Efficient Periodicity Mining in Time Series Databases Using Suffix Trees

Faraz Rasheed, Mohammed Alshalalfa, and Reda Alhajj, *Associate Member*, *IEEE*

Abstract—Periodic pattern mining or periodicity detection has a number of applications, such as prediction, forecasting, detection of unusual activities, etc. The problem is not trivial because the data to be analyzed are mostly noisy and different periodicity types (namely symbol, sequence, and segment) are to be investigated. Accordingly, we argue that there is a need for a comprehensive approach capable of analyzing the whole time series or in a subsection of it to effectively handle different types of noise (to a certain degree) and at the same time is able to detect different types of periodic patterns; combining these under one umbrella is by itself a challenge. In this paper, we present an algorithm which can detect symbol, sequence (partial), and segment (full cycle) periodicity in time series. The algorithm uses suffix tree as the underlying data structure; this allows us to design the algorithm such that its worst-case complexity is $O(k.n^2)$, where k is the maximum length of periodic pattern and n is the length of the analyzed portion (whole or subsection) of the time series. The algorithm is noise resilient; it has been successfully demonstrated to work with replacement, insertion, deletion, or a mixture of these types of noise. We have tested the proposed algorithm on both synthetic and real data from different domains, including protein sequences. The conducted comparative study demonstrate the applicability and effectiveness of the proposed algorithm; it is generally more time-efficient and noise-resilient than existing algorithms.

Index Terms—Time series, periodicity detection, suffix tree, symbol periodicity, segment periodicity, sequence periodicity, noise resilient.

1 Introduction

A time series is a collection of data values gathered generally at uniform interval of time to reflect certain behavior of an entity. Real life has several examples of time series such as weather conditions of a particular location, spending patterns, stock growth, transactions in a superstore, network delays, power consumption, computer network fault analysis and security breach detection, earthquake prediction, gene expression data analysis [1], [10], etc. A time series is mostly characterized by being composed of repeating cycles. For instance, there is a traffic jam twice a day when the schools are open; number of transactions in a superstore is high at certain periods during the day, certain days during the week, and so on. Identifying repeating (periodic) patterns could reveal important observations about the behavior and future trends of the case represented by the time series [33], and hence would lead to more effective decision making. In other words, periodicity detection is a process for finding temporal regularities within the time series, and the goal of analyzing a time series is to find whether and how frequent a periodic pattern (full or partial) is repeated within the series.

- *F. Rasheed and M. Alshalalfa are with the Department of Computer Science, University of Calgary, Calgary, Alberta, Canada T2N1N4. E-mail: {frasheed, msalshal}@ucalgary.ca.*
- *R. Alhajj is with the Department of Computer Science, University of Calgary, Calgary, Alberta, Canada T2N1N4, and the Department of Computer Science, Global University, Beirut, Lebanon. E-mail: alhajj@ucalgary.ca.*

Manuscript received 23 Nov. 2008; revised 19 July 2009; accepted 11 Oct. 2009; published online 28 Apr. 2010.
Recommended for acceptance by J. Li.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2008-11-0618.
Digital Object Identifier no. 10.1109/TKDE.2010.76.

A time series is mostly discretized [18], [20], [6], [7], [14] before it is analyzed. Let $T = e_0, e_1, e_2, \ldots, e_{n-1}$ be a time series having n events, where e_i represents the event recorded at time instance i; time series T maybe discretized by considering m distinct ranges such that all values in the same range are represented by one symbol taken from an alphabet set, denoted by $\sum$. For example, consider the time series containing the hourly number of transactions in a superstore; the discretization process may define the following mapping by considering different possible ranges of transactions; {0} transactions: a, {1-200} transactions: b, {201-400} transactions: c, {401-600} transactions: d, $\{>600\}$ transactions: e. Based on this mapping, the time series $T = 243, 267, 355, 511, 120, 0, 0, 197$ can be discretized into $T' = cccdbaab$.

In general, three types of periodic patterns can be detected in a time series: 1) symbol periodicity, 2) sequence periodicity or partial periodic patterns, and 3) segment or full-cycle periodicity. A time series is said to have symbol periodicity if at least one symbol is repeated periodically. For example, in time series $T = abd\ acb\ aba\ abc$, symbol a is periodic with periodicity $p = 3$, starting at position zero ($stPos = 0$). Similarly, a pattern consisting of more than one symbol maybe periodic in a time series; and this leads to *partial periodic patterns*. For instance, in time series $T = bbaa\ abbd\ abca\ abbc\ abcd$, the sequence ab is periodic with periodicity $p = 4$, starting at position 4 ($stPos = 4$); and the partial periodic pattern $ab**$ exists in T, where $*$ denotes *any* symbol or *don't care* mark. Finally, if the whole time series can be mostly represented as a repetition of a pattern or segment, then this type of periodicity is called segment or full-cycle periodicity. For instance, the time series $T = abcab\ abcab\ abcab$ has segment periodicity of

5 ($p = 5$) starting at the first position ($stPos = 0$), i.e., T consists of only three occurrences of the segment $abcab$.

It is not necessary to always have perfect periodicity in a time series as in the above three examples. Usually, the degree of perfection is represented by confidence, which is 100 percent in the above three examples. On the other hand, real-life examples are mostly characterized by the presence of noise in the data and this negatively affects the confidence level. The confidence of a pattern is defined as the ratio of its actual frequency in the series over its expected perfect frequency in the series. The actual and expected perfect frequency are both the same in the the above three examples; however, in the time series $T_x = abefd\ abcde\ acbcd\ abefa$, the pattern ab starting at position 0 with $p = 5$ has four and five as its actual and expected perfect frequencies, respectively; so the confidence of ab is $\frac{4}{5}$.

In addition to detecting periodicity in the full time series, seeking periodicity only in a subsection of the series is also possible. For instance, in time series $T = bbcdbabababababccdaccab$, the sequence ab is periodic starting from positions 5 to 12. This type of periodicity is very common in real life. For example, there might be a specific periodic pattern of road traffic during school days, but not during summer or winter vacation. Unfortunately, most of the existing algorithms [6], [7], [14], [16] only detect periods that span through the entire time series.

To incorporate all of the above factors, a period in a time series maybe represented by 5-tuple (S, p, $stPos$, $endPos$, $Conf$), where S and $endPos$ denote, respectively, the periodic pattern and the end position of the section to be analyzed (it is set to be the length of the series in case all of the series should be analyzed). For instance, (ab, 2, 5, 12, 1) represents a period in $T = bbcdbabababababccdaccab$; here, it is worth noting that the confidence is one because the analysis considers only subsection of the series, from positions 5 to 12. Both (ab, 5, 0, 16, 0.8) and (a, 5, 0, 15, 1) represent periods in $T = abefd\ abcde\ acbcd\ abefa$.

To sum up, time series do exist frequently in our daily life and their analysis could lead to valuable discoveries. However, the problem is not trivial and none of the approaches described in the literature is comprehensive enough to handle all the related aspects. In other words, we argue the need to develop a noise-resilient algorithm that could tackle the problem well by being capable of: 1) identifying the three different types of periodic patterns, 2) handling asynchronous periodicity by locating periodic patterns that may drift from their expected positions up to an allowable limit, and 3) investigating periodic patterns in the whole time series as well as in a subsection of the time series. Realizing the significance of this problem, we incrementally developed an algorithm that covers all of these aspects. At each step in the development process, we conducted comprehensive testing of the algorithm before new features are incorporated. Our initial findings, as described in [23], [24], [25], have been articulated into a version of the algorithm that covers the first two points mentioned above. In addition to these features, the algorithm proposed in this paper can detect periodic patterns found in subsections of the time series, prune redundant periods, and follow various optimization strategies; it is also applicable to biological data sets and has been analyzed for time performance and space consumption.

We present an efficient algorithm that uses suffix tree as the underlying data structure to detect all the above-mentioned three types of periodicity in a single run. The algorithm is noise-resilient and works with replacement, insertion, deletion, or any mixture of these types of noise [24]. It can also detect periodicity within a subsection of a time series and applies various redundant period pruning techniques to output a small number of useful periods by removing most of the redundant periods. The algorithm looks for all periods starting from all positions which have confidence greater than or equal to the user-provided periodicity threshold. The worst time complexity of the algorithm is $O(k.n^2)$. Finally, the different aspects of the algorithm, its applicability, and effectiveness have been demonstrated using both real and synthetic data.

Contributions of our work can be summarized as follows:

1. the development of suffix-tree-based comprehensive algorithm that can simultaneously detect symbol, sequence, and segment periodicity;
2. finding periodicity within subsection of the series;
3. identifying and reporting only useful and nonredundant periods by applying pruning techniques to eliminate redundant periods, if any;
4. detailed algorithm analysis for time performance and space consumption by considering three cases, namely the worst case, the average case, and the best case;
5. a number of optimization strategies are presented; they do improve the running time as demonstrated in the related experimental results, although they do not improve the time complexity;
6. the proposed algorithm is shown to be applicable to biological data sets such as DNA and protein sequences and the results are compared with those produced by other existing algorithms like SMCA [13];
7. various experiments have been conducted to demonstrate the time efficiency, robustness, scalability, and accuracy of the reported results by comparing the proposed algorithm with existing state-of-the-art algorithms like CONV [6], WARP [7], and partial periodic patterns (ParPer) [12].

The rest of the paper is organized as follows: Related work is presented in Section 2. The periodicity detection problem in time series is formulated in Section 3. The proposed algorithm is presented in Section 4 along with algorithm analysis and the utilized optimization strategies. Experimental results are reported in Section 5 using both real and synthetic data. Section 6 is conclusions and future research directions.

2 Related Work

Existing literature on time series analysis roughly covers two types of algorithms. The first category includes algorithms that require the user to specify the period, and then look only for patterns occurring with that period. The second class, on the other hand, are algorithms which look for all possible periods in the time series. Our algorithm

described in this paper could be classified under the second category; it does more than the other algorithms by looking for all possible periods starting from all possible positions within a prespecified range, whether the whole time series or a subsection of the time series.

Another way to classify existing algorithms is based on the type of periodicity they detect; some detect only symbol periodicity, and some detect only sequence or partial periodicity, while others detect only full cycle or segment periodicity. Our single algorithm can detect all the three types of periodicity in one run.

Earlier algorithms, e.g., [14], [21], [34], [2], require the user to provide the expected period value which is used to check the time series for corresponding periodic patterns. For example, in power consumption time series, a user may test for weekly, biweekly, or monthly periods. However, it is usually difficult to provide expected period values; and this approach prohibits finding unexpected periods, which might be more useful than the expected ones. For instance, a fraud maybe detected by spotting unexpected behavior in credit card usage; discovering such unexpected periods out of the scope of this paper, thought it is being addressed as part of our research project.

Recently, there are few algorithms, e.g., [6], [7], [16] which look for all possible periods by considering the range ($2 \leq p \leq \frac{n}{2}$). One of the earliest best known work in this category has been developed by Elfeky et al. [6]. They proposed two separate algorithms to detect symbol and segment periodicity in time series. Their first algorithm (CONV) is based on the convolution technique with reported complexity of $O(nlogn)$. Although their algorithm works well with data sets having perfect periodicity, it fails to perform well when the time series contains insertion and deletion noise. Realizing the need to work in the presence of noise, Elfeky et al. later presented an $O(n^2)$ algorithm (WARP) [7], which performs well in the presence of insertion and deletion noise. WARP uses the time warping technique to accommodate insertion or deletion noise in the data. But, besides having $O(n^2)$ complexity, WARP can only detect segment periodicity; it cannot find symbol or sequence periodicity. Also, both CONV and WARP can detect periodicity which last till the very end of the time series, i.e., they cannot detect patterns which are periodic only in a subsection of the time series.

Sheng et al. [27], [28] developed an algorithm which is based on Han's [12] ParPer algorithm to detect periodic patterns in a section of the time series; their algorithm utilizes optimization steps to find dense periodic areas in the time series. But their algorithm, being based on ParPer, requires the user to provide the expected period value. ParPer runs in linear $O(n)$ time for a given period value, which is very difficult to provide. However, its complexity would increase to $O(n^2)$ time if it is to be augmented to look for all possible periods. Also, ParPer can only detect partial period patterns; i.e., it cannot detect symbol and sequence periodicity.

Recently, Huang and Chang [13] presented their algorithm for finding asynchronous periodic patterns, where the periodic occurrences can be shifted in an allowable range along the time axis. This is somehow similar to how our algorithm deals with noisy data by utilizing the time tolerance window for periodic occurrences.

To sum up, the approaches described in the literature are dedicated to tackle one side of the problem; and each approach incorporates certain techniques. However, our approach described in this paper is capable of reporting all types of periods and could function when noise is present in the data up to a certain level; combining different techniques under one umbrella produced our powerful and effective algorithm.

3 Problem Definition

Consider a time series $T = e_0, e_1, e_2, \ldots, e_{n-1}$ of length n, where e_i denotes the event recorded at time i; and let T be discretized into symbols taken from an alphabet set $\sum$ with enough symbols, i.e., $|\sum|$ represents the total number of unique symbols used in the discretization process. In other words, in a systematic way T can be encoded as a string derived from $\sum$. For instance, the string $abbcabcdcbab$ could represent one time series over the alphabet $\sum = \{a, b, c, d\}$. Researchers have mostly investigated time series to identify repeating patterns and some researchers studied exceptional patterns (outliers) in time series. In this paper, we concentrate on the first case; we need to develop an algorithm capable of detecting in an encoded time series (even in the presence of noise) symbol, sequence, and segment periodicity, which are formally defined next. We start by defining confidence because it is not always possible to achieve perfect periodicity and hence we need to specify the degree of confidence in the reported result.

Definition 1 (Perfect Periodicity). *Consider a time series T, a pattern X is said to satisfy perfect periodicity in T with period p if starting from the first occurrence of X until the end of T every next occurrence of X exists p positions away from the current occurrence of X. It is possible to have some of the expected occurrences of X missing and this leads to imperfect periodicity.*

Definition 2 (Confidence). *The confidence of a periodic pattern X occurring in time series T is the ratio of its actual periodicity to its expected perfect periodicity. Formally, the confidence of pattern X with periodicity p starting at position $stPos$ is defined as:*

$$conf(p, stPos, X) = \frac{Actual\_Periodicity(p, stPos, X)}{Perfect\_Periodicity(p, stPos, X)},$$

where $Perfect\_Periodicity(p, stPos, X) = \lfloor \frac{|T| - stPos + 1}{p} \rfloor$ and $Actual\_Periodicity(p, stPos, X)$ is computed by counting (starting at $stPos$ and repeatedly jumping by p positions) the number of occurrences of X in T.

For example, in $T = \mathbf{a}bbca\mathbf{a}bcdb\underline{ac}cdb\mathbf{a}bbca$, the pattern ab is periodic with $stPos = 0$, $p = 5$, and $conf(5, 0, ab) = \frac{3}{4}$. Note that the confidence is 1 when perfect periodicity is achieved.

Definition 3 (Symbol Periodicity). *A time series T is said to have symbol periodicity for a given symbol s with period p and starting position $stPos$ if the periodicity of s in T is either perfect or imperfect with high confidence, i.e., s occurs in T at most of the positions specified by $stPos + i \times p$, where p is the period and integer $i \geq 0$ takes consecutive values starting at 0.*

For example, in $T = bacdbabcdaccdabc$, the symbol a is periodic with $stPos = 1$ and $p = 4$, i.e., a occurs in T at positions 1, 5, 9, and 13.

Definition 4 (Sequence Periodicity). *A time series T is said to have sequence periodicity (or partial periodicity) for a pattern (partial periodic pattern) X starting at position $stPos$, if $|X| \geq 1$ and the periodicity of X in T is either perfect or imperfect with high confidence, i.e., X occurs in T at most of the positions specified by $stPos + i \times p$, where p is the period and integer $i \geq 0$ takes consecutive values starting at 0.*

For example, in $T = \mathbf{a}bbca\mathbf{a}bcdb\mathbf{a}bcdb\mathbf{a}bbca$, the pattern ab is partially periodic starting from $stPos = 0$ and period value of 5.

Note that symbol periodicity is a special case of sequence periodicity, where $|X| = 1$. But these two are defined separately in order to keep the terminology consistent with the previous works described in the literature.

Definition 5 (Segment Periodicity). *A time series T of length n is said to have segment periodicity for a pattern X with period p and starting position $stPos$ if $p = |X|$ and the periodicity of X in T is either perfect or imperfect with high confidence, i.e., X occurs in T at most of the positions specified by $stPos + i \times p$, where integer $i \geq 0$ takes consecutive values starting at 0.*

Definition 6 (Periodicity in Subsection of a Time Series). *A time series T possesses symbol, sequence, or segment periodicity with period p between positions $stPos$ and $endPos$ (where $0 \leq stPos < endPos \leq |T|$), if the investigated period satisfies Definition 3, Definition 4, or Definition 5, respectively, by considering only subsection $[stPos,endPos]$ of T.*

For example, in $T = abd\mathbf{a}bbca\mathbf{a}bcdbaccdb\mathbf{a}bbca$, the pattern ab is periodic with $p = 5$ in the subsection [3, 19] of T.

Definitions 3, 4, 5, and 6 will not be applicable in case the time series contains some insertion or deletion noise. Unfortunately, it is not possible to avoid noise in real-life applications. Thus, to be able to tackle noisy time series, we introduce the concept of *time tolerance*, denoted tt, where a periodic occurrence is counted valid if it is found within $\pm tt$ positions of the expected position.

Definition 7 (Periodicity with Time Tolerance). *Given a time series T which is not necessarily noise-free, a pattern X is periodic in subsection $[stPos,endPos]$ of T with period p and time tolerance $tt \geq 0$ if X is found at positions $stPos$, $stPos + p \pm tt, stPos + 2p \pm tt, \ldots, endPos - p \pm tt$.*

For example, in time series $T = abce\ dabc\ cabc\ aabc\ babc\ c$, $freq(ab,\ 4,\ 0,\ 18,\ tt = 1) = 5$ and $conf(ab,\ 4,\ 0,\ 18,\ 1) = \frac{5}{5} = 1$, while based on Definitions 4 and 2, $conf(ab,\ 4,\ 0,\ 18) = \frac{1}{5} = 0.2$ and $conf(ab,\ 4,\ 5,\ 18) = \frac{4}{4} = 1$.

4 Periodicity Detection—Our Approach

Our algorithm involves two phases. In the first phase, we build the suffix tree for the time series and in the second phase, we use the suffix tree to calculate the periodicity of various patterns in the time series. One important aspect of our algorithm is *redundant period pruning*, i.e., we ignore a redundant period ahead of time. As immediate benefit of redundant period pruning, the algorithm does not waste time to investigate a period which has already been identified as redundant. This saves considerable time and also results in reporting fewer but more useful periods. This is the primary reason why our algorithm, intentionally, reports significantly fewer number of periods without missing any existing periods during the pruning process.

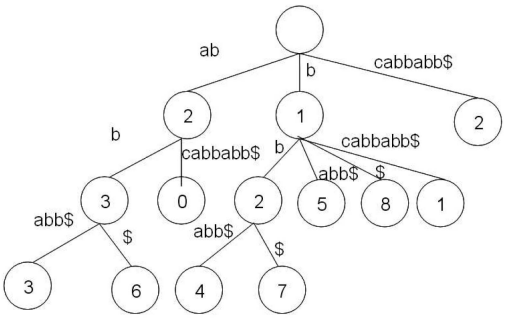

Fig. 1. The suffix tree for the string $abcabbabb\$$.

4.1 First Phase—Suffix-Tree-Based Representation

Suffix tree is a famous data structure [9] that has been proven to be very useful in string processing [9], [11], [5], [19]. It can be efficiently used to find a substring in the original string, to find the frequent substring and other string matching problems. A suffix tree for a string represents all its suffixes; for each suffix of the string there is a distinguished path from the root to a corresponding leaf node in the suffix tree. Given that a time series is encoded as a string, the most important aspect of the suffix tree, related to our work, is its capability to very efficiently capture and highlight the repetitions of substrings within a string. Fig. 1 shows a suffix tree for the string $abcabbabb\$$, where $\$$ denotes end marker for the string; it is a unique symbol that does not appear anywhere in the string.

The path from the root to any leaf represents a suffix for the string. Since a string of length n can have exactly n suffixes, the suffix tree for a string also contains exactly n leaves. Each edge is labeled by the string that it represents. Each leaf node holds a number that represents the starting position of the suffix yield when traversing from the root to that leaf. Each intermediate node holds a number which is the length of the substring read when traversing from the root to that intermediate node. Each intermediate edge reads a string (from the root to that edge), which is repeated at least twice in the original string. These intermediate edges form the basis of our algorithm presented later in this section.

A suffix tree can be generated in linear time using Ukonen's famous algorithm described in [32]. Ukonen's algorithm works online, i.e., a suffix tree can be extended as new symbols are added to the string. A suffix tree for a string of length n can have at most $2n$ nodes and the average depth of the suffix tree is of order $log(n)$ [8], [26]. It is not necessary to always keep a suffix tree in memory; there are algorithms for handling disk-based suffix tree, making it a very preferred choice for processing very large sized strings such as time series [20] and DNA sequences which grow in billions [15], [29].

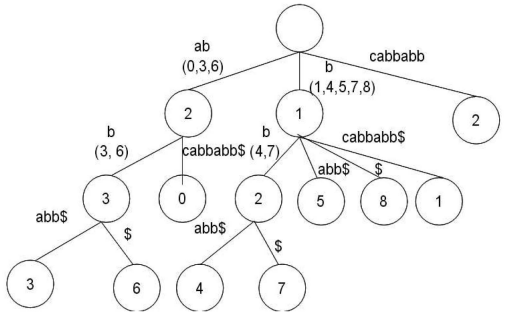

Fig. 2. Suffix tree for string $abcabbabb\$$ after bottom-up traversal.

Once the tree as presented in Fig. 1 is constructed, we traverse the tree in bottom-up order to construct what we call *occurrence vector* for each edge connecting an internal node to its parent. We start with nodes having only leaf nodes as children; each such node passes the values of its children (leaf nodes) to the edge connecting it to its parent node. The values are used by the latter edge to create its *occurrence vector* (denoted $occur\_vec$ in the algorithm). The occurrence vector of edge e contains index positions at which the substring from the root to edge e exist in the original string. Second, we consider each node v having a mixture of leaf and nonleaf nodes as children. The occurrence vector of the edge connecting v to its parent node is constructed by combining the occurrence vector(s) of the edge(s) connecting v to its nonleaf child node(s) and the value(s) coming from its leaf child node(s). Finally, until we reach all direct children of the root, we recursively consider each node u having only nonleaf children. The occurrence vector of the edge connecting u to its parent node is constructed by combining the occurrence vector(s) of the edge(s) connecting u to its child node(s). Applying this bottom-up traversal process on the suffix tree shown in Fig. 1 will produce the occurrence vectors reported in Fig. 2. The periodicity detection algorithm uses the occurrence vector of each intermediate edge (an edge that leads to a nonleaf node) to check whether the string represented by the edge is periodic. The tree traversal process is implemented using the nonrecursive explicit stack-based algorithm presented in [30], which prevents the program from throwing the stack-overflow-exception.

4.2 Periodicity Detection Algorithm

As mentioned in the previous section, we apply the periodicity detection algorithm at each intermediate edge using its occurrence vector. Our algorithm is linear-distance-based, where we take the difference between any two successive occurrence vector elements leading to another vector called the *difference vector*. It is important to understand that we actually do not keep any such vector in the memory but this is considered only for the sake of explanation. Table 1 presents example occurrence and difference vectors. Each value in the difference vector is a candidate period starting from the corresponding occurrence vector value (the value of $occur\_vec$ in the same row). Recall that each period can be represented by 5-tuple $(X, p, stPos, endPos, conf)$, denoting the pattern, period value, starting position, ending position, and confidence, respectively. For each candidate period $(p = diff\_vec[j])$, the algorithm scans the occurrence vector starting from its corresponding value $(stPos = occur\_vec[j])$, and increases the frequency count of the period *freq(p)* if and only if the occurrence vector value is periodic with regard to $stPos$ and p. This is presented formally in Algorithm 1.

TABLE 1
An Example Occurrence Vector and Its Corresponding Difference Vector

occur_vec	diff_vec
0	3
3	9
12	4
16	5
21	3
24	3
27	11
38	7
45	3
48	

Algorithm 1. Periodicity Detection Algorithm
Input: *a time series of size n;*
Output: *positions of periodic patterns;*

1. for each occurrence vector occur_vec of size k for pattern X, repeat
1.1 for $j = 0; j < \frac{n}{2}; j++$;
1.1.1 $p = occur\_vec[j+1] - occur\_vec[j]$;
1.1.2 $StPos = occur\_vec[j]$;$endPos = occur\_vec[k]$;
1.1.3 for $i = j; i < k; i++$;
1.1.3.1 if $(stPos \bmod p == occur\_vec[i] \bmod p)$ *increment* $count(p)$;
1.1.4 endfor
1.1.5 $conf(p) = \frac{count(p)}{Perfect\_Periodicity(p,stPos,X)}$;
1.1.6 if $(conf(p) \geq threshold)$ *add p to the period list;*
1.2 endfor
2 endfor
EndAlgorithm

4.3 Periodicity Detection in Presence of Noise

We have already presented the noise-resilient features of the algorithm earlier in [24], but we briefly review them here for the sake of completeness. Three types of noise generally considered in time series data are replacement, insertion, and deletion noise. In replacement noise, some symbols in the discretized time series are replaced at random with other symbols. In case of insertion and deletion noise, some symbols are inserted or deleted, respectively, randomly at different positions (or time values). Noise can also be a mixture of these three types; for instance, RI type noise means the uniform mixture of replacement (R) and insertion (I) noise. Algorithm 1 performs well when the time series is either perfectly periodic or contains only replacement noise and performs poorly in the presence of insertion or deletion noise. This is because insertion and deletion noise expand or contract the time axis leading to shift of the original time series values. For example, time series $T = abcabcabc$ after inserting symbol b at positions 2 and 6 would be $T' = abbcabcabbc$. The occurrence vector for

symbol a in T is $(0, 3, 6)$, while it is $(0, 4, 7)$ in T'. It is very clear that when the time series is distorted by insertion and/or deletion noise, linear-distance-based algorithms do not perform well. Actually, this is not only the case with linear-distance-based algorithms, it is also true for periodicity detection algorithms in general [7].

In order to deal with this problem, we introduce the concept of time tolerance (see Definition 7) into the periodicity detection process. The idea is that periodic occurrences can be drifted (shifted ahead or back) within a specified limit called time tolerance (denoted as tt in the algorithm). This means that if the periodic occurrence is expected to be found at positions $x, x+p, x+2p, \ldots$, then with time tolerance, occurrences at $x, x+p \pm tt, x+2p \pm tt, \ldots$ would also be considered valid. For example, in

$$T = abcde\ abdcec\ abcdca\ abdd\ abbca$$
$$01234\ 567890\ 123456\ \ 7890\ \ 12345,$$

the occurrence vector for pattern $X = ab$ is $(0, 5, 11, 17, 21)$. According to Algorithm 1, $conf(ab, p = 5, 0, 21) = \frac{2}{5}$. But, $conf(ab, 5, 0, 21, tt = 1) = \frac{5}{5}$, by considering time tolerance of ± 1. The occurrences at positions 11 and 17 are counted because they are $+1$ position away from the expected positions 10 and 16, respectively, and the occurrence at position 21 is counted because it is -1 position away from the expected position 22. It is important to note that our algorithm maintains the *moving reference*, i.e., when the occurrence at position 11 is counted with $p = 5$, the next expected occurrences are taken as $16 + 5 \times i$. This is the reason why the occurrence at position 17 is counted because it is $+1$ position away from 16 and the next reference is set at position 17; this propagates into the next positions to be checked, i.e., the next expected occurrences are checked at positions $22 + 5 \times i$. The modified algorithm with time tolerance taken into consideration is presented in Algorithm 2.

Algorithm 2. Noise Resilient Periodicity Detection Algorithm
Input: *a time series of size n and time tolerance value tt;*
Output: *positions of periodic patterns;*

1. for each occurrence vector occur_vec of size k for pattern X, repeat
1.1 for $j = 0; j < \frac{n}{2}; j++$;
1.1.1 $p = occur\_vec[j+1] - occur\_vec[j]$;
1.1.2 $StPos = occur\_vec[j]$;$endPos = occur\_vec[k]$;
1.1.3 for $i = j; i < k; i++$;
1.1.3.1 $A = occur\_vec[i] - currStPos$;
1.1.3.2 $B = Round(\frac{A}{p})$;
1.1.3.3 $C = A - (p \times B)$;
1.1.3.4 if $((-tt \leq C \leq +tt)$ *AND* $(Round((preOccur - currSTPos)p) \neq B))$
1.1.3.4.1 $currStPos = occur\_vec[i]$;
1.1.3.4.2 $preOccur = occur\_vec[i]$;
1.1.3.4.3 increment $count(p)$;
1.1.3.4.4 $sumPer+ = (p + C)$;
1.1.4 endfor
1.1.5 $meanP = sumPer - \frac{p}{(count(p)-1)}$;
1.1.6 $conf(p) = \frac{count(p)}{Perfect\_Periodicity(p,stPos,X)}$;
1.1.7 if $(conf(p) \geq threshold)$ *add p to the period list;*
1.2 endfor
2 endfor
EndAlgorithm

Algorithm 2 contains three new variables, namely A, B, and C; A represents the distance between the current occurrence and the current reference starting position; B represents the number of periodic values that must be passed from the current reference starting position to reach the current occurrence. Variable C represents the distance between the current occurrence and the expected occurrence. For instance, assume the occurrence vector of the last example, namely $(0, 5, 11, 17, 21)$, is modified into $(0, 5, 16, 22, 26)$. For this, when the current occurrence $(occur\_vec[i])$ is 16, $A = 16 - 5 = 11$, $B = \frac{11}{5} = 2$, $C = 11 - 5 \times 2 = 1$. The second condition at line 1.1.3.4 of Algorithm 2 is also very interesting. It checks to see if the current occurrence is the repetition of an already counted periodic value. Consider how the occurrence vector $(0, 5, 15, 16, 22, 26)$ would have been affected if this condition was not included in Algorithm 2: $count(p = 5, stPos = 0)$ would be 6 as the occurrences at positions 15 and 16 are within ± 1 range of the expected position 15; so they would be counted as valid occurrences; this scenario is avoided by the condition in line 1.1.3.4 of Algorithm 2. Finally, Algorithm 2 calculates the average period value as in the presence of insertion and/or deletion noise, the first difference or the candidate period might be misleading. Suppose the occurrence vector is $(0, 5, 11, 17, 23, 29, 35, 42)$; based on Algorithm 2, the period value would be $p = 5$; but the average period value is $P = 5.85 \cong 6$, which reflects more realistic information.

4.4 Periodicity Detection in a Subsection of Time Series

So far, the periodicity detection algorithm calculates all patterns which are periodic starting from any position less than half the length of the time series $(stPos < \frac{n}{2})$ and continues till the end of the time series or till the last occurrence of the pattern. This is general assumption in most of the periodicity detection algorithms [6], [7]. But, in real-life situations, there might be a case that a pattern is periodic only within a section and not in the entire time series. For example, the traffic pattern during semester break near schools is different than during the semester, movie rental pattern in winter is different than in summer, hourly number of transactions at a superstore may show different periodicity during Holidays season (15 November to 15 January) than regular periodicity. By considering time series $T = bcdabababababcdabcadad$, pattern ab is perfectly periodic only in the range [3, 12]. Such type of periodicity might be very interesting in DNA sequences in particular and in regular time series in general.

In order to detect such periodicity, we employ the concept introduced in [28], where two parameters *dmax* and *minLength* are utilized. The first parameter *dmax* denotes the maximum distance between any two occurrences of a pattern to be part of the same periodic section. Consequently, having the distance between two occurrences more than *dmax* may potentially mark the end of one periodic section and/or the start of new periodic section for the same pattern. For instance, consider the time series

$$T = bcdabababa\ \ babccdadbc\ \ dabcadad$$
$$0123456789\ 0123456789\ 01234567$$

here, $(3, 5, 7, 9, 11, 21)$ is the occurrence vector for $X = ab$. With $dmax < 10$, the last occurrence at position 21 would not be counted as valid because $21 - 11 = 10$, which would result in reporting the period $(ab,\ 2,\ 3,\ 12, conf = 1)$. Similarly, the second parameter *minLength* is used to specify the minimum length of the periodic section in order to prohibit reporting large number of very short useless periodic sections. Hence, in $T = bcababababdebcdebabab$, the occurrence vector for $X = ab$ is $(2, 4, 6, 8, 17, 19)$. With $dmax = 5$ (or $dmax < 9$) and $minLengh = 6$ (or $4 <$ *minLength*< 9), the section between positions 2 and 9 would be reported as periodic, but the section between 17 and 20 would be ignored. The modified algorithm is presented next as Algorithm 3.

Algorithm 3. Noise Resilient Periodicity Detection Algorithm within a Subsequence
Input: *a time series of size n, time tolerance value tt and a range* [$stPos$,$endPos$];
Output: *positions of periodic patterns;*

1. for each occurrence vector occur_vec of size k for pattern X, repeat
1.1 for $j = 0; j < \frac{n}{2}; j++$;
1.1.1 $p = occur\_vec[j+1] - occur\_vec[j]$;
$StPos = occur\_vec[j]$;
$endPos = occur_v ec[k]$;
1.1.2 $currStPos = stPos$; $preOccur = -5$;
1.1.3 for $i = j; i < k; i++$;
1.1.3.1 $A = occur\_vec[i] - currStPos$;
1.1.3.2 $B = Round(\frac{A}{p})$;
1.1.3.3 $C = A - (p \times B)$;
1.1.3.4 if $((-tt \leq C \leq +tt)$ *AND* $(Round((preOccur - currSTPos)p) \neq B))$
1.1.3.4.1 $currStPos = occur\_vec[i]$;
1.1.3.4.2 $preOccur = occur\_vec[i]$;
1.1.3.4.3 increment $count(p)$;
1.1.3.4.4 $sumPer+ = (p + C)$;
1.1.3.5 if $((i \neq j)$ *AND* $(i < (k-1))$ *AND* $((occur\_vec[i+1] - occur\_vec[i]) > dmax)$ *AND* $((currStPos - stPos) > minLength))$
1.1.3.5.1 $endPos = currStPos$;
1.1.3.5.2 break (exit current for loop);
1.1.4 endfor
1.1.5 $meanP = sumPer - \frac{p}{(count(p)-1)}$;
1.1.6 $conf(p) = \frac{count(p)}{Perfect\_Periodicity(p,stPos,X)}$;
1.1.7 if $(conf(p) \geq threshold)$ add p *to the period list;*
1.2 endfor
2 endfor
EndAlgorithm

4.5 Redundant Period Pruning Techniques

Periodicity detection algorithms generally do not prune or prohibit the calculation of redundant periods; the immediate drawback is reporting a huge number of periods, which makes it more challenging to find the few useful and meaningful periodic patterns within the large pool of reported periods. Assume the periods reported by an algorithm are as presented in Table 2. Looking carefully at this result, it can be easily seen that all these periods can be replaced by just one period, namely $(X = ab, p = 5, stPos = 0)$; and all other periods maybe considered as just the mere repetition of period X.

TABLE 2
An Example Result of a Periodicity Detection Algorithm

Pattern(X)	Period(p)	Starting Position(stops)
ab	5	0
ab	10	0
ab	25	0
a	5	0
a	15	0
b	5	1
b	10	1
a	5	10
ab	5	20

Empowered by redundant period pruning, our algorithm not only saves the time of the users observing the produced results, but it also saves the time for computing the periodicity by the mining algorithm itself. We implemented the redundant period pruning techniques as prohibitive steps which prohibit the algorithm from handling redundant periods. Some of the redundant period pruning approaches are outlined next.

If $(X, p, stPos)$ exists then $(X, k \times p, stPos)$ would never be addressed, where $k > 0$ is an integer, e.g., if (*a*, 3, 0) exists then (*a*, 6, 0) and (*a*, 9, 0) are redundant.
If $X \subseteq Y$, and $(Y, p, stPos)$ exists then $(X, p, stPos)$ would never be calculated, e.g., if $(abc, 5, 0)$ exists then $(*bc, 5, 0)$ and $(bc, 5, 1)$ are redundant.
If $X \subseteq Y$, and $(Y, p, stPos)$ exists then $(X, kp, stPos)$ would never be calculated, where $k > 0$ is an integer, e.g., if $(abc, 5, 0)$ exists then $(ab*, 15, 0)$ and $(ab, 15, 0)$ are redundant.
If $(X, p, stPos)$ exists then $(X, p, k \times stPos)$ would never be calculated, where $k > 0$ is an integer, e.g., if $(a, 3, 0)$ exists then $(a, 3, 9)$ and $(a, 3, 27)$ are redundant.

4.6 Optimization Strategies of the Algorithm

There are some optimization strategies that we selected for the efficient implementation of the algorithm. These strategies, though simple, have improved the algorithm efficiency significantly. We do not include these into the algorithm pseudocode so as to keep it simple and more understandable. Some of these strategies are briefly mentioned in the text below.

1. Recall that each edge connecting parent node v to child node u has its own occurrence vector that contains values from leaf nodes present in the subtree rooted at u. Accordingly, edges are sorted based on the number of values they have to carry in their occurrence vectors; and edges that qualify to be processed in each step of the algorithm are visited in descending order based on the size of their occurrence vectors. This is beneficial as we calculate

the periods at each intermediate node, and we need to sort the occurrence vector at each intermediate node. Processing edges (indirectly) connected to the largest number of leaf nodes first will lead to better performance because this will mostly decrease the amount of work required to sort newly formed occurrence vectors.

2. We do not physically construct the occurrence vector for each intermediate edge as that would mostly result in huge number of redundant subvectors. Rather, a single list of values is maintained and each intermediate edge keeps the starting and ending index positions of its occurrence list, and sorts only its concerned portion of the list. As the sorting might also require a huge number of shifting of elements, we maintain the globally unified occurrence vector as a linked list of integers so that the insertion and deletion of values do not disturb large part of the list.
3. Periodicity detection at the first level (for edges directly connected to the leaves) is generally avoided because experiments have shown that in most cases, the first level does not add any new period. This is based on the observation that in time series periodic patterns mostly consist of more than one symbol. This step alone improves the algorithm efficiency significantly.
4. Intermediate edges (directly or indirectly) connected to significantly small number of leaves can also be ignored. For example, if the subtree rooted at the child node connected to an edge contains less than 1 percent leaves, there is less chance to find a significant periodic pattern there. This leads to ignoring all intermediate edges (present at deeper levels) which would mostly lead to multiples of existing periods. Experiments have shown that, in general, this strategy does not affect the output of the periodicity detection algorithm.
5. Very small periods (say less than five symbols) may also be ignored. Periods which are larger than 30 percent (or 50 percent) of the series length are also ignored so that the infrequent patterns do not pollute the output. We also ignore periods which start from or after index position $\frac{n}{2}$, where n is the length of the time series. Similarly, intermediate edges which represent the string of length $\geq \frac{n}{2}$ are also ignored; that is, we only calculate periodicity for the sequences which are smaller than or equal to half the length of the series.
6. Similarly, periods smaller than edge value (the length of the sequence so far) are ignored. For example, in the series $abababab\$$, if the edge value is 4 then we do not consider the period of size 2, which would otherwise mean that $abab$ is periodic with $p = 2$, $st = 0$; this does not make much sense; rather the case should be expressed like $abab$ is periodic with $p = 4$, $st = 0$.
7. The collection of periods is maintained with two-levels indexing; separate index is maintained on period values and starting positions. This facilitates fast and efficient search of periods because we check the existing collection of periods a number of times.

It is worth mentioning that the strategies outlined in points 3 and 4 are optional and can be activated according to the nature of the data, its distribution, and the sensitivity of the results.

4.7 Algorithm Analysis

The proposed algorithm, as a linear-distance-based algorithm, has an advantage of inherent simplicity and flexibility. This allows us to improve the algorithm according to the requirements of different types of data sets and for problems with alternate nature. In this section, we discuss the complexity of the algorithm and its extensibility issues.

The algorithm requires only a single scan of the data in order to construct the suffix tree; and the suffix tree is traversed only once to find all periodic patterns in the time series. An additional traversal of the suffix tree might be performed to sort the edges by the size of their occurrence vectors, but this is not among the necessary requirements of the algorithm.

The algorithm basically processes the occurrence vectors of the intermediate edges. A suffix tree would have at most n intermediate edges because it contains at most $2n$ nodes and always n leaves. This means that we have to process at most n occurrence vectors. An occurrence vector may at most contain n elements (because we have n leaf nodes). Since we consider each element of the occurrence vector as candidate starting position of the period and each difference value is candidate period, the complexity of processing an occurrence vector of length n would be $O(n^2)$. Since there can be at most n occurrence vectors, the worst-case complexity of the algorithm comes out as $O(n^3)$, which is not a very good theoretical result. But fortunately, this is not the case in general, i.e., the proposed algorithm depicts the $O(n^3)$ complexity in very rare cases. One interesting observation about periodic patterns is that generally they are not too long. The length of periodic patterns is independent of the size of the time series. For example, in the Wal-Mart hourly number of transactions data that we considered in [25], the most dominant period is 24 resulting in periodic patterns of $size \leq 24$. The second most frequent pattern is found to be 168 (24×7)—a weekly pattern. It is obvious that the length of frequent patterns is independent of the time series length. Hence, we may also define a notion of maximum pattern length (MPL) and we may require the algorithm to look for patterns having their length less than or equal to MPL. With this constraint, it is trivial to prove that the worst-case complexity of the algorithm would be $O(k.n^2)$, where k is the maximum pattern length that we are looking for. The proof of this follows from the fact that the length of periodic patterns at the first level of the suffix tree is at least 1, and as we go up toward the root, the length of each pattern would increase at each level by at least 1; that is, in total we need to process the occurrence vectors present at the top k levels of the suffix tree. The cost of processing k levels would be $O(k.n^2)$ because at each level we have the sum of the sizes of all occurrence vectors as n, and the worst-case complexity of processing a level is $O(n^2)$.

To analyze the average-case complexity of the proposed algorithm, it is worth noting that the average depth of the suffix tree has been reported [8], [26] to be of order $log(n)$.

Accordingly, the average-case complexity of the proposed algorithm would be $O(n^2.logn)$.

Finally, lets cover the best-case cost of the algorithm. The best case for the periodicity detection algorithm is achieved when the time series is perfectly periodic with full periodicity, e.g., $T = abcdabcdabcd$. In such case, the embedded periodicity would be detected in the first few levels and in the first level if the periodic pattern itself does not contain repetition. An example pattern that contains a repetitive subpattern is $T = ababcababcababc$, where the subpattern ab is getting repeated in the full-cycle periodic pattern $ababc$. As we apply several redundant period pruning strategies (See Section 4.6), we do not calculate periodicity for patterns which are composed as repetition of the existing periodic patterns. Hence, the algorithm in this case would work only on the first few levels of the suffix tree. Similarly, in case two consecutive differences are the same, we do not traverse the occurrence vector for the repeating difference. For example, if we have $occur\_vec = 0, 3, 6, 9, 12, 15$, then we would traverse the occurrence vector only for $p = 3$ with $stPos = 0$, and not for $p = 3$ with $stPos$ taking the value $3,\ 6,\ 9,\ or\ 12$), because all of these periods would be the same. Hence, the occurrence vector would be traversed only once for perfectly periodic time series. The complexity of the algorithm would be $O(n)$ in the best case because at each level of the suffix tree the sum of the sizes of all occurrence vectors is n, and we are only processing the first few levels in the best case. This is reflected as great improvement in performance compared to the worst-case analysis; it is actually better than the known algorithms which look for all period sizes starting from any position in the series and with any periodic pattern length.

To analyze the cost of the memory used by the algorithm, we need to analyze the space complexity of the suffix tree and the occurrence vectors. The suffix tree requires linear space to be stored in memory [9], [32]. This is true because a suffix tree may at most contain $2 \times n$ nodes, where n is the length of the time series. Since each node can be stored in a constant memory space (few bytes), the space complexity of the suffix tree is $O(n)$ or linear. As presented earlier (in Section 4.6), we do not copy the occurrence vectors for each edge of the tree, rather we keep a single-linked list to which we add new values as the tree is traversed in bottom-up fashion; and for each edge we keep updating the start and end pointers for its occurrence vector; thus, we only need to keep at maximum n occurrence values in the list. Hence, the space required to run our algorithm is linear in terms of the length of the series irrespective of the type, distribution, number of patterns, and their length.

Finally, the proposed algorithm can be modified to work with online time series periodicity mining by using the algorithms existing for online construction of the suffix tree, e.g., [32]. In fact, Ukkonen's algorithm [32] that we are using is an online algorithm for constructing the suffix tree. However, the proposed algorithm has been only tested to work with offline time series mining, and we are currently working on modifying the algorithm for online periodicity mining. At the end, it is worth mentioning that the suffix tree may grow large and it is not necessary to keep it in memory, instead we will benefit from the algorithms [4], [31] already available for the disk-based implementation of the suffix tree.

5 Experimental Evaluation

In this section, we present the results of several experiments that have been conducted using both synthetic and real data; we also report the results of testing various characteristics of our algorithm against other existing algorithms. Hereafter, our algorithm is denoted as *STNR* (Suffix-Tree-based Noise-Resilient algorithm). As mentioned in Section 2, there are two types of algorithms described in the literature: algorithms in the first category find periodic patterns for a specific period value, and those in the other category check the time series for each and every period. From the first type, we compare STNR with ParPer [12]; and from the second type, we compare STNR with CONV [6] and WARP [7].

5.1 Accuracy

The first set of tests are dedicated to demonstrate the completeness of STNR in the sense that it should be able to find a period once it exists in the time series. We test how STNR satisfies this on both synthetic and real data.

5.1.1 Synthetic Data

The synthetic data have been generated in the same way as done in [6]. The parameters controlled during data generation are data distribution (uniform or normal), alphabet size (number of unique symbols in the data), size of the data (number of symbols in the data), period size, and the type and amount of noise in the data. A datum may contain replacement, insertion, and deletion noise or any mixture of these types of noise.

For an inerrant datum (perfectly periodic with 0 percent noise), the algorithm can find all periodic patterns with 100 percent confidence regardless of the data distribution, alphabet size, period size, and data size. This is an immediate benefit of using the suffix tree which guarantees identifying all repeating patterns. Since the algorithm checks the periodicity for all repeating patterns, the algorithm can detect all existing periods in the inerrant data. Results with noisy data are presented later.

5.1.2 Real Data

For real data experiments, we have already used the Walmart data in testing a previous version of the algorithm and the results reported in [25] demonstrated the efficiency of the algorithm for the general case where the period lasts till the end of the time series; we compared our algorithm with CONV [6]. Additionally, in this paper, we used the data set (denoted *PACKET*) which as described next contains the packet Round Trip Time (RTT) delay [27]. These data have been selected to demonstrate the adaptability of STNR for handling cases where periods are contained only within a subsection of the time series. The data have been discretized uniformly into 26 symbols.

The Packet data have been used in [27], where the authors found the dense periodic patterns, i.e., the area where the time series is mostly periodic; the three regions which were found periodic for symbol a and its patterns are: [148187-155084], [181413-186097], and [300522-304372]. Accordingly, the target of this set of experiments is to confirm that STNR can also find the periodicity within a subsection of the time series. Table 3 presents some of the results produced by

TABLE 3
Periodic Patterns in Packet Data

Pattern	Period	StPos	EndPos	Confidence
aa	2	146698	155081	0.42
aa	2	180136	186061	0.42
aa	2	297362	304371	0.47
aaa	3	146772	155064	0.44
aaa	3	180897	186065	0.43
aaa	3	297525	304367	0.47
aaa*a	5	147030	155044	0.44
aaaa*	5	182390	186064	0.36
aaa**	5	297480	304324	0.44
****aa*	7	147203	155042	0.46
aaa	7	148841	155042	0.46
aa*****	7	149394	155070	0.46
a**aaaa	7	182091	186059	0.4
a*a*aa*	7	299362	304324	0.49

STNR when run on the Packet data. The algorithm does find all the periods in the dense periodic region, mostly at superior confidence level. One important point to note here is that there are very few periodic patterns in dense regions for periods 4 and 6, as they are multiples of an existing reported period, namely 2; STNR has not reported these as they are redundant and do not carry any new information.

5.2 Time Performance

This section reports the results of the experiments that measure the time performance of STNR compared to ParPer, CONV, and WARP. We test the time behavior of the compared algorithms by considering three perspectives: varying data size, period size, and noise ratio.

5.2.1 Varying Data Size

First, we compare the performance of STNR against ParPer [12], which finds the periodic patterns for a specific period. The synthetic data used in the testing have been generated by following uniform distribution with alphabet size of 10 and embedded period value of 32. The algorithms have been run on these data by varying the time series size from 100,000 to 1,000,000. The results are presented in Fig. 3. Being linear, ParPer performs better than STNR. This is because ParPer (and other similar algorithm(s)) have been designed to find only patterns with a specified periodicity while STNR is general and finds the periodicity for all the patterns which are periodic for any period value starting and ending anywhere in the time series. In case ParPer and the other similar algorithms are extended to find patterns with all period values, their complexity would jump up to $O(n^2)$. Even with $O(n^2)$ complexity, ParPer only finds partial periodic patterns while STNR can find all the three types of periodic patterns in the data, namely symbol, sequence, and segment periodicity. Further, compared to STNR, ParPer does not detect periodicity within subsection of a time series and it is not resilient to noise (especially insertion/deletion).

Due to the differences highlighted above, the quality of the results produced by STNR could be classified as better than that produced by ParPer algorithm [12]. For instance, STNR is capable of finding out (using segment periodicity) that the electrical power consumption has the weekly pattern as presented in [6] where Elfeky et al. analyzed a data set which reflects the daily power consumption rates of some customers over a period of one year. In addition,

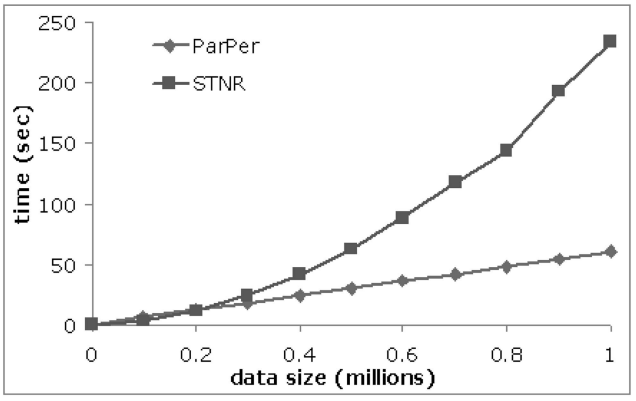

Fig. 3. Time performance of STNR compared with partial periodic patterns algorithm.

STNR can even achieve this result when the series contains insertion and deletion noise and when the periodicity is only found in a subsection of the time series and not in the entire time series; ParPer is not capable of achieving this [12]. Similarly ParPer cannot detect the periodicity of singular events (termed as symbol periodicity) which might be prevalent in the time series.

In the second set of experiments, we prepared a time series with uniform distribution containing 10 unique symbols and embedded period size of 32. The size of the series has been varied from 100,000 to 100,000,000 (100 millions). The results of STNR are compared with those reported by CONV [6] and WARP [7] algorithms of Elfeky; the curves are plotted in Fig. 4, where it can be seen that STNR performs worse than CONV, but better than WARP. Actually CONV performs better because its runtime complexity is $O(nlogn)$. Although the complexity of WARP is $O(n^2)$, STNR performs better than WARP because STNR applies various optimization strategies, most notably the adopted redundant period pruning techniques. This also supports our claim that the value of k (*maximum period length* or MPL for short) is generally very small compared to the time series length and does not grow with the time series length. We conducted this set of experiments on very large series in order to prove that STNR is extendable or scalable in practice.

5.2.2 Varying Period Size

This set of experiments is intended to show the behavior of STNR by varying the embedded period size. For this experiment, we fixed the time series length and the number of alphabets in the series and vary the embedded period size from 5 to 100. The time taken by STNR for both uniform and normal distribution has been recorded and the corresponding curves are plotted in Fig. 5. From Fig. 5, we can easily observe the effect of changing period value on time for ParPer [12] and CONV [6]. The results show that the time performance of both STNR and CONV does not change significantly and remains more or less the same. But ParPer does get affected as the period size increased; this is true because when the period value is large, the partial periodic pattern is large as well, and so is the max-subpattern tree [12]. Time performance of CONV remains the same as it checks for all possible periods irrespective of the data set.

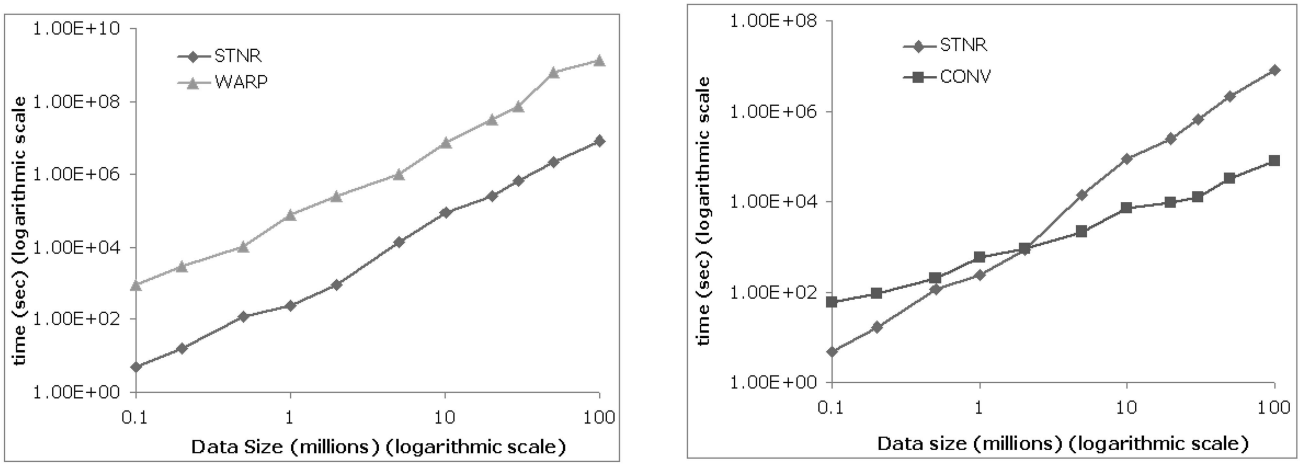

Fig. 4. Time performance of the algorithm compared with convolution and time warping algorithms.

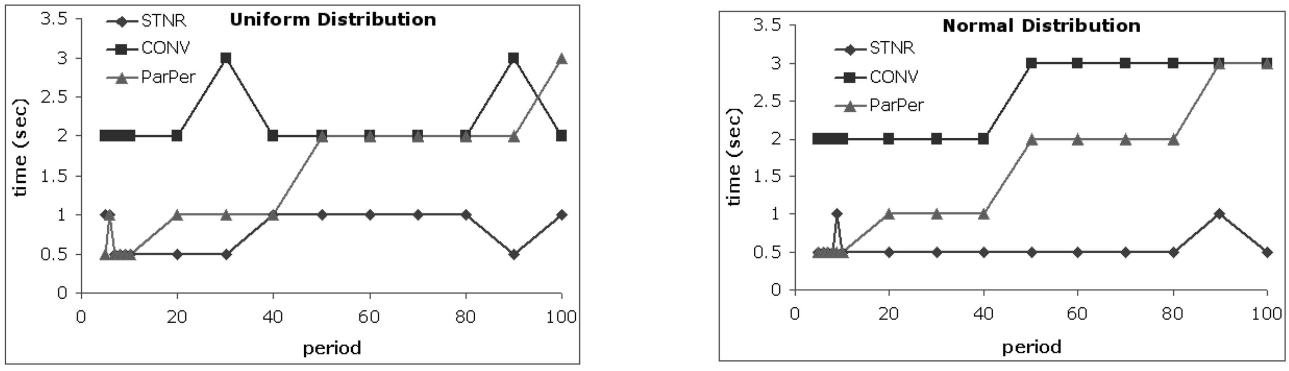

Fig. 5. Time behavior with varying period size (alphabet $\text{size} = 10$).

5.2.3 Varying Noise Ratio

The next set of experiments measure the impact of noise ratio on the time performance of STNR. For this experiment, we fixed the time series length, period size, alphabet size, and data distribution and measured the impact of varying noise ratio on time performance of the algorithm. We tested two sets of data; one contains replacement noise and the other contains insertion noise. The results are plotted in Fig. 6. Again these experiments have been conducted using the three algorithms: ParPer, STNR, and CONV. It is true that STNR takes more time when the noise ratio is small (10-15 percent); but when the noise ratio is increased, STNR tends to take the similar time. As we ignore the intermediate edges (or the periodic patterns) which carry less than 1 percent leaf nodes (or patterns which appear rarely), mostly noise is caught into these infrequent patterns and does not affect the time performance of STNR by reasonable margin. As the results show, the noise ratio does not effect the time performance of CONV and ParPer.

5.2.4 Effect of Data Distribution

Finally, we tested the time performance of STNR, ParPer [12], WARP [7], and CONV [6] for the combination of data distribution and period size. We tested two series; one has been generated with uniform data distribution while the other inhibits normal distribution. Period size of 25 and 32 are embedded in the time series and the time performance

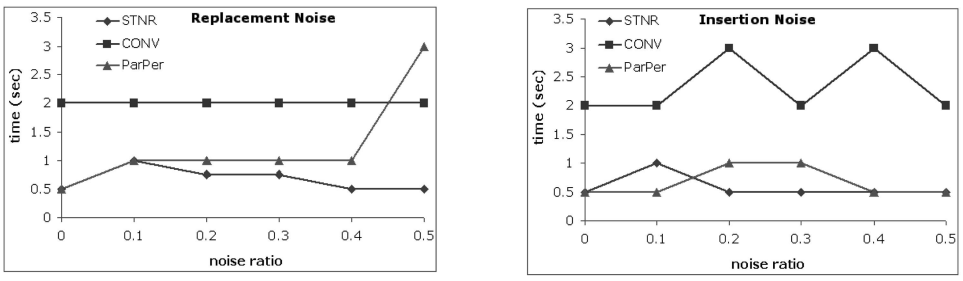

Fig. 6. Time performance of the algorithm by varying noise ratio.

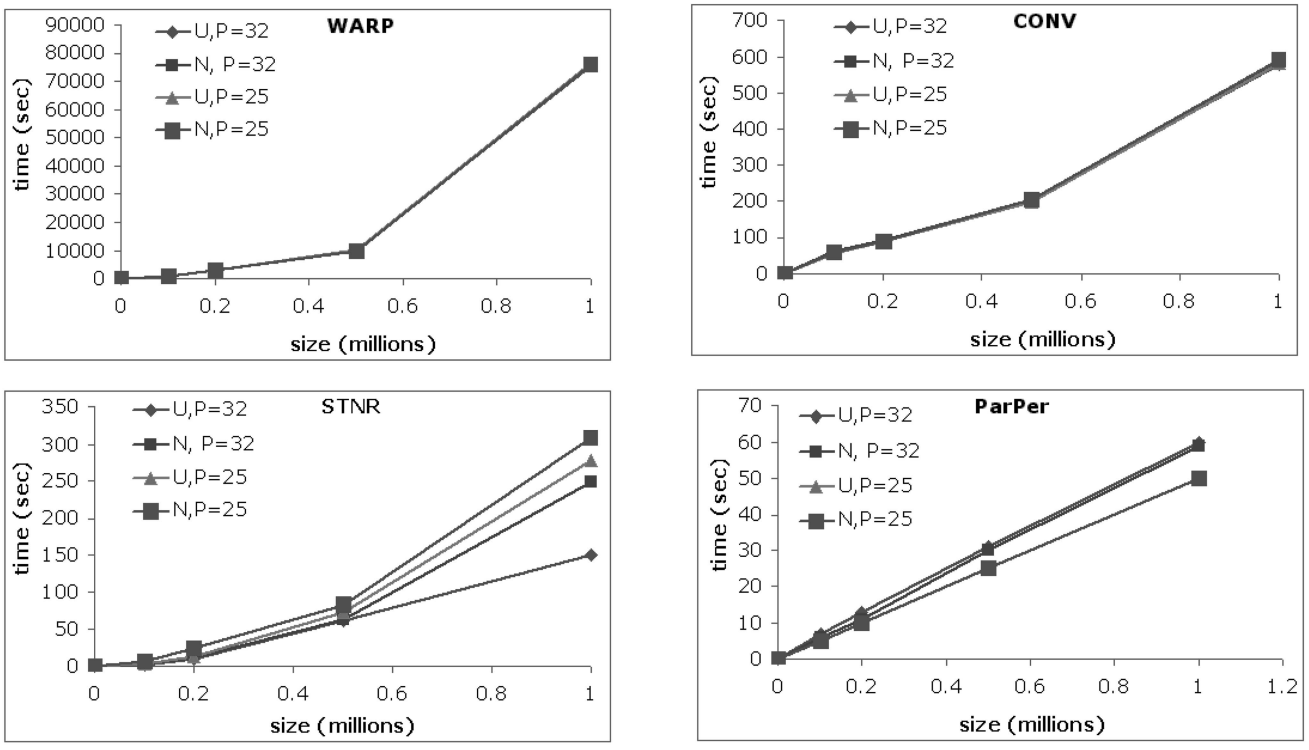

Fig. 7. Time performance with different data distribution.

is measured between 100,000 and 1,000,000 series length. The results are presented in Fig. 7. For STNR, the uniform distribution seems to take less amount of time compared to the normal distribution. Since the shape of the suffix tree depends on the data distribution, it is very understandable that the different data distributions would take different amounts of time. But, an important observation is that the time behavior or the time pattern is similar for both uniform and normal distributions and for different period size combinations. For CONV and WARP, the data distribution does not affect the time performance as well because they check the periods exhaustively. The time performance of ParPer does not get affected by the data distribution but it seems to take more time when the period size is increased (as pointed out earlier).

5.3 Optimization Strategies

As presented in Section 4.6, we use several optimization strategies to improve the time performance of the algorithm in practice. In this section, we will present the experimental results demonstrating the gain in time consumption of the algorithm to process the same data set with and without some of the optimization strategies. Here, we will present three experiments to show the effect of the employed optimization techniques; we mainly test for: 1) sorted and unsorted occurrence vectors of edges in terms of the number of values they carry (strategy 1 in Section 4.6), 2) calculating the periodicity and executing STNR by including/excluding the edges at the first level of the suffix tree (strategy 3 in Section 4.6), and 3) ignoring or considering the edges whose occurrence vectors carry fewer values (strategy 4 in Section 4.6).

5.3.1 Tree Traversal Guided by Sorted Edges

As mentioned earlier, the tree traversal is guided by the number of leaves a node (or edge) carry. The edge which leads to more leaves (i.e., the subtree rooted at the immediate child following the edge has more leaves) is traversed first and the edge which leads to fewer number of leaves is traversed later. This results in fewer changes in the occurrence vectors maintained by STNR. Fig. 8 presents the time taken by STNR to process the same data set with and without employing sorting on occurrence vectors. As expected, STNR conserves time when using the sorted occurrence vectors. The gap reduces when the noise ratio is increased as well when there is more noise in the data, the number of patterns also increases and hence the leaves tend to be divided evenly among edges.

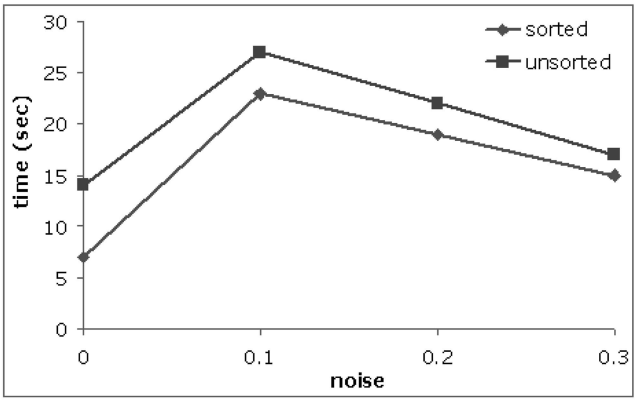

Fig. 8. Time performance with and without sorted edges.

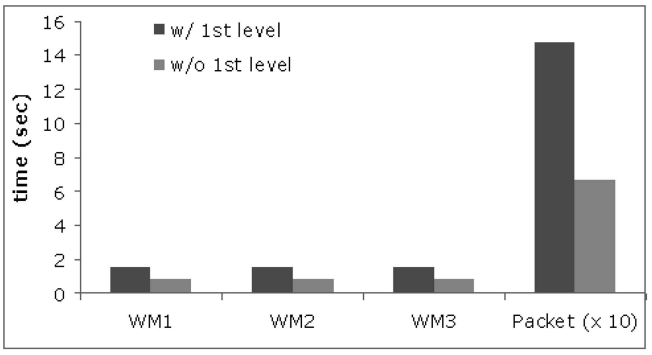

Fig. 9. Execution time with and without checking first level occurrence vectors.

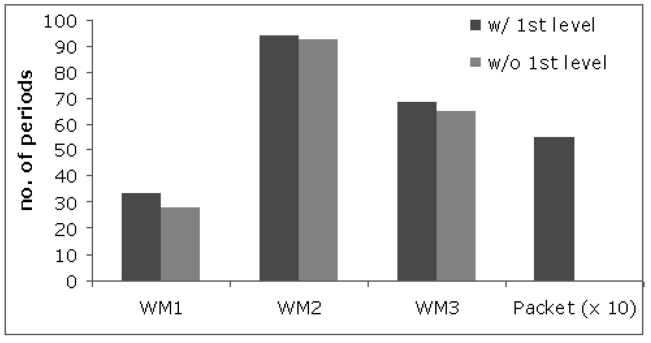

Fig. 10. Number of periods detected with and without checking first level occurrence vectors.

5.3.2 Periodicity Detection at First Level

Periodicity detection can be avoided for occurrence vectors at the first level of the suffix tree for many data sets because usually the patterns at first level are subsets of the larger patterns. But this depends heavily on the data set and the periodic pattern size (or length). We used real data sets to test this feature. We used three Wal-Mart hourly transaction count data sets and PACKET. The results are presented in Figs. 9 and 10. In these experiments, we observed the runtime and the number of nonredundant periods detected by STNR with and without calculating the periodicity based on occurrence vectors at the first level of the tree. The time performance of the algorithm improves significantly when occurrence vectors at the first level are ignored. Time improvement is expected because occurrence vectors at the first level always contain more elements (leaves) than those at other levels. By not considering the first level, there is hardly any difference in the number of periods detected in the Wal-Mart data sets because almost all the patterns in these data sets consist of more than one symbol (the daily pattern) and hence are caught at levels deeper than the first level. But as PACKET is concerned, the algorithm could not find a single periodic pattern when the first level is ignored. This is because all periodic patterns in PACKET consist of single symbol and do not have their supersets at the deeper levels of the tree. Therefore, we may conclude that although ignoring the occurrence vectors at the first level considerably improves the time performance, it may miss some very useful patterns if the data set contains very small periodic patterns (usually symbol periodicity only). Hence, this strategy should be followed very carefully.

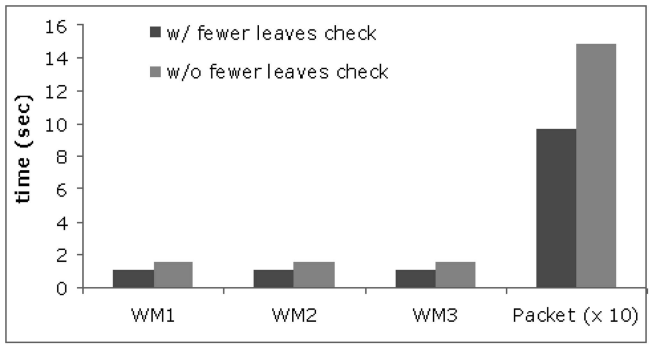

Fig. 11. Execution time with and without fewer leaves check.

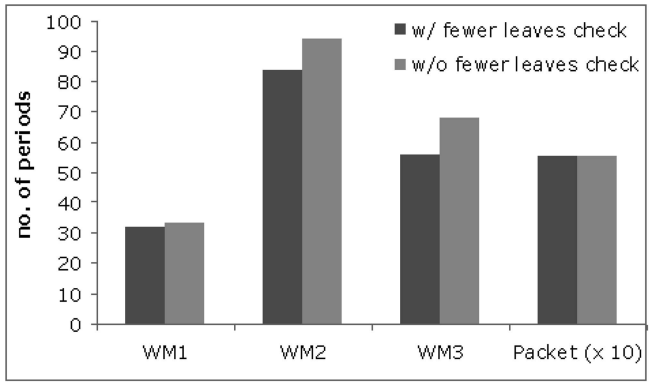

Fig. 12. Number of periods detected with and without fewer leaves check.

5.3.3 Ignoring Edges Carrying Fewer Leaves

We may ignore edges (or nodes) that carry very small number of leaves say 1 or 2 percent leaves. Such edges usually do not lead to any valid periodic patterns. Since the synthetic data sets maybe biased, we decided to run this set of experiments again on the Wal-Mart and PACKET data sets. Here, we ignored all edges which carry less than 3 percent leaves. The results are presented in Figs. 11 and 12. The results show that ignoring such minority edges hardly affects the number of nonredundant periods detected in the considered data sets. Still there is a possibility that some (although very small number of) periods might be missed because of this strategy, but again it does conserve the algorithm's running time.

Here, it is important to note that these strategies might be very useful when the data set is very large and disk-based implementation is to be used where only a small portion of the suffix tree is to be kept in memory.

5.4 Noise Resilience

We have already demonstrated in [24] the noise-resilient features of the algorithm where we compared the resilience to noise of STNR and the other algorithms based on each of the five combinations of noise, i.e., replacement, insertion, deletion, insertion-deletion, and replacement-insertion-deletion. The results show that our algorithm compares well with WARP [7] and performs better than AWSOM [22], CONV [6], and STB [25]. The latter three algorithms do not perform well because they only consider synchronous periodic occurrences while STNR and WARP perform better because we take into account asynchronous periodic

TABLE 4
Periodic Pattern Found in $P09593$

Per	Stpos	EndPos	confidence	pattern	repetitions
11	104	338	0.9	ggpgse	19
11	105	313	0.95	gpgsegpkgtg	18

occurrences which drift from the expected position within an allowable limit.

5.5 Experiments with Biological Data

Biological data, e.g., DNA or protein sequences, also exhibit periodicity for certain patterns. DNA sequences are constructed using four alphabets A, T, C, and G, while protein sequences are based on 20 alphabets (from A to T). These sequences have their own specific properties such as the periodic patterns are only found in a subsection of the sequence and do not span the entire series, the pattern occurrence may drift from the expected position, there is a concept of alternative substrings where a substring may replace another substring without any change in the semantics. For example, in the data set considered in [23], TA may replace TT. We applied our algorithm on DNA sequences in [23]. In this section, we present two experiments where we applied the algorithm for detecting the periodicity in protein sequences. The two protein sequences namely P09593 and P14593 can be retrieved from the Expert Protein Analysis System (ExPASy) database server (www.expasy.org).

The protein sequence P09593 is S antigens protein in Plasmodium falciparum $v1$ strain. S antigens are soluble heat-stable proteins present in the sera of some infected individuals. The sequence of S antigen protein is different among different strains [3]. Diversity in S antigen is mainly due to polymorphism in the repetitive regions. It has been shown that the repeated sequence is *ggpgsegpkgt* with periodicity of 11 amino acids, and this pattern repeats itself 19 times [17]. Some of the periodic patterns with period value of 11, found using STNR are presented in Table 4. Note that the index position starts from zero, as is the case with all the examples and experimental results reported in this paper. The second pattern is just a rotated version of the original pattern *ggpgsegpkgt* with 18 repeats while the first pattern starts ($StPos = 104$) just before the second pattern ($StPos = 105$) with 19 repeats. Thus, our algorithm finds not only the expected pattern of *ggpgsegpkgt*, but also shows that its shifted version *gpgsegpkgtg* is also periodic. The pattern exists in the middle of the series and is repeated contiguously. We have discovered 18 repeats of shifted pattern instead of 19, and the reason is that the repetition which starts at position 282 is *gpgsespkgtg*; it is different from the expected pattern by one amino acid at position 6, where it has s instead of g. Since STNR does consider alternatives, we were able to detect that pattern. The first result also hints this by showing the pattern *ggpgse* repeating 19 times. The shifted pattern *gpgsegpkgtg*, detected by STNR, is an interesting result that has not been reported in any protein database or in any other studies. Together with our collaborators in the Department of Molecular Biology in the School of Medicine at the University of Calgary, we are investigating the practical significance of this discovery.

TABLE 5
Periodic Pattern Found in $P14593$

Per	stpos	StPosMod	EndPos	confidence	pattern	repetitions
4	106	2	162	0.33	d	5
4	108	0	369	1	gn	66
4	115	3	369	1	agn	64
4	122	2	369	0.94	aagn	58

For the second experiment, we applied STNR on the protein sequence P14593 which is the code for circumsporozoite protein. It is the immunodominant surface antigen on the sporozoite. The sequence of this protein has interesting repeats which represent the surface antigen of the organism. Pattern $aagn$ was reported in [17] to be the repeating pattern in the sequence with 65 repeats, but the repeating pattern in the UniProtKB protein database is reported to be $gn(a/d)(a/e)$. Some of the periodic patterns with period of 4 found by STNR are presented in Table 5. The algorithm does detect the periodic pattern $aagn$ with 58 repeats. But if we combine the four results, we get an interesting pattern. For the combination of patterns, we have also included a column named *StPosMod* in Table 5 which represents the modulus of the start position (StPos) and the period value (Per). If we join the last three patterns, we get the pattern $gnaa$, a shifted version of the pattern $aagn$, with maximum repetitions of 66. When we combine all the four patterns, we get the pattern $gn(a/d)a$, which is similar to the pattern reported in the UniProtKB database.

Both protein sequences studied are antigen proteins which are used for immune responses. The repeating sequence is used to allow the antibodies to detect the antigens. To further analyze the antigen protein sequences, all antigen protein sequences are analyzed to discover similar patterns among difference sequences. This helps in understanding the immune mechanism against each antigen. Designing a drug to target the repeats in the antigen will have significant impact on drug and antibody design industry.

6 Conclusions and Future Work

In this paper, we have presented STNR as a suffix-tree-based algorithm for periodicity detection in time series data. Our algorithm is noise-resilient and run in $O(k.n^2)$ in the worst case. The single algorithm can find symbol, sequence (partial periodic), and segment (full cycle) periodicity in the time series. It can also find the periodicity within a subsection of the time series. We performed several experiments to show the time behavior, accuracy, and noise resilience characteristics of the data. We run the algorithm on both real and synthetic data. The reported results demonstrated the power of the employed pruning strategies.

Currently, we are working on online periodicity detection where the suffix tree is constructed online, i.e., extended while the algorithm is in operation. This type of stream mining has a large number of applications especially in sensor networks, ubiquitous computing, and DNA sequence mining. We are also implementing the disk-based solution of the suffix tree to extend capabilities beyond the virtual memory into the available disk space. In [6], the

authors used fast Fourier transforms to reduce the complexity from $O(n^2)$ to $O(nlogn)$; and in [22], the authors used wavelet transform to solve the problem in linear time. Along these lines, we are working on a suitable transform to reduce the runtime of the algorithm. The ideas of dense periodic regions and minimum period size calculated in [27] are also useful and can be incorporated to further improve the performance of the algorithm.

REFERENCES

[1] M. Ahdesmäki, H. Lähdesmäki, R. Pearson, H. Huttunen, and O. Yli-Harja, "Robust Detection of Periodic Time Series Measured from Biological Systems," *BMC Bioinformatics,* vol. 6, no. 117, 2005.

[2] C. Berberidis, W. Aref, M. Atallah, I. Vlahavas, and A. Elmagarmid, "Multiple and Partial Periodicity Mining in Time Series Databases," *Proc. European Conf. Artificial Intelligence,* July 2002.

[3] H. Brown et al., "Sequence Variation in S-Antigen Genes of Plasmodium falciparum," *Molecular Biology and Medicine,* vol. 4, no. 6, pp. 365-376, Dec. 1987.

[4] C.-F. Cheung, J.X. Yu, and H. Lu, "Constructing Suffix Tree for Gigabyte Sequences with Megabyte Memory," *IEEE Trans. Knowledge and Data Eng.,* vol. 17, no. 1, pp. 90-105, Jan. 2005.

[5] M. Dubiner et al., "Faster Tree Pattern Matching," *J. ACM,* vol. 14, pp. 205-213, 1994.

[6] M.G. Elfeky, W.G. Aref, and A.K. Elmagarmid, "Periodicity Detection in Time Series Databases," *IEEE Trans. Knowledge and Data Eng.,* vol. 17, no. 7, pp. 875-887, July 2005.

[7] M.G. Elfeky, W.G. Aref, and A.K. Elmagarmid, "WARP: Time Warping for Periodicity Detection," *Proc. Fifth IEEE Int'l Conf. Data Mining,* Nov. 2005.

[8] J. Fayolle and M.D. Ward, "Analysis of the Average Depth in a Suffix Tree under a Markov Model," *Proc. Int'l Conf. Analysis of Algorithms,* pp. 95-104, 2005.

[9] D. Gusfield, *Algorithms on Strings, Trees, and Sequences.* Cambridge Univ. Press, 1997.

[10] E.F. Glynn, J. Chen, and A.R. Mushegian, "Detecting Periodic Patterns in Unevenly Spaced Gene Expression Time Series Using Lomb-Scargle Periodograms," *Bioinformatics,* vol. 22, no. 3 pp. 310-316, Feb. 2006.

[11] R. Grossi and G.F. Italiano, "Suffix Trees and Their Applications in String Algorithms," *Proc. South Am. Workshop String Processing,* pp. 57-76, Sept. 1993.

[12] J. Han, Y. Yin, and G. Dong, "Efficient Mining of Partial Periodic Patterns in Time Series Database," *Proc. 15th IEEE Int'l Conf. Data Eng.,* p. 106, 1999.

[13] K.-Y. Huang and C.-H. Chang, "SMCA: A General Model for Mining Asynchronous Periodic Patterns in Temporal Databases," *IEEE Trans. Knowledge and Data Eng.,* vol. 17, no. 6, pp. 774-785, June 2005.

[14] J. Han, W. Gong, and Y. Yin, "Mining Segment-Wise Periodic Patterns in Time Related Databases," *Proc. ACM Int'l Conf. Knowledge Discovery and Data Mining,* pp. 214-218, 1998.

[15] E. Hunt, R.W. Irving, and M.P. Atkinson, "Persistent Suffix Trees and Suffix Binary Search Trees as DNA Sequence Indexes," Technical Report TR2000-63, Univ. of Glasgow, Dept. of Computing Science, 2000.

[16] P. Indyk, N. Koudas, and S. Muthukrishnan, "Identifying Representative Trends in Massive Time Series Data Sets Using Sketches," *Proc. Int'l Conf. Very Large Data Bases,* Sept. 2000.

[17] M.V. Katti, R. Sami-Subbu, P.K. Rajekar, and V.S. Gupta, "Amino Acid Repeat Patterns in Protein Sequences: Their Diversity and Structural-Function Implications," *Protein Science,* vol. 9, no. 6, pp. 1203-1209, 2000.

[18] E. Keogh, J. Lin, and A. Fu, "HOT SAX: Efficiently Finding the Most Unusual Time Series Subsequence," *Proc. Fifth IEEE Int'l Conf. Data Mining,* pp. 226-233, 2005.

[19] R. Kolpakov and G. Kucherov, "Finding Maximal Repetitions in a Word in Linear Time," *Proc. Ann. Symp. Foundations of Computer Science,* pp. 596-604, 1999.

[20] N. Kumar, N. Lolla, E. Keogh, S. Lonardi, C.A. Ratanamahatana, and L. Wei, "Time-Series Bitmaps: A Practical Visualization Tool for Working with Large Time Series Databases," *Proc. SIAM Int'l Conf. Data Mining,* pp. 531-535, 2005.

[21] S. Ma and J. Hellerstein, "Mining Partially Periodic Event Patterns with Unknown Periods," *Proc. 17th IEEE Int'l Conf. Data Eng.,* Apr. 2001.

[22] S. Papadimitriou, A. Brockwell, and C. Faloutsos, "Adaptive, Hands Off-Stream Mining," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* pp. 560-571, 2003.

[23] F. Rasheed, M. Alshalalfa, and R. Alhajj, "Adapting Machine Learning Technique for Periodicity Detection in Nucleosomal Locations in Sequences," *Proc. Eighth Int'l Conf. Intelligent Data Eng. and automated Learning (IDEAL),* pp. 870-879, Dec. 2007.

[24] F. Rasheed and R. Alhajj, "STNR: A Suffix Tree Based Noise Resilient Algorithm for Periodicity Detection in Time Series Databases," *Applied Intelligence,* vol. 32, no. 3, pp. 267-278, 2010.

[25] F. Rasheed and R. Alhajj, "Using Suffix Trees for Periodicity Detection in Time Series Databases," *Proc. IEEE Int'l Conf. Intelligent Systems,* Sept. 2008.

[26] Y.A. Reznik, "On Tries, Suffix Trees, and Universal Variable-Length-to-Block Codes," *Proc. IEEE Int'l Symp. Information Theory,* p. 123, 2002.

[27] C. Sheng, W. Hsu, and M.-L. Lee, "Mining Dense Periodic Patterns in Time Series Data," *Proc. 22nd IEEE Int'l Conf. Data Eng.,* p. 115, 2005.

[28] C. Sheng, W. Hsu, and M.-L. Lee, "Efficient Mining of Dense Periodic Patterns in Time Series," technical report, Nat'l Univ. of Singapore, 2005.

[29] N. Välimäki, W. Gerlach, K. Dixit, and V. Mäkinen, "Compressed Suffix Tree—A Basis for Genome-Scale Sequence Analysis," *Bioinformatics,* vol. 23, pp. 629-630, 2007.

[30] A. Al-Rawi, A. Lansari, and F. Bouslama, "A New Non-Recursive Algorithm for Binary Search Tree Traversal," *Proc. IEEE Int'l Conf. Electronics, Circuits and Systems (ICECS),* vol. 2, pp. 770-773, Dec. 2003.

[31] Y. Tian, S. Tata, R.A. Hankins, and J.M. Patel, "Practical Methods for Constructing Suffix Trees," *VLDB J.,* vol. 14, no. 3, pp. 281-299, Sept. 2005.

[32] E. Ukkonen, "Online Construction of Suffix Trees," *Algorithmica,* vol. 14, no. 3, pp. 249-260, 1995.

[33] A. Weigend and N. Gershenfeld, *Time Series Prediction: Forecasting the Future and Understanding the Past.* Addison-Wesley, 1994.

[34] J. Yang, W. Wang, and P. Yu, "InfoMiner+: Mining Partial Periodic Patterns with Gap Penalties," *Proc. Second IEEE Int'l Conf. Data Mining,* Dec. 2002.

Faraz Rasheed received the BSc degree in computer science from the University of Karachi, Pakistan, in 2003 and the MSc degree in computer engineering from Kyung Hee University, South Korea, in 2006. He is a PhD candidate in the Department of Computer Science at the University of Calgary under the supervision of Professor Reda Alhajj. He published more than 15 papers in international conferences and journals. He received several prestigious scholarships, including departmental research awards, provincial and national awards like the prestigious NSERC CGS-D and iCore graduate scholarships. His research interests include time series data mining, periodicity detection in time series databases, and algorithm design and analysis.

Mohammed Alshalalfa received the BSc degree in molecular biology and genetics from Middle East Technical University (METU), Ankara, Turkey. During his undergraduate studies, he worked in the Plant Biotechnology lab at METU. In the last year of his undergraduate studies, he visited the Department for Molecular Biomedical Research (DMBR), Ghent University, Belgium, for two months to study the effect of viral infection on transcription factors activity. In September 2006, he joined the graduate program in the Department of Computer Science at University of Calgary, Canada, where he received the MSc degree in July 2008. Currently, he is a PhD candidate in the Department of Computer Science at the University of Calgary under the supervision of Professor Reda Alhajj. So far, he has published more than 20 papers in refereed international conferences and journals. He is focusing on microarray data analysis and applications of data mining technique into microarray data. He received several prestigious scholarships and recently distinguished himself by ranking at the very top of the iCore graduate scholarship. His research interests are in the areas of genomics, proteomics, computational biology, social networks, and bioinformatics, where he is successfully adapting different machine learning and data mining techniques for modeling and analysis.

Reda Alhajj received the BSc degree in computer engineering from Middle East Technical University, Ankara, Turkey, in 1988. After completing the BSc degree with distinction from METU, he was offered a full scholarship to join the graduate program in computer engineering and information sciences at Bilkent University in Ankara, where he received the MSc and PhD degrees in 1990 and 1993, respectively. Currently, he is a professor in the Department of Computer Science at the University of Calgary, Alberta, Canada. He published more than 300 papers in reputable journals and fully refereed international conferences. He served on the program committee of several international conferences including the IEEE ICDE, IEEE ICDM, IEEE IAT, SIAM DM, etc. He also served as guest editor of several special issues and is currently the program chair of the IEEE IRI 2009, CaSoN 2009, ASONAM 2009, ISCIS 2009, MS 2009, and OSINT-WM 2009; he is maintaining the same positions for 2010. He is on the editorial board of several journals and associate editor of the *IEEE Systems, Man, and Cybernetics-Part C*. He frequently gives invited talks in North America, Europe, and the Middle East. He has a research group of 10 PhD and 8 MSc students working primarily in the areas of biocomputing and biodata analysis, data mining, multiagent systems, schema integration and reengineering, and social networks and XML. He received Outstanding Achievements in Supervision Award from the Faculty of Graduate Studies at the University of Calgary. He is an associate member of the IEEE.