Appl Intell (2010) 32: 267–278
DOI 10.1007/s10489-008-0144-9

STNR: A suffix tree based noise resilient algorithm for periodicity detection in time series databases

Faraz Rasheed · Reda Alhajj

Published online: 4 September 2008
© Springer Science+Business Media, LLC 2008

Abstract Periodicity detection has been used extensively in predicting the behavior and trends of time series databases. In this paper, we present a noise resilient algorithm for periodicity detection using suffix trees as an underlying data structure. The algorithm not only calculates symbol and segment periodicity, but also detects the partial (or sequence) periodicity in time series. Most of the existing algorithms fail to perform efficiently in presence of noise; although noise is an inevitable constituent of real world data. The conducted experiments demonstrate that our algorithm performs more efficiently compared to other algorithms in presence of replacement, insertion, deletion or a mixture of any of these types of noise.

Keywords Time series · Periodicity detection · Suffix tree · Segment periodicity · Sequence periodicity · Noise resilient

1 Introduction

Periodicity detection in time series databases has recently been an active research area and already received the attention of several research groups. It is a process of finding whether a given series, or a pattern within the series, is repeating itself at regular intervals or not. Periodicity detection is used in predicting future events or trends in the time series; it is a crucial step towards better and more effective decision making.

F. Rasheed · R. Alhajj (✉)
Department of Computer Science, University of Calgary, Calgary, Alberta, Canada
e-mail: alhajj@ucalgary.ca

F. Rasheed
e-mail: frashed@ucalgary.ca

Periodic patterns are frequently found in time series such as road traffic pattern, power consumption over time, transaction record, weather information, geological and astronomical observations, patient physiological data, DNA sequences, and so on. Periodicity detection is useful to predict the behavior and the future trends of the time series [4]. Periodic pattern mining is also a useful tool in predicting the stock price movement, computer network fault analysis and detection of security breach, earth-quake prediction, and gene expression analysis [17, 18].

Time series consists of a sequence of ordered values collected, generally, after uniform interval of time. Let T be a time series of length n and D_i be a feature collected at time i, then the time series can be represented as $T = D_0, D_1, D_2, \dots, D_{n-1}$. For periodicity detection, generally, the time series is discretized into finite symbols taken from an alphabet set, denoted $\sum$. For instance, if $\sum$ contains $\{a, b, c\}$, then a possible time series $T = D_0, D_1, D_2, \dots, D_{n-1}$ can be discetized as $T = aabacabbc$.

Three types of periodicity are considered in the time series literature. The first type is called the segment or full-cycle periodicity, where the time series is a result of approximate repetition of a segment of the series. The second type is symbol periodicity, where it is calculated if the individual symbols of a time series are getting repeated periodically. Thirdly, we have the partial periodicity, where a pattern (of length ≥ 1) in the time series is getting repeated periodically. For example, the time series $T = abcabcabbabc$ exhibits the segment periodicity, $P_{seg}(p, st) = (3, 0)$, of 3 starting at position zero because the pattern abc is of length 3, and it is repeating regularly. Similarly, the time series $T = abcdabbdabcdabbd$ contains the symbol periodicity of 4 for symbol a starting at position zero, i.e., $P_{sym}(char, p, st) = (a, 4, 0)$. The latter series T

also contains $P_{sym}(b,4,1)$, $P_{sym}(d,4,3)$, $P_{sym}(c,8,2)$, and $P_{sym}(b,8,6)$. The same series $T = abcd\ abbd\ abcd\ abbd$ exhibits the partial periodicity of 4 for the pattern $ab*d$ or for the pattern $ab\{b,c\}d$, starting at position zero.

Various researchers have presented separate algorithms for detecting each of these types of periodicity. However, very few attempts have been made to capture all three types of periodicity using a single algorithm. We have presented our periodicity detection algorithm using suffix trees as an underlying structure that allows us to efficiently capture all the three types of periodicity. It starts with building the suffix tree of the given time series, which helps us in capturing the collection of occurrences (which we call occurrence vector) of all the repeating patterns in the time series. We then check if the repetitions are periodic and calculate the periodic strength (confidence) of these patterns.

Another problem not considered by most of the existing approaches is that they do not perform well in the presence of noise in the data. Since noise is an inevitable component of real world time series, it is a vital argument to state that the algorithm should be noise resilient and be able to detect the periodicity, even in the presence of a reasonable amount of noise.

Three basic types of noise generally considered in time series analysis are replacement, insertion and deletion (or any mixture of them). While some algorithms may accommodate the replacement noise, most algorithms perform poorly in the presence of insertion and deletion noise.

In this paper, we improve our previous algorithm by incorporating the time tolerance window so as to make it more resilient to insertion and deletion noise. The reported experimental evaluation demonstrates that the algorithm is more resilient to insertion and deletion noise than other existing algorithms.

The rest of the paper is organized as follows. Section 2 presents the related literature in the periodicity detection field. Our approach is explained in Sect. 3. Section 4 contains the results of experimental evaluation, while the paper is concluded with mention of future work in Sect. 5.

2 Related work

Time series analysis has received considerable attention of the research community; and, there are some approaches out there capable of detecting different types of periodicity from a time series data. For instance, Indyk et al. [1] presented their periodic trends algorithm which can find the segment periodicity only, i.e., to detect if the entire time series can be achieved by the repetition of a sequence of symbols. For example, the time series *abcabcabc* has the segment periodicity of 3. Their algorithm favors the larger periods. Recently, Elfeky et al. [2] presented two algorithms to find symbol and segment periodicity in time series data. Their algorithms favor the shorter periods.

The methods developed by the above mentioned researchers ([1] and [2]) use the shift and compare approach, where they compare the (right) shifted copy of the time series with the original time series to see how many matches there are, and what symbols these matches are for. Indyk et al. [1] use naive approach, while Elfeky et al. [2] use convolution based approach that reduces the time complexity to $O(n \log n)$.

Our algorithm does not favor any period size, i.e., the period size does not influence the accuracy of the algorithm. We prefer the shorter periods and longer sequences as far as their strength is acceptable and conclude that the sequences of multiple-length would have at least the same periodic strength. For example, if a sequence (ab) is periodic with period 5 (say p), then we conclude that it is also periodic with period 10 ($2p$), 15 ($3p$), and so on. Similarly, we prefer the larger sequences. For example, if a time series is periodic for the sequence $abc**$ with period value of 5 starting at position 0 with periodicity strength of 0.8, then the subsequences abc, ab, a, b and c all would be periodic with period 5 starting, respectively, at positions 0, 0, 0, 1 and 2, with periodicity strength greater than or equal to 0.8.

Unlike the work of Elfeky et al. [2], we do not need two separate algorithms to find the symbol and segment periodicity. Our single algorithm can detect both segment and symbol periodicity in just a single pass over the data once the series is represented using the suffix tree structure.

Unlike both the works of Indyk et al. [1] and Elfeky et al. [2], our algorithm may also find the periodicity of sequences comprising of more than one symbol. We believe that the periodicity of such sequences carries more useful and interesting information than that of single symbol. More importantly, it results in reporting much fewer numbers of periods. For example, for a sequence *abcabcabc*, the symbol periodicity detection algorithm of Elfeky et al. [2] would report 5 periods; representing them using the three-tuple form (symbol, period, starting position), they are (a, 3, 0), (b, 3, 1), (c, 3, 2), (a, 3, 3), and (b, 3, 4). On the other hand, our algorithm would report only a single period (abc, 3, 0), which is clearly easier to handle and much more useful and meaningful information.

There are some researchers, e.g., [6–8], who do use the linear distance-based algorithm like us. However, since they are not using suffix trees, but the simple sequence, they do miss certain useful periods because they only consider the adjacent inter-arrivals. For example, consider a symbol that occurs in series at positions 0, 3, 5, 8, 10. Since their algorithms (like ours) use the linear distance, they miss the period 5 starting from 0. But, this is not the case with our algorithm because the period of 5 is captured in the lower (or deeper) level node of the tree or for the larger sequence.

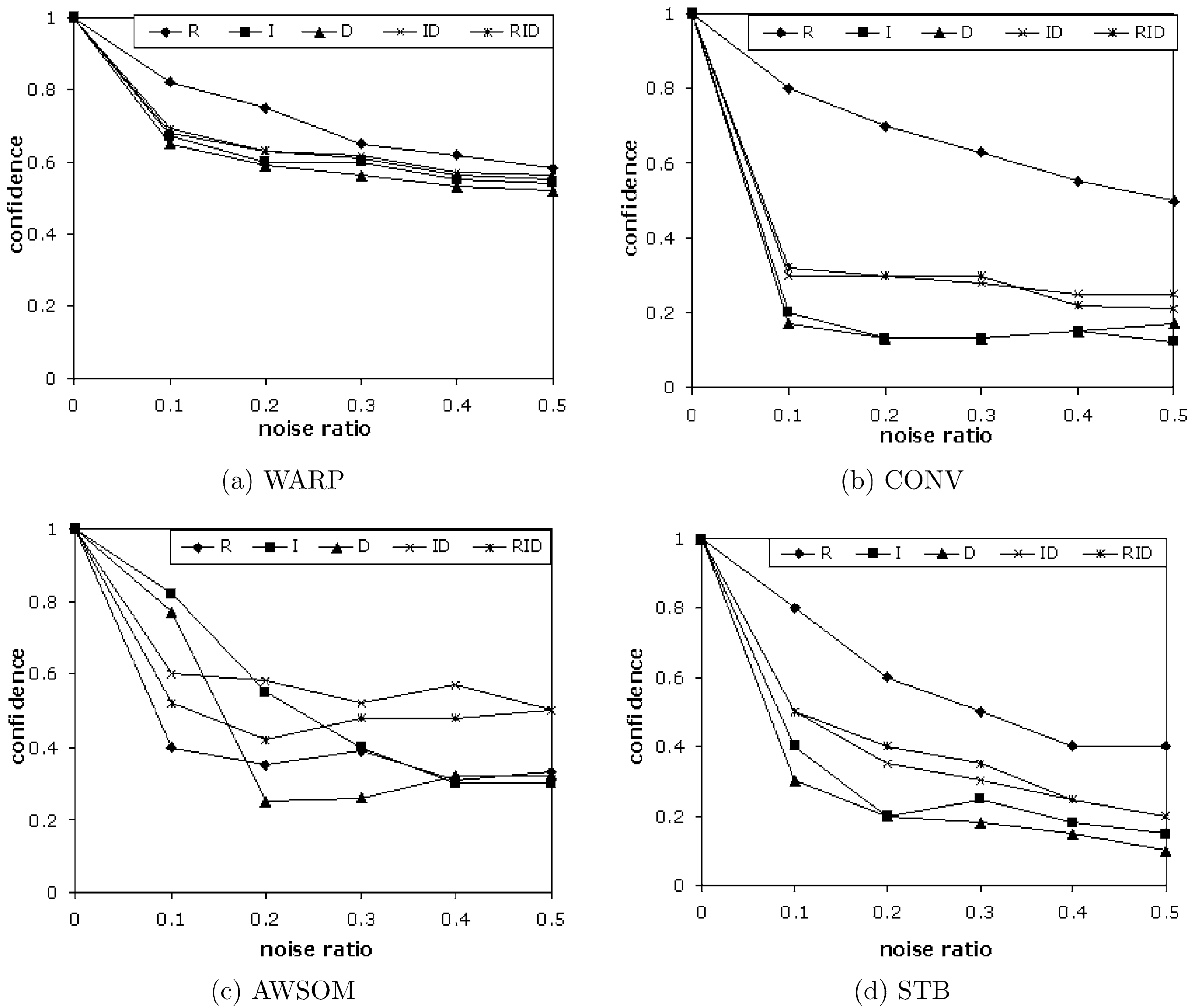

Fig. 1 Noise resilience of previous algorithms

Also, the algorithms described in [6–8] require another pass over the time series in order to output the periodic patterns, while our algorithm requires only a single pass over the series to detect and report the periods existing in the series.

Elfeky et al.'s convolution based algorithm [2] performs poorly in the presence of insertion and deletion noise, which he improved in his most recent publication [13] using the time warping concept. Papadimitriou et al. [14] presented their algorithm for periodicity detection using wavelet transform which runs in linear time but can only detect periods which are of powers of two. Our previous algorithm called Suffix Tree Basic (STB) [15] uses suffix tree as an underlying data structure to find symbol, sequence and partial periodicity using a single algorithm, but its performance also degrades when insertion and deletion noise are introduced in the time series.

Elfeky et al. [13] presented a comparison of existing algorithms for noise resilience, where they compared how the existing algorithms perform, especially in the presence of insertion and deletion noise. They presented their algorithm, called WARP, which uses the time warping concept to expand and shrink the time axis.

Although, their algorithm performs well in the presence of noise, it is only capable of detecting the segment (or full-cycle) periodicity in the time series, while our algorithm can detect symbol, segment as well as partial (sequence) periodicity. With the improvements presented in this paper, our algorithm can accommodate insertion and deletion noise at least as effectively as the WARP algorithm of Elfeky et al. [13].

Figure 1 shows how existing algorithms (WARP, Convolution (CONV) and AWSOM) perform in the presence of

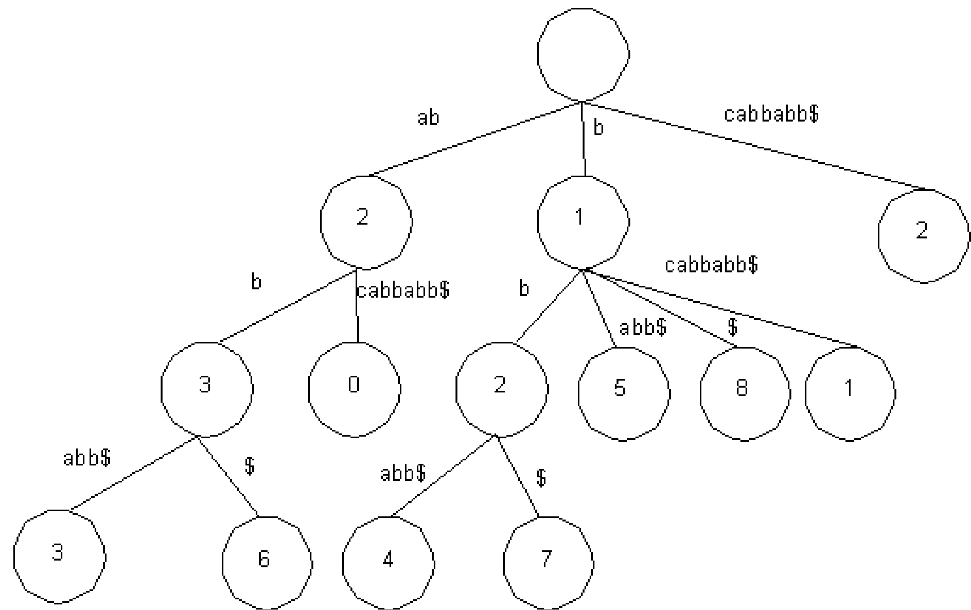

Fig. 2 The suffix tree for the string *abcabbabb*$

difference combination of replacement (R), insertion (I) and deletion (D) noise.

3 Suffix tree based noise resilient (STNR) algorithm

Based on the above analysis, we developed our algorithm for periodicity detection; the proposed algorithm uses suffix trees as an underlying data structure. In the following subsections, we would give a brief discussion on suffix trees and how we adapted the concept for periodicity detection. Later, we would explain how the algorithm accommodates the noise in the data.

3.1 Suffix trees

Suffix tree is a commonly used data structure [3] that has been proved to be very useful in string processing, e.g., [3, 9–11]. It can be efficiently used to find a substring in the original string, find the frequent substring; it can also be used to solve other substring matching problems. Each of the branches of the suffix tree represents a suffix of the original string. Hence, a suffix tree for a string of length n has n branches, and thus n leaf nodes. For example, Fig. 2 shows the suffix tree for the string *abcabbabb*$ where $ denotes the terminating symbol.

Each leaf node in the tree has an integer value showing the starting position of the substring achieved through the path from the root to that leaf in the original string. Since there are exactly n suffixes for a string, each starting at one of the index positions, there are n leaf nodes in the tree. Each internal node (nodes that are neither leaf nor the root) has the integer value representing the length of the substring so far achieved while traversing from the root to the node. A suffix tree can have a maximum of $2n$ nodes, but mostly having periodicity and repetition in the time series; there are less than $2n$ nodes in the suffix tree for these series.

We use the well-known Ukonen algorithm [5] to construct the suffix tree for a given time series; this algorithm runs in linear time. The algorithm gives us the collection of 'edges', each having the starting node number, end node number, the first character index and the last character index and the value. For example, the edge from the root with label *ab* in Fig. 2 is represented as: starting node number: 0, end node number: 1, first character index: 0, last character index 1, and the value: 2. Thus, this edge can be represented in five-tuple (0, 1, 0, 1, 2). The subsequent edge labeling *b* is represented as (1, 4, 2, 2, 3), and the edge labeling *cabbabb*$ is represented as (1, 5, 2, 9, 0). The algorithm for the suffix tree traversal is presented in Fig. 3.

3.2 Periodicity detection

Once we have the marked and labeled suffix tree, we invoke the periodicity detection algorithm given in Fig. 4. The process articulated in the invoked algorithm traverses the tree in bottom-up fashion using explicit stack [12]. During the traversal, each leaf node passes its value to the parent. The internal nodes after receiving the values from all of their children collect these in the collection called the *occurrence vector*. An occurrence vector is represented in our algorithm as '*occur_vect*'. The tree in Fig. 2 after performing this step is presented in Fig. 5, where each internal node has its own occurrence vector. In fact, this vector shows the index positions in the original time series where this sequence or pattern (starting from the root to the node in question) appear.

Fig. 3 Suffix tree traversal algorithm

SuffixTreeTraversal
1. Initialize *rootOccurSt* and *rootOccurLength* and stack 's'
2. With each children edge 'e' (edges having $stn = 0$ of the root edge
   2.1. Sort children edges
   2.2. $e.pntVal = 0$ // parent value
   2.3. $e.pntOccurSt = rootOccurSt$
   2.4. $e.pnOccurLength = rootOccurLength$
   2.5. push e to stack 's'
3. while (stack is not empty)
   3.1. $e = s.pop()$
   3.2. if edge is already marked
      3.2.1. if e does not originate from root AND e carries more than 'w'% leaves
         3.2.1.1. *CalculatePeriod* for e
      3.2.2. if *e.pntOccurSt* is blank
         3.2.2.1. $e.pntOccurSt = e.occurSt$
         3.2.2.2. $e.pntOccurLength = e.occurLength$
      3.2.3. else
         3.2.3.1. *Join&Sort*($e.pntOccurSt$, $e.pntOccurLength$, $e.occurSt$, $e.occurLength$)
   3.3. else if edge has not been marked yet
      3.3.1. if e leads to leaf $e.val = N - (e.lci - e.fci) + 1 + e.pntVal$
         3.3.1.1 *occur.add*($e.val$)
      3.3.2. else $e.val = e.lci - e.fci + 1 + e.pntVal$
         3.3.2.1. find and sort all children edges of e
         3.3.2.2. With each child edge 'ce'
            $ce.pntVal = e.val$
            $ce.pntOccurSt = e.occurSt$
            $ce.pnOccurLength = e.occurLength$
            $s.push(ce)$
      3.3.3. mark 'e'

Recall that there are a maximum of $2n$ nodes in the suffix tree representing a string of length n and there are n leaf nodes in the suffix trees; hence, the tree should have significantly less than n such vectors. If there are $x < n$ such vectors, and y_k represents the length of the occurrence vector k, then $y_k < n$, where $0 \le k \le (x - 1)$.

The second step is to calculate another vector from each of these occurrence vectors, which we call the difference vector (or '*diff_vect*' in our algorithm). Let V be the occurrence vector of length m, $V = v_0, v_1, \ldots, v_{m-1}$. The difference vector D would always have the length $(m - 1)$, and would be $D = v_1 - v_0, v_2 - v_1, \ldots, v_{m-1} - v_{m-2}$, which is calculated by simply taking the difference of the consecutive values, and thus called the difference vector. Actually, the difference vectors contain the candidate periods. Each of these periods (with some exceptions mentioned later) is checked and the corresponding periodic strength is calculated.

Consider the occurrence and difference vectors presented in Table 1. For each candidate period, we calculate the periodicity strength or confidence $\tau(p, st)$ where p represents the period value, st is the starting position of the periodic sequence.

Let v_j and s_j represent the jth entry in the difference and occurrence vectors of a pattern ptn, respectively, and i be a positive integer. The algorithm checks each and every subsequent entry (s_k) in the occurrence vector, starting from s_j, and increments *count(p, st)* by 1 if and only if:

$$s_k = (s_j + i v_j) \pm tt;$$

$$0 \le (i \times v_j \times max(occur_vect)), \quad 0 \le tt \le v_j,$$

where tt represents the time tolerance threshold value. The $count(p, st)$ represents the frequency count of the occurrence of a sequence starting from st with a period value p. If the length of the time series is n, then the periodicity

Table 1 Occurrence and difference vectors for the sequence *ab*

Index	*occur_vect*	*diff_vect*
0	0	3
1	3	9
2	12	4
3	16	5
4	21	3
5	24	3
6	27	11
7	38	7
8	45	3
9	48	

Fig. 4 Periodicity detection algorithm

```
CalculatePeriod: Edge e
1. if length of the pattern > half of series length then return
2. current = e.st; // initialize current with the starting node of the edge
3. for (i = 1; i < e.len; i++)
    3.1. diffValue = current.next.value − current.value;
/*ignore this occurrence if period < length of pattern OR period > 33% of series length OR
period starting position > half of series length */
    3.2. if (diffValue < e.value OR diffValue == 1 || diffValue > (0.33 × T.Length) OR
            current.value > (0.5 × T.Length))
        3.2.1. current = current.next;
        3.2.2. continue;
    3.3. Initialize p as candidate period
    3.4. p.val = diffValue;
        p.stops = current.value;
        p.fci = p.stPos;
        p.length = e.value;
    3.5. if (p exists in PeriodCollection)
        3.5.1 current = current.next;
        3.5.2. continue;
    3.6. p.foundPosCount = 0;
    3.7. initialize A, B, C, sumPerVal = 0;
        preSubCurValue = −5;
        initialize currStPos = p.stops;
        subCurrent = current;
    3.8. for (j = i; j <= e.len; j++)
        3.8.1. A = subCurrent.value − currStPos;
        3.8.2. B = Round(A/p.val);
        3.8.3. C = A − (p.val × B);
        3.8.4. if (C >= (−1× tolWin) AND C <= tolWin)
            3.8.4.1. if(Round(((preSubCurValue − currStPos)/p.periodValue)) NOT EQUALS B)
            3.8.4.1.1. preSubCurValue = subCurrent.value;
            3.8.4.1.2. currStPos = subCurrent.value;
            3.8.4.1.3. p.foundPosCount++;
            3.8.4.1.4. sumPerVal += (p.val + C);
3.8.5. subCurrent = subCurrent.next;
    3.9. initialize y with zero;
    3.10. if(((T.Length − 1 − p.stops)%p.val) >= e.value) then y = 1;
        else y = 0;
    3.11. p.conf = p.foundPosCount /Floor(((T.Length −1 − p.stops)/p.val) +y);
    3.12. if (p.conf >= minConfidence)
        3.12.1. p.avgVal = (sumPerVal −p.val)/(p.foundPosCount −1);
        3.12.2. Add this period 'p' to period collection
    3.13. current = current.next;
```

strength τ is calculated as:

$$\tau(p, st) = \frac{count(p, st)}{\lfloor \frac{n-st}{p} + y \rfloor},$$

$$y = \begin{cases} 1 & \text{iff } ((n - st) \bmod p) > \text{edges value}, \\ 0 & \text{otherwise.} \end{cases}$$

The periodicity strength is the ratio between the frequency of occurrences of a sequence and the maximum possible number of occurrences for the same sequence. For example, for the sequence $abcabbabc\$$, $\tau(3, 2, c)$ is $\frac{2}{3}$, as there are 2 occurrences of c, while the maximum possible occurrences are 3.

3.3 Noise resilience

Most algorithms for periodicity detection do not perform well in the presence of insertion and deletion noise. The reason for this is the expansion or contraction of the time axis of the data due to these types of noise.

Assume a time series T, without noise, looks like $T = abc\ abd\ abb\ aba$, exhibiting the periodicity of 3 for the pattern ab starting from position zero. Then, assume that due to the insertion noise a value c is inserted at position 5. The distorted time series would look like $T' = abc\ ab\underline{c}\ dab\ bab\ a$.

Now, if we apply a regular algorithm that compares the time series after every three position for $p = 3$, it would lose

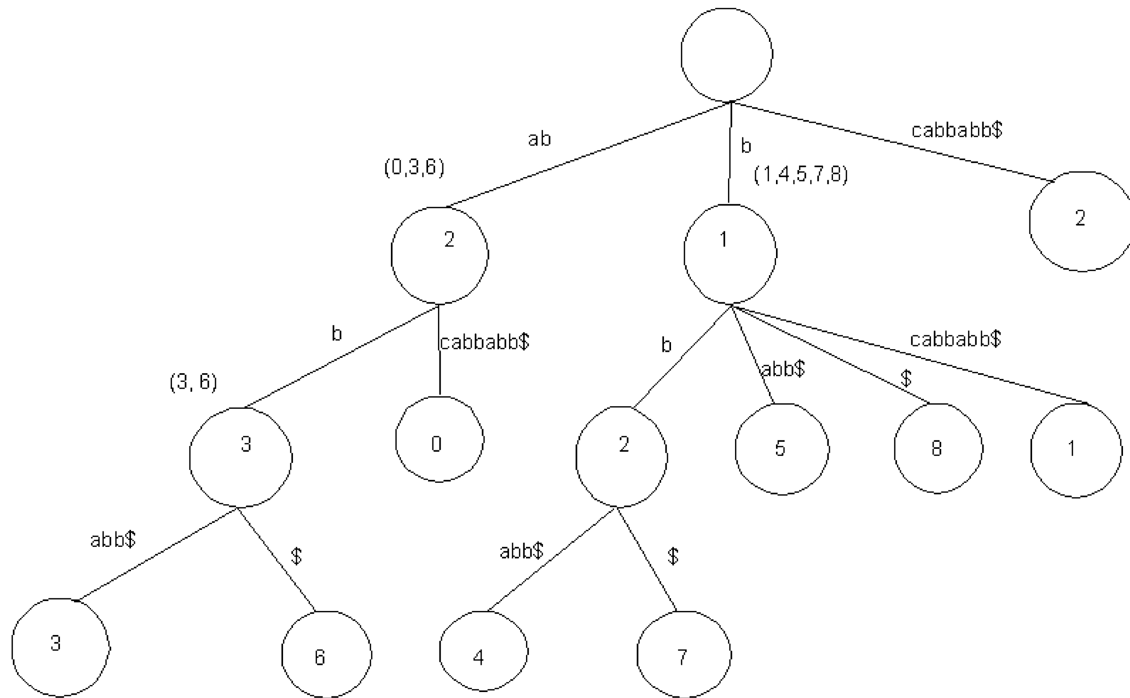

Fig. 5 The suffix tree for the string *abcabbabb*$ with substring occurrences

the track after the noise. But, if we invoke our algorithm to look for the occurrence of pattern within some time tolerance window (say ± 1 in the considered case), it would still be able to find the period 3 for the pattern '*ab*', starting from position zero.

As the insertion and deletion noise ratio is increased in the data, the periodicity detection gets more and more challenging, making it tougher for the algorithm to find the embedded periods in the series. The case of deletion noise is more severe; here few values from the original series are missed (or removed).

Consider the same original series $T = abc\ abd\ abb\ aba$, and assume that we introduce the deletion noise of 1 character at position 4; then, the distorted series would look like, $T' = abc\ ada\ bba\ ba$. Again, the traditional algorithms would find it very difficult to detect that there is a period 3 for the pattern *ab* starting from position zero. But, if we contract the time axis by 1 (having time tolerance window of ± 1), we can find that there are 3 out of 4 occurrences of the pattern *ab* with the period of 3 starting from position zero. The situation becomes worse as the noise ratio is increased.

Our algorithm works by first setting the 'anchor' at the expected position (determined by $p.val \times B$ in step 3.8.3. of the periodicity detection algorithm presented in Fig. 4). Then, the algorithm determines whether the occurrence lies within the time tolerance window of the expected position. If so, the occurrence is counted as valid and frequency count is increased. For example, we get the occurrence vector which looks as follows: (2, 5, 8, 18).

Assume we are calculating period 3 starting at position 2 with time tolerance of ± 1. Since we keep on updating the *currStPos* with the last valid position found (based on the assumption that the time axis expansion/contraction can start at any position), assume that we have checked until occurrence value 8 and we are then checking 18. For this, $A = 18 - 8 = 10$, $B = Round(\frac{10}{3}) = 3$, $C = 10 - (3 \times 3) = 1$. Since $C = 1$ is within the time tolerance window of ± 1, it would be counted as a valid occurrence.

It is important to keep on updating the *currStPos* with the last valid occurrence found, as otherwise the algorithm may still lose track; consider the series $T = abcd\ abcd\ abcd\ abcd$, and assume that after deletion noise it becomes $T' = abcd\ abca\ bcab\ cd$, it is very clear that for the pattern *abc* with $p = 4$, time tolerance window of ± 1, the algorithm can't find the last occurrence if the anchor is set statically at the first position. This is because the occurrence vector for *abc* is (0, 4, 7, 10), and with static anchor at first occurrence 0, the expected positions are (0, 4, 8, 12). The occurrence 7 would be caught as it lies within time tolerance window of ± 1, but it can not find the occurrence 10. For this, we have implemented the algorithm with the dynamic anchor that moves with each valid occurrence. Hence, in this case, the anchor would move from 0 to 4, 4 to 7, and 7 to 10. When we consider occurrence 10 with anchor at 7, it lies within the time tolerance window ($7 + 4 = 11$, $11 - 1 = 10$). Another important point is to consider the fact that there might be a case where we have more than one occurrence within the time tolerance window.

Suppose the occurrence vector is (2, 7, 12, 17, 18, 22). If the time tolerance (*tolWin*) is ±1, $p = 5$, $stops = 2$, then both 17 and 18 are within *tolWin* of the expected position 17 when the anchor is at position 12. To avoid this, we have step 3.8.4.1. in the algorithm presented in Fig. 4. Finally, because of the moving anchor, we cannot set the period value with the first difference as we used to do in our previous algorithm [15], because there is a possibility that the first difference is due to some noise in the data. Consider the occurrence vector (2, 8, 13, 18, 23, 28), where the first difference is $6 = (8 - 2)$, while as the series progresses the regular difference is mostly 5. Hence, we calculate the average difference of valid occurrences and set it as the average period value. Experiments have shown that not considering this step results in misleading periods in the results.

3.4 Redundant period pruning

We do not calculate the periodicity for all the periods, rather the algorithm detects and avoid the redundant period, thus significantly improves the period calculation space. The period pruning approaches that our algorithm uses are explained in our previous work called *STB* and presented in [15].

4 Experimental analysis

We performed various experiments to analyze the behavior of the algorithm in the presence of noise. Experiments are performed with replacement, insertion, deletion and a mixture of these types of noise in the data. We compared our algorithm with four existing algorithms: WArping foR Periodicity detection WARP [13], Convolution based algorithm CONV [2], Adaptive handSome Off-stream Mining AWSOM [14] and our previous algorithm STB [15]. The obtained results show that the algorithm proposed in this paper performs quite well compared with the other algorithms.

4.1 Accuracy

In order to test the accuracy, we test the algorithm for various period sizes, distributions (uniform and normal), and time series length. The algorithm was able to detect the periodicity in the data with 100% confidence. This is similar to our previous algorithm STB [15], and the other popular algorithms described in the literature.

4.2 Real data analysis

For real data experiments, we used the Walmart data (denoted *WALMART*) which contains the record of hourly number of transactions performed at a Walmart store. The data contains the record of around 15 months of data with the expected period value of 24; the number of transactions are discretized into five levels; *very low* (*0 transaction*), *low* (*less than 200 transactions*), *medium* (*between 200 to 400 transactions*), *high* (*between 400 to 600 transactions*) *and very high* (*between 600 to 800 transactions*) mapped, respectively, to symbols a, b, c, d and e.

The Walmart data is plotted in Fig. 6 (reproduced from [2]). This is the same dataset used in [2]. We run our algorithm with periodicity threshold values ranging from 0.8 to 0.4 and observed: (1) the number of periods captured by the algorithm, (2) the dominating period, (3) the periodic pattern of the dominating period, (4) the number of index values filled by the symbols, and (5) the number of don't care values represented by *. The results are presented in Table 2.

As expected, the proposed algorithm finds more periods at the lower periodicity threshold. The expected period 24 is captured immediately even at the periodicity threshold

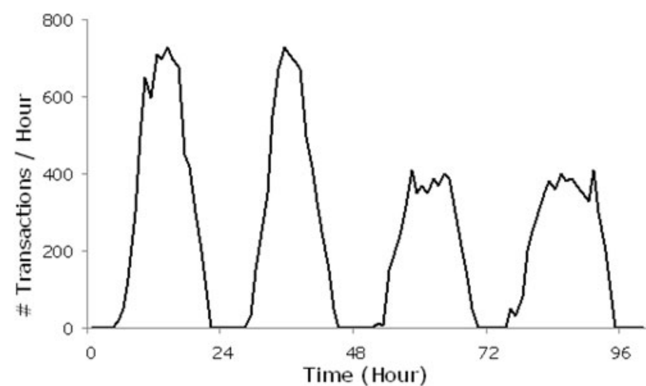

Fig. 6 Hourly number of transactions in a Walmart store [2]

Table 2 Periodicity detection algorithm output for Walmart data

Periodicity threshold	No. of periods	Dominating period	Sequence	No. of index filled	No. of '*' (don't care)
0.8	4	24 (1/4)	AAA*******************AA	5	19
0.7	8	24 (4/8)	AAA***BBBC************AA	9	15
0.6	11	24 (5/11)	AAAA*BBBBC************AA	11	13
0.5	18	24 (8/18)	AAAABBBBBC********DD**AA	14	10
0.4	27	24 (12/27)	AAAABBBBBCCDD*****DDCAAA	19	5

Table 3 Covolution based algorithm [2] output for Walmart data

Periodicity threshold	No. of periods (segment periodicity)	No. of periods (symbol periodicity)	No. of patterns (period = 24)
0.8	4	2328	6
0.7	101	2460	7
0.6	532	2728	10
0.5	1054	3164	13

Table 4 Raw output of the algorithm run on 15 months Walmart data

Series Length: 10992

Periodicity Threshold: 0.6

Period	StPos	StPos (Mod)	Threshold	SymbolString
24	7	7	0.64	BB
24	32	8	0.62	BC
24	142	22	0.74	AAAAAA
24	1895	23	0.7	AAAAA
24	2333	5	0.63	BBB
168	222	54	0.6	BBBCC
168	5491	115	0.73	DB
336	4450	82	0.6	DDDDDDDDD
3022	5393	2371	1	CCCC
3023	5392	2369	1	CCCCC
3186	5391	2205	1	CCCCCC

Number of Periods: 11

Period	Count
24	5
168	2
336	1
3022	1
3023	1
3186	1

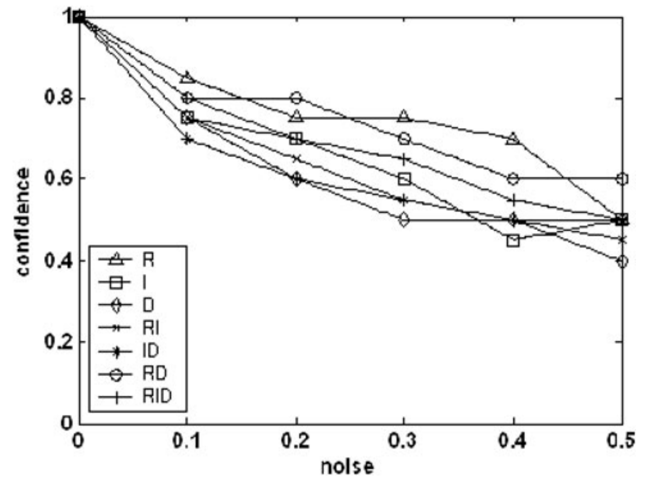

Fig. 7 Resilience to noise of the suffix tree based noise resilient algorithm

of 0.8. The periodic pattern keeps on materializing as the periodicity threshold is lowered. One important observation from the above results is that the algorithm never filled the middle index positions in the sequence. The reason may be inferred from the data plot in Fig. 6. The peak hours are obviously not periodic on daily basis, rather they are periodic mostly weekly, which is captured in our results at the period of 168 (24×7); it is the second most dominating period, and it usually fills the middle index positions. Another important observation is that our algorithm results in very few yet accurate, meaningful, useful and non-redundant periods. For the same data, the algorithms in [2, 13] produce hundreds of periods, most of which are redundant. Table 3 presents the number of periods detected by convolution based segment and symbol periodicity detection algorithm [2] and number of unique patterns of period 24 which is similar to the *number of index filled* in Table 2.

As reported in Table 2, the algorithm can be used to capture the periodic trends in the data. For example, in case of *Walmart* data, it shows that the initial and the closing hours generally have the least number of transactions. The number of transactions increases as the day progresses, which is also evident in Fig. 6, which demonstrates that the data follows the normal distribution. As presented in the results of Tables 2 and 3, our algorithm produces significantly lesser number of periods without missing out any non-redundant period. This is because our algorithm does not calculate redundant periods, e.g., those which are multiple of an existing periods or which represent the pattern whose super-pattern has already been found periodic with same period value. Since algorithms like [2, 13] do not check redundant periods during the algorithm, they must be checked after the main algorithm has finished. Finally the Table 4 presents how the raw output of the algorithm looks.

4.3 Noise resilience

For our experiments, we used a synthetic time series of length 10,000 containing 4 symbols with the embedded period size of 10. Symbols are uniformly distributed and the

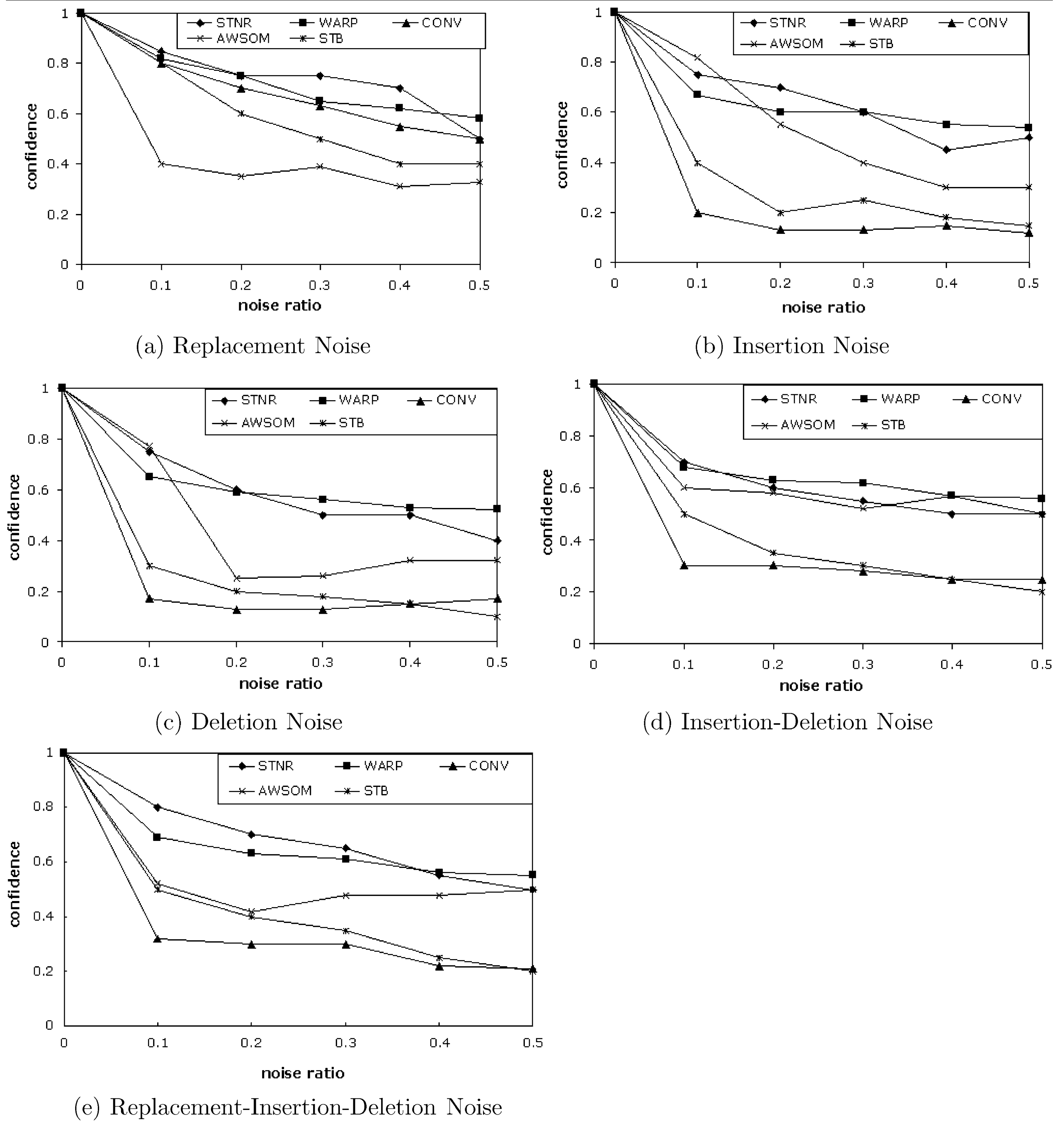

Fig. 8 Resilience to noise of STNR compared with WARP, CONV, AWSOM, STB

time series is generated in the same way as done in [2]. We introduced 7 combinations of replacement, insertion and deletion noise and gradually increased the noise ratio from 0.0 to 0.5, and note the confidence (periodic strength) at which the actual period of 10 is detected. The time tolerance window for all the experiments is ± 2. Figure 7 presents the result of this experiment.

For most of the combinations of noise, the algorithm detects the period at the confidence higher than 0.5. The worst results are found with the deletion noise, which is quite understandable; actually, the deletion noise not only disturbs the actual periodicity, but also affects the patterns in the data by removing some of the symbols at random.

In the next set of experiments, we compared the resilience to noise of our STNR algorithm with the other algorithms based on each of the 5 combinations of noise, i.e., replacement, insertion, deletion, insertion-deletion, and replacement-insertion-deletion. The results of this set of experiments are presented in Fig. 8.

The results plotted in Fig. 8 demonstrate the consistent superiority of our algorithm over most of the other algorithms described in the literature. Further, our algorithm compares favorably with WARP; it even performs better than WARP for certain noise ratios. This turns our algorithm into a more attractive choice because it is capable of detecting different types of periodicity.

5 Conclusion and future work

In this paper, we have presented a novel algorithm that uses suffix tree as the underlying structure and also employs the concept of time tolerance window for noise resilience. Our single algorithm can detect symbol, sequence (partial) and segment (full-cycle) periodicity as well as present the patterns that are periodic. The algorithm reports only the significant periods by ignoring the redundant and repeating periods. It is important to note that if there is a periodic pattern in the data, which satisfies our definition of periodicity, the algorithm would find it which means that during the period pruning, no useful period is ignored. Another important aspect of the algorithm is that it detects the redundant period ahead of time; before calculating its strength which saves a significant amount of time and is contrary to other approaches where the redundant periods are pruned after the main algorithm completion. We tested the algorithm on both real and synthetic data in order to test its accuracy, effectiveness of reported results, and the noise resilience characteristics. The algorithm uses the concept of *timetolerancewindow* where an occurrence found within a specified tolerance window is counted valid during periodicity detection. The algorithm can accommodate various combinations of noise and report the periods with acceptable degree of confidence. It performs well in comparison with existing algorithms. Besides higher noise ratio ($>= 0.5$), STNR performs better than existing approaches.

As future work, we plan to extend our work to consider the online periodicity detection in continuous stream of data. Moreover, we want to adapt the algorithm to detect the periodicity within subsequences of the time series as presented in [16]. Although the Ukonen's suffix tree construction algorithm has been proved to perform in linear order of space and time [3, 5], an online periodicity detection algorithm using disk-based suffix tree is our next goal. There are disk based implementations of the suffix tree construction [19, 20], which might be used to devise an online algorithm that can detect periodicity in very large time series database.

References

1. Indyk P, Koudas N, Muthukrishnan S (2000) Identifying representative trends in massive time series data sets using sketches. In: Proceedings of the international conference on very large data bases, Sept 2000
2. Elfeky MG, Aref WG, Elmagarmid AK (2005) Periodicity detection in time series databases. IEEE Trans Knowl Data Eng 17(7):875–887
3. Gusfield D (1997) Algorithms on strings, trees, and sequences. Cambridge University Press, Cambridge
4. Weigend A, Gershenfeld N (1994) Time series prediction: forecasting the future and understanding the past. Addison-Wesley, Reading
5. Ukkonen E (1995) Online construction of suffix trees. Algorithmica 14(3):249–260
6. Ma S, Hellerstein J (2001) Mining partially periodic event patterns with unknown periods. In: Proceedings of IEEE international conference on data engineering, Apr 2001
7. Yang J, Wang W, Yu P (2002) InfoMiner+: Mining partial periodic patterns with gap penalties. In: Proceedings of IEEE international conference on data mining, Dec 2002
8. Berberidis C, Aref W, Atallah M, Vlahavas I, Elmagarmid A (2002) Multiple and partial periodicity mining in time series databases. In: Proceedings of the European conf, artificial intelligence, Jul 2002
9. Grossi R, Italiano GF (1993) Suffix trees and their applications in string algorithms. In: Proceedings of South American workshop on string processing, Sep 1993, pp 57–76
10. Dubiner M et al (1994) Faster tree pattern matching. J Assoc Comput Mach 14:205–213
11. Kolpakov R, Kucherov G (1999) Finding maximal repetitions in a word in linear time. In: Proceedings of the annual symposium on foundations of computer science, pp 596–604
12. Al-Rawi A, Lansari A, Bouslama F (2003) A new non-recursive algorithm for binary search tree traversal. In: Proceedings of IEEE international conference on electronics, circuits and systems, vol 2, pp 770–773, UAE, Dec 2003
13. Elfeky MG, Aref WG, Elmagarmid AK (2005) WARP: time warping for periodicity detection. In: Proceedings of IEEE international conference on data mining, pp 138–145
14. Papadimitriou S, Brockwell A, Faloutsos C (2003) Adaptive, hands off-stream mining. In: Proceedings of the international conference on very large databases
15. Rasheed F, Alshalalfa M, Alhajj R (2007) Adapting machine learning technique for periodicity detection in nucleosomal locations in sequences. In: Proceedings of the international conference on intelligent data engineering and automated learning, IDEAL'07, Dec 2007, Birmingham, UK. LNCS series. Springer, Berlin
16. Wang Y, Zhou L, Feng J, Wang J, Liu Z-Q (2006) Mining complex time-series data by learning Markovian models. In: Proceedings of IEEE international conference on data mining, pp 1136–1140
17. Ahdesmäki M, Lähdesmäki H, Pearson R, Huttunen H, Yli-Harja O (2005) Robust detection of periodic time series measured from biological systems. BMC Bioinformatics 6:117
18. Glynn EF, Chen J, Mushegian AR (2006) Detecting periodic patterns in unevenly spaced gene expression time series using Lomb–Scargle periodograms. Bioinformatics 22(3):310–316
19. Cheung C-F, Yu JX, Lu H (2005) Constructing suffix tree for gigabyte sequences with megabyte memory. IEEE Trans Knowl Data Eng 17(1):90–105
20. Tian Y, Tata S, Hankins RA, Patel JM (2005) Practical methods for constructing suffix trees. VLDB J 14(3):281–299

Faraz Rasheed is a Ph.D. candidate in the Department of Computer Science at the University of Calgary. He received his B.Sc. in Computer Science from University of Karachi, Pakistan in 2003 and M.Sc. in Computer Engineering from Kyung Hee University, South Korea in 2006. He published several papers in international conferences in journals. His research interests include time series data mining, periodicity detection in time series databases and algorithm design and analysis.

Reda Alhajj received his B.Sc. degree in Computer Engineering in 1988 from Middle East Technical University, Ankara, Turkey. After he completed his B.Sc. 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 his M.Sc. and Ph.D. degrees in 1990 and 1993, respectively. Currently, he is Professor in the Department of Computer Science at the University of Calgary, Alberta, Canada. He published over 275 papers in refereed international journals and conferences. He served on the program committee of several international conferences including IEEE ICDE, IEEE ICDM, IEEE IAT, SIAM DM; program chair of IEEE IRI 2008, OSIWM 2008, SONAM 2009, IEEE IRI 2009. He is editor in chief of International Journal of Social Networks Analysis and Mining, associate editor of IEEE SMC—Part C and he is member of the editorial board of the Journal of Information Assurance and Security; he has been guest editor for a number of special issues and edited a number of conference proceedings. Dr. Alhajj's primary work and research interests are in the areas of biocomputing and biodata analysis, data mining, multiagent systems, schema integration and re-engineering, social networks and XML. He currently leads a research group of 7 Ph.D. and 9 M.Sc. candidates. Dr. Alhajj recently received with Dr. Jon Rokne donation of equipment valued at $5 million from RBC and Teradata for their research on Computational Intelligence and Bioinformatics research.