue are sorted in increasing order of their timestamps.

STAGGER is outlined in the following steps:

- Initialize the priority queue by the events that correspond to the arrival of a number of symbols equal to w_1, w_2, \dots, w_m (e.g., events at Times t_1, t_2 , and t_6 of Figure 1(b)).
- Get the next event from the priority queue.
- If the event corresponds to considering a new window w_i (e.g., events at Times t_1, t_2 , and t_6 of Figure 1(b)), then apply a convolution-based algorithm [4] over the window w_i (Section 3.2), sharing the results from the previous window w_{i-1} (Section 3.3.2).
- If the event corresponds to sliding the window w_i (e.g., events at Times t_3, t_5 , and t_{10} of Figure 1(b)), then update the results of this window (Section 3.3.1).

- Analyze the results from the previous step (Section 3.2) to obtain potential period lengths p_j , and their corresponding maximal periodic patterns.
- For each p_j , build a max-subpattern tree [6] R_{p_j} using p_j 's corresponding maximal periodic pattern (Section 4).
- Schedule the next event that corresponds to sliding the current window w_i a number of positions equal to $w_i/2$, and insert that event in the priority queue (e.g., in Figure 1(b), the event at Time t_5 schedules the event at Time t_8).

3. Periodicity Detection

Periodicity detection, in our terms, stands for discovering potential rates at which the data stream is periodic. For example, a data stream of the closing price of a specific stock may have a periodicity rate of 7 that describes its weekly periodic pattern. A periodic pattern describes the periodic behavior at, not necessarily all, the points in the period. For example, the closing price of a specific stock may be *high* every Friday and *low* every Tuesday, yet may not have any regularity on the other week days. That description of periodic patterns implies that the technique devised for periodicity detection should consider symbol periodicities. Recall that a symbol may represent an event in an event data stream or a nominal discrete level in a discretized real-valued data stream.

3.1. Symbol Periodicity

In a data stream S , a symbol s is said to be periodic with a period of length p if s exists “almost” every p timestamps. For example, in the data stream $S = \text{abcabbabdb}$, the symbol b is periodic with a period of length 4 since b exists every four timestamps (positions 1, 5 and 9). Moreover, the symbol a is periodic with a period of length 3 since a exists almost every three timestamps (positions 0, 3, and 6 but not 9). Let $\pi_{p,l}(S)$ denote the projection of a data stream S according to a period p starting from position l :

$$\pi_{p,l}(S) = e_l, e_{l+p}, e_{l+2p}, \dots, e_{l+(m-1)p},$$

where $0 \leq l < p$, $m = \lceil (n-l)/p \rceil$, and n is the length of S . For example, if $S = \text{abcabbabdb}$, then $\pi_{4,1}(S) = \text{bbb}$, and $\pi_{3,0}(S) = \text{aaab}$. Intuitively, the ratio of the number of occurrences of a symbol s in a certain projection $\pi_{p,l}(S)$ to the length of this projection indicates how often this symbol occurs every p timestamps. This ratio, however, is not quite accurate since it captures all the occurrences including the outliers. In the example above, $\pi_{3,0}(S) = \text{aaab}$ implies that the symbol b is periodic with a period of length 3

with a frequency of $1/4$, which is not quite true. As another example, if for a certain S , $\pi_{p,l}(S) = \text{abcbac}$, then the symbol changes every p timestamps and hence no symbol should be considered periodic with a period of length p . We remedy this problem by considering only the consecutive occurrences. A consecutive occurrence of a symbol s in a certain projection $\pi_{p,l}(S)$ indicates that the symbol s has reappeared in S after p timestamps from the previous appearance. Let $\mathcal{F}_2(s, S)$ denote the number of times the symbol s occurs in two consecutive positions in the data stream S . For example, if $S = \text{abbaaabaa}$, then $\mathcal{F}_2(\text{a}, S) = 3$ and $\mathcal{F}_2(\text{b}, S) = 1$. Therefore, the ratio of the number of consecutive occurrences of a symbol s in a certain projection $\pi_{p,l}(S)$ to the length of this projection ($\frac{\mathcal{F}_2(s, \pi_{p,l}(S))}{\lceil (n-l)/p \rceil - 1}$) indicates how often the symbol s occurs every p timestamps in a data stream S .

Definition 1 *If a data stream S of length n contains a symbol s such that $\exists l, p$ where $0 \leq l < p$, and $\frac{\mathcal{F}_2(s, \pi_{p,l}(S))}{\lceil (n-l)/p \rceil - 1} \geq \tau$ where $0 \leq \tau \leq 1$; then s is said to be periodic in S with a period of length p at position l with respect to periodicity threshold τ .*

For example, in the data stream $S = \text{abcabbabdb}$, $\frac{\mathcal{F}_2(\text{a}, \pi_{3,0}(S))}{\lceil 10/3 \rceil - 1} = 2/3$, thus the symbol a is periodic with a period of length 3 at position 0 with respect to a periodicity threshold $\tau \leq 2/3$. Similarly, the symbol b is periodic with a period of length 3 at position 1 with respect to a periodicity threshold $\tau \leq 1$.

The main advantage of Definition 1 is that not only does it determine the candidate periodic symbols and their corresponding periods, but also it locates their corresponding positions. In other words, each symbol that exhibits periodicity according to Definition 1 produces a frequent single-symbol periodic pattern.

Definition 2 *In a data stream S of a finite alphabet Σ , a pattern of length p is a sequence $q = q_0 \cdots q_{p-1}$, such that $q_i \subseteq \Sigma$.*

Definition 3 *If a data stream S of length n contains a symbol s that is periodic with a period of length p at position l , then a frequent single-symbol periodic pattern $q = q_0 \cdots q_{p-1}$ of length p is formed by setting all q_i 's to the empty set except for q_l that is set to $\{s\}$.*

For example, in the data stream $S = \text{abcabbabdb}$, for a periodicity threshold that is less than $2/3$, the pattern⁴ $\text{a} * *$ is a frequent single-symbol periodic pattern of length 3, and so is the single-symbol periodic pattern $*\text{b}*$.

⁴For simplicity, the symbol $*$ denotes the empty set, and the set brackets are omitted when any q_i is a singleton (contains only one element). Note that $*$ is a place-holder for only one element.

As will be described in Section 4, frequent single-symbol periodic patterns are used to form a maximal periodic pattern that is the root node of the max-subpattern tree of the corresponding period length.

Given these definitions, in the next sections, we discuss how STAGGER detects all the potential period lengths and all the corresponding frequent single-symbol periodic patterns in one pass over the data stream. At the core of STAGGER is a symbol periodicity detection algorithm [4], called SPD, that deals with data stream portions (windows) of known lengths. We present the SPD algorithm (Section 3.2) partly to render this paper self-contained, but also to facilitate introducing the multiple sliding windows online incremental technique for infinite data streams (Section 3.3).

3.2. The SPD Algorithm

Assume that the period length p is known for some symbols of a specific portion, say S , of a data stream. Then, the problem is reduced to mining S for the frequent single-symbol periodic patterns of period length p . In other words, the problem is to detect the symbols that are periodic with period length p within S . A way to solve this simpler problem is to shift S by p positions, denoted as $S^{(p)}$, and then compare $S^{(p)}$ to S . For example, if $S = \text{abcabbabcb}$, then shifting S three positions results in $S^{(3)} = **\text{abcabba}$. Comparing S to $S^{(3)}$ results in four symbol matches. If the symbols are mapped in a particular way, we can deduce that these four matches are actually two for the symbol a and two for the symbol b .

The SPD algorithm [4] relies on two main ideas to detect symbol periodicities in fixed-length data stream portions. The first idea is to use the concept of convolution in order to shift and compare the data stream portion for all possible values at once. The second idea is to obtain a mapping scheme for the symbols, which reveals, upon comparison, the identity of which symbols that match and their corresponding positions. The remaining part of this section describes these ideas in detail. The convolution is covered in Section 3.2.1 while the mapping scheme is covered in Section 3.2.2.

3.2.1. Convolution

A convolution [2] is defined as follows. Let $X = [x_0, x_1, \dots, x_{n-1}]$ and $Y = [y_0, y_1, \dots, y_{n-1}]$ be two finite length sequences of numbers⁵, each of length n . The convolution of X and Y is defined as another finite length sequence $X \otimes Y$ of length n such that

$$(X \otimes Y)_i = \sum_{j=0}^i x_j y_{i-j}$$

⁵The general definition of convolution does not assume equal-length sequences. We adapt the general definition to conform to our problem, in which convolutions only take place between equal-length sequences.

for $i = 0, 1, \dots, n-1$. Let $X' = [x'_0, x'_1, \dots, x'_{n-1}]$ denote the reverse of the vector X , i.e., $x'_i = x_{n-1-i}$. Taking the convolution of X' and Y , and obtaining its reverse leads to the following:

$$(X' \otimes Y)'_i = (X' \otimes Y)_{n-1-i} = \sum_{j=0}^{n-1-i} x'_j y_{n-1-i-j} = \sum_{j=0}^{n-1-i} x_{n-1-j} y_{n-1-i-j},$$

i.e.,

$$\begin{aligned} (X' \otimes Y)'_0 &= x_0 y_0 + x_1 y_1 + \dots + x_{n-1} y_{n-1}, \\ (X' \otimes Y)'_1 &= x_1 y_0 + x_2 y_1 + \dots + x_{n-1} y_{n-2}, \\ &\vdots \\ (X' \otimes Y)'_{n-1} &= x_{n-1} y_0. \end{aligned}$$

In other words, the component of the resulting sequence at position i corresponds to shifting one of the input sequences i positions and comparing it to the other input sequence.

The SPD algorithm performs the following steps: (i) Convert the data stream portion S into two finite sequences of numbers $\Phi(S)$ and $\Phi(S)'$, where $\Phi(S)'$ is the reverse of $\Phi(S)$ (based on the mapping scheme Φ described in Section 3.2.2), (ii) Perform the convolution between the two sequences $\Phi(S)' \otimes \Phi(S)$, and (iii) Reverse the output $(\Phi(S)' \otimes \Phi(S))'$. The component values of the resulting sequence correspond to shifting and comparing the data stream portion for all possible values.

It is well known that convolution can be computed by the fast Fourier transform (FFT) [8] as follows:

$$X \otimes Y = \text{FFT}^{-1}(\text{FFT}(X) \cdot \text{FFT}(Y)).$$

This computation reduces the time complexity of the convolution to $O(n \log n)$. The brute-force approach of shifting and comparing the data stream portion for all possible values has the time complexity $O(n^2)$.

3.2.2. Mapping Scheme

Let $S = e_0, e_1, \dots, e_{n-1}$ be a data stream portion of length n , where e_i 's are symbols from a finite alphabet Σ of size σ . Let Φ be a mapping for the symbols of S such that $\Phi(S) = \Phi(e_0), \Phi(e_1), \dots, \Phi(e_{n-1})$. Let $C(S) = (\Phi(S)' \otimes \Phi(S))'$, and $c_i(S)$ be the i th component of $C(S)$. The mapping scheme of the SPD algorithm satisfies two conditions: (i) While matched symbols contribute a non-zero value in the product $\Phi(e_j) \cdot \Phi(e_{i-j})$, unmatched symbols contribute 0, and (ii) the value of each component of $C(S)$, $c_i(S) = \sum_{j=0}^{n-1-i} \Phi(e_{n-1-j}) \cdot \Phi(e_{n-1-i-j})$, identifies the matched symbols and their corresponding positions.

The SPD algorithm maps the symbols to the binary representation of increasing powers of 2. For example, if a data stream contains only the 3 symbols a, b, and c, then a possible mapping would be a : 001, b : 010, and c : 100,

which correspond to power values of 0, 1, and 2, respectively. Hence, a data stream portion of length n is converted to a binary vector of length σn . For example, let $S = \text{accabb}$, then S is converted to the binary vector $\bar{S} = 001100100100001010010$. Using convolution results in a sequence $C(\bar{S})$ of length σn . Considering just the n positions $0, \sigma, 2\sigma, \dots, (n-1)\sigma$ that are the exact start positions of the symbols, gives back the sequence $C(S)$. The latter derivation of $C(S)$ can be written as $C(S) = \pi_{\sigma,0}(C(\bar{S}))$ using the notation defined in Section 3.1.

The SPD algorithm modifies the definition of convolution as follows:

$$(X \otimes Y)_i = \sum_{j=0}^i 2^j x_j y_{i-j}.$$

Notice that this definition still preserves that

$$X \otimes Y = \text{FFT}^{-1}(\text{FFT}(X) \cdot \text{FFT}(Y)).$$

The reason for adding the coefficient 2^j is to get a different contribution for each match. For example, Figure 2 illustrates an example where $S = \text{accabb}$. The powers of 2 in the value of $c_1(S)$ are 1, 11, and 14. Examining those powers modulo 3, which is the size of the alphabet in this particular example, results in 1, 2 and 2, respectively, which correspond to the symbols b, c, and c, respectively.

\bar{T} :	0	0	1	1	0	0	1	0	0	1	0	0	0	0	1	0	1	0	0	1	0	
$\bar{T}^{(3)}$:			0	0	1	1	0	0	1	0	0	1	0	0	0	0	1	0	1	0	1	0
$c_3(\bar{T}) = c_1(T) =$																						
\bar{T} :	0	0	1	1	0	0	1	0	0	1	0	0	0	0	1	0	1	0	0	1	0	0
$\bar{T}^{(12)}$:																						
$c_{12}(\bar{T}) = c_4(T) =$																						

Figure 2. A clarifying example for the mapping scheme

Figure 2 gives another example for $c_4(S)$ that has only one power of 2, which is 6, that corresponds to the symbol a since $6 \bmod 3 = 0$ and a was originally mapped to the binary representation of 2^0 . This means that comparing S to $S^{(4)}$ results in only one match of the symbol a. Moreover, the power value of 6 reveals that the symbol a is at position 0 in $S^{(4)}$. Therefore, the power values reveal not only the number of matches of each symbol at each period, but also the corresponding starting positions. This latter observation complies with the definition of symbol periodicity. Hereby, we direct the reader to [4] for a complete discussion of that algorithm.

3.3. Online Symbol Periodicity Detection

The idea of STAGGER is to perform the SPD algorithm incrementally on multiple sliding windows of expanding

bounded by the buffer size allowed by the system for buffering the data stream. Therefore, $w_m/2$ is the maximum period length value that we can discover.

4. Mining Periodic Patterns

The max-subpattern tree [6] has proved to be efficient in mining periodic patterns in time series data. Han et al. [6] have presented a two-pass algorithm for building such a tree. Aref et al. [1] have presented an incremental version of the algorithm that maintains the max-subpattern tree during continuously arriving data. However, the algorithm in [1] requires two passes over the new data and a possible reprocessing of the previously seen data. In this section, we propose a new incremental technique that fits the data stream model. Data is not stored and hence a reprocessing of the previously seen data is not possible. The produced results are approximate compared to the exact results produced by the two-pass algorithms [6, 1]. Yet, this approximation is unavoidable and is empirically reasonable as the accuracy exceeds 90%.

We give an overview of the max-subpattern tree [6] (Section 4.1). Then, we introduce how STAGGER maintains the max-subpattern tree incrementally over a data stream (Section 4.2). Subsequently, we introduce the ‘‘hysteresis’’ approach for maintaining the periodicity thresholds over the data stream (Section 4.3).

4.1. The Max-Subpattern Tree

Figure 3 gives an example of a max-subpattern tree built with a period of length 4. Every node represents a candidate periodic pattern and has a count that reflects the number of occurrences of this pattern in the data. Recall that, according to Definition 2, a pattern is a sequence of subsets over the entire alphabet. A node is a parent to another if the pattern of the latter node is a subpattern of the pattern of the former node. A pattern q' is called a subpattern of another pattern q if for each position i , $q'_i \subseteq q_i$. The direct link between a parent node and a child node is labeled by the difference between their two patterns. Notice that a pattern may be a subpattern of more than one pattern. Therefore, the data structure is actually a graph rather than a tree. In order to preserve the data structure as a tree and decrease the complexity of insertion and search, not all the parent-child links are kept. The dotted lines in Figure 3 represent those links that are not kept.

Clearly, the root node of the max-subpattern tree should represent the candidate periodic pattern that all the other candidate periodic patterns are subpatterns of. Hence, the root node pattern is called the *maximal* periodic pattern Q . The algorithm of [6] determines Q by extracting all the *single-symbol* patterns of length p from a first pass over the data. The frequent single-symbol patterns are determined with respect to the periodicity threshold, and Q is computed by the union operation of all these frequent single-symbol patterns. For example, if the frequent single-symbol periodic patterns of length 4 are $*b**$, $*c**$, $a***$, and $**d*$, then $Q = a\{b, c\}d*$.

A second pass over the data combines every p symbols and inserts the formed pattern, filtered by the maximal periodic pattern, into the max-subpattern tree. Once it is built, the max-subpattern tree is traversed in order to detect those actual frequent periodic

patterns with respect to the periodicity threshold. Note that the count of every pattern is actually the count of its corresponding node plus all the counts of its parent nodes (both the direct parent and the other hidden parents).

Note that in STAGGER, that first pass is not conducted. The periodicity detection technique (Section 3) reveals not only the potential period lengths but also their single-symbol periodic patterns that form their corresponding maximal periodic patterns.

4.2. Approximate Incremental Technique

The idea of our proposed technique for mining periodic patterns in data streams is to maintain the max-subpattern tree over the continuous arrival of new data. The arrival of new data may update the maximal periodic pattern Q due to the discovered single-symbol periodic patterns. Accordingly, the max-subpattern tree should be updated. Let Q and Q' be the maximal periodic patterns before and after the update, respectively. Let c_j and c'_j be the components at position j of Q and Q' , respectively. If $c'_j \neq c_j$, then updating c_j to c'_j implies deletion and/or insertion of one or more symbols. For example, if $Q = a\{b, c\}d*$ and $Q' = a\{b, e\}df$, then updating Q to Q' implies deleting the symbol c from position 2, and inserting the symbols e and f at positions 2 and 4, respectively.

STAGGER updates the max-subpattern tree in two steps: a deletion step followed by an insertion step. Let Q^t be the pattern resulting from deleting the necessary symbols from Q , e.g., in the previous example, deleting the symbol c from position 2 results in $Q^t = abd*$. If the original max-subpattern tree R contains a node representing the pattern Q^t , then STAGGER converts that node to be the root of the updated max-subpattern tree. Otherwise, STAGGER creates a new root node representing the pattern Q^t . There are two aspects concerning that intermediate resulting tree. First, the counts must be fixed since some deleted nodes were parents to some retained nodes (e.g., in Figure 3, deleting the root node means a missing count of 10 for all its children nodes). Second, the non-linked children from R should be added. To consider both of these two aspects simultaneously, STAGGER scans the original max-subpattern tree R and inserts R 's patterns into the intermediate max-subpattern tree with their corresponding counts. For example, in Figure 3, where $Q = a\{b, c\}d*$, assume that $Q' = a\{b, e\}df$, then $Q^t = abd*$. The intermediate max-subpattern tree is given in Figure 4(a). Notice that, without fixation, the intermediate max-subpattern tree would have had only two nodes: one for the pattern $abd*$ with a count of 40, and one for the pattern $ab**$ with a count of 2.

In the insertion step, STAGGER creates a new root node for the updated max-subpattern tree that represents the updated maximal periodic pattern Q' . Nevertheless, the updated max-subpattern tree is not accurate since it has no information about those inserted symbols. In other words, the counts of the newly created nodes along the path from the new root to the old root are set to 0. The only way to have their real counts is to reprocess the previous data, which is not possible. This is why STAGGER is considered an approximate algorithm. For example, Figure 4(b) shows the max-subpattern tree updated from Figure 4(a) to have $Q' = a\{b, e\}df$ as its root node. Although the previously seen data might contain patterns that had the symbol f in position 4, the fact that this sym-

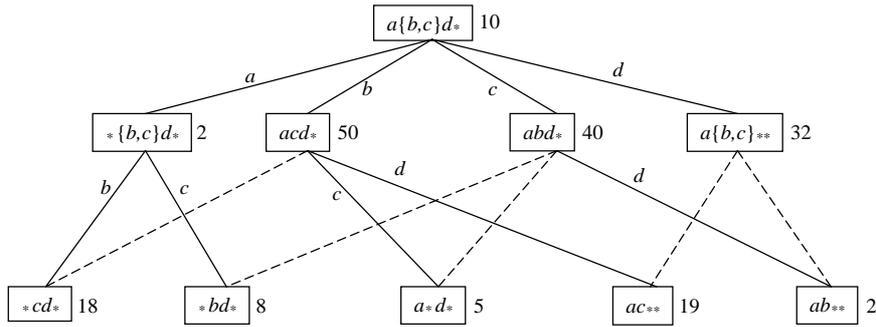
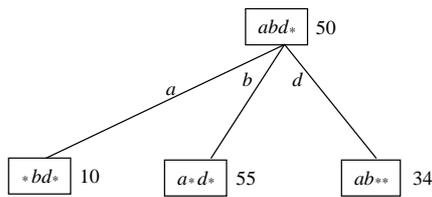
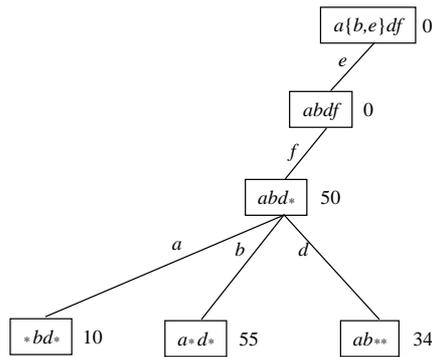


Figure 3. An example of a max-subpattern tree



(a) After deleting the symbol c



(b) After inserting the two symbols e and f

Figure 4. Updating the max-subpattern tree

bol was not frequent before had prohibited us from including any pattern containing this symbol in the max-subpattern tree.

Now that the max-subpattern tree is updated to reflect the change of the maximal periodic pattern, the newly arrived data patterns are inserted into the updated tree. Thus, the max-subpattern tree now represents approximately the whole data stream.

4.3. The “Hysteresis” Thresholding Approach

The proposed technique, as well as any traditional data mining technique, handles the patterns as follows. As long as a pattern, say q , is not frequent, q is not included in the max-subpattern tree. As soon as q becomes frequent, it is added to the max-subpattern

tree, and its history starts. As long as q is frequent, it is kept in the max-subpattern tree updating its history. As soon as q becomes infrequent, it is removed from the max-subpattern tree losing its history information. The frequency of the pattern is determined by a single threshold. This approach is illustrated in Figure 5(a). A disadvantage of this approach is that it fails to handle a pattern that oscillates between being frequent and being infrequent. Such a pattern, say q , will lose all the history information (max-subpattern tree counts) as soon as it becomes infrequent. When q becomes frequent again, it will be treated as a newly appeared frequent pattern which has no history information.

One can think of several approaches for overcoming that disadvantage. One of which is to not remove an infrequent pattern from the max-subpattern tree as it may become frequent later. This approach, however, suffers the disadvantage that the max-subpattern tree will be huge and will contain several infrequent patterns, especially when a data stream has a rather lengthy transient component. Another approach is to have another max-subpattern tree for those patterns that are infrequent but are candidates to be frequent later. Those patterns are determined by a lower threshold value than the original one. Yet, this approach migrates the problem to that other max-subpattern tree, which will suffer the same disadvantage with a lower threshold value.

We propose a tradeoff approach whose idea comes from the Physics domain. Hysteresis represents the *history* dependence of physical systems⁶. The proposed approach utilizes two thresholds, as illustrated in Figure 5(b). A pattern is considered frequent if its frequency is above the higher threshold value, and is considered infrequent if its frequency is below the lower threshold value. All the patterns whose frequencies are above the lower threshold value are kept in the max-subpattern tree. However, only those whose frequencies are above the higher threshold value are considered frequent patterns. In other words, The max-subpattern tree is built using the lower threshold value, yet the frequent periodic patterns are extracted using the higher threshold value. The distance between the two threshold values acts as a period of time during which a pattern is given a chance to stabilize rather than to oscillate.

⁶If you push on something, it will yield, and when you release, if it does not spring back completely, then it is exhibiting hysteresis.

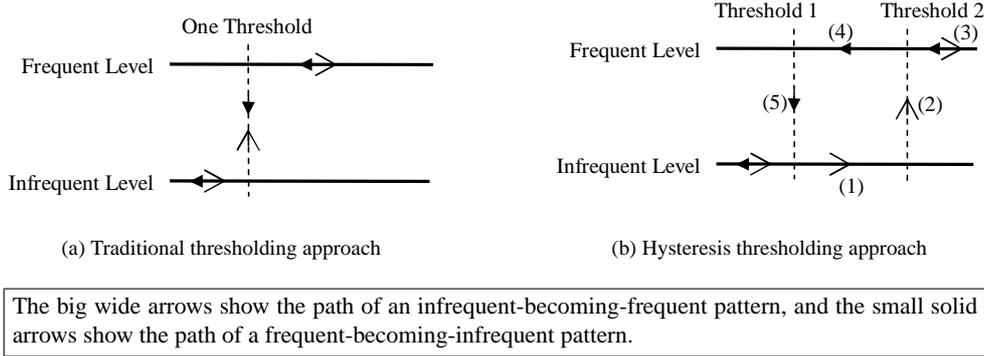


Figure 5. Hysteresis versus traditional thresholding approaches

5. Experimental Study

This section contains the results of an experimental study that examines STAGGER for different aspects. In Section 5.1, experiments are conducted using synthetic data in order to examine the accuracy and the time performance of STAGGER. In Section 5.2, the practicality and usefulness of the output are explored using real data experiments.

We generate controlled synthetic data by tuning some parameters, namely, data distribution, period length, alphabet size, and noise amount. Both uniform and normal data distributions are considered. First, inerrant data is generated by repeating a partial pattern that is randomly generated from the specified data distribution. Then, noise is introduced randomly and uniformly over the whole data stream. Noise is introduced by altering a symbol by another, inserting a new symbol, or deleting the current symbol at a randomly selected position in the data stream. Unless stated otherwise, data streams lengths of 1M symbols are used with alphabet size of 10, and the values collected are averaged over 100 runs. With respect to the period lengths, we consider values that divide the whole data stream length as well as values that do not. We thus inspect STAGGER bias, if any, towards any particular set of period length values.

STAGGER is the first one-pass online incremental periodicity mining algorithm for data streams that combines both periodicity detection and periodic patterns mining. Hence, there is no direct comparison between STAGGER and any of the algorithms in the literature. However, we compare our periodicity detection technique to the periodicity detection algorithm, AWSOM [11]. AWSOM uses Wavelets to approximate data streams and hence discovers potential periodicity rates.

5.1. Synthetic Data Experiments

The first experiment studies the accuracy of the proposed incremental technique for mining periodic patterns. The fact that STAGGER is approximate in building the max-subpattern tree in one pass implies how the accuracy is measured. The approximate tree is compared to the accurate tree that is built offline in a two-pass algorithm [6]. The relative error of each node count in the tree is computed and averaged over the total number of nodes in

the tree. In Figure 6, we use the symbols “U” and “N” to denote the uniform and the normal distributions, respectively; and the symbol “P” to denote the embedded period length.

When a single sliding window is used, Figure 6(a) shows expectedly that increasing the window length increases the accuracy since the effect of transient frequent periodic patterns is reduced. Furthermore, as the stream data continuously arrives, the accuracy increases, as shown in Figure 6(b), since the data becomes progressively more stable. Therefore, the frequent periodic patterns are determined and the probability of adding new frequent periodic patterns is very low. Both Figures 6(a) and 6(b) show that STAGGER is highly accurate (in the 90% level).

Figure 6(c) gives the results of inspecting the “hysteresis” thresholding approach by varying the distance between the two threshold values. A zero distance means only one threshold value, that is the traditional thresholding approach. Figure 6(c) shows that the “hysteresis” thresholding approach increases the accuracy of the approximate max-subpattern tree. As the distance between the two threshold values increases, the accuracy increases since we allow more time for a pattern not to lose its history (max-subpattern tree counts). After some distance level, the accuracy does not increase more since the allowed time is more than enough for all patterns to keep their history.

The next experiment inspects the accuracy of STAGGER with respect to the discovered potential period lengths. The accuracy measure that we use is the ability of STAGGER to detect all the period lengths that were artificially embedded into the synthetic data. To discover a period length accurately, it is not enough to discover it at any periodicity threshold value. In other words, the period lengths discovered with a high periodicity threshold value are better candidates than those discovered with a lower periodicity threshold value. Therefore, we define the confidence of a discovered period length to be the minimum periodicity threshold value required to detect this period length. The accuracy is measured by the average confidence of all the period lengths that are embedded artificially into the synthetic data. Figure 7 gives the results of this experiment. Recall that synthetic data is generated such that the embedded period lengths are: $P, 2P, \dots$

Using a single sliding window, Figure 7(a) shows that increasing the window length increases the accuracy since this allows the algorithm to detect more period lengths (of larger lengths). Sub-

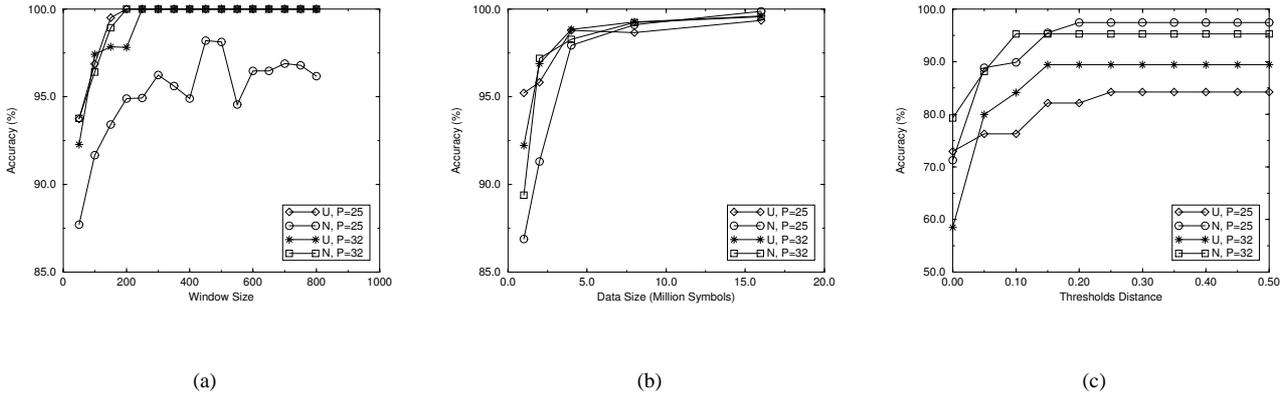


Figure 6. Accuracy of the approximate max-subpattern tree

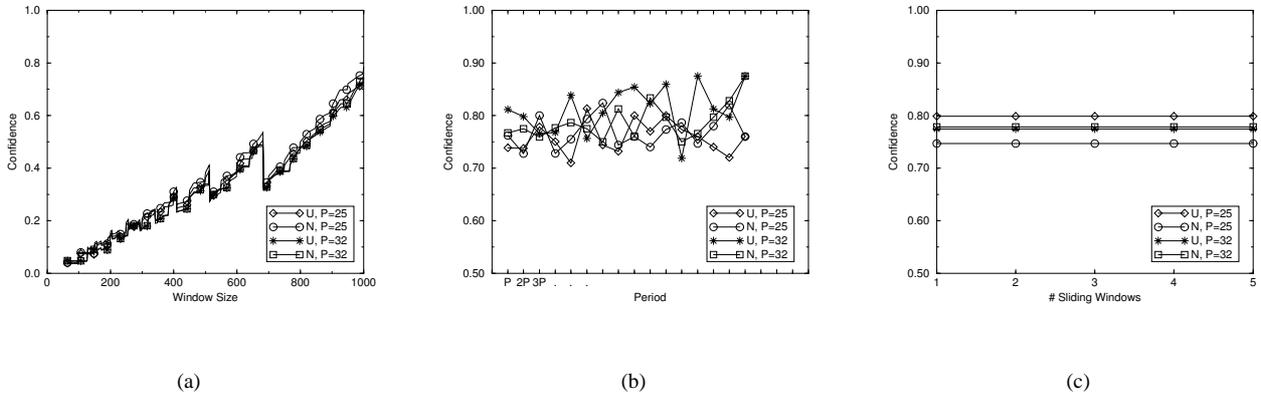


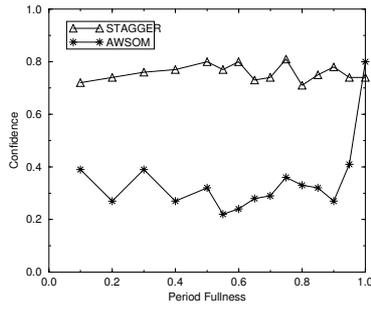
Figure 7. Accuracy of symbol periodicity detection

sequently, using a single sliding window of maximum buffer size, Figure 7(b) shows an unbiased behavior with respect to the embedded period length. Moreover, Figure 7(c) shows that the accuracy does not rely on the number of expanding sliding windows as far as the largest sliding window is selected to be as large as the buffer size in order to detect all possible period lengths. Both Figures 7(b) and 7(c) show that STAGGER achieves an average of 80% accuracy, i.e., STAGGER detects all the artificially embedded period lengths at a periodicity threshold value of 80%.

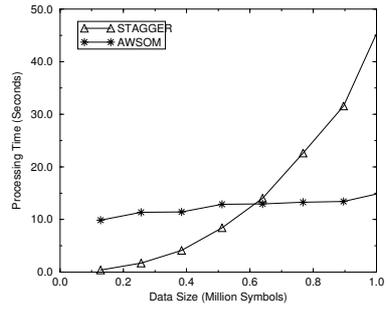
To compare STAGGER to the periodicity detection algorithm of [11] (AWSOM), we consider both the time performance and the accuracy of the discovered potential periodicity rates. AWSOM has two main limiting features. First, AWSOM discovers only periodicity rates of lengths that are powers of two. This limitation is due to the use of Wavelets to approximate the data stream. Periodicity rates of lengths that are not powers of two will be missed by AWSOM. The second limitation of AWSOM is that it discovers periodicities that are complete, e.g., full sinusoidal periods. If a period has only few periodic symbols, AWSOM fails to discover

such a period. On the other hand, STAGGER does not suffer either disadvantage as (i) it uses convolution to consider all possible period lengths, and (ii) it defines the periodicity in terms of its symbols. Figure 8(a) shows empirically this latter property. The experiment compares STAGGER to AWSOM based on their ability to discover semi-full periods. The experiment considers only periodicity rates of powers of two. Figure 8(a) shows that both algorithms achieve 80% accuracy at a full period (1.0 period fullness). However, for semi-full periods, STAGGER outperforms AWSOM with an order of magnitude.

On the other hand, Figure 8(b) shows that AWSOM outperforms STAGGER with respect to the processing time. AWSOM is a Wavelet-based algorithm that takes $O(n)$ time to compute. However, STAGGER takes $O(n \log n)$ time to compute the convolution. In conclusion, STAGGER trades processing time for more accurate periodicity rates, more general periodicities in terms of symbols, and more discovered information (the periodic patterns themselves). In contrast, AWSOM trades accuracy for processing time by the Wavelet approximation that may lead to discarding

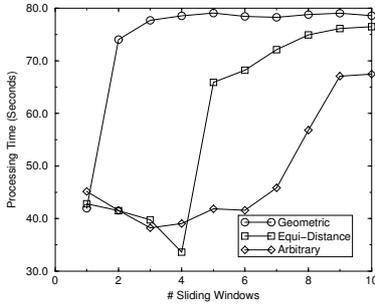


(a) Accuracy

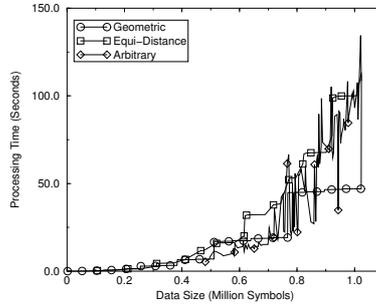


(b) Time Performance

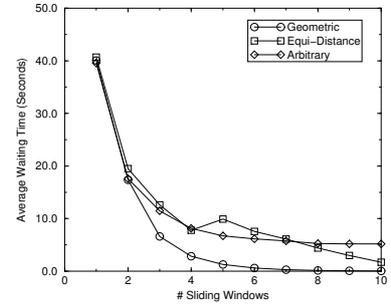
Figure 8. STAGGER versus AWSOM



(a)



(b)



(c)

Figure 9. Time performance of STAGGER

potential periodicities.

To evaluate the time performance of STAGGER, we observe that the number of expanding sliding windows is the key factor of the processing time, as depicted in Figure 9(a). We have three different settings for the lengths of the expanding sliding windows: arbitrary, geometric, and equi-distance settings. Let m be the number of expanding sliding windows of lengths $w_1 < w_2 < \dots < w_m$ such that w_m is selected to be as large as the buffer size. For all $i = 1, 2, \dots, m - 1$, while the arbitrary setting selects w_i randomly, the geometric setting selects w_i to equal $w_{i+1}/2$, and the equi-distance setting selects w_i to equal $(i/m)w_m$. Figure 9(a) shows that the arbitrary setting gives the least processing time while the geometric setting gives the most processing time.

Figure 9(b) shows that the processing time of both the geometric and the equi-distance settings increases gradually over time (as the data stream increases in size), whereas the processing time of the arbitrary setting increases irregularly. The importance of the gradual increase is that it gives a bound on the data arrival rate that STAGGER can manage without dropping any data sym-

bols. For example, Figure 9(b), drawn with 10 expanding sliding windows, shows that STAGGER with the geometric setting has processed 1M symbols in approximately 100 seconds. This means that STAGGER can cope with an arrival rate of 10,000 symbols per second. Of course, this high rate is due to the high speed processor we used in our experiments.

The most important advantage of having expanding sliding windows is shown in Figure 9(c), that is a much better average waiting time over a single sliding window. Clearly, the more the number of expanding sliding windows, the less average waiting time between consecutive outputs. Moreover, Figure 9(c) shows that the geometric setting of the expanding sliding windows has the best performance with respect to the average waiting time.

5.2. Real Data Experiments

A real-world database that contains sanitized data of timed sales transactions for some stores over a period of 15 months serves the purpose of real data experiments. Timed sales transactions data, of size 130 Megabytes is streamed to STAGGER. AI-

